http://researchcommons.waikato.ac.nz/

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

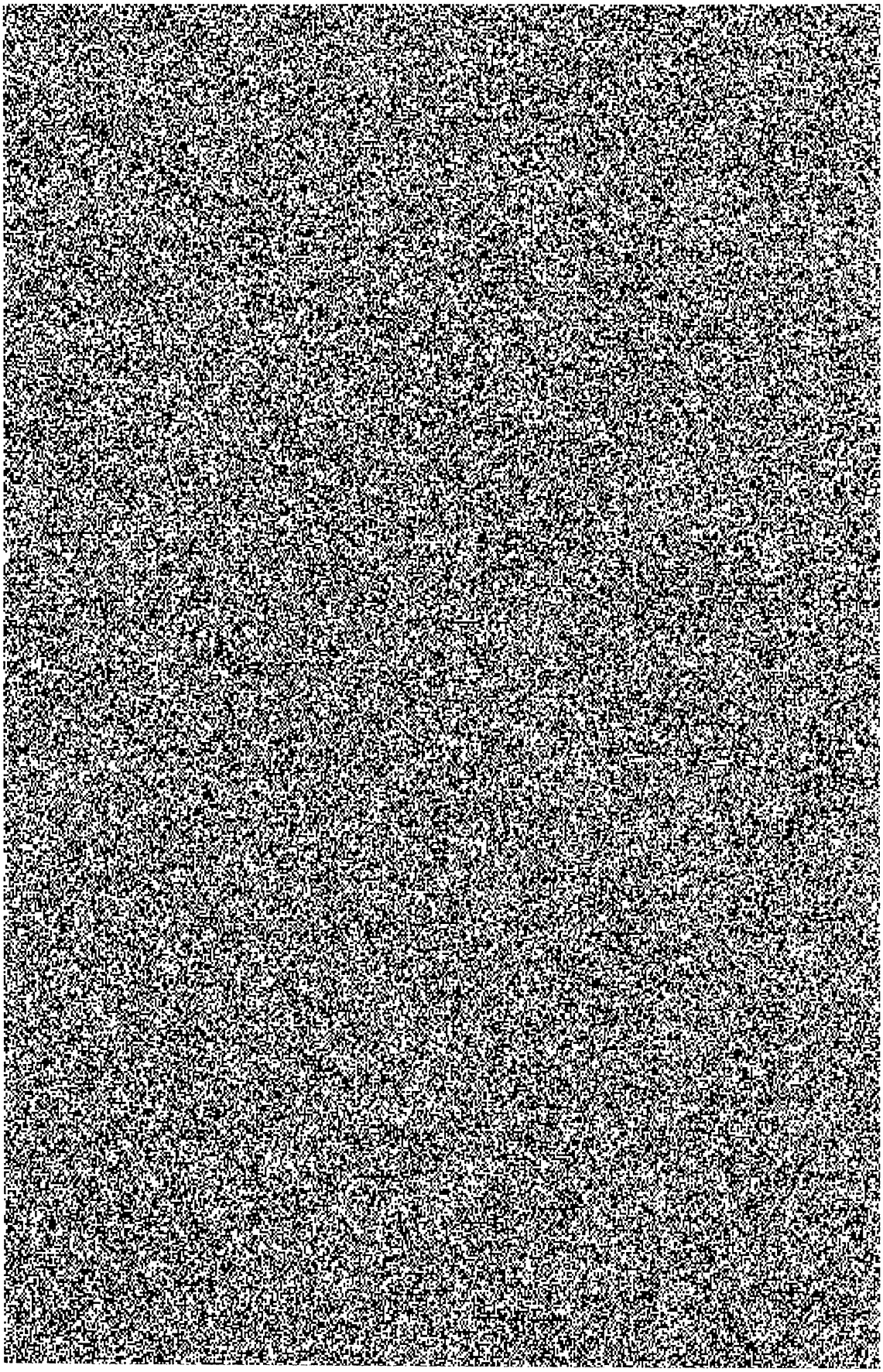
School of Computing and Mathematical Sciences
The University of Waikato
Te Whare Wānanga o Waikato
Hamilton, New Zealand

Compression and Cryptology

Sean A. Irvine

This thesis is submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy at The University of Waikato.

22 August 1997



Compression and Cryptology

by Sean A. Irvine

© 1997 Sean A. Irvine. All rights reserved.

Typeset using LaTeX on a NeXT workstation. The Gnuplot, Diagram, and Tailor applications for the NeXT were used in the preparation of illustrations. Printed at the University of Waikato, New Zealand. The frontispiece depicts a raster image of a PPM encoded file.

Abstract

Currently data compression and encryption are carried out as two separate activities. A simpler communication system would result if these two activities could be combined. The security properties of lossless data compression techniques are investigated. It is argued that compression is necessary for security but demonstrated that current compression techniques are not sufficient. Both coding techniques (arithmetic coding and Huffman coding) and modelling techniques (splaying, Ziv-Lempel, and prediction by partial matching—PPM) are examined. It is shown that all these systems have significant security flaws. Explicit attacks are given and descriptions of algorithms to automate the attacks are provided. Chosen plaintext attacks are given for Huffman coding, PPM, and splaying. Known plaintext attacks are given for arithmetic coding, splaying and Ziv-Lempel compressors. A ciphertext only attack is given for the Ziv-Lempel compressors. These results are used to highlight why better models of natural languages have important applications in cryptology. In particular, it is explained how a better model of a source leads naturally to increased resistance from dictionary and ciphertext only attacks.

Some of the attacks presented involve novel methods not previously reported in the literature. In particular, it is shown how a backtracking search can be used for known and chosen plaintext attacks against fixed to variable length binary encoders (Huffman coding and splaying). It is shown how a good model of the source (a PPM model) can be used in an automatic ciphertext only attack against simple substitution ciphers and Ziv-Lempel compressors. The attack against simple substitution ciphers is at least comparable to previously published automated approaches. In some cases analytical or asymptotic results concerning the complexity of the attacks are given. It appears that compression leads naturally to increased ciphertext only resistance, but current compressors contain no operations likely to thwart known or chosen plaintext attacks.

Acknowledgements

During the course of this research I have been in the fortunate position of having access to the facilities and expertise of both the Computer Science and Mathematics Departments of the University of Waikato. This has enabled me to consume more than my fair share of disk space (and free lunches). But more importantly it has given me two intersecting circles of friends and two points of view on many problems over the duration of this work. All of these people deserve my thanks.

I am deeply indebted to my three excellent supervisors Professor John Cleary, Doctor Ingrid Rinsma-Melchert, and Professor Ian Witten. Not only have they supplied me with the technical expertise and encouragement necessary for the successful completion of the thesis they have also been the instigators of many interesting and thought provoking conversations on a wide range of topics.

Special thanks are due to Doctor Colin Boyd, who while in New Zealand on leave from the University of Manchester (now at Queensland Institute of Technology), helped us understand error correction in arithmetic coding. Special thanks also to Doctor Brendan McKay of Australian National University for help in deriving some of the bounds for Ziv-Lempel coding. I thank Bill Teahan for helping with experiments using PPM on various occasions.

For the arduous task of proofreading I thank, in addition to my supervisors, Callum Irvine, Doctor Wayne Schou, and Bill Teahan.

Many of my fellow students and friends have helped in various ways, including those moments of relaxation when I was trying to avoid thinking about this thesis. The following spring to mind, my apologies for any glaring omissions: Julie Blake, Jeff Clarke, James Conwell, Sandy Davidson, Mark Hall, Brent Haskell, Dr Stefan Henton, Stuart Inglis, Callum Irvine, Glen Irvine, Heather Kennedy, Shane Legg, Dr Craig Nevill-Manning, Nicole More, Dr Wayne Schou, Raymond Sellars, Richard and Cassandra Shepherd, Tony Smith, Len Trigg, Dr Anping Wang, Yuchuan Wang, Derek Wong, Nan Zhu, and Alec Zwart.

Lastly, but not least, I thank my parents, without whom this thesis would never have happened.

Financially I have been supported primarily by two scholarships. I thank Professor Ian Graham for arranging the school scholarship, and Academic Board for the University of Waikato Postgraduate Scholarship. I also acknowledge the support of the Mathematics Department in providing sessional assistance work and for the opportunity to give some lectures.

Contents

Chapter 1

Looking After Information

Very few people in modern society live a day without using some electronic communication network: the banking system (automatic teller machines, electronic funds transfer, point of sale transactions), the telephone system, electronic mail, the World Wide Web, or cable television. The distinction between all these systems is becoming blurred with time, and it is not unusual to be able to access a service from any one of these sources. In the future we can look forward to widespread use of smart cards and digital cash, as well as additional network services. For the success of these global systems it is important that privacy and integrity of information be maintained. The broad subject dealing with these security issues is called ***cryptology***.

In the past, cryptology was considered by many to be a mystical activity, much like alchemy, astrology, dowsing, and exorcism. Until recently it was practised only by heads of state, religious leaders, lovers, spies, and of course the military. Indeed most of the major advances in cryptology have been made in times of war. The importance of cryptology to the military is the principle reason for the cloud of censorship which still hangs over the field. Even today many countries, including the United States and France, retain Draconian laws designed to prevent the dissemination of cryptologic research and security products.

Only since World War II, with the advent of computers and global communication networks, has open research on cryptology been carried out. Cryptology has now entered the commercial sector where it is used to maintain the integrity of modern financial systems, to keep commercially sensitive information confidential, and for authentication of digital information. In parallel there has been increased public awareness of privacy issues; many countries, including New Zealand [NZPSA93], now have legislation to protect aspects of an individual's privacy.

Communication systems are often vulnerable to eavesdropping and active attacks because they extend outside the jurisdiction of their users, do not always have a clearly defined owner, cover wide physical areas, are unnervingly complex, and often carry sensitive information. With many networks it is possible for someone to connect in at any

point and read, delete, modify, or insert information. Different types of network have different security risks. Many wire and optic based networks are specifically designed to make it easy for someone to connect onto the network at any point. For a microwave based network, it is nearly impossible for an attacker to delete or modify a message, but eavesdropping is very easy, and with a little more effort, so is inserting extra messages. In contrast, it is much harder to eavesdrop on a wire-based network since a physical connection to the network is required. However, once such a connection is established it is often all too easy to delete, modify, and insert messages.

Storage systems are also vulnerable. Even though they are easier for authorities to control, data tends to reside on storage systems for long periods of time, giving unauthorized personnel more time to attempt access to the information. Encryption alone is insufficient to guarantee security, since even though a thief might be unable to read the information on a stolen disk, it may be the only copy in existence, and hence could be used to extort money from its owners. Many other threats of this type exist. Anderson [And94] discusses how several automatic teller machine systems have been attacked. These attacks range from the outrageous on the thieves' part to downright stupidity on the banks part. For instance, one organized crime ring set up an entire branch of a bank complete with money machines, which they used to obtain account numbers and the corresponding personal identification numbers. In another case, the operations manual for a particular bank's money machines contained a fourteen digit number which would cause the machine to emit cash without a card (supposedly, to be used for testing the machine). Yet the bank concerned was surprised when money mysteriously started disappearing out of the machine.

Despite the exponential increases in the amount of network bandwidth available for the communication of digital information, the demand for faster and bigger links has not diminished, nor is it likely to in the near future. It is therefore hardly surprising that the benefits of data compression are widely recognized. As noted by Bell, Cleary, and Witten [BCW90], evidence suggests that users are always interested in achieving a two or threefold increase in capacity at no cost.

Data compression initially consisted of a number of ad hoc techniques, but in the last twenty years powerful and universal techniques have been developed. Many digital communications are now routinely compressed before transmission and decompressed at the receiver. Often this is actually faster than sending the original uncompressed data [WNC87].

The techniques discussed in this thesis are primarily about compressing and securing information flowing over a communication network, but are equally applicable to the storage of information. Security issues involved in actual processing of information or physical security issues like preventing disk theft are not considered. Likewise, some security issues, such as key management and detection of replay attacks, must be considered at a level above the techniques presented here.

1.1 The Thesis

It has been recognized that directly encrypting redundant sources leads to weaknesses; in particular, encrypting redundant sources (such as English text) leads to dictionary attacks [Luc88, Wil94]. Compression removes redundancy from a source and therefore is one way to protect a cryptosystem against this type of attack.

In this thesis the converse proposition that *data compression techniques can be used to provide security* is investigated. If this were to prove true it would be a useful and practical result as the two currently separate activities of compression and encryption could be combined, resulting in a simpler and faster communication system. Unfortunately, proving security is a very tricky business. Proofs are only available for a few algorithms, and then only by making (reasonable) assumptions about the computational difficulty of certain problems.

We fail to uphold the thesis; all of our results show significant security flaws in the data compression schemes investigated. Cursory examination of many data compressors hinted at reasonable security. However, every system closely examined was found to have weaknesses. In many instances various ways of strengthening the security of the compressor are apparent, but such embellishments would likely detract from the overall goal of having a simpler communication system. Further, most enhancements while of obvious cryptologic origin do nothing to increase the security of the compressor itself. This is made clear by noting that the security benefits of the enhancement can be considered or argued for independently of the compressor. The more useful enhancements are complex enough that one is better off continuing to use a well-understood cryptosystem.

Several contributions are made in the thesis. Considerable negative evidence for the proposition is presented, to the point where the proposition is no longer tenable. This evidence consists primarily of a variety of different attacks on several different cryptosystems. The methods of attack themselves represent another contribution. In particular, both the backtracking strategies (used in Chapter 6) and the use of compression models to break cryptosystems (used in Chapter 7 and 8) represent novel kinds of attack. These methods are general enough that they should find application against other cryptosystems. The importance of having a good model of the source being transmitted is highlighted.

1.2 Extended Summary

It is argued that compression is necessary for security but demonstrated that current compression techniques are not sufficient. Both coding techniques and modelling techniques are examined. It is shown that arithmetic coding, Huffman coding, splaying, Ziv-Lempel compression, and prediction by partial matching all have significant security flaws. Explicit attacks are given and descriptions of algorithms to automate the attacks are provided. These arguments and results are developed in the following way.

In Chapter 2 classical and modern cryptology are reviewed. Classical systems are discussed primarily so that various kinds of attack can be introduced. This is followed by a discussion of practical modern systems applicable to electronic communication. Directions of current research in general cryptology are examined.

Chapter 3 introduces data compression and relevant information theory. Several modern data compression techniques are reviewed and explained. The distinction between the compression model and the coder is highlighted. Elementary security considerations are mentioned.

Chapter 4 deals with ciphertext only attacks against compressed files. The results are primarily empirical and serve mainly to indicate that compressors are not as good at removing redundancy as expected. Various statistical tests of randomness are applied and the results indicate that some compressors, in particular those based on Ziv-Lempel coding, do not result in random outputs, which is the expected output of a perfect compressor.

The security of arithmetic coding is discussed in Chapter 5. The results show that a static arithmetic coder is insecure, and that arithmetic coding is equivalent to the subset sum problem. These results imply that most security must come from the model.

Chapter 6 presents the first complete cryptanalysis of a compression system, the splay tree compression algorithm of Jones [Jon88]. A simple known plaintext attack is presented and is shown to run in linear time for certain sequences. The attack is also shown to be applicable to Huffman coding.

Chapter 7 discusses the security of prediction by partial matching (PPM). The results of Bergen [BH92, BH93] are reviewed and extended in several ways. It is also shown how the PPM data compressor can be used to break simple substitution ciphers.

Chapter 8 details attacks against the Ziv-Lempel family of cryptosystems. Analytic results are derived for a known plaintext attack. The PPM attack against simple substitutions is extended to a ciphertext only attack against Ziv-Lempel coding. The importance of having good models is emphasized since these attacks are possible because the attacker has a better model of the source than the compressor.

Chapter 9 collects the results together and their overall significance with relation to the thesis is considered. Directions for further research are included.

1.3 Comments on Jargon and Notation

Like any specialized subjects, data compression and cryptology come with their own special jargon. The terminology used in this thesis is a mixture from the two subjects, although we probably draw more heavily on the jargon of cryptology than that of data compression. An attempt has been made to keep our terminology as generic as possible. For example, by using the term 'encoders' rather than encrypters or compressors. Most of the terminology used is defined in Chapters 2 and 3.

The mathematical notation used in the thesis follows standard conventions but is included here for completeness.

If S is a set then $|S|$ denotes the cardinality of the set S. We use $\lfloor x \rfloor$ to denote the floor function of x and $\lceil x \rceil$ to denote the ceiling function, $\lg x$ for $\log_2 x$, $\mathbb{B}$ to denote a binary alphabet $\{0,1\}$, $\mathbb{Z}$ to denote the set of integers, $\mathbb{Z}^+$ to denote the set of positive integers, $\mathbb{N} = \mathbb{Z}^{0+}$ to denote the set of nonnegative integers, and $\mathbb{R}$ to denote the set of real numbers. Bitwise exclusive-or is represented by $\oplus$ and bitwise equivalence by $\equiv$. Pr, Var, and E are used to denote probabilities, variances, and expectations, respectively. The greatest common divisor of two integers is denoted by gcd and the least common multiple by lcm.

On occasion order notation is used. The relevant definition is included below. Details regarding order notations can be found in standard texts dealing with algorithms [BB88, CLR90, Knu69, Sed88].

DEFINITION 1.1: We say that f is ***big oh*** of g, written $f = O(g)$, if there exist constants c and n_1 such that $f(n) \leq cg(n)$ for all $n \geq n_1$.

Some common complexity classes are also referred to. Let **P** denote the set of problems solvable in polynomial time on a standard Turing machine; and **NP** (nondeterministic polynomial) denote the set of problems whose solutions can be checked in polynomial time. It is known that $\mathbf{P} \subseteq \mathbf{NP}$ [Pap94]. The reverse inclusion is widely believed to be false. Finally, **NP**-complete denotes the set of problems polynomially equivalent to three-satisfiability [GJ79]. The **NP**-complete problems represent the hardest problems in **NP** and have no known polynomial time algorithms. However, if a polynomial time algorithm was found for any one of these problems then all of them could be solved in polynomial time. Detailed explanations of these complexity classes are given in [GJ79, Pap94].

When a sequence consists of symbols drawn from an alphabet S we may write $s = s_0 s_1 \cdots s_{n-1}$ where $s_i \in S$ and n is the length of the sequence. Further, we write $s \in S^*$ and $|s| = n$. When the length is clear from the context, or is not important, the superscript is omitted and we write $\mathbf{s}$. The empty string is denoted by Λ. To show concatenation the strings are simply written one after the other. By a^n we mean symbol a repeated n times. The symbol $\coprod$ is used in a manner analogous to the $\prod$ used for products; thus

$$\coprod_{i=1}^{n} a_i = a_1 a_2 \cdots a_n.$$

If S is a binary sequence then $\overline{S}$ denotes the complement of S; for example if $S = 0110111$ then $\overline{S} = 1001000$. The symbol $\sqcup$ is used to denote a space, and $\hookleftarrow$ is used to denote a carriage return.

Some of the experiments discussed are applied to the *Calgary Corpus* [BCW90]. This corpus contains fourteen files as detailed in Figure 1.1. In addition *The Hobbit* [Tol66] was used and dubbed book3.

Title	Bytes	Brief description
bib	111261	ASCII text, bibliography
book1	768771	ASCII text, Thomas Hardy's *Far from the Madding Crowd*
book2	610856	ASCII text, Ian Witten's *Principles of Computer Speech*
geo	102400	32-bit numbers, seismic data
news	377109	ASCII text, usenet news
obj1	21504	VAX executable
obj2	246814	Apple Macintosh executable
paper1	53161	ASCII text, technical paper
paper2	82199	ASCII text, technical paper
pic	513216	CCITT facsimile test image
progc	39611	C source code
progl	71646	Lisp source code
progp	49379	Pascal source code
trans	93695	EMACS terminal session transcript

Figure 1.1: The Calgary Corpus.

Chapter 2

Introduction to Cryptology

The subject of cryptology is the study of security, and can be further subdivided into two main branches. ***Cryptography*** is the design and provision of security systems and **cryptanalysis** is the breaking of such systems.

In this chapter various cryptosystems, some secure but most insecure are reviewed. They serve to demonstrate the large number of techniques used by cryptographers and the equally large repertoire of attacks available to the cryptanalyst.

Section 2.1 introduces the terminology needed in the remainder of the chapter. Modern cryptology has many aspects other than encryption, and although this thesis is primarily concerned with encryption, other activities are reviewed in Section 2.2. Generic attacks are discussed in Section 2.3. This leads, in Section 2.4, to the presentation of various classical cryptosystems which serve mainly to identify the kinds of weaknesses that have been exploited in the past. In particular, the dangers of using a redundant source is highlighted in various ways. Section 2.5 discusses modern cryptosystems. These systems are believed to be reasonably secure but cryptanalysis is given where appropriate. Section 2.5 also serves to introduce public-key cryptosystems and the possibility of cryptography based on quantum mechanics. A brief summary is given in Section 2.6.

2.1 Special Terminology

Later, communication systems involving both data compression and encryption will be examined and the terminology used reflects our aims. Figure 2.1 depicts the relevant operations and information flows.

The combined operation of compressing and encrypting information is called ***encoding*** and the combined operation of decompression and decrypting is called ***decoding***. The ***plaintext*** (or ***message***) is the data prior to encoding. The output of encoding is called the ***ciphertext***. Decoding the ciphertext restores the plaintext. The plaintext can be any digital data; it might already be compressed or encrypted. The ***encoder*** is a set of

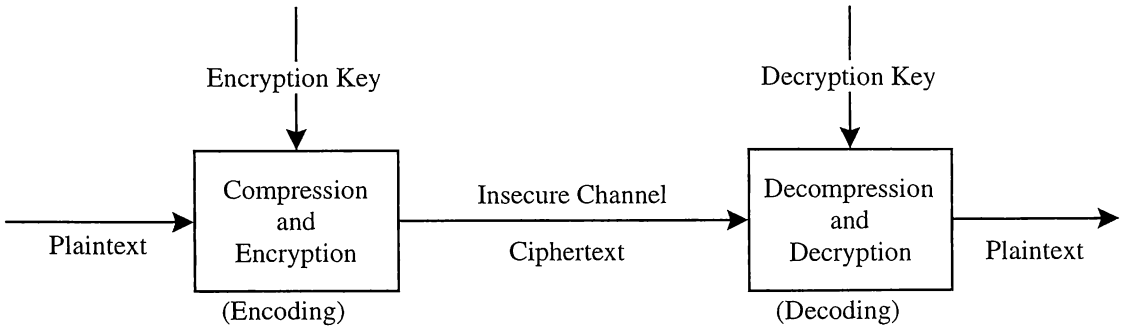


Figure 2.1: A communication system using both data compression and data encryption techniques.

functions which map plaintexts to ciphertexts. A particular ***encoding function*** is selected by the ***encryption key***. The encoding operation is denoted by $C = E_{k_0}(M)$ where M is the plaintext, C is the ciphertext, k_0 is the encryption key, and E_{k_0} is the encoding function. For each encoding function there exists an inverse encoding function called the ***decoding function***, which uses the ***decryption key***, k_1 to reverse the effect of the encoding function; thus $D_{k_1}(C) = M$ and $D_{k_1} = E_{k_0}^{-1}$. Collectively, the decoding functions form the ***decoder***. The design of encoders and decoders is called ***cryptography***, the attempted breaking of encoders and decoders is called ***cryptanalysis***. Jointly, these activities are called ***cryptology***.

In many systems the encryption and decryption keys are the same, $k_0 = k_1$. Such systems are called ***symmetric***, and in such a system it makes sense to talk about the ***key*** $k = k_0 = k_1$; otherwise the system is ***asymmetric***. The Data Encryption Standard (DES) [NBS77] is a well-known example of a symmetric system. Public-key cryptosystems [DH76b] are examples of asymmetric systems.

The security offered by any encoder is dependent only on the keys k_0 and k_1; it is assumed that an attacker is aware of the mechanisms of the encoder and decoder. This makes sense because changing keys is much simpler than changing the entire system.

The set of possible keys is called the ***key space***, the set of possible messages the ***message space***, and the set of possible ciphertexts the ***ciphertext space***. For many systems, two or all of these spaces are the same. The key space should be large enough so as to make it infeasible for an attacker to try every possible key in a reasonable amount of time. Often binary or textual spaces will be used, but all the techniques discussed are sufficiently general to be applicable to any data with a digital representation.

Modern electronic communication employs many more symbols than the twenty-six capital letters typically used in classical systems. Most computers use the American Standard Code for Information Interchange (ASCII) alphabet. The ASCII alphabet contains both upper and lowercase letters, digits, various punctuation and mathematical symbols, a space symbol, and several nonprinting codes used for formatting and other special pur-

poses.

When using a communication network, it is normal to encode incrementally. This poses some restrictions on the type of encoders and decoders which can be used. In extreme cases a message may have to be encoded bit by bit, but it is more common for the system to operate on a symbol-by-symbol basis. Operating systems often provide automatic buffering to networks so that practical encoders and decoders can work with multiple symbols at once if necessary.

A somewhat arbitrary distinction is drawn between a ***block encoder*** and a ***stream encoder***. A stream encoder operates on a symbol-by-symbol basis and a block encoder operates on several symbols at once.

A typical block encoder takes a n-bit message and produces a n-bit ciphertext; that is, the 2^n possible messages get mapped onto 2^n possible ciphertexts. Effectively this is just a permutation chosen by the key out of $(2^n)!$ possible permutations. However, since the cardinality of the key space is typically less than $(2^n)!$, not every permutation is possible. If the encoder also performs compression then the output ciphertext could be shorter or longer than the plaintext.

In cryptology no attempt is made to hide the existence of a message, the security comes from the key and encoder. Another activity, ***steganography***, offers security by trying to hide the existence of a message, for example, by writing in invisible ink, or by hiding the message in a picture. Steganographic techniques are used in modern digital systems to mark documents in hidden ways so that illicit copying and dissemination can be traced [BLMG94, Max94, KZ95]. There also exist programs to hide a message in common picture file formats. Figure 2.2 is an example of a steganographic message.[1]

2.2 Cryptologic Activities

Originally, cryptology involved only the encryption of messages so that they could be transmitted or stored securely; but modern cryptology has protocols for many security issues. The formal protocols for these activities are often quite complicated, although many are based on relatively simple concepts. For example, the technique of ***cut and choose*** frequently used by children (and sometimes adults) to divide anything fairly is found in some protocols. Many of these protocols do for the electronic world what has been possible, and in many cases trivial, in the paper world. The following list of activities represents current area of research in cryptology. Protocols for the activities are discussed more fully in [Sch94].

- ***Anonymous Message Broadcast***: A protocol invented to solve the ***Dining Cryptographers Problem*** [Cha88]:

[1] Reproduced from [DP84] by permission of John Wiley & Sons Ltd.

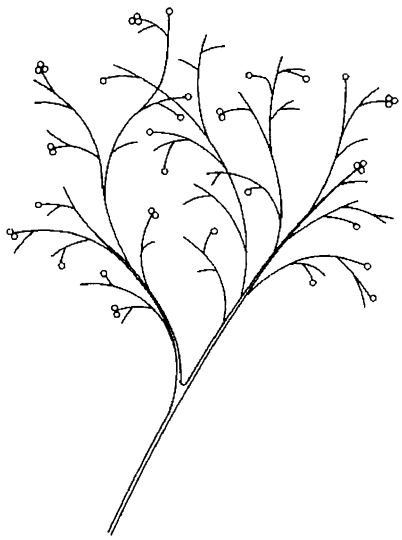


Figure 2.2: Example of steganography. Here a binary message is hidden in a plant like picture. Starting with the stem the message can be read off in a clockwise direction to give 1001 0011 0010 1001 1100 0011 0000 0111 0010 1101 1010 0010 0011 1100 0110.

> Three cryptographers are sitting down to dinner at their favourite three-star restaurant. Their waiter informs them that arrangements have been made with the maître d'hôtel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been the NSA [National Security Agency]. The three cryptographers respect each other's right to make an anonymous payment, but they wonder if the NSA is paying.

The problem is solved in the following way:

> Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer to his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see—the one he flipped and the one his left-hand neighbour flipped–fell on the same side or on different sides. If one of the cryptographers is the payer he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number of differences indicates that NSA is paying (assuming that the dinner was paid for only once). Yet, if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.

- ***Authentication***: Modern cryptology also deals with access control for computers, automatic teller machines, and so on, and the specification of password algorithms. Passwords themselves are no longer stored, instead the image of the password under a one-way function is stored. This is an important application of cryptology because current password methods are not particularly good. For example, there exists

programs that will crack about 21% of the passwords on an average UNIX system in a week [Kle90] (see [Sch94] for a survey).

- ***Bit Commitment***: Bit commitment is the prediction of or commitment to a bit which is not revealed until a later time. The bit cannot be changed in the intervening time. Bit commitment protocols allow coins to be flipped and poker to be played over the telephone [Blu82, GM82].

- ***Computing with Encrypted Data*** (Hiding information from an oracle): In these protocols a user can have a function evaluated at x by someone else such that the other person does not learn the value of x. Currently this is only possible for a few functions [AFK89].

- ***Digital Money***: Credit cards and point-of-sale cards are useful but have their limitations. In particular they always leave an audit trail. Many people feel the need to be able to exchange money without a trace. Digital money protocols allow such transfers and provide for the detection of counterfeit money [Cha85, CFM90, Hay90].

- ***Digital Signatures***: Digital signatures are a major part of modern cryptology. Protocols for digital signatures provide the digital equivalent of a handwritten signature. To be useful a signature needs to be unforgeable, unalterable, nonreuseable, and nonrepudiable. Various protocols achieve these goals to differing degrees. Extensions to the basic digital signature include ***undeniable digital signatures*** [CA90], ***fail-stop digital signatures*** [PW90], and ***group signatures*** [Cha91].

- ***Key Exchange***: Prior to secure communication the sender and receiver must agree on a key to use during communication. Traditionally this was a particularly thorny problem, and the key was assumed to be transported by a secure channel, usually a trusted courier. Preparation for communication in advance of the actual communication was required; so it was necessary to know in advance who you were going to communicate with. Further, this always begged the question of why the secure channel couldn't be used for all messages. The invention of public-key cryptography made it possible for two strangers to agree on a key for future communication over an insecure channel in such a way that no one else learned the value of the key (see [Sch94] for a survey).

- ***Multiparty Computation***: These protocols allow the value of a function to be calculated with unknown inputs [Sal90]. It can solve problems like enabling a group of people to determine their average salary without anyone learning the salary of anyone else. It can allow two people to decide who is older without either age being revealed.

- ***Oblivious Transfer***: This protocol is used when two possible messages exist (say one meaningful and one nonsense). The recipient is guaranteed of receiving exactly

one of them and the sender cannot tell which message was received [Kil90]. This can be used when you wish to send someone only half the bits of a certain number (for example, to help someone factor a large number but to not actually reveal the factor).

- ***Random Sequence Generation***: Generating unpredictable random sequences is actually a very interesting and challenging problem. Several methods exist for producing sequences with certain statistical properties [Knu81, Gol67, BBS86, BP82]. However, many of the these methods are simple rules and although the sequences show 'statistical randomness', they are predictable and are not suitable for use in cryptology. Many cryptologic protocols require long sequences of unpredictable (pseudo)random bits and considerable research has gone into producing such sequences. Powerful tests of randomness have been formulated. Randomness is considered in more detail in Chapter 4.

- ***Secret Sharing***: Secret sharing allows a message to be divided up amongst a number of agents so that no single agent can reconstruct the message [Bla79, Sha79b]. Various schemes exist depending on the number of agents and the number needed to reconstruct the message. In the case of just two agents this is called ***secret splitting***.

- ***Secure Elections***: Before computers can be used in general elections it must be proven that the protocols prevent cheating and maintain the privacy of the voters. In the ideal case only authorized voters can vote, they can only vote once, individual votes remain secret, and nobody can change anybody else's vote. A multitude of protocols satisfying these conditions to differing degrees have been proposed (see [Sch94] for a survey).

- ***Simultaneous Contract Signing***: Two people have agreed to a contract but neither wishes to be the first to sign. Traditionally this is solved by having the contract signed simultaneously in a face-to-face environment or by using a third person to arbitrate. Electronically this problem can be solved by having the parties sign the contract incrementally. Essentially each party signs the contract with probability p, as the protocol progresses p tends towards 1 until, when $p = 1$, both parties are fully committed to the contract. A modification to simultaneous contract signing allows 'registered mail' to be sent electronically. In this protocol, called ***digital certified mail*** [Dif77], the recipient must return a receipt for the mail. The protocol is fair in that the message cannot be read nor receipt confirmed until both are received. The protocol can be generalized to enable the ***simultaneous exchange of secrets*** [EGL85].

- ***Subliminal Channels***: The notion of a subliminal channel is the idea of communicating a secret message using a seemingly innocuous message [Sim84, Sim85]; and is therefore associated with steganography. A simple example of a subliminal channel

is the number of words in a sentence. Better subliminal channels include a key. The subliminal channel can also be used to mark documents in particular ways which if necessary can be revealed at a later date. Subliminal channels are often found in digital signature schemes; although there exist digital signature schemes which have been proven to be free of subliminal channels [Des90].

- ***Timestamping***: This is like digital signatures but concerned with dates rather than personal signatures [HS91].

- ***Zero-Knowledge Proofs***: This is a collection of several different techniques for convincing someone that you know a piece of information without actually revealing any of the information [GMW86, GMR89, BCD88].

2.3 Attacks Against Cryptosystems

When breaking a system the attacker can be interested in obtaining the plaintext for a given ciphertext or in finding the key used to create the ciphertext. In general, finding the key is the more rewarding activity since a key may have been used to encode several messages.

As mentioned earlier it is usual to assume that an attacker has full knowledge of the algorithm in use. Such knowledge may be obtained by reading the appropriate literature, reverse engineering, theft, and so on. However, the ability of the attacker may be limited in other ways which are independent of the algorithm. For example, the attacker may have limited computational ability (so as to be unable to perform exhaustive key searches in reasonable time), or may only be able to intercept and not transmit messages. These restrictions are used to classify various kinds of attack.

- ***Exhaustive search.*** In this attack the cryptanalyst tries decoding ciphertext with each possible key in turn. The difficulty of the attack is proportional to the size of the key space, thus a large key space can make this attack infeasible. Immunity to the attack is also possible if multiple decryptions correspond to meaningful messages.

- ***Ciphertext only attack.*** The attacker has some ciphertext from which the plaintext or key is to be obtained. The difficulty of this task is dependent on the length of the ciphertext available and the redundancy present in the ciphertext. If there is no redundancy in the ciphertext then this class of attack will yield no information about the plaintext (other than its length). Attacks of this class make heavy use of statistics (about the source language) and often involve guessing likely pieces of plaintext.

- ***Known plaintext attack.*** In this case some of the guessing in the ciphertext only attack is removed because the attacker has some plaintext known to correspond to

intercepted ciphertext. For many classical systems this extra information allows the key to be trivially determined. Obtaining this extra information is much easier than one might expect—many such instances occurred during and prior to World War II [Kah66]. These attacks vary from a portion of plaintext known to occur somewhere in the ciphertext to knowing the exact correspondence between some plaintext and ciphertext.

- ***Chosen plaintext attack***. In this case the attacker is able to choose messages to be encoded. This enables the attacker to choose particular plaintext which might increase the chance of determining the key. In the more powerful ***adaptive chosen plaintext attack***, a number of such messages may be encoded with each new message dependent on all previous chosen-plaintexts.

- ***Chosen ciphertext attack***. In this attack, the cryptanalyst can choose different ciphertexts to be decrypted and obtain the resulting plaintext. This attack is used only when the attacker wishes to determine the key being used. Public-key cryptosystems are often susceptible to this attack.

- ***Chosen key attack*** (related-key cryptanalysis). An attack where the cryptanalyst knows the key in advance [Bih92]. It is used to investigate the effect of different keys on the same message during the design of a cryptosystem. Ideally flipping one bit of the key should result in roughly half the bits of the ciphertext being flipped.

A ***passive attack*** against a system is merely an attempt to read what is being transmitted or stored. The more ominous ***active attack*** is when an effort is made to insert, delete, modify, or replay messages.

When assessing the security of modern systems it is usual to assume that a known plaintext attack is possible, and in most cases that a chosen plaintext attack is also possible. Security is always a matter of degree and the importance of the message to be transmitted will dictate the precautions to be taken. The strength of a cipher is a negative quality in that security depends on the inability of cryptanalysts to find a feasible way of breaking it. One day it may be possible to use complexity theory to prove the difficulty of breaking ciphers, but this is not currently possible. Currently, the best one can do is to show that breaking a cryptosystem is equivalent to solving certain computational problems generally agreed to have no polynomial-time solution. Such problems include factoring integers, extracting discrete logarithms, and the various **NP**-complete problems. Strictly speaking, showing equivalence to a **NP**-complete problem should only be a preliminary step. The subtlety is that security requires the vast majority of problem instances be hard, but often **NP**-complete problems have large subclasses for which fast algorithms exist. An example of this occurs in Chapter 5.

2.4 Classical Cryptosystems

This section gives several examples of older cryptosystems but is in no way meant to be exhaustive. The examples are used to illustrate the various types of encoder possible and the attacks which can break them. Kahn [Kah66] gives a detailed and entertaining history of cryptology and cryptosystems up to and including World War II. Analysis of these systems is also given by Denning [Den82].

2.4.1 Simple Substitution

In a ***simple substitution*** each symbol of the alphabet is replaced by another predetermined letter of the alphabet. Using the substitution:

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

the phrase BILBO BAGGINS[2] would be encoded as ROPRJ RIDDOTE. The permutation selected is considered to be the key.

For an alphabet with n symbols there are $n!$ possible permutations. With such a large number of possible substitutions an ***exhaustive search*** for the correct permutation is infeasible, even for $n = 26$. However, simple substitutions do nothing to obscure the statistical properties of the source language and the cryptosystem is easily broken by ***frequency analysis***.

Figure 2.3 lists the letter frequencies for three English texts. Provided a simple substitution ciphertext is long enough it is a trivial exercise to match the letter frequency distribution of the ciphertext with that of English and thereby determine both the corresponding plaintext and the key. This is still true even if the symbols of the ciphertext alphabet differ in appearance from those of English. Of course a few possibilities may have to be tried before the correct substitution is found. Even for the three books in Figure 2.3 there is some variation in the distributions.

The cryptanalyst has access to much more information than just the frequency of individual letters. A cryptanalyst can also make use of higher-order statistics, such as digram and trigram frequencies. Figure 2.4 gives the digram frequencies for `book3`. In order of frequency the twenty most common digrams (excluding those containing space) are: TH, HE, IN, AN, ND, ER, RE, OU, NG, HA, ED, ON, AT, OR, TO, AR, AS, EN, HI, and ES. Notice that many digrams occur more frequently than some individual letters. Other digrams such as QJ are exceedingly rare in English (although it does occur in this thesis!). Because letter frequencies are different for different languages, given a long enough ciphertext, it is often possible to determine the language of the plaintext before actually attempting to decode the ciphertext! The concept of a digram generalizes to an n-gram,

[2]Bilbo Baggins is a character from the writings of J. R. R. Tolkien.

Letter	book1	book2	book3	Total	%
␣	142173	103388	97218	342779	19.18
E	72875	56989	48765	178629	10.00
T	51993	42382	36402	130777	7.32
A	48803	33790	32422	115015	6.44
O	45651	31872	30792	108315	6.06
N	41421	32827	27787	102035	5.71
I	39906	35553	24462	99921	5.59
S	37638	32006	22960	92604	5.18
H	38538	20624	26724	85886	4.81
R	33134	28182	22014	83330	4.66
D	26892	14793	20137	61822	3.46
L	23491	19034	17644	60169	3.37
U	16134	15035	10824	41993	2.35
C	13265	17984	6860	38109	2.13
M	14609	11394	8559	34562	1.93
F	12650	12019	9027	33696	1.89
W	14824	5902	10824	31550	1.77
G	12878	7380	10356	30614	1.71
P	10025	13611	5310	28946	1.62
Y	12402	6985	7404	26791	1.50
B	10595	7215	7837	25647	1.43
V	5446	5025	3542	14013	0.78
K	5039	1963	3833	10835	0.61
X	866	2519	302	3687	0.21
Q	534	1372	317	2223	0.12
Z	264	1035	227	1526	0.09
J	721	43	325	1476	0.08
Total	732767	561309	492874	1786950	100.00

Figure 2.3: The frequencies of the letters in three English novels.

Figure 2.5 lists every n-gram occurring more than 1000 times in `book3` (this time the alphabet is full ASCII).

Frequency analysis works by exploiting statistical regularities or redundancy in the source. If a source were to emit letters of the Roman alphabet with uniform probability and with no correlation between symbols then a simple substitution would give unconditional security against ciphertext only attacks. This is because each possible decryption is equally likely for each possible key. Since compression removes redundancy from a source, it is immediately apparent why compression is advocated prior to encryption.

Techniques have been developed for the automatic solution of simple substitutions (applied to redundant sources) which are effective even on short ciphertexts [Har94, Luc88, PR79]. In Chapter 7 a new method for the automatic solution of such ciphers, using a compression technique, is presented.

2.4.2 The One-Time Pad

It is possible to increase security of a simple substitution by using digram or trigram substitutions. The Vigenère, Playfair, and Beaufort ciphers are examples of digram substitutions [Kah66, Den82]. These ciphers can also be broken by statistical attacks, although longer ciphertexts are required.

Additional security can be gained by using different substitutions for different symbols. Such systems are called ***polyalphabetic substitutions***. In the extreme case where a different substitution is applied to every symbol, ***unconditional secrecy*** is achieved. This system is called a ***one-time pad*** or Vernam cipher and is completely secure even from an attacker with unlimited computational power. No extra information (except the length) can be gained from the ciphertext. The one-time pad is used on the Washington-Moscow hotline and by the Russian secret service. The disadvantage of the one-time pad is that the key is the same length as the message. The problem is not so much in communicating the length of key required but in the actual generation of such keys. Considerable research has gone into developing ways to produce (nearly) random sequences from a smaller key.

One way of implementing a one-time pad encodes the message according to the function $c_i = m_i + k_i (\mathrm{mod}\, n)$ where n is the number of symbols in the alphabet, m_i, c_i, and k_i are the ith message, ciphertext, and key symbols respectively. For the binary case the decoder is exactly the same function and can be implemented using exclusive-or, $c_i = m_i \oplus k_i$. It is important that the key is never reused. Reused key bits lead to detectable statistical regularities from which the reused portion of the key may be recovered.

2.4.3 Transposition Ciphers

In a ***transposition cipher***, the content of a message is obscured by rearranging groups of symbols; a transposition is thus a permutation. Originally, matrices were used. The

	␣	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	Total
␣	1609	2027	54	78	13268	18711	3457	3602	2168	966	1	1287	2460	2053	6811	4314	850	1	5364	10317	10527	831	1	1107	17	5335	2	97218
A	11658	6	770	1055	817	2613	620	758	4024	124	16	10	1276	1048	258	302	579	0	1037	1134	996	147	162	2903	10	28	71	32422
B	5386	512	250	0	20	48	0	4	9	87	0	3	560	293	12	501	1	0	25	2	13	83	0	3	0	25	0	7837
C	2958	865	1	46	9	501	3	1	3	744	0	5	17	2	499	227	0	0	213	221	149	313	0	3	75	5	0	6860
D	3182	1799	0	37	229	3833	1	8	8	1521	0	4	1238	5	6570	591	9	0	854	11	9	221	0	5	0	2	0	20137
E	1827	2	2009	863	1817	1581	550	933	13608	736	28	1208	2585	2241	2083	97	1137	0	5545	2064	2131	250	3042	1913	28	391	96	48765
F	3800	317	1	0	48	497	343	6	5	547	0	37	380	54	39	2772	1	0	95	8	20	45	0	8	0	4	0	9027
G	2711	884	2	0	107	277	3	199	0	844	0	5	7	0	4169	88	1	0	224	14	9	808	0	0	0	4	0	10356
H	7078	20	0	1081	5	128	4	1403	2	1	0	2	1	0	31	8	17	0	50	1076	14321	0	0	1485	1	10	0	26274
I	5042	1713	979	238	707	672	634	440	3259	2	1	600	1930	529	677	376	390	0	1749	883	1365	279	235	1597	16	134	15	24462
J	305	2	2	0	2	1	1	0	0	0	0	0	0	0	8	1	0	0	3	0	0	0	0	0	0	0	0	325
K	600	520	0	961	7	42	3	0	0	285	0	1	131	0	330	434	1	0	327	183	3	0	0	5	0	0	0	3833
L	2729	2553	793	316	191	1700	321	312	18	1952	0	61	2952	12	318	922	464	0	164	289	415	1033	0	102	0	13	13	17643
M	2989	882	0	1	15	1104	4	4	10	1077	0	0	53	93	13	1524	2	0	177	266	12	322	0	5	0	6	0	8559
N	2416	7142	0	1	55	3299	3	46	1	7779	0	300	6	26	247	3668	1	0	512	99	28	1609	0	548	0	1	0	27787
O	5813	3	1245	1306	1114	164	1673	1333	1946	376	92	11	1527	1119	2149	1635	546	0	1973	1255	3400	1	78	894	0	1136	3	30792
P	1760	465	0	0	1	454	3	7	3	111	0	1	89	266	18	499	381	0	75	509	6	557	0	1	99	5	0	5310
Q	262	0	0	1	0	9	0	2	1	2	0	0	0	0	5	0	0	0	0	35	0	0	0	0	0	0	0	317
R	1759	3389	366	440	458	5825	646	705	304	1190	1	2	96	70	5	3566	400	0	455	2	842	1417	0	67	0	9	0	22014
S	6571	3371	29	4	571	3149	21	347	38	2482	0	185	364	250	1015	630	131	0	1002	764	612	1079	1	157	0	187	0	22960
T	16563	3576	30	240	4	1150	433	25	951	3100	0	3	216	2	1950	1656	207	0	667	3009	678	1812	0	6	55	89	0	36402
U	1108	317	980	181	63	3	297	215	291	1	186	4	286	276	42	4675	160	316	410	534	467	0	0	9	1	2	0	10824
V	500	748	5	0	94	681	0	0	0	402	0	2	261	3	53	415	0	0	372	0	0	4	0	1	0	1	0	3542
W	7266	327	3	1	391	344	1	4	15	0	0	48	42	1	30	1820	4	0	63	230	195	0	0	2	0	37	0	10824
X	2	14	0	0	0	248	0	0	0	17	0	0	0	0	11	9	0	0	0	0	0	1	0	0	0	0	0	302
Y	1321	923	318	10	144	1708	6	1	60	0	0	54	11167	216	444	50	28	0	658	55	203	3	23	3	0	0	9	7404
Z	3	45	0	0	0	23	0	1	0	116	0	0	0	0	0	12	0	0	0	0	0	9	0	0	0	0	18	227
Total	97218	32422	7837	6860	20137	48765	9027	10356	26724	24462	325	3833	17644	8559	27787	30792	5310	317	22014	22960	36401	10824	3542	10824	302	7404	227	492873

Figure 2.4: The frequency of digrams in book3. Notice that there is obviously one less digram than singletons.

88060	␣	4959	,␣	2542	ea	1708	ri	1353	␣was	1102	on␣
48542	e	4909	␣the␣	2485	␣to	1707	nt	1337	ey␣	1092	em
34683	t	4896	nd␣	2462	is	1702	ey	1322	en␣	1087	f␣th
31839	a	4883	and	2449	al	1695	er␣	1308	all	1086	␣T
30540	o	4656	ou	2400	␣m	1688	n␣t	1306	␣we	1077	us
27503	n	4598	␣an	2370	␣l	1682	␣a␣	1300	I	1075	ch
26154	h	4554	.	2320	of␣	1643	ot	1270	ut␣	1066	im
23162	i	4380	y␣	2309	re␣	1635	oo	1267	was␣	1062	ad␣
22493	s	4229	r␣	2275	ing␣	1622	el	1267	wh	1060	␣tha
21895	r	4162	ng	2222	me	1609	fo	1264	for	1056	sh
19876	d	4113	and␣	2220	␣of␣	1594	↩↩	1264	n␣the	1055	e␣h
17451	l	4064	␣and	2186	’	1592	un	1263	e␣s	1054	␣they
15728	e␣	3999	ha	2180	to␣	1576	ee	1257	hey␣	1052	The
14003	␣t	3824	ed	2139	at␣	1550	␣r	1255	co	1051	t␣th
13204	he	3751	k	2118	te	1539	Th	1241	␣wi	1049	his␣
12740	th	3729	o␣	2091	␣g	1536	is␣	1238	ld	1045	so
11071	d␣	3723	␣and␣	2057	ne	1535	␣be	1236	␣on	1036	␣it
10775	u	3704	␣b	2050	se	1531	␣p	1232	es␣	1031	they␣
10511	␣a	3612	on	2029	d␣t	1525	-	1223	hat␣	1031	ul
10272	w	3589	␣i	2022	␣he	1524	om	1214	his	1026	!
10031	␣th	3540	or	1993	␣n	1520	id	1211	s␣a	1026	ca
9876	g	3509	v	1986	␣wa	1520	wi	1211	␣was␣	1025	do
9158	↩	3507	at	1952	ro	1510	␣hi	1210	␣in␣	1023	ra
9035	the	3355	ar	1945	as␣	1488	ll␣	1204	la	1019	f␣the
8831	f	3330	␣f	1943	il	1486	,␣an	1201	ke	1019	of␣t
8656	t␣	3303	to	1924	li	1481	,␣and	1201	or␣	1017	yo
8150	m	3293	as	1905	no	1479	‘	1184	ir	1007	had
7957	␣the	3282	en	1901	ho	1455	her	1178	e␣a		
7785	s␣	3209	ing	1890	l␣	1455	lo	1167	go		
7666	in	3183	hi	1872	ere	1454	t␣t	1167	ly		
7239	y	3106	es	1864	be	1422	hat	1166	d␣a		
7041	he␣	3084	.␣	1840	␣to␣	1412	was	1158	␣fo		
6968	an	3022	f␣	1831	␣ha	1410	ur	1152	bo		
6751	c	3016	ve	1820	ow	1400	gh	1147	f␣t		
6563	nd	2952	st	1811	ut	1393	n␣th	1138	tha		
6484	b	2948	ll	1797	ad	1381	e␣w	1119	.↩		
6292	␣w	2864	ed␣	1792	e␣t	1378	␣no	1117	sa		
6150	␣h	2860	wa	1781	de	1375	d␣th	1116	␣wh		
5806	er	2846	it	1771	a␣	1369	ere␣	1115	they		
5684	␣s	2829	g␣	1767	in␣	1369	hey	1114	pe		
5674	,	2743	of	1760	,␣a	1369	m␣	1114	ver		
5521	re	2716	ng␣	1756	h␣	1359	,␣and␣	1114	␣B		
5444	n␣	2707	␣d	1737	we	1358	␣e	1110	et		
5313	␣o	2584	␣c	1719	T	1357	␣he␣	1105	d␣the		
5244	p	2570	␣of	1718	␣in	1354	ti	1105	e↩		
5186	the␣	2552	le	1712	ai	1353	B	1104	␣y		

Figure 2.5: The most common n-grams in book3.

BILB
OBAG
GINS

Figure 2.6: Example of a matrix transposition cipher.

message was written into a matrix in row-major order and then read out in column major order. Encoding BILBOBAGGINS using a 3×4 matrix, Figure 2.6, gives BOGIBILANBGS if the columns are read off in order. By reading the columns in a different order, say 3–2–1–4, different ciphertexts can be obtained, LANIBIBOGBGS. The order to read the columns and the size of the matrix constitute the key. The technique can be extended to d dimensions. More generally if $f : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$ is any permutation over $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ then f can be used as a transposition cipher for messages of length n. The message can be blocked to encode longer messages. For example, if

$$f(x) = \begin{cases} 3 & \text{if } x = 1 \\ 2 & \text{if } x = 2 \\ 4 & \text{if } x = 3 \\ 1 & \text{if } x = 4 \end{cases}$$

Then, BILB OBAG GINS would be encoded as BIBL GBOA SIGN. Of course in practice values of n much larger than 4 are used, say $n = 64$. Transposition ciphers tend to be easy to recognize because the relative frequencies of the letters in the ciphertext will closely match those of the source language. Many transposition ciphers can be attacked by a technique called ***anagramming***; a method of rearranging letters into their original positions [Sin66].

2.4.4 Rotor Machines

The ciphers discussed so far have been suitable for pencil and paper work. Mechanization of coding came with the need for more frequent encryption, particularly during World War II, and demand led to the widespread use of rotor machines. These machines are largely based on the earlier disc encryption devices such as the Jefferson cylinder, the Wheatstone disk, and Bazeries' cylinder [Kah66]. Rotor machines are essentially polyalphabetic substitutions which change for each symbol encoded. The most famous rotor machine is undoubtedly the Enigma[3] used extensively by Nazi Germany during World War II. Other kinds of rotor machines include those made by Hagelin [Hag94].

Rotor machines work in the same manner as an odometer. There are a number of wheels, the ***rotors***, which turn. When the first rotor completes a revolution the second rotor is moved on by one position and so on.

[3]The Enigma machine was actually patented (#1657411) in the United States by one of its inventors, Arthur Scherbius, in 1928!

A team of Polish cryptographers broke a simplified version of the Enigma. The attack was extended to the full Enigma by British cryptanalysts, including Alan Turing, working at Bletchley Park. An important weakness of the Enigma was that no letter was ever encoded as itself. These attacks against the Enigma exploited this cryptographic weakness and the German operation of the machine [Kah66]. One of the first computers, the Colossus, was also located at Bletchley Park and was used in the cryptanalysis of other German systems.

2.5 Modern Cryptosystems

The invention of computers led to faster cryptanalysis of the classical systems. At the same time the door was opened on much more complex encryption functions which were infeasible to carry out by hand.

The move to automated systems allowed the use of complex mathematical transformations and novel approaches. We illustrate modern cryptology by discussing the Data Encryption Standard, public-key cryptosystems, and quantum cryptology.

2.5.1 The Data Encryption Standard

The Data Encryption Standard (DES) [NBS77] is a cryptosystem supported by the US Federal government which basically transforms 64-bit input blocks into 64-bit output blocks using a 56-bit key. The DES can be used for data encryption and data authentication. The same algorithm is used to both encode and decode. The DES is used extensively throughout the world and intensively within the United States. Its use in the United States has been endorsed by a number of authorities including various commercial and banking committees [ABA79, ANSI25, DoT84]. The latest federal review of the DES was in 1992 and its use was approved for another five years. Its widespread use makes it the most important cryptosystem currently in use.

An input block consisting of 64-bits, T, is first transposed under an initial permutation IP (Figure 2.7), giving $T_0 = IP(T)$. After it has passed through sixteen iterations of a function f, it is transposed under the inverse transformation, IP^{-1} (Figure 2.8). These permutations add nothing to the security of the DES but they do jumble the bits from the way they would appear in ASCII.

Figures 2.7 and 2.8 should be read in row major order. Between the IP and IP^{-1} permutations, sixteen iterations of a function f are performed; these combine substitution and transposition. Let T_i denote the result of the ith iteration, and let L_i and R_i denote the left and right halves of T_i, respectively; that is $T_i = L_i R_i$, where $L_i = t_1 \cdots t_{32}$, $R_i = t_{33} \cdots t_{64}$. Then $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$ where K_i is a 48-bit key described later. After the last iteration, the left and right halves are not exchanged;

instead $R_{16}L_{16}$ is input to the final permutation IP^{-1}. This is so the algorithm can be used to both encode and decode.

The function f is the most complex piece of the algorithm. First R_{i-1} is expanded to a 48-bit block $E(R_{i-1})$ using the bit selection table, E (Figure 2.9).

Next, the exclusive-or of $E(R_{i-1})$ and K_i is calculated and the result is broken into eight 6-bit blocks $B_1, \ldots, B_8$ where $E(R_{i-1}) \oplus K_i = B_1 B_2 \cdots B_8$. Each 6-bit block B_j is then used as input to a selection (substitution) function (***S-box***) S_j, which returns a 4-bit block $S_j(B_j)$. These blocks are concatenated together, and the resulting 32-bit block is transposed by the permutation, P (Figure 2.10). The integer corresponding to $b_1 b_6$ selects a row in the table, while $b_2 b_3 b_4 b_5$ selects a column. The value of $S_j(B_j)$ is the 4-bit integer in that row and column.

Thus, the block returned by $f(R_{i-1}, K_i)$ is $P(S_1(B_1) \cdots S_8(B_8))$. See Figure 2.11 for the S-boxes.

Key calculation: Each iteration i uses a different 48-bit key K_i derived from the initial key K. The key K is input as a 64-bit block with 8 parity bits in positions $8, 16, \ldots, 64$. The permutation *PC1* (permuted choice 1) discards the parity bits and transposes the remaining 56 bits according to *PC1* (Figure 2.12)

The result $PC1(K)$ is then split into two halves C and D of 28 bits each. The blocks C and D are then successively shifted left to derive each key K_i. Letting C_i and D_i denote the values of C and D used to derive K_i, we have $C_i = LS_i(C_{i-1})$, $D_i = LS_i(D_{i-1})$, where LS_i is a left circular shift according to Figure 2.13.

The key K_i is given by $K_i = PC2(C_i D_i)$, where *PC2* is given in Figure 2.12.

The DES is implemented in both software and hardware. The current fastest DES chip was developed at Digital Equipment Corporation [Ebe93] and can encrypt or decrypt at 1 Gbit/s which is equivalent to 15.6 million blocks per second. A software implementation running on a 33 MHz Intel 80486 can do 40 600 blocks per second. Despite its age the DES has held up very well and has resisted many years of cryptanalysis. It still appears to be safe against all but the most powerful (or richest) adversaries.

There are several modes in which the DES can operate [NBS80]. The two principal modes are ***electronic code book*** and ***ciphertext block chaining***. In electronic code book mode encoding proceeds exactly as described above. Identical plaintext blocks will give rise to identical ciphertext blocks. In ciphertext block chaining mode each block to be encoded is exclusive-ored with the previous ciphertext block before being encoded. In this mode, it is extremely unlikely that identical plaintext blocks will give identical ciphertext only blocks. The ciphertext block chaining mode is more secure since it is not vulnerable to a dictionary style attack. An instance of this is given in Chapter 4.

There are concerns that the key size is too small. Several designs for special purpose machines to exhaustively search the keyspace of the DES have been reported [Sch94]; most recently being [Wie93]. These proposals show that for a few million dollars a machine could

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Figure 2.7: The initial permutation *IP* used in the DES.

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Figure 2.8: The inverse of *IP*.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Figure 2.9: The bit selection table *E*.

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Figure 2.10: The permutation *P*.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S_1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S_2
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S_3
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S_4
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S_5
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	S_6
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S_7
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	S_8
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Figure 2.11: The eight S-boxes used in DES.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC1

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

PC2

Figure 2.12: The permutations *PC1* and *PC2*.

i	number of left shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Figure 2.13: DES shifting table for the key.

be built which could search the keyspace in four and a half hours. It is not known whether any such machine exists. But incidents have been reported where large purchases of DES chips have been made [Sch94]. In particular, the former Soviet Union is known to have purchased a large quantity of DES chips from a German company. The French intelligence agency (Direction Generale de la Securite Exterieure—DGSE) is also a likely candidate to have such a machine since they have a large budget and are legally encouraged to spy.

Differential cryptanalysis [BS93] is a relatively new kind of attack applicable to iterated cryptosystems. It is currently the best known attack against the DES, requiring 2^{47} chosen plaintexts. Don Coppersmith, one of the designers of the DES, has stated that differential cryptanalysis was known to the NSA when the DES was designed, and the S-boxes of the DES were subtly altered to give the best defence against this attack [BS93]. In fact, nearly any variation on the DES's S-boxes leads to a weaker cipher. Differential cryptanalysis analyzes the evolution of differences in ciphertext when two related plaintexts are encrypted using the same key. This type of attack is therefore unlikely to be applicable to stream based data compressors.

2.5.2 Public-Key Cryptosystems

In 1976, Diffie and Hellman [DH76a, DH76b] published a seminal paper which revolutionized cryptology. They invented[4] an asymmetric cryptosystem using the concept of a

[4] As is often the case with this sort of thing, the NSA has made an unsubstantiated claim that they were aware of two-key cryptography a decade earlier than this.

public-key cryptosystem in which there are two keys, a ***private key***, and a ***public key***. It is assumed computationally infeasible to deduce the private key from the public key. A variety of schemes for implementing public-key systems quickly followed, mostly based on seemingly intractable problems in pure mathematics, such as factoring and the subset sum problem. Many of these systems were subsequently broken sometimes leading to improved algorithms for the corresponding mathematical problem. Surprisingly, two of the earliest proposals, the Rivest-Shamir-Adleman (RSA) cryptosystem and the ElGamal cryptosystem, have stood the test of time, and we discuss these systems below. Schemes based on the subset sum problem (knapsack public-key cryptosystems) are discussed briefly in Chapter 5. Schneier [Sch94] gives extensive coverage of these and a multitude of other public-key schemes.

Public-key cryptosystems tend to be slow when compared to symmetric cryptosystems. For this reason they are frequently used to agree on a key, called a ***session key***, for use with a symmetric cryptosystem. The bulk of the communication then takes place using a symmetric cryptosystem with the agreed key.

The RSA algorithm [RSA78] rests its security on the difficulty of factoring large integers. To generate the two keys, two large (typically 512 bit) prime numbers p and q are selected and their product calculated, $n = pq$. Next an encryption key, e, is chosen such that $\gcd(e, (p-1)(q-1)) = 1$. Finally, the decryption key, d, is the multiplicative inverse of e modulo $(p-1)(q-1)$; that is $d \equiv e^{-1} \bmod (p-1)(q-1)$. The value e can be found with the extended Euclidean algorithm. The integers e and n form the public key and the integer d is the private key. The numbers p and q should also be kept secret or discarded.

Technically, breaking this form of RSA may not be as hard as factoring n since it has never been proven that it is necessary to factor n to recover a message from the ciphertext and e. Variants of RSA, however, have been proven to be as hard as factoring n.

The fastest known factoring algorithm is the ***number field sieve*** [LLMP90] running tantalizingly close to polynomial-time, with time complexity $O(e^{(\ln n)^{1/3}(\ln \ln n)^{2/3}})$. A slightly older method the ***double large prime variation of the multiple polynomial quadratic sieve*** [Sil87] is actually faster for numbers having less than about 120 digits due to implementation difficulties with the number field sieve. The quadratic sieve method was recently used to factor a 129 digit (429 bit) number originally thought to be secure for use in RSA [AGLL94]. The authors of [AGLL94] suggest this is probably the largest single computation ever completed. Over 600 people and 1600 computer were involved using an equivalent of several million mips years. They conclude that numbers as large as 512 bits are vulnerable to factoring with current technology and algorithms.

There exist chosen ciphertext attacks against RSA when RSA is used for signing messages as well as encryption. There is also an attack for the situation where two or more people have the same modulus n; further e should be large otherwise the ***low exponent attack*** [Has86] can be used.

A particularly nasty kind of attack is the ***dictionary attack***, applicable to many public-

key cryptosystems [Wil94]. This attack takes advantage of the public key and redundant nature of the source. The cryptanalyst guesses likely pieces of plaintext and encodes these using the public key. Large amounts of memory and disk space are used to store these plaintext-ciphertext pairs. Then when some ciphertext is intercepted a simple table lookup is used. Wilson [Wil94] found that for English encoded using ASCII, 80% of a message could be determined in this manner when using a two gigabyte table. If necessary, unknown blocks can then be guessed using the surrounding blocks as context. The guesses can be checked and modified as desired.

The dictionary attack works because the dictionary does not need to be too large to cover common English. Compression can be used to restore security since in a compressed file each possible block is (nearly) equally likely, thereby making the required dictionary size too large.

The ElGamal scheme [ElG85] can be used for both digital signatures and encryption. Its security rests on the difficulty of extracting ***discrete logarithms***. The discrete logarithm problem is to find an integer x where $a^x = b(\mathrm{mod}\, n)$. There need not always be a solution. The difficulty of this problem is comparable to factoring. To generate a key pair, first choose a large prime p and two random numbers g and x such that both are less than p. Let $y = g^x(\mathrm{mod}\, p)$. The public key is y, g, and p. The private key is x. To encrypt a message M, first choose a random k such that $\gcd(k, p-1) = 1$. Then compute $a = g^k(\mathrm{mod}\, p)$ and $b = y^k M(\mathrm{mod}\, p)$. The pair (a, b) is the ciphertext. To decrypt a and b, compute $b/a^x(\mathrm{mod}\, p)$. Since $a^x \equiv g^{kx}(\mathrm{mod}\, p)$, then $b/a^x \equiv y^k M/a^x \equiv g^{xk} M/g^{xk} \equiv M(\mathrm{mod}\, p)$.

ElGamal is also susceptible to dictionary attacks [Wil94]. Once again all that is need is a large representative sample of text and knowledge of the public key. The attack works in the same way as the dictionary attack against RSA.

2.5.3 Quantum Cryptography

Recently cryptosystems have been proposed which derive their security from the laws of quantum mechanics, rather than intractability of mathematical problems. These systems rely on the uncertainty principle: any attempt to read the message by a cryptanalyst will disturb the quantum state and will be detected. An experimental prototype has been constructed [BB89, BBBSS91]. For a review see [BBE92].

In contrast, new probabilistic models of computation based on quantum mechanics may be significantly more powerful in the complexity-theoretic sense than the probabilistic Turing machine. Recently, Shor [Sho94] has shown that a quantum computer of specific design could factor integers in expected polynomial time. Currently these results are theoretical and it is unlikely that a quantum computer will be forthcoming in the near future; although some large corporations, such as Hitachi, are actively working in this area.

2.6 Summary

Originally, cryptology was only about encryption and this was carried out with pencil and paper. Since the plaintext was a natural language, redundancy was high and this was exploited in statistical attacks. Increased demand for encryption during the World Wars led to mechanization of encryption. Longer and more complicated substitutions and transpositions could then be used. At this time known plaintext attacks became much more important since it was often difficult to break the systems with only ciphertext. The invention of computers greatly simplified the more arduous tasks of cryptanalysis but at the same time gave rise to much more complex ciphers such as the DES. The invention of public-key cryptography heralded a new era in cryptography by solving the long standing key exchange problem and by introducing a plethora of new security issues such as digital signatures. It was proven that the one-time pad was unconditionally secure but no other cipher has been proven secure. Such proofs are only likely once complexity theory has advanced far enough to make meaningful statements about average complexities. Quantum cryptography is becoming a reality and the prospect of quantum computers, which only a few years ago were considered as science fantasy, may soon be realized.[5]

However, this thesis is about the relationships between data compression and cryptography. Even the most elementary of ciphers, the simple substitution, offers excellent security against ciphertext only attack for a purely random source. Compression is a technique for removing redundancy from a source, to make the source more random. The next chapter introduces the ideas of data compression and in Chapter 4 we measure just how much redundancy remains after compression.

[5]There are now even serious papers in mainstream journals dealing with the possibility of teleportation [Sud93]. Many of these incredible developments have come to light as corollaries to the research on quantum cryptography.

Chapter 3

Introduction to Compression

Compression has always been related to cryptography. Even before the proliferation of computer networks the problems of communication were the same: there was a need to communicate messages privately and in a cost effective manner.

Many early methods employed aspects of both compression and cryptology. For example, in the 1800s large codebooks were compiled listing common phrases and corresponding numerical codes. By transmitting the shorter codes money was saved on the cost of telegrams; and as the codebooks had limited circulation some level of privacy was achieved.

In this chapter the techniques of data compression are surveyed. In Section 3.1, the nature of information is examined and the entropy measure invented by Claude Shannon is presented. Shannon's theory provides a quantitative way to measure 'the amount of information'. A brief history of data compression is given in Section 3.2, leading to the distinction between a model and a coder. The optimal coding technique of arithmetic coding is presented along with the older and less perfect Huffman coding in Section 3.3. A variety of modern compression algorithms are presented in Section 3.4. Finally, in Section 3.5 the use of compression in cryptology is addressed.

3.1 Information Theory

Compression is about removing redundancy from a source or a message. Natural languages like English are highly redundant. For instance reading text with spaces removed and normalized to uppercase is not difficult:

BILBOBAGGINSISAFAMOUSHOBBIT.

In English prose, spaces typically account for fifteen to twenty percent of all the symbols; so just eliminating the space would lead to considerable compression. However, for the purpose of reading, redundancy is useful as it allows us to read faster and recover from misprints. Understanding speech would be impossible if spoken language did not contain redundancy. For these reasons text compressors need to be reversible in the sense that the original can be reconstructed from the compressed version.

The amount of redundancy in a source is related to its ***entropy***. In physics, entropy is used to measure the amount of disorder or uncertainty in a system. The use in information theory is analogous; it measures the amount of information. Entropy is small where there is a lot of order and large when there is a lot of disorder. It is impossible to give a precise entropy for any particular message since the information content depends on the observer. Nevertheless, entropy is the best guide available when it comes to specifying how many bits are required to specify a message.

Information is inextricably bound up with choice. Where there is choice there is information, since the choice made must be specified. For instance, to choose between two equally likely alternatives exactly one bit of information is required.

More generally, suppose events $1, \ldots, n$ occur with probabilities $p_1, \ldots, p_n$ summing to one. Shannon [Sha48, Sha49] established that the correct way to measure the amount of choice in the events, or the entropy, is given by

$$H = -\sum_{i=1}^{n} p_i \lg p_i \text{ bits.}$$

For example suppose all n events are equally likely, $p_i = 1/n$, then

$$\begin{aligned} H &= -\sum_{i=1}^{n} \frac{1}{n} \lg \frac{1}{n} \\ &= \lg n. \end{aligned}$$

From this formula one can determine that the maximum number of objects which can be distinguished in a game of twenty questions is $2^{20} = 1048576$. Further, to construct a block code for an alphabet of n symbols $\lceil \lg n \rceil$ bits are needed per block.

The formula for entropy can, at least in principle, be applied to any probability distribution (a slight generalization is needed to handle continuous distributions) and gives the expected number of bits required to encode an arbitrary event drawn from the given distribution. The entropy formula does not specify how the encoding is to be carried out. The invention of arithmetic coding (discussed in Section 3.3.2) solves this problem by coding events arbitrarily close to their entropy.

For a given language the ***entropy of the language*** (also called the ***rate of the language***) is the uncertainty of the language per symbol $H = H(M)/|M|$, for a message M. The ***absolute entropy of a language*** is the maximum number of bits that can be coded in each symbol assuming that each possible sequence is equally likely. If there are n symbols in the alphabet, the absolute rate is $h = \lg m$. The ***redundancy*** in the language is defined to be $R = h - H$.

The ***unicity distance*** is the expected minimum length for a cryptogram to have a unique solution. For symmetric cryptosystems is is given by $U = H(K)/R$ where $H(K)$ is the entropy of the possible keys.

There are various published numerical values for the entropy of twenty-seven letter English. The first determination was made by Shannon [Sha51] who obtained the value of 1 bit/symbol by using human subjects to predict text. A gambling strategy was later used in a more sophisticated analysis to obtain a value of 1.25 bits/symbol [CK50]. Books on cryptanalysis typically quote a value of about 1.5 bits/symbol [Til88]. These estimates are based on gram analysis. A trigram analysis of 583 million words yielded an entropy of 1.75 bits/symbol [BDDLM92]. Other determinations are 1.65 bits/symbol [Bar55] and 1.7 bits/symbol [JJ68]. More recently, a compression scheme has been used to demonstrate an upper bound of 1.46 bits/symbol [TC96]. A value of about 2.4 bits/symbol [TC96] is likely to be accurate for the more general case where casing, punctuation, and digits are retained. Because English is still evolving and is different for different people there can never be a definitive value given for the entropy of English.

3.2 Early Compression

Morse code is a good example of an early compression technique. Comparison of the codes in Figure 3.1 with the frequency of English letters (Figure 2.3) reveals that Morse cleverly assigned the shortest codes to the most frequent symbols. Intuitively, if symbols which occur frequently have the shortest codes then on average messages will be shorter. The disadvantages of Morse code become apparent when something other than English text, say a page of numbers, needs to be transmitted. In this case the resulting message will be long because the digits have longer representations in Morse code. The more powerful techniques discussed in Section 3.4 overcome this limitation by adapting dynamically to a message. A second problem with Morse code is that symbols are considered in isolation; that is, without reference to the surrounding context. Better codes are produced by considering surrounding symbols. This is easily seen in English where the probability of a U occurring goes up dramatically just after coding a Q.

We are primarily concerned with ***lossless compression***; that is compression where the original message can be exactly reconstructed from the compressed version. Lossless compression is particularly appropriate for compressing natural languages, source code, program executables, and financial data. Sometimes lossless methods are unnecessary; noisy sources like speech, music, and images do not need exact reproduction when decompressed. Compressors for this situation are called ***lossy***.

In the past a variety of ad hoc techniques which took advantage of particular vagaries in the source were widely used. Although these techniques have been supplanted by adaptive compressors, a few are mentioned to illustrate the kinds of redundancy known to occur in practice.

- ***Compaction***: Although strictly a form of lossy compression, compaction does not lead to loss of semantic information. It typically involves replacing all whitespace

A	·–	N	–·	0	–––––
B	–···	O	–––	1	·––––
C	–·–·	P	·––·	2	··–––
D	–··	Q	––·–	3	···––
E		R	·–·	4	····–
F	··–·	S		5	
G	––·	T	–	6	–····
H		U	··–	7	––···
I		V	···–	8	–––··
J	·–––	W	·––	9	––––·
K	–·–	X	–··–		
L	·–··	Y	–·––		
M	––	Z	––··		

Figure 3.1: The Morse code.

symbols (spaces, tabs, carriage returns, etc.) and groups of consecutive whitespace symbols with a single space symbol. A more risky form of compaction forces all the message into a single case so that a smaller alphabet can be used.

- ***Run-Length Encoding***: In run-length encoding, consecutive identical symbols are replaced by a single instance of the symbol and a count indicating the number of times the symbol is to be repeated. The technique works best for small alphabets, particularly binary, and is widely used to compress bilevel images. Direct application of run-length encoding is generally ineffective for ASCII text; but was used extensively over slow communication links to compress runs of blanks in card or line printer images.

- ***Packing***: For text, the high bit of ASCII is always zero. By using the high bits eight symbols can be encoded in seven bytes. For alphabets with not more than sixteen symbols, such as decimal or hexadecimal digits, two symbols can be stored per byte. Similar bounds can be derived for other alphabet sizes.

- ***Differential Coding***: When compressing sorted records, it is often beneficial to store differences between consecutive records. The technique works well for static lexicons and consecutive lines of raster images.

The formula for entropy shows that the information content of a source is determined by the underlying probability distribution. This in turn indicates a logical distinction between the model (which specifies the probability distribution) and the actual coding of events drawn from the distribution. In data compression this is referred to as the model-coder paradigm [BCW90]. The idea is that a compressor should consist of two parts: a model which gathers statistics, uses prior information and heuristics to select a probability distribution; and a coder used to produce a compact representation of events generated

by the model. From a security perspective these are two distinct areas where security may be found or where one could attempt to introduce some secret transformation.

3.3 Coders

It is by no means obvious that information can be encoded according to its entropy. Shannon's noiseless coding theorem [Sha48] only proves that it is impossible to encode information in less space than its entropy on average.

In this section, Huffman coding and arithmetic coding are introduced. For a long time Huffman coding was thought to be the best possible. The invention of arithmetic coding showed that in fact not only was Huffman not the best possible when incremental transmission is required, but that it is possible to code information arbitrarily close to its entropy.

3.3.1 Huffman Coding

An important advance was made in 1952 when Huffman [Huf52] invented a coding scheme that came close to Shannon's theoretical bound. It can be shown that the redundancy of Huffman codes, defined as the average code length less the entropy, is bounded by $p + \lg(2 \lg e/e)$, where p is the probability of the most likely symbol [Gal78]. This slight imperfection in the coding results from the output of an integral number of bits per symbol encoded. In particular, at least one bit is output for each symbol encoded. Arithmetic coding, discussed in the next section, overcomes this limitation.

Despite its imperfections Huffman coding is still useful and is widely used; often in variations such as adaptive Huffman coding [Gal78, Knu85, Vit86].

A Huffman code is elegantly represented by a binary tree. Figure 3.2 gives a Huffman code for English letters based on the percentages given in Figure 2.3. The corresponding code is given in Figure 3.3.

The entropy for this Huffman model of English is given by

$$-\sum_{k=\text{␣}}^{\text{Z}} p_k \lg p_k = 4.093 \text{ bits/symbol.}$$

The expected length of the corresponding Huffman code is

$$\sum_{k=\text{␣}}^{\text{Z}} |h(k)| p_k = 4.135 \text{ bits/symbol}$$

where $h(k)$ is the Huffman code for symbol k. A naïve block code for the same source requires $\lceil \lg 27 \rceil = 5$ bits/symbol.

To construct a Huffman code for an ensemble with probabilities $p_1, \ldots, p_n$ repeat the following steps until only one element remains:

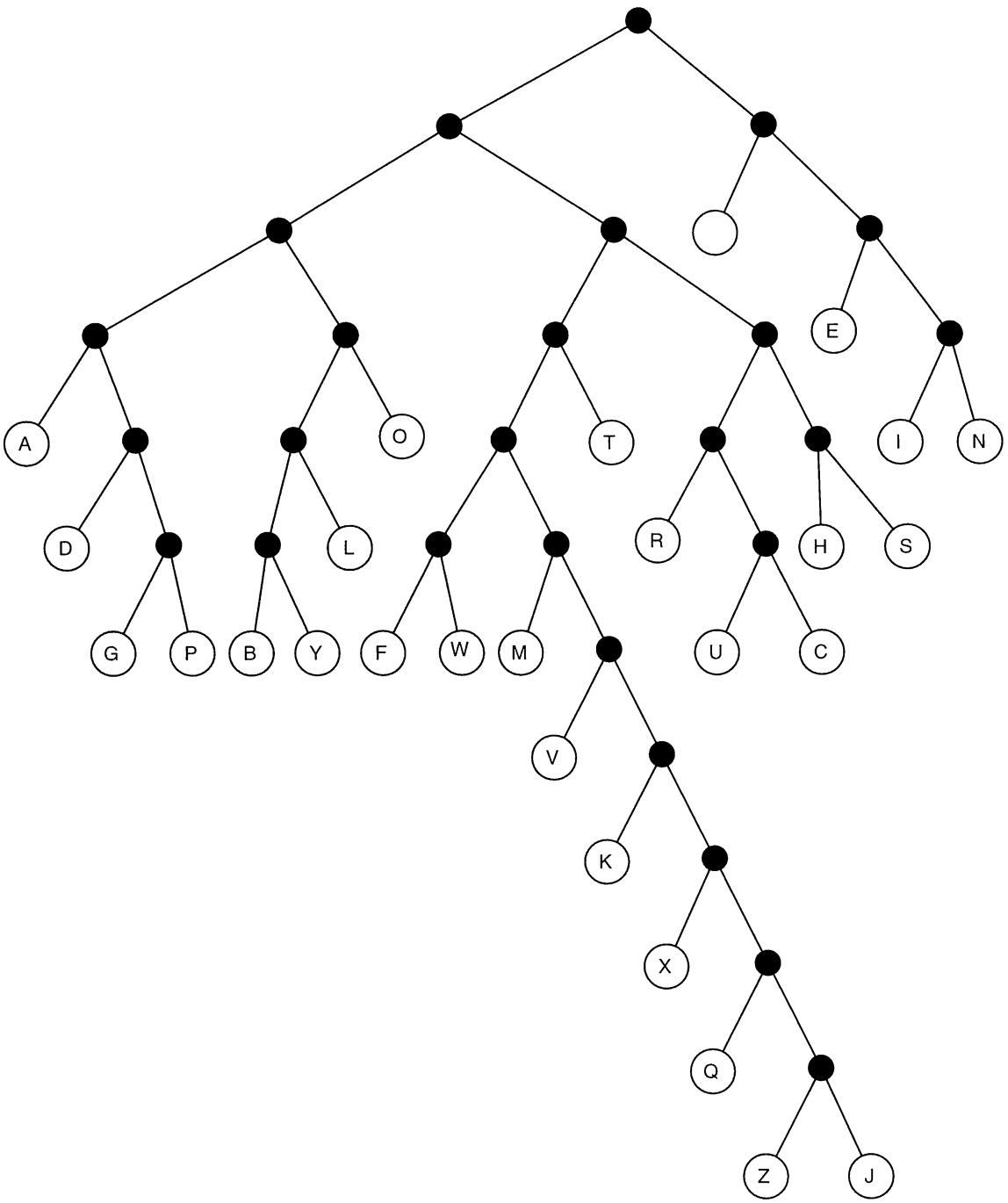


Figure 3.2: A Huffman tree based on the frequency of individual letters in English. The blank circle represents the space symbol.

Symbol	Code	Symbol	Code
␣	10	N	1111
A	0000	O	0011
B	001000	P	000111
C	011011	Q	0100111110
D	00010	R	01100
E	110	S	01111
F	010000	T	0101
G	000110	U	011010
H	01110	V	0100110
I	1110	W	010001
J	01001111111	X	010011110
K	01001110	Y	001001
L	00101	Z	01001111110
M	010010		

Figure 3.3: An example Huffman code based on the frequency of individual letters and the space in English text.

- Locate the two symbols with the smallest probability.
- Replace these symbols with a new symbol containing both of them and whose probability is the sum of the individual probabilities.

Notice that is necessary that the probability distribution be known to the encoder and decoder prior to transmission of the actual message. The relations between the symbols can conveniently be specified by a binary tree, as in Figure 3.2. The code for each symbol is then found by traversing the tree from the root to the required symbol outputting a zero for a left branch and one for a right branch.

In Chapter 6 Huffman coding is shown to be insecure.

3.3.2 Arithmetic Coding

Arithmetic coders are used in the coding part of many modern data compressors. It is superior to Huffman coding: arithmetic coding will always represent information at least as compactly as Huffman coding. Huffman encoding can only output an integral number of bits per input symbol (and at least one bit is output for each input symbol). Arithmetic coding dispenses with this limitation by allowing fractions of bits to be output per symbol.

A brief explanation of arithmetic coding is given here, followed by an illustration of its operation with a static model.

In arithmetic coding, each message is uniquely represented by a subinterval of the half closed interval $[0, 1)$. Although the messages can be drawn from alphabets of arbitrary symbols our examples will use everyday alphabets like the Roman alphabet, ASCII, or

binary. Longer messages result in smaller intervals and it takes more bits to represent a small interval than a larger one. The interval used for a particular message is determined by the content of the message, and the probability of the message according to the model. The more likely messages result in larger subintervals and therefore shorter code lengths. The output of the arithmetic coder is a sequence of bits representing the interval which contains the message.

Except for short messages, it is computationally infeasible to make a direct transformation from a message to a subinterval, especially when the coder is used on-line. Most models encode the message one symbol at a time; that is, encode the message ***incrementally***. Thus 'transmit message m', or 'encode m', etc. should be understood to mean 'encode message m one symbol at a time'.

At each stage of the encoding (or decoding) there is a ***current interval*** $[l, h)$ which consists of a ***lower bound*** l and ***upper bound*** h. To avoid wasting codespace the current interval is initialized to $[0, 1)$.

Each symbol encoded causes the current interval to be narrowed in accordance with the symbol's probability. More probable symbols cause less narrowing than unlikely symbols. Knowing with complete certainty that a symbol will occur means no narrowing occurs (because it is unnecessary to transmit that symbol), whereas if a symbol with zero probability occurs, the coding interval width is narrowed to zero and thus takes an infinite number of bits to specify.

The concept of arithmetic coding was developed independently by several people, but the first published version is due to Elias [Abr63]. Further development was done by Langdon [Lan84] and Rissanen [Ris76, Ris79]. It was made accessible by Witten, Neal, and Cleary [WNC87].

Assume we have an alphabet of n symbols $\{s_0, s_1, \ldots, s_{n-1}\}$. Let θ define a total ordering of the alphabet, in particular we could have $\theta(s_i) = i$ for $0 \leq i < n$. We ignore the distinction and write 'symbol i' to mean the ith symbol in the ordering θ. Let p_i denote the probability of symbol i, and let P_i denote the cumulative probability of all the symbols up to but not including symbol i (the sum of the probability of all those symbols for which $\theta(s_j) < \theta(s_i)$);

$$P_i = \sum_{j=0}^{i-1} p_j.$$

Let $\delta = h - l$ be the width of the current interval $[l, h)$. If symbol s_i is encoded the new interval $[l', h')$ will be given by

$$\begin{aligned} \delta' &= p_{\theta(s_i)}\delta, \\ l' &= l + P_{\theta(s_i)}\delta, \\ h' &= h - (1 - P_{\theta(s_i)+1})\delta. \end{aligned}$$

One can easily verify that $\delta' = h' - l'$.

To illustrate the operation of arithmetic coding we now present an example using a static model. Suppose our messages are drawn from English with the alphabet limited to the upper case letters. Figure 3.4 is a static-model based on statistics derived from book1.

Symbol s_i	$\theta(s_i)$	$p_{\theta(s_i)}$	$P_{\theta(s_i)}$	Coding Interval
E	25	0.123394	0.876606	[0.876606, 1.000000)
T	24	0.088036	0.788570	[0.788570, 0.876606)
A	23	0.082634	0.705936	[0.705936, 0.788570)
O	22	0.077297	0.628639	[0.628639, 0.705936)
N	21	0.070134	0.558505	[0.558505, 0.628639)
I	20	0.067569	0.490936	[0.490936, 0.558505)
H	19	0.065253	0.425683	[0.425683, 0.490936)
S	18	0.063729	0.361954	[0.361954, 0.425683)
R	17	0.056103	0.305851	[0.305851, 0.361954)
D	16	0.045534	0.260317	[0.260317, 0.305851)
L	15	0.039775	0.220542	[0.220542, 0.260317)
U	14	0.027318	0.193224	[0.193224, 0.220542)
W	13	0.025100	0.168124	[0.168124, 0.193224)
M	12	0.024736	0.143388	[0.143388, 0.168124)
C	11	0.022460	0.120928	[0.120928, 0.143388)
G	10	0.021805	0.099123	[0.099123, 0.120928)
F	9	0.021419	0.077704	[0.077704, 0.099123)
Y	8	0.020999	0.056705	[0.056705, 0.077704)
B	7	0.017940	0.038765	[0.038765, 0.056705)
P	6	0.016974	0.021791	[0.021791, 0.038765)
V	5	0.009221	0.012570	[0.012570, 0.021791)
K	4	0.008532	0.004038	[0.004038, 0.012570)
X	3	0.001466	0.002572	[0.002572, 0.004038)
J	2	0.001221	0.001351	[0.001351, 0.002572)
Q	1	0.000904	0.000447	[0.000447, 0.001351)
Z	0	0.000447	0.000000	[0.000000, 0.000447)

Figure 3.4: A simple static model based on statistics from an English novel. (Note: $p_{25} + P_{25} = 0.876606 + 0.123394 = 1$)

Consider transmitting the message GABRIEL.[1] The initial range is $[0, 1)$. After seeing the first symbol, G, the interval is narrowed to $[0.099123, 0.120928)$ which is the range allocated to this symbol. The next symbol to be encoded is A, ($\theta(\text{A}) = 23$) the current range of the interval is $\delta = 0.120928 - 0.099123 = 0.021805$. Using the recurrence relations we discover $l' = 0.099123 + 0.705936\delta = 0.11451593448$ and $h' = 0.120928 - (1 - 0.78857)\delta = 0.11631776885$. The complete encoding is shown in Figure 3.5.

It is not necessary for the decoder to know both ends of the interval. Any value in the final interval suffices. In our example any of 0.11459667027, 0.114596670275, 0.114596670418719, 0.11459667079 would suffice—along with an infinite number of other possibilities.

[1] Gabriel is the name of one of the characters in book1.

Symbol	Interval After Coding
G	[0.09912300000, 0.12092800000)
A	[0.11451593448, 0.11631776885)
B	[0.11458578259, 0.11461810750)
R	[0.11459566920, 0.11459748272)
I	[0.11459655952, 0.11459668206)
E	[0.11459666694, 0.11459668206)
L	[0.11459667027, 0.11459667088)

Figure 3.5: Illustration of the encoding process. The interval is shown accurate to eleven decimal places.

The entropy of GABRIEL in this model is

$$-\log p_{\mathrm{G}} - \log p_{\mathrm{A}} - \log p_{\mathrm{B}} - \log p_{\mathrm{R}} - \log p_{\mathrm{I}} - \log p_{\mathrm{E}} - \log p_{\mathrm{L}}$$

$$\approx 9.22 \qquad \text{(logarithm to base 10)}.$$

This is why it takes ten ($10 = \lceil 9.22 \rceil$) decimal digits to encode the message. Actually, the size of the final range is 6.1×10^{-10}, the logarithm (base 10) of which is 9.22. In binary, 31 bits would be required which is a significant saving over the 56 bits used by ASCII—and a very simplistic model was used. There is a slight saving over the 35 bits required for a five bit block code for 26 symbols.

Now consider the decoding process. Assume 0.11459667030 was transmitted. We can tell immediately that the first symbol was G because 0.11459667030 lies within the code space allocated to symbol G. The decoder now simulates the action of the encoder by narrowing the range to $[0.099123, 0.120928)$. The next character must be A because an A will result in the range $[0.11451593448, 0.1131776885)$ which contains the transmitted value. Proceeding like this, the decoder can identify the whole message.

In the example, no mention was made of how the decoder knows when to stop. In practice either the message length must be transmitted prior to the actual message or a special end-of-file symbol must be added to the input alphabet. The latter option tends to be nicer because using an end-of-file character does not require the encoder to know the length of the message in advance.

Another advantage of arithmetic coding is that a controlled amount of redundancy can be incorporated by allocating a proportion of the coding region to a new symbol ERROR [BCIRW97]. Whenever the decoder receives the symbol ERROR it knows that an error has occurred. Thus arithmetic coding allows us to control the amount of redundancy (by varying the proportion of the coding region allocated to ERROR), and, further, this amount can be altered dynamically while coding is in progress. Critical pieces of the message could be encoded with higher redundancy than the rest of the message to ensure correct reception.

The security of arithmetic coding is extensively analyzed in Chapter 5.

3.4 Modern Techniques for Compression

There are many possible models for generating the probabilities. Modern compression systems tend to use ***adaptive models*** (models in which the probability of a symbol can change during the encoding of a message). Adaptation makes a compressor suitable for a much wider class of inputs, even for messages in which the statistics vary over the course of the message. In contrast, a ***static model*** uses a fixed set of probabilities throughout the encoding of a message. The static model is useful when random access to the compressed information is needed.

3.4.1 Context Modelling

In the ***prediction by partial matching*** (PPM) compressor [CW84b, BCW90], the previously transmitted symbols are used to condition the probability of the next symbol. The PPM model is adaptive and changes as the message is compressed. There are many variants of the basic approach depending on how many symbols are used to predict the next, whether or not multiple predictions are used, and how shorter context models are used when necessary. The predictions are based on simple frequency counts of the transmission so far. Better predictions can be made when longer contexts are used: it is much easier to guess what comes after motorcycl than after cl.

The ***order*** of a model is the maximum number of symbols used to predict the next symbol. In practice an order-o model will sometimes base its prediction on less than o symbols. By convention the order-(-1) model predicts each symbol with equal probability and the order-0 model predicts each symbol with probability proportional to the number of times it has occurred previously.

The general PPM method requires the predictions of all orders (up to some maximum) to be blended together to give an overall probability for the next symbol. The most general approach is

$$\Pr(s_j = \phi) = \sum_{i=-1}^{o} w_i p_i(\phi),$$

where the w_i are a set of weights normalized to sum to 1, s_j is the next symbol, and $p_i(\phi)$ is the probability of symbol ϕ according to an order-i model. Calculating the sum is computationally expensive and there is no single 'right' way to determine the weights to use. The probabilities $p_i(\phi)$ are rational and based on the frequency counts:

$$p_i(\phi) = \frac{f_i(\phi)}{F_i} \qquad \text{where} \qquad F_i = \sum_{\phi} f_i(\phi).$$

However, the formula for $p_i(\phi)$ just given leads to the ***zero-frequency problem*** citeWB, since symbols not previously encountered are given zero probability. The formula for $p_i(\phi)$ must be modified so that symbols not previously encountered in a given context can be represented. If the context itself has never been seen before then all of the $f_i(\phi)$ and F_i

will be zero. However, we are guaranteed that some shorter context has been seen before; in the worst case the order-(-1) model can be used. The order-(-1) model always predicts every symbol.

In practice full blending is not used; instead each context assigns a probability, called an ***escape probability***, to a novel symbol occurring. The PPM variants differ in the way escape probabilities are assigned. When a novel symbol is seen the escape probability is used followed by the prediction of the next shorter context. Several escapes may be made before a context is reached which predicts the symbol. In the worst case the order-(-1) model makes the prediction. Cleary and Witten [CW84b] show that escape probabilities are equivalent to weighting. We now mention how the probabilities are determined in some variants.

- **PPMA**: This method uses the plausible assumption that novel symbols are more likely when the context has only been seen a few times. The probability of the escape symbol is $e_i = 1/(F_i + 1)$ and the probability for the other symbols becomes

$$p_i(\phi) = \frac{f_i(\phi)}{F_i + 1}.$$

- **PPMB**: In this method no prediction is made unless the symbol has occurred more than once in the current context. This is done by subtracting one from all counts with the subtracted counts being combined to give the escape probability. The idea is to filter anomalous events. Symbols are not predicted until they are seen twice. Thus, $e_i = q_i/F_i$ where q_i is the number of different symbols seen in the given context. The probabilities for the remaining symbols become

$$p_i(\phi) = \frac{f_i(\phi) - 1}{F_i}.$$

- **PPMC**: This popular method is similar to PPMB except symbols are predicted immediately. Thus,

$$e_i = \frac{q_i}{F_i + q_i} \qquad \text{and} \qquad p_i(\phi) = \frac{f_i(\phi)}{F_i + q_i}.$$

- **PPMD**: This method is a small modification to PPMC suggested by Howard [How93] where each count is incremented by a 1/2. It sets $e_i = q/2F_i$ and

$$p_i(\phi) = \frac{f_i(\phi) + \frac{1}{2}}{F_i + 1}.$$

  The use of ***deterministic scaling*** [Tea95] also leads to better performance.

- Other approaches dubbed **PPMP**, **PPMX**, and **PPMXC** based on a Poisson process model have also been proposed [WB91].

The PPM method can be ***bounded*** if some maximum order is specified in advance or ***unbounded*** [Tea95] if contexts are allowed to be arbitrarily long. A tree representation is suitable for both approaches. Each node in the tree contains a frequency count $f_i(\phi)$ for the given context. To prevent overflow it is on occasion necessary to rescale the frequency counts. Scaling often actually leads to improved compression as it gives increased importance to recently encoded text. Scaling is appropriate for most text where the subject matter varies slowly over the course of the document.

3.4.2 Dictionary Based Compression

Context modelling is not the only popular approach to adaptive data compression. Another kind of compression uses a specially constructed dictionary to achieve compression. In a dictionary based compression scheme, groups of consecutive symbols, called ***phrases***, are replaced with indices into some dictionary. The dictionary is constructed so as to contain a list of phrases expected to occur frequently. To achieve compression the indices must occupy less space than the phrase they encode. Dictionary coding is also called macro coding and codebook coding. Since the number of bits used in indices can be chosen to align with machine words efficient implementations are possible. Typically a multiple of four or eight bits is chosen for the indices. Since in some schemes the number of phrases can grow without bound it is necessary to be able to encode arbitrarily large integers. Methods for doing this are discussed in the next section.

These systems achieve good compression because a single dictionary reference may represent many characters. For every dictionary scheme there is an equivalent statistical scheme achieving the same compression [BCW90]. Eventually, dictionary methods will probably be completely replaced by statistical approaches. Dictionary schemes are currently still widely because they offer rapid decompression.

The construction of the dictionary is one of the more important aspects of the system. A dictionary which closely matches the text to be compressed will yield good compression. The dictionary can be static, semi-adaptive, or adaptive. The maximum length of phrases stored in the dictionary may be fixed or unbounded. Better compression is achieved by having an adaptive dictionary which allows longer phrases.

DEFINITION 3.1: A ***dictionary*** $D = (M, C)$ is a finite set of phrases M and a function C that maps M onto a set of codes, where $M \subseteq A^*$ for an alphabet A. Without loss of generality the output codes are assumed to be over $\{0,1\}^*$.

DEFINITION 3.2: The set M is ***complete*** if every infinite string over the input alphabet A is also in M^*; that is, any input string can be formed by the concatenation of phrases from M.

DEFINITION 3.3: The function C obeys the ***prefix property*** if no string $C(m)$ is a prefix of another string $C(s)$, for $s, m \in M$ and $s \neq m$.

For reversible compression of any input to be possible the set M must be complete and C must obey the prefix property.

Aside from constructing the dictionary there is also the problem of matching the input to the dictionary. Optimal parsing is hard: it sometimes requires the entire input to be examined before anything can be output. There are reasonable heuristics which approach optimal parsing; frequently, greedy parsing is used. In greedy parsing the next longest match is found. The extra compression gain obtained by using optimal parsing is minimal and is at the expense of increased execution time.

Static dictionaries are constructed by considering a sample of representative text prior to actually compressing any messages. Such a static dictionary is assumed to be available prior to transmission to both the encoder and decoder. A ***semi-adaptive dictionary*** is constructed for the text to be compressed in an initial pass. Determining the optimal dictionary is **NP**-complete in the size of the text. Further, the dictionary itself must be transmitted along with the compressed text. Most modern dictionary systems use ***adaptive dictionaries*** which are constructed incrementally (by both the encoder and decoder) as the message is transmitted.

The most popular dictionary schemes are from the Ziv-Lempel family of compressors. In the Ziv-Lempel coding scheme [ZL77], text is compressed by providing references to earlier text. In fact, there are many variants of the Ziv-Lempel coding scheme. The coding scheme presented in [ZL77] differs in several respects to the later scheme [ZL78]. In this section a brief overview of the different forms of Ziv-Lempel coding are given.

- **LZ77**: The first form of Ziv-Lempel coding [ZL77]. Pointers are used to denote phrases in a fixed-size window that precedes the coding position. Thus the window is essentially the phrases of the dictionary. There is a maximum length for substrings that may be replaced by a pointer (usually about twenty). The window is initially spaces. Matches may overlap with the text. The longest match is coded as a triple (i, j, a) where i is the offset of the longest match from the lookahead buffer, j is the length of the match, and a is the first symbol which failed to match. The window size is typically about eight kilobytes.

- **LZR**: The same as LZ77 except pointers denote any position in the already encoded text [RPE81]. The pointer i and length j are encoded with a variable length code, since they can grow to arbitrary sizes. The LZR method tends to be slow since the search time and memory requirements increase as more and more text is processed.

- **LZSS**: The same as LZ77 except pointer and characters are distinguished by a flag bit [Bel86]. This avoids the presence of an explicit character in each triple.

- **LZB**: The same as LZSS but using a variable size for pointers [Bel87].
- **LZH**: The same as LZB but encodes the pointers using Huffman coding [Bre87].
- **LZ78**: The second form of Ziv-Lempel coding [ZL78]. The input text is broken into phrases where each phrase is the longest matching phrase seen previously plus one character. Each phrase is encoded as an index to its prefix plus the extra character. The number of phrases can grow unboundedly. In practical implementations when memory is exhausted the coding starts again from scratch. The LZ78 scheme is asymptotically optimal for a stationary ergodic source although convergence is relatively slow.
- **LZW**: Like LZ78 but the output consists solely of pointers [Wel84]. This is achieved by initializing the dictionary to contain every character in the alphabet. The LZW approach is perhaps the most common LZ variant in practice.
- **LZC**: This variant of Ziv-Lempel coding is used in the UNIX `compress` program. The output consists solely of pointers. The dictionary is rebuilt when compression performance drops or memory is exhausted.
- **LZT**: Like LZC except a recency heuristic is used to discard phrases when the memory is exhausted [Tis87].
- **LZMW**: The same as LZT but phrases are built by concatenating the previous two phrases [MW84].
- **LZJ**: The output contains pointers only. Pointers indicate a substring anywhere in the previous characters [Jak85].
- **LZFG**: Pointers select a node in a trie, the strings in the trie are from a sliding window [FG89].

The security of Ziv-Lempel compression is examined in Chapter 8.

3.4.3 Coding Arbitrary Integers

Some Ziv-Lempel compression methods call for the encoding of arbitrarily large integers and a variety of techniques have been developed for doing so (see [BCW90] for a brief survey). It is normally desirable for small integers to have shorter codes than large ones. The best code is determined by the probability distribution of the integers in the actual application. Where such a distribution is known, arithmetic coding can be used to give an optimal encoding.

Assuming such a distribution is not available or unknown, then other methods must be used. A trivial approach is to double-up all the bits in the canonical binary representation and then flip the last bit. In this way the value and end of the number are readily

determined. For example, $19 = 10011_2$ would be encoded as 1100001110. Using this scheme every number except 0 starts with 1. By coding $x + 1$ rather than x, two bits can be saved since there is no need to store the leading 1.

Marked improvement is obtained by prefixing a number with its length in bits with the length expressed in the double-bit method. The number n is therefore encoded as $D(\lceil \lg n \rceil)n_2$ where D represents the doubling-up code. For example, 19 becomes 11001010011 (although for 19 this is worse than doubling each bit, for larger numbers the length method is always superior). Even better codes can be formed by prefixing the length by the length of the length, etc.

3.4.4 Other Compression Techniques

There is another large class of compression techniques called ***state-based modelling*** discussed in detail in [BCW90]. In principle this approach is more powerful than the context modelling used by PPM or dictionary coding as the general finite-state machine is able to capture some regularities (particularly those involving counting) that context models cannot. However, actual compressors using state-based models, like dynamic Markov models [HC86, CH87], do not harness this potential power and can be proven to be equivalent to finite context models [BM89].

One adaptive approach is ***dynamic Markov coding*** (DMC) [CH87] which starts with a small initial model and expands by the addition of new states as they are required. Frequency counts are maintained for each transition and new states are cloned when a transition becomes sufficiently popular (as governed by a heuristic).

The SEQUITUR grammatical inference method [NWM94] represents another type of compressor. SEQUITUR attempts to derive a hierarchical grammar for a sequence using a greedy heuristic. Although originally developed as a programming by demonstration tool, it can be used for compression. When used adaptively as a compressor its performance exceeds popular dictionary coders such as the UNIX `compress` utility.

Another compression technique, BLOCK, considers a message in blocks [BW94]. Each block is subjected to a reversible process which attempts to make the block easier to compress by conventional methods. The transformation groups symbols together so that the probability of finding a symbol close to another instance of the same symbol is high. A variety of simpler algorithms can then be used to compress the sorted blocks.

Yet another approach, WORD [Mof87], resembles PPM except that the alphabet consists of entire words rather than individual symbols. The message is expressed as an alternating sequence of words and nonwords (spaces, punctuation, etc.). Separate models are used for the compression of the words and nonwords.

3.5 Connection Between Cryptography and Compression

It has long been recognized that redundancy is the bane of cryptography [Sha49]. It was shown in Chapter 2 that the presence of redundancy allowed many classical ciphers to be broken with ciphertext only attacks.

The conventional cryptographic approach to hiding redundancy is not with compression but with ***diffusion***. Diffusion works by spreading the redundancy as evenly as possible over the ciphertext. Some ciphers such as the DES manage to do this very successfully. The related notion, ***confusion***, is about obscuring the relationship between the plaintext and ciphertext [Sha49].

There are also a number of other somewhat ad hoc techniques for reducing redundancy. For example, in ***homophonic coding*** common symbols are mapped onto a variety of different symbols in the ciphertext, in an attempt to get a flatter distribution. Boyd [Boy90] has developed such codes for use in conjunction with arithmetic coding.

Since compression is an ideal method for removing redundancy, it is surprising that compression rarely seems to have been seriously advocated as a technique for improving security. While it seems to be common lore that 'compression before encryption is a good thing', modern texts on cryptology, such as [Sch94], devote at best a page to compression. An exception is [Til88] which contains an entire chapter on Huffman coding. Part of this reticence may be due to difficulties in giving detailed analysis of the security properties of practical compression schemes. Compression schemes are very hard to analyze in this way, primarily because their effectiveness is deeply dependent on the message compressed. In contrast, the techniques used to achieve diffusion are applicable to all messages. From a cryptologic point of view, it is unwise to accept a system as secure when it is not possible to determine any security properties for it.

There have been some instances where the benefits and necessity of compression have been highlighted. As seen in Chapter 2, dictionary attacks are easily thwarted by compressing messages prior to encryption.

3.6 Using Compressors as Cryptosystems

Before a compressor can be used as a cryptosystem, it is necessary to introduce a key mechanism. In evaluating various methods, the following desiderata are proposed:

- The mechanism must be simple. A time-consuming key mechanism ameliorates the benefit of using a compressor for encryption, in that one may be better off to use a standard fast cryptosystem on the output of the compressor.
- The mechanism must be global. Ideally the key mechanism should afford at least some protection to every bit of the output. It is especially important that the early

bits of the output have some protection. This requirement eliminates certain schemes whereby extra bits are used to pad the compressed output at random locations, or where a selection of output bits are flipped according to some key schedule.

- The key should be short. To be useful the key mechanism should not call for keys much longer than those used in conventional cryptosystems. That is, while any compressor can be made unconditionally secure by exclusive-oring each bit of output with a one-time pad it is unlikely to be practical except in specialized applications. Note, however, that if one is intent on unconditional secrecy then compression before encryption represents the most economical way of using the bits of a one-time pad.
- The mechanism should not degrade the compression by more than an additive constant or at worst a near unity multiplicative constant. In particular, any mechanism that ends up producing output longer than the original plaintext is pointless.

These desiderate will be used when evaluating various key mechanisms in Chapters 7 and 8.

Chapter 4

Compression and Randomness

The primary motivation for data compression has always been making messages smaller so they can be transmitted quicker or stored in less space. Compression is achieved by removing redundancy from the message, resulting in a more 'random' output. Detectable regularities in compressed output indicate a less than perfect compressor. In theory, a detectable regularity could be used as the basis for further compression. In practice, this is often difficult to accomplish because of the requirement that the compressor run in reasonable time and space. In contrast, a cryptanalyst is prepared to expend considerable computation in an attempt to exploit such a weakness and recover the message.

Randomness and cryptology have always had close links. The design of stream ciphers is largely an exercise in the design of unpredictable random number generators. This is because a truly random sequence of bits can be used to give unconditional secrecy (for example, by a simple bit-by-bit exclusive-or of the random sequence and the message). Most of the research centres on expanding a short random key (often called the ***seed*** in this context) into a long random sequence. Using a seed is desirable since it can be committed to memory or easily exchanged and removes the requirement of the communicating parties exchanging long random sequences of bits.

Removal of redundancy makes ciphertext only attacks more difficult because the cryptanalyst has less statistical leverage. The ideal compressor would remove all redundancy and achieve a random output. In this chapter, tests of randomness are applied to a variety of compressors to gauge how well they achieve this goal. The results indicate that several compressors leave detectable regularities in the output. In particular, the results show that the splay tree, Block, and Ziv-Lempel compressors produce nonrandom outputs. This randomness deficiency is exploited in Chapter 8 where a ciphertext only attack is developed for Ziv-Lempel compressors. The better compressors are found to leave detectable regularities comparable to those of the DES. In particular, PPMD [Tea95] passes all the tests suggesting PPMD is more likely to be immune from ciphertext only attack than the other compressors tested.

In Section 4.1 the nature of randomness is briefly discussed. This is a contentious

issue and only superficial coverage is given. Section 4.2 explains popular methods for the generation of random sequences. Section 4.3 introduces four tests of randomness. These are applied to a variety of sources in Section 4.4. The remaining sections present other uses for the tests and discuss the results.

4.1 Randomness

The study of randomness is fraught with philosophical difficulties. Nobody really doubts that a sequence of a thousand consecutive heads from flipping a coin is a dubious event; nevertheless, *it is a possible outcome*, and for a truly random coin, *it is just as likely as any other outcome*. Despite this fact we would normally want to reject a sequence of one thousand consecutive heads as nonrandom.

Real randomness is hard to find. For example, it is a result of Kolmogorov complexity theory [LV93] that while the majority of binary sequences are random it is impossible to constructively exhibit even a single instance of such a sequence. The situation is particularly troublesome for finite sequences, where it is impossible to make a sharp distinction between random and nonrandom sequences. For instance, it is ridiculous to claim a particular finite sequence is random, but that by flipping a single bit the sequence becomes nonrandom. Further, the single coin flip is never random in practice. The only way a coin can be made to fall with equal probability on each side is for both sides to be absolutely identical, in which case we cannot tell on which side it landed.[1] Quantum physics shows that there is randomness in the world but that no detector can ever hope to accurately harness this randomness. Local physical sources of randomness (such as geiger-counter pulses and zener diode noise) are of little use to cryptology since it is necessary in most circumstances to reproduce the same random sequence at both ends of the communication link. An exception is in the generation of keys where randomness from physical devices is useful.

Kolmogorov complexity answers randomness questions by considering the length of the shortest program capable of producing a given sequence. The result is only accurate up to an additive constant. A sequence having high complexity $C(x) = |x| + O(1)$ is considered random. Roughly speaking, this says that random sequences cannot be compressed or that the shortest description of a random sequence is itself. Unfortunately, the Kolmogorov complexity of a given sequence is a noncomputable measure. This means we can never be completely convinced that a sequence is random. However, it is possible to decide with high probability that a sequence is nonrandom if some alarming statistical regularity is

[1] Given a coin with probability p of landing on heads, $0 < p < 1$, it is possible using multiple flips of this coin to distil out a random sequence. The coin is flipped twice, if both flips give the same result the flipping operation is repeated. If the flips are head-tail, then output a 1; otherwise, if the flips were tail-head, output a 0. The binary sequence so produced will be random in that the occurrence of a zero or one is equiprobable at each point in the sequence. This of course assumes there is no correlation between consecutive flips.

found in the sequence.

For each possible (computable) regularity a ***test for randomness*** can be devised.[2] The test will reject all strings having the given regularity. Clearly, if a sequence passes such a test it could still be nonrandom (it could have a statistical regularity of a different type). However, if a sequence passes a number of different randomness tests then this increases our confidence in the randomness of the sequence.

There is no end to the number of tests of randomness that can be conceived. It follows that every finite sequence must fail at least some of the possible tests. In particular, a given sequence x will always fail the special test which just checks the input for x and fails if it finds x.

A sequence can fail a test for two reasons. The sequence may deviate considerably from the expected outcome, or alternatively, the sequence may be too close to the expected result. As an example, consider rolling a six-sided dice, six million times. Then the expected number of times each value would occur is one million. In practice one would not expect this to be the exact outcome. Exactly one million occurrences of each number would be very suspicious. Likewise, if all six million rolls are the same number this is an extremely suspicious event.

Knuth [Knu81] identifies two main classes of tests: ***empirical tests*** and ***theoretical tests***. Empirical tests are those tests which are applied directly to the sequence while theoretical tests are those tests which are applied to the mechanism which generates the sequence. For a dice, an example of an empirical test is the frequency with which each number is produced; a theoretical test might involve checking that the area of each face on the dice was the same.

We will be primarily concerned with the analysis of binary sequences. Much of the literature on testing deals with sequences of real numbers uniformly distributed over the closed interval $[0, 1]$. The distinction need not be of concern as it is always possible to represent one form of a sequence by the other.

4.2 Generating Random Sequences

If producing good unpredictable random sequences was a trivial matter then most of contemporary cryptology would be unnecessary. Random sequences can be used as one-time pads to give unconditional secrecy. Unfortunately, a deterministic computer can never produce an unbounded random sequence. This is because any real computer only has a finite amount of state, thus any sequence produced must ultimately be periodic. The random number generators shipped with many computers are notoriously bad. For instance the low order bit of the `rand()` function provided under UNIX alternates between

[2]The name *test for randomness* is somewhat misleading. In practice a better name would be *test for violation of randomness* since such tests can only tell us when a sequence is nonrandom.

0 and 1. Recently, commonly used high-quality generators have led to incorrect results in Monte Carlo simulations of physical processes [FLW92, Pet92]. Generators with good statistical properties are known [PM88] and these are suitable for most simulation and numerical analysis. The extra requirement that generators be unpredictable rules out these generators for use in cryptology.

A popular form of random number generator are the ***linear congruential generators*** [Knu81] which have the form

$$X_{n+1} = (aX_n + b) \bmod m$$

where X_{n+1} is the next number in the sequence and X_n is the current number. The number X_0 is the seed. The variables a, b, and m are constants, and when appropriately chosen give the maximum period length of m. The generator given by $a = 7^5 = 16807$, $b = 0$, and $m = 2147483647$ has been recommended as a minimal standard [PM88]. These generators are very fast, requiring only a few operations per bit. However, they have been broken [Boy89] and are not suitable for use in cryptology. Generalized forms of these generators have also been studied and found to be insecure [Kra92].

Another popular class of random number generators are ***linear feedback shift registers*** (LFSRs) which consist of a finite width register and a ***tap sequence***. Each time a bit is needed, all of the bits in the shift register are shifted right and the LFSR outputs the least significant bit. The new left-most bit is computed by exclusive-oring the other bits in the register, in accordance with the tap sequence. An n-bit LFSR can have maximum period $2^n - 1$ [Gol67]. These generators are also easily broken [MT79]. Even so, LFSRs are often used as the building blocks for more complicated designs [Rue92].

Slower but more secure generators have been proposed which rest their security on complexity-theoretic assumptions. They are typically based on problems similar to those in public-key cryptosystems. Indeed there are proposals based on RSA [Sha81, BB88] and discrete logarithms [BM84]. Currently the most efficient and secure generator is the ***Blum Blum Shub*** (BBS) generator [BBS86]. It is also called the ***quadratic residue generator***.

DEFINITION 4.1: An integer n is a ***Blum integer*** if it is the product of two distinct primes p and q both of which are congruent to 3 modulo 4.

To use the BBS generator a Blum integer n is chosen along with another random integer x relatively prime to n, $\gcd(x, n) = 1$. Compute $x_0 = x^2 \bmod n$. This is the seed for the generator. The ith pseudorandom bit is the least significant bit of x_i where $x_i = x_{i-1}^2 \bmod n$. This can also be written $x_i = x_0^{2^i \bmod ((p-1)(q-1))} \bmod n$. Using the second form any particular bit of the sequence can be computed directly which is a desirable property in applications requiring random access. The generated bits cannot be predicted by anyone not knowing the factorization of n [BBS86].

The generator is slow but a few speedups are known. For instance it has been

proven [ACGS88] that it is actually safe to use the least significant $\lg |x_i|$ bits, where $|x_i|$ is the length of x_i in bits, rather than just the least significant bit.

Another option is to use some large publicly available sequence of 'random' bits, such as those distilled from certain astronomical sources. The key is then an index indicating where and when to start observing the random source. If the source is large enough it will be infeasible for a cryptanalyst to find the portion of the sequence used.

A large number of other proposed generators are discussed in [Rue92].

4.3 Empirical Tests

Empirical tests are used to detect statistical regularities in a sequence. This means that significant deviations from what is expected of a truly random sequence will be detected. Empirical tests work by examining a length of sample sequence, n, for a certain property. A sequence is rejected when it exhibits nonrandom behaviour in the property being tested. An empirical test T can be considered as a function with signature $T : A^n \rightarrow \{\text{ACCEPT}, \text{REJECT}\}$ where A is the alphabet of possible symbols. In most cases we will take $A = \{0, 1\}$. The function T partitions the set A^n of n-symbol sequences into a set of bad sequences B_T and a set of good sequences G_T defined by

$$B_T = \{s^n : T(s^n) = \text{REJECT and } s^n \in A^n\}, \qquad G_T = A^n \backslash B_T.$$

Normally, $|B_T| \ll |G_T|$ since there are more random sequences than nonrandom sequences.

The length n should be quite large ($n > 5000$ is typical) to ensure the test has adequate information to work with. The ratio $\rho = |B_T|/|A^n|$ is called the ***rejection rate***, and is the probability that an arbitrary sequence is rejected. In practical tests ρ should be small, $\rho \leq 0.01$.

Whether or not a defect in a sequence will be detected depends on the extent of the defect, on the length of the sequence examined, and on the rejection rate. There is a trade off between the ability to detect defects and the frequency of false alarms.

A statistical test, T, is typically implemented as a polynomial-time computable function f_T that maps the sequences of length n into the real numbers: $f_T : A^n \rightarrow \mathbb{R}$ with f_T such that the probability distribution of the real-valued random variable $f_T(R^n)$ can be determined, where R^n denotes a truly random sequence of length n. Two thresholds t_1 and t_2 are then specified such that

$$\Pr[f_T(R^n) \leq t_1] + \Pr[f_T(R^n) \geq t_2] = \rho.$$

Essentially these thresholds give the cutoffs for sequences with detectable regularities and with statistics too close to the expected value. The set B_T of cardinality $\rho|A^n|$ can then be defined by

$$B_T = \{s^n \in A^n : f_T(s^n) \leq t_1 \text{ or } f_T(s^n) \geq t_2\}.$$

Typically f_T is chosen so that $f_T(R^n)$ has the normal distribution or the χ^2-distribution. The normal distribution results when a large number of independent and identically distributed random variables are summed. The χ^2-distribution with d degrees of freedom results when the squares of d independent and normally distributed random variables with mean zero and variance one are summed.

Suppose our observations fall into k categories and that $n \gg k$ observations are made. Let p_i be the probability that an observation falls into category i. For the simple case where all categories are equally likely, $p_i = 1/k$. Finally, let f_i be the actual number of observations falling in category i. Then the χ^2-statistic is given by

$$V = \sum_{i=1}^{k} \frac{(f_i - np_i)^2}{np_i}.$$

Since $\sum p_i = 1$, we have $k-1$ degrees of freedom: intuitively this is because any particular p_i can be calculated given all the other p_i's.

The χ^2 test is only accurate when the expected number of observations in each category is at least five [Knu81]. This is why n must be considerably larger than k. In general, the larger n the more likely a bias in the sequence will be detected (assuming, of course, that such a bias exists). However, if n is very large, local nonrandom behaviour might not be detected.

4.3.1 The Poker Test

In a ***poker test*** [BP82, Knu81] the sequence is divided up into nonoverlapping blocks of length m. Thus, the first block consists of the first m symbols in the sequence, the second block consists of the next m symbols, and so on. The special case of $m = 1$ is called the ***frequency test***. The purpose of the test is to ensure that each possible block occurs with roughly the same frequency. The test can be applied for several values of m in order to get a more comprehensive result. For each block of length m there are $|A|^m$ possible patterns, where $|A|$ is the number of symbols in the alphabet. For a binary alphabet there are 2^m possible patterns of length m.

Let the length of the sequence be n, then $B = \lfloor n/m \rfloor$ blocks of length m can be extracted from the sequence. Any remaining bits are ignored. A count is kept of the number occurrences of each pattern: $f_0, f_1, \ldots, f_{2^m-1}$. These counts clearly sum to the total number of blocks:

$$B = \left\lfloor \frac{n}{m} \right\rfloor = \sum_{i=0}^{2^m-1} f_i.$$

For a random sequence we expect the B blocks to be uniformly distributed, thus the probability for any individual pattern is 2^{-m}. The χ^2-statistic is therefore given by

$$V = \sum_{i=0}^{2^m-1} \frac{(f_i - Bp_i)^2}{Bp_i} = \frac{2^m}{B} \sum_{i=0}^{2^m-1} f_i^2 - B.$$

Because there are 2^m patterns the result must be compared to the χ^2-distribution with $2^m - 1$ degrees of freedom giving the following criteria for evaluating our sequences:

if $V > \chi^2_{2^m-1}(0.0005)$	extreme failure (E)
else if $V > \chi^2_{2^m-1}(0.005)$	failure (F)
else if $V > \chi^2_{2^m-1}(0.025)$	suspect (S)
else if $V < \chi^2_{2^m-1}(0.9995)$	extreme failure (E*)
else if $V < \chi^2_{2^m-1}(0.995)$	failure (F*)
else if $V < \chi^2_{2^m-1}(0.975)$	suspect (S*)
else	pass (P)

The first three criteria test for sequences whose statistics vary considerably from what is expected for a random sequence. The next three test for sequences which are too random. If the sequence passes all the criteria it is said to have passed the test.

4.3.2 The Autocorrelation Test

In the ***autocorrelation test*** [BP82, Knu81] a sequence is compared to shifts of itself. It is a measure of how dependent bits of a sequence are on each other. The test simply counts the number of bits that match in the shifted sequences; and normalizes the result by the length of the overlapping portion of the sequence. Portions not overlapping are ignored. For example, given the sequence 01101101110000 the autocorrelation for a shift of one is 7/13 as can be seen from the alignment below:

0	1	1	0	1	1	0	1	1	1	0	0	0	0
	0	1	1	0	1	1	0	1	1	1	0	0	0	0

In general, given a sequence $s^n = s_1 s_2 \cdots s_n$ with $s_i \in \{0, 1\}$, let n_0 be the number of zero bits in s^n and n_1 the number of one bits in s^n; thus $n = n_0 + n_1$.

DEFINITION 4.2: The ***autocorrelation*** of s^n with ***shift*** $0 \leq d < n$ is given by

$$A(d) = \frac{1}{n-d} \sum_{i=1}^{n-d} s_i \equiv s_{i+d},$$

where $\equiv$ denotes bitwise equivalence. If $d = 0$, then we speak of the ***in-phase correlation*** and if $d \neq 0$ we speak of the ***out of phase correlation***.

Considering the case $d = 0$ we obtain

$$A(0) = \frac{1}{n} \sum_{i=1}^{n} s_i \equiv s_i = \frac{1}{n} \sum_{i=1}^{n} 1 = 1,$$

for every sequence. Such a value indicates a perfect correlation. Perfect anticorrelation corresponds to a value of zero.

Consider the autocorrelation of a random sequence when $d \neq 0$. The probability of a zero at any position is n_0/n and a one is n_1/n. Only when the two symbols match will any contribution be made to the autocorrelation, thus

$$E[A(d)] = \frac{n_0^2 + n_1^2}{n^2},$$

and

$$\begin{aligned}
\mathrm{Var}[A(d)] &= \frac{1}{n-d}\sum_{i=1}^{n-d}(s_i \equiv s_{i+d} - E[A(d)])^2 \\
&= \frac{1}{n-d}\left[\frac{n_0^2+n_1^2}{n^2}(n-d)\left(1-\frac{n_0^2+n_1^2}{n^2}\right)^2 + \frac{2n_1n_0}{2}(n-d)\left(0-\frac{n_0^2+n_1^2}{n^2}\right)^2\right] \\
&= \frac{2n_0n_1(n_0^2+n_1^2)}{n^4}
\end{aligned}$$

(Note that for most random sequences, $n_0 \approx n_1 \approx n/2$ giving $E[A(d)] \approx 1/2$ and $\mathrm{Var}[A(d)] \approx 1/4$).

A standard hypothesis test is applied for any particular sequence to see if its autocorrelation suggests a nonrandomness. Let ρ be the significance level, usually chosen to be 1% or 5%, then the sequence passes the test if

$$E[A(d)] - z_{\rho/2}\sqrt{\mathrm{Var}[A(d)]/(n-d)} \leq A(d) \leq E[A(d)] + z_{\rho/2}\sqrt{\mathrm{Var}[A(d)]/(n-d)}$$

and fails it otherwise.

By applying the test for several different values of d a more accurate picture of a sequence's autocorrelation is obtained. An autocorrelation differing greatly from 1/2 indicates a nonrandom sequence. However, if using a significance level of 5% we would expect on average one such failure per twenty shifts in a random sequence. An examination of the autocorrelation plot (Section 4.5) will reveal any regularity in the failures.

4.3.3 The Compression Test

Any data compressor can be used as a test for randomness. Random sequences do not have short descriptions and therefore cannot be compressed. Therefore, any sequence which is compressible must contain detectable regularities and thus cannot be random. By using a universal data compressor, known to be optimal in the information theoretic sense, there is a good chance of discovering nonrandomness in a sequence. If the sequence only compresses by a small amount, or expands, then the sequence passes the test. Accurate results can only be obtained if long sequences are used.

Let q denote the size of the alphabet. It is known that the worst-case input to an adaptive compressor elicits an expansion of $O\left(\frac{q+1}{2}\frac{\lg n}{n}\right)$ bits per symbol [CW84a, CW84b, BCW90]. However, such worst-case scenarios do not necessarily correspond to random

inputs. Therefore, consider the effect of encoding a random input, using a fixed order PPM model.

For the analysis it is assumed that the model is finite and complete; in that it contains all possible contexts. Any cost incurred in constructing the model is considered to be fixed overhead which is amortized over the length of the sequence. The remaining cost is then due to the escape probabilities and scaling mechanism. Further, since the model is complete only the costs incurred by the highest order need be considered.

An optimal coding of a random sequence of length n would use $\lceil n \lg q \rceil$ bits. In PPMC the probability of symbol i, in a given context, is $p_i = f_i/(F+q)$, $1 \leq i \leq q$, and where, $F = \sum f_i$, is the total number of symbols transmitted so far. The remaining probability, $q/(F+q)$, is assigned as an escape probability. Given a random input, all the f_i will have approximately the same value (because in a random sequence each block occurs about as often as other blocks of the same size); thus $f_i = F/q$. Finally, the values of the f_i are halved whenever F threatens to exceed some predefined maximum M. The quantity F increases by 1 for each symbol transmitted. After F reaches the value $M/2$ for the first time, count halving will occur every time another $M/2$ symbols have been transmitted. Thus, the expected length in bits per symbol of a random input compressed by PPMC is given by

$$\begin{aligned} E &= \frac{2}{M} \sum_{F=M/2}^{M} -\lg \frac{F/q}{(F+q)} \\ &= -\frac{2}{M} \lg \prod_{F=M/2}^{M} \frac{F/q}{(F+q)} \\ &= -\frac{2}{M} \left[-\frac{M}{2} \lg q - q + O(1/M) \right] \\ &= -\lg(1/q) + \frac{2q}{M} + O(1/M^2). \end{aligned}$$

This implies an excess over the optimal coding of

$$\frac{2q}{M} + O(1/M^2)$$

bits per symbol.

For $M = 16384$ and $q = 256$ (the values used in our application of the test) this gives an excess of about 0.031 bits per symbol. Therefore, an expansion of 0.031 or more bits per symbol, indicates a sequence which is not easily compressed. However, on short files results considerably worse than this may be observed because of additional costs incurred in construction of the model.

Similar analysis holds for other variants of PPM. Clearly, the excess can be reduced by increasing M or decreasing q. The real cause of the excess, however, is a failure to reduce the escape probability once all the possible symbols have been seen in a given context. This suggests that the compression of messages may be improved by up to 0.031 bits per

symbol simply by reducing the escape probability to zero once all the possible symbols in a context have been seen. Although the reduction of escape probabilities has been advocated previously [BCW90], we know of no implementation which actually does so.

4.3.4 The Maurer Universal Statistical Test

The ***Maurer test*** [Mau91, Mau92] theoretically supersedes many other tests discussed in the literature and is designed to detect cryptographic weaknesses. The test has two main advantages over many other statistical tests:

- The test is able to detect any of a very general class of defects, not just one specific defect. Any defect which can be modelled by an ergodic stationary source with finite memory will be detected with high probability.
- The test measures the actual amount by which the security of a cipher system would be reduced if the tested sequence were used as the key source. It therefore measures the effective key size.

The test essentially measures how compressible a sequence is. It does not actually compress the sequence, but calculates a quantity related to the length of the compressed sequence. The disadvantage of the test is that extremely long sequences are required for accurate results, far longer than those needed by the poker test or autocorrelation test.

The test takes three parameters $L, Q, K \in \mathbb{Z}^+$. The input is partitioned into adjacent nonoverlapping blocks of length L. The parameters are related by the equation $n = (Q+K)L$, where K is the number of test steps, Q is the number of initialization steps and n is the length of the sequence. Thus, in total there are $Q+K$ blocks $b_0, b_1, \ldots, b_{Q+K-1}$ and each block is of length L.

For $Q \leq m \leq Q+K-1$ the sequence is scanned for the most recent occurrence of block b_m; that is, the least $i \in \mathbb{Z}^+$ such that $i \leq m$ and $b_m = b_{m-i}$. Define

$$\alpha_m = \begin{cases} i & \text{if such an } i \text{ exists} \\ m & \text{otherwise} \end{cases}$$

Let the computable test f_U be defined by (the average logarithm of K terms)

$$f_U(s^n) = \frac{1}{K} \sum_{m=Q}^{Q+K-1} \lg \alpha_m.$$

Pseudocode for implementing the test is given in Figure 4.1. A table is used to store the values α_m.

Maurer [Mau91, Mau92] recommends choosing L between 8 and 16 inclusive, $Q \geq 5 \times 2^L$ and K as large as possible (at least 10^4). This choice for Q ensures with high probability

```
real Maurer (L, K, Q, ⟨b⟩)

int L, K, Q;
block ⟨b⟩;

begin
        int i, m;
        int α[];
        real sum ← 0.0;

        for (i ← 0; i < 2^L; i++)
                α[i] ← 0;
        for (m ← 0; m < Q; m++)
                α[b_m] ← m;
        for (m ← Q; m < Q + K; m++) begin
                sum ← sum + lg(m − α[b_m]);
                α[b_m] ← m;
        end;
        return (sum/K);
end;
```

Figure 4.1: Pseudocode for the Maurer's universal statistical test.

that every L-bit pattern occurs at least once in the first Q blocks. This is important for determining the threshold values t_1 and t_2. The values t_1 and t_2 are best given by

$$t_1 = \mu - y\sigma$$

$$t_2 = \mu + y\sigma$$

where

$$\mu = E[f_U(R^n)]$$

and

$$\sigma = \sqrt{\mathrm{Var}[\lg \alpha_m]/K} \approx \sqrt{\mathrm{Var}[f_U(R^n)]},$$

and y is chosen such that $\Phi(-y) = \rho/2$, where $\Phi(x)$ is the normal distribution function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, \mathrm{d}t.$$

Maurer [Mau91, Mau92] has shown that the expected values and the variance for a random sequence are given by

$$E[f_U(R^n)] = 2^{-L} \sum_{i=1}^{\infty} (1 - 2^{-L})^{i-1} \lg i,$$

$$\mathrm{Var}[f_U(R^n)] = \frac{2^{-L}}{K} \sum_{i=1}^{\infty} (1 - 2^{-L})^{i-1} \lg^2 i - E^2[f_U(R^n)].$$

The variance result assumes the α terms are statistically independent, and this is a good approximation for $L \geq 8$.

4.4 Example Application

In order to gauge how well various compressors remove redundancy, the tests described in the previous section were applied to a number of compressors. For comparison, the conventional cryptosystems DES [NBS77] and `crypt` [Sun88] were also included. The randomness of the DES has been studied previously [Fel88, CR89]. An ASCII file is included as a control. A sequence produced by a nonlinear additive feedback random number generator with period approximately $16(2^{31} - 1)$ was also tested [Coh88]. All the files used are summarized in Figure 4.2.

Results for the poker test, autocorrelation test, compression test, and Maurer's test appear in Figures 4.5–8 respectively. For the poker test and Maurer's test every block size between one and sixteen inclusive was used. The number of initialization steps in the Maurer test was $Q = 10(2^L)$. Shifts of up to two hundred bits were used in the autocorrelation test.

Type	Size (bytes)	Description
Random	3000000	A sequence produced by the `random()` function offered on the NeXT computer.
ASCII	1891806	The concatenation of the ASCII coded forms of book1, book2, and book3.
Crypt	1891806	The ASCII file encrypted using the `crypt` command provided under UNIX. This program essentially implements a one-rotor machine similar to the Enigma.
DES	1891808	The ASCII file encrypted using the DES algorithm in electronic code book mode with key $0011011100110000110101001101_2$. The file is two bytes longer as it needs to be padded out to a multiple of eight bytes.
DES′	1891808	The ASCII file encrypted using the DES algorithm in ciphertext block chaining mode with key $0011011100110000110101001101_2$. The file is two bytes longer as it needs to be padded out to a multiple of eight bytes.
Splay	1485768	The ASCII file encoded using the semisplay tree compression algorithm [Jon88].
LZH	850839	The ASCII file compressed using the `lharc` program, which uses the LZH compression method [Bre87].
GZIP	710864	The ASCII file compressed using the UNIX `gzip` program operating in its best compression mode.
Block	577943	The ASCII file compressed using the BLOCK algorithm of Burrows and Wheeler [BW94].
PPMC(3)	568019	The ASCII file compressed using variant C of the PPM algorithm running at order three and with a bounded amount of memory (65 K) [CW84b].
PPMD(5)	507368	The ASCII file compressed using variant D of the PPM algorithm running at order five and with an unbounded amount of memory [Tea95].
PPMD(5)+BC	491516	The ASCII file compressed using variant D of the PPM algorithm running at order five, with an unbounded amount of memory, and using a model primed with the Brown Corpus.
JPG	71054	A colour image of an aeroplane encoded using the Joint Photographic Experts Group (JPEG) standard.

Figure 4.2: A collection of sources for which randomness tests were applied.

Input	Block size															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Random	P	P	P	P	P	P	P	P	S*	P	P	P	P	P	P	P
ASCII	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
Crypt	S	E	P	E	F	S	E	E	E	E	E	E	E	E	E	E
DES	P	P	S	E	E	E	E	E	E	E	E	E	E	E	E	E
DES′	P	P	P	P	P	S	P	P	P	P	P	P	P	P	P	S*
Splay	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
LZH	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GZIP	E	E	E	E	E	E	E	E	E	E	E	E	E	F	S	E
Block	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	
PPMC(3)	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	
PPMD(5)	S	P	S	P	P	P	P	P	P	P	P	P	P	P	P	
PPMD(5)+BC	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	
JPG	E	E	E	E	E	E	E	E	E	E	E	E	E			

Figure 4.3: The results of the poker test. An E indicates an extreme failure (worse than one chance in a thousand), F a failure (worse than one chance in a hundred), S a suspect (worse than five chances in a hundred), and P a pass. An asterisk indicates the result was better than should be expected. A blank indicates that not enough information was gathered to make the result accurate.

File	Passes	Rating
Random	99.5%	Pass
ASCII	0.0%	Fail
Crypt	89.5%	Fail
DES	95.0%	Pass
DES′	99.5%	Pass
Splay	20.0%	Fail
LZH	79.05%	Fail
GZIP	33.0%	Fail
Block	1.0%	Fail
PPMC(3)	89.3%	Fail
PPMD(5)	99.5%	Pass
PPMD(5)+BC	99.0%	Pass
JPG	80.5%	Fail

Figure 4.4: The results of the autocorrelation test. Passes indicates the percentage of shifts for which the sequence had no obvious autocorrelation.

File	PPMC(3)	PPMD(5)	LZH	Overall
Random	112.35	111.48	100.00	Pass
ASCII	30.03	26.82	44.97	Fail
Crypt	112.07	110.79	100.00	Pass
DES	108.96	92.60	98.28	Suspect
DES′	112.34	112.61	100.00	Pass
Splay	54.56	59.16	57.43	Fail
LZH	112.04	113.34	100.00	Pass
GZIP	112.20	114.11	100.01	Pass
Block	108.11	107.30	97.27	Pass
PPMC(3)	112.26	114.31	100.01	Pass
PPMD(5)	112.37	114.43	100.01	Pass
PPMD(5)+BC	112.35	114.43	100.01	Pass
JPG	110.14	107.12	99.74	Pass

Figure 4.5: The results of the compression test expressed as percentages of the original files. A number greater than 100 indicates expansion.

Input	Block size															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Random	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
ASCII	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
Crypt	P	E	P	E	P	E	P	P	P	P	P	P	S	E	S	E
DES	P	P	P	F	P	P	P	F	P	S	P	E	F	E	E	E
Splay	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
LZH	E	P	E	E	E	E	E	E	E	E	E	E	E	E	E	
GZIP	F	P	F	E	E	E	E	E	F	F	F	F	P	P	P	
Block	E	E	E	E	E	E	E	E	E	E	E	E	E	E		
PPMC(3)	E	E	E	E	S	F	P	F	S	P	P	F	S	E		
PPMD(5)	P	P	P	P	P	P	P	P	P	P	P	P	P	P		
PPMD(5)+BC	P	P	P	P	P	P	P	P	P	P	P	P	P			
JPG	E	P	E	E	E	E	E	E	E	E	E	E				

Figure 4.6: The results of the Maurer universal test. An E indicates an extreme failure, F a failure, S a suspect, and P a pass. A blank indicates that not enough information was gathered to make the result accurate.

Based on the results the sources can be ranked (from least random to most random): ASCII, JPG, Splay, Block[3], GZIP, LZH, PPMC(3), Crypt, DES, DES′, PPMD(5), PPMD(5)+BC, Random.

As expected the ASCII file does badly in all the tests. The colour image fails all the tests except for the compression test. The semisplay and Block compressors also perform badly and apparently much redundancy remains in the file. However, only a single symbol splay model was used and some improvement should occur in a splay compressor that took bigrams and trigrams into account. The Block compressor does extraordinarily badly in the autocorrelation test. The Ziv-Lempel compressors, LZH and GZIP, pass only the compression test. PPMC(3) does only marginally better. The `crypt` program passes the compression test and does well in the Maurer test. The DES passes the autocorrelation test, does well in the Maurer test, but no so well in the compression test. The remaining three sources, PPMD(5), PPMD(5)+BC, and Random pass all the tests; they are the only ones to pass the poker test.

4.5 Other uses for Autocorrelation

It is instructive to consider a plot of the autocorrelation versus the shift for a sequence. Figure 4.7 shows such a plot for the file ASCII. The regularities in this plot should be immediately obvious. In particular, there is a large peak at shift eight and at every multiple of eight. These shifts correspond to moving the sequence a multiple of eight bits; that is, some number of symbols. It is not surprising to see higher than usual correlation in these cases because the most significant bit in ASCII is always zero. Further, the alignment of common symbols, such as space and e, also contribute to the high correlation. In fact, the reason why the peak at shift eight is smaller than the other peaks is because very few spaces become aligned with a shift of one symbol.

Another example, Figure 4.8, shows a similar plot for geo. This time the peaks occur every thirty-two bits suggesting that the information in geo is encoded using a thirty-two bit data structure. This is known to be the case [BCW90].

Autocorrelation is therefore a tool for determining the phase of a data file and is useful for compression techniques such as PPMD [Tea95]. Similar techniques are used in cryptography to find the period of repeated keys [Kah66, Sin66].

Figure 4.9 shows a plot for a file which passes the autocorrelation test, PPMD(5). Notice that the scale on the ordinate is much smaller than on the previous plots. Further, there is no discernible pattern to the peaks on the plot. Obviously, it is desirable for cryptosystems to have such an irregular autocorrelation profile. The corresponding plots for the DES and Block are given in Figures 4.10 and 4.11 respectively.

[3]In the tests a very early version of the BLOCK algorithm was used. This version used Huffman coding as the final stage. Since then newer implementations have been developed and considerably more is now known about this compressor [Fen96]. It is likely that these newer implementations would perform better.

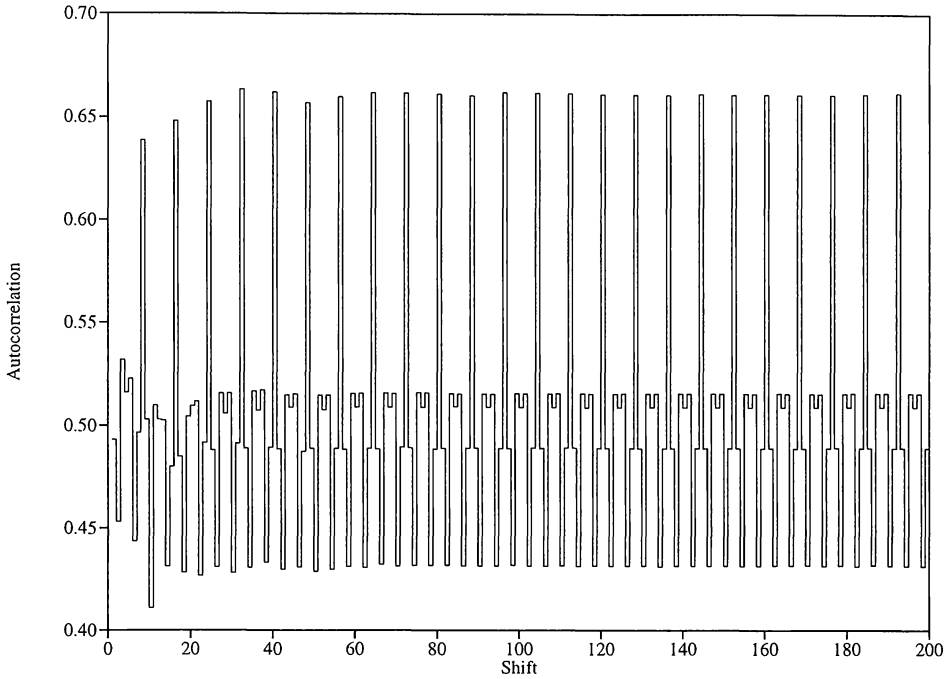


Figure 4.7: Autocorrelation plot for ASCII, illustrating regular peaks at every eight bits corresponding to shifts of whole symbols.

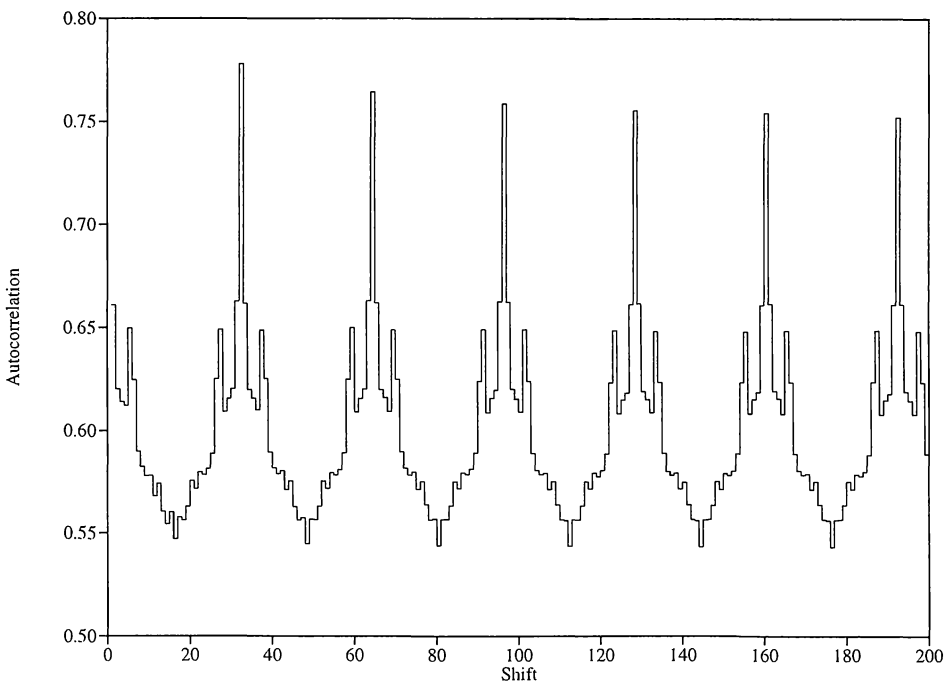


Figure 4.8: Autocorrelation plot for geo, illustrating regular peaks at every thirty-two bits.

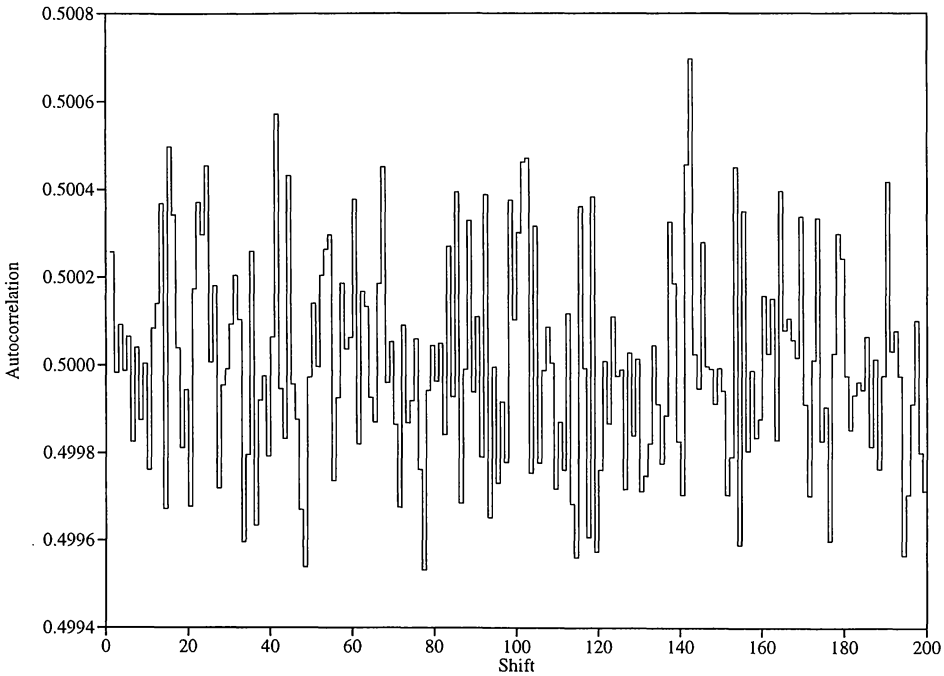


Figure 4.9: Autocorrelation plot for PPMD(5), illustrating irregular peaks and no discernible pattern.

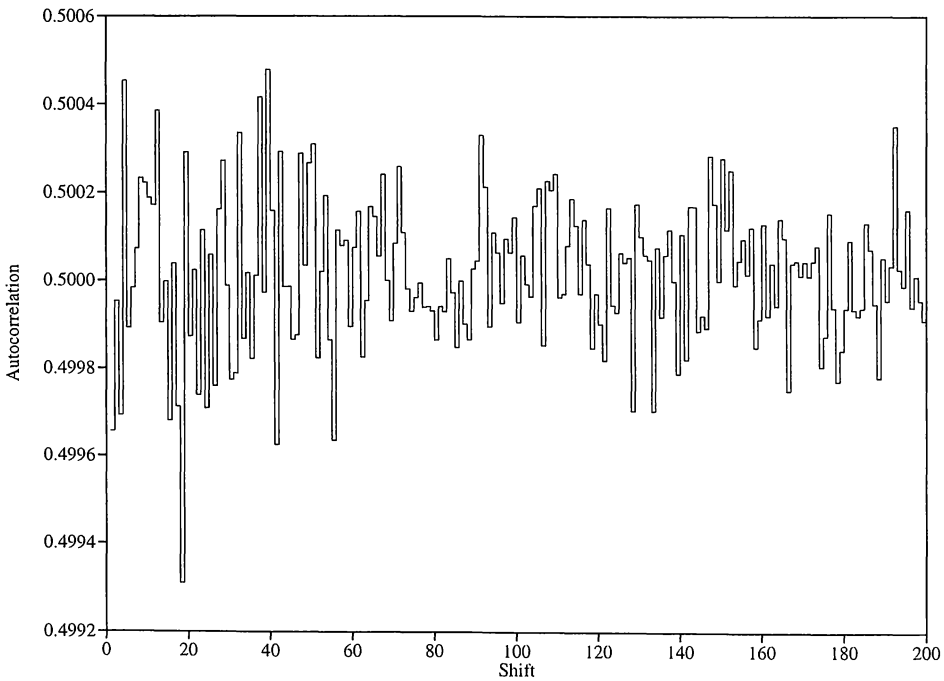


Figure 4.10: Autocorrelation plot for the DES, illustrating irregular peaks and no discernible pattern.

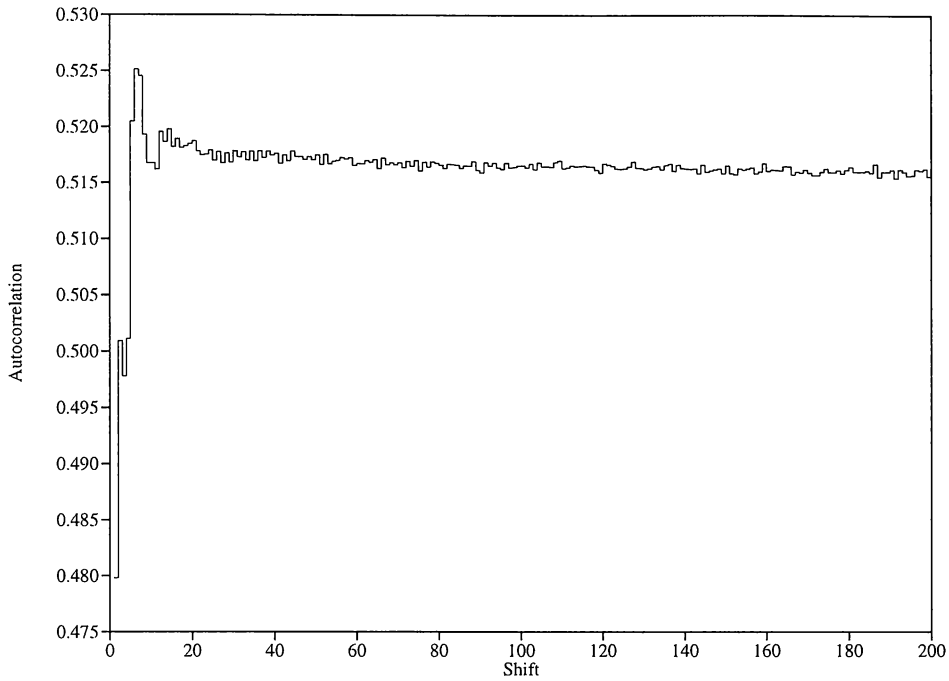


Figure 4.11: Autocorrelation plot for Block, indicating an unusual autocorrelation spectrum.

The plot for Block is particularly unusual. For any shift greater than three there is definite correlation. This suggests the Block compressor could be improved by taking these correlations into account.

4.6 The Wider View

The detection of a regularity in a compressor indicates that it is not optimal. In theory this indicates that the compressor could be improved. However, in many cases it might be computationally expensive to remove the remaining redundancy and in some cases it might not be clear how to remove it at all. In practice there will always be a trade-off between compression and execution costs. For example, the results above indicate that a splay compressor can be improved by subjecting the output to further compression using a PPM approach. But it is unclear how the regularities detected in the LZH compressor could be removed.

Many improvements will simply not be worth it from a compression point of view. Even in current algorithms a number of approximations are used in an effort to make the compressors run faster or use less space [BCW90]. However, from a cryptologic point of view such improvements could easily make the difference between a secure system and an insecure one.

Currently, there is no ciphertext only attack against the DES which is significantly

better than an exhaustive search. Further, the DES has been extensively studied for over two decades, so this lack of attack is not from a lack of trying. It seems reasonable to conjecture that PPMD(5), which outperforms the DES in all the tests, will have no ciphertext only attack. Further argument for this point is given in Chapter 9.

For those systems which performed worse than the DES the situation is less hopeful. The `crypt` program does only slightly worse than DES and yet there are attacks against it [RW84]. The Ziv-Lempel compressors, Block, and splaying do considerably worse than `crypt`, suggesting the possibility of a ciphertext only attack. Indeed, in Chapter 8 an automatic ciphertext only attack for Ziv-Lempel compressors is given.

The success of the PPM compressor suggests that it may be possible to use a compression program as a random number generator. Indeed, the DES has been suggested as a random number generator by the same argument [Gai77]. It was found that repeated application of order-3 PPMC caused a file to be expanded by about 12% per iteration. At each stage the resulting 'compressed' file passed all four randomness tests. It seems likely the majority of expansion is due to the blending and escape mechanisms used in PPM. A more theoretical approach to this problem would be helpful in settling this question. A general study on the performance of compressors on random inputs might give some fruitful insight into building compressors that cope better with unexpected (that is, difficult to predict) input. Of course, every compressor must expand some random inputs slightly, otherwise it would be impossible to compress other longer sequences.

Chapter 5

The Security of Arithmetic Coding

Arithmetic coding was introduced in Chapter 3 as an optimal coding technique. Since arithmetic coding is used as the coder for several compression schemes it is appropriate that its security properties be investigated. The security of arithmetic coding is analyzed from two perspectives. These results have previously been published in [CIR95, ICR95].

We first consider the difficulty of determining the state of an arithmetic coder using a chosen sequence. It is found that a $w + 2$ symbol sequence is sufficient to determine a w-bit probability. Although the detailed analysis is only for the static binary case, it is indicated how the attack could be extended to multisymbol and adaptive situations.

Second, the use of a key in arithmetic coding is considered. A number of different key schemes are found to be equivalent to the subset sum problem, a known **NP**-complete problem. However, the resulting subset sums typically fall into a class of sums known to be solvable in polynomial time. In particular, they can be solved by the short-vector algorithm [LO85].

One way of viewing the operation of arithmetic coding is that it generates as output a single real number which is uniformly distributed (the individual bits are random) with respect to the probability distribution of the model. Sufficient leading bits from the real number output are transmitted to ensure that the original text can be decoded. The analysis in Section 5.2 approximates the actual (finite precision) encoder with an infinite precision one. In the binary case, the output for a particular plaintext sequence can then be expressed as a polynomial in the unknown probability. Section 5.3 shows how to infer the unknown probability by solving the polynomial. Section 5.4 examines the practical (finite precision) case and shows by example that the infinite precision approximation gives results sufficiently close to the actual answer that there is no difficulty in computing the finite precision probability. Section 5.5 extends the attack to known plaintext. Section 5.6 considers related schemes.

In Section 5.7, the subset sum problem is introduced. It is shown that even when a solution is known to exist, finding the solution is **NP**-complete. In Section 5.8, a key is added to the arithmetic code and the resulting cryptosystem is found to be equivalent to the subset sum problem. Unfortunately, the sums that arise are found to fall into a special class of sums solvable in polynomial time.

We conclude that arithmetic coding does not greatly enhance the security of the driving model.

5.1 Introduction

Since arithmetic coding is optimal in the information-theoretic sense it is not unreasonable to assume that the addition of arithmetic coding to a communication system cannot weaken the system. However, because practical implementations of arithmetic coding use finite precision arithmetic, they are not quite optimal, and a fractional increase in the length of the output, to the order of $O(2^{-w})$ for w-bit arithmetic is incurred [Boy90, How93]. The losses are caused by the use of finite-precision arithmetic, the periodic scaling of frequency counts (only a problem for adaptive systems), and the cost of encoding a termination symbol. Since 32-bit arithmetic is readily available, this fractional increase is easily made negligible.

The security of arithmetic coding in conjunction with an adaptive model was first studied by Bergen [BH92, BH93]. Her results were highly dependent on the model and did not directly address the security of arithmetic coding itself. Further, her attacks required long chosen plaintexts, whereas the attacks presented here only require short plaintexts and often a known rather than chosen plaintext will suffice. The first suggestion which we are aware of for the use of arithmetic coding in encryption is [Rub79b], which contains an example of a multiple known plaintext attack against arithmetic coding. This topic is also discussed by Witten and Cleary [WC88].

To simplify the analysis we initially consider a single binary probability distribution where the probability of either symbol occurring is fixed but unknown. We then consider the problem of how to extract the probability given some sample output from the encoder. Once the probability is known, all future text can be decoded. It is shown that if the probability is stored to w-bits of precision, then using chosen plaintext, the probability can be determined using $w + 2$ symbols. For many known plaintexts the attack will take $w + m + O(\log m)$ symbols, where m is the length of an initial sequence (0^m or 1^m) containing just one of the two possible symbols.

This is a very strong negative result for the use of arithmetic coding on its own as an encryption technique. Stronger for example, than the results in [BH92] which required assumptions about the model. This negative result seems to carry over to a wide range of related variations in the encoding mechanism (the probability is adapted, the initial state of the encoder is unknown, the alphabet has more than two symbols, and so on) so long

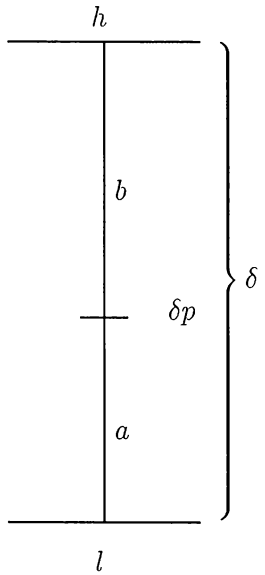


Figure 5.1: A binary arithmetic coder.

as it is possible to repeatedly reset the encoder and start a new encoding sequence. If it is not possible to reset, then it is an open question as to how hard it is to attack an encoder using these variations.

5.2 A Static Model

Consider encoding the output of a binary source with alphabet $\{0, 1\}$ using a static model. Let p be the probability used to encode a 0 and $q = 1 - p$ be the probability used to encode a 1. Neither probability can be zero or one as we must always allow some probability of each symbol occurring. The model is set up as illustrated in Figure 5.1 where l is a lower bound and h an upper bound on the output real number. δ is the difference between the upper and lower bound. Initially $l = 0$, $h = 1$, and $\delta = 1$. At each succeeding step the interval δ is narrowed and either l is increased or h is decreased.

Assume we have an infinite precision arithmetic coder. The following recursive relations, where l', δ', and h' are the new values for l, δ, and h, define characteristic polynomials for l, h, and δ.

If a 0 is encoded then

$$\begin{aligned} l' &= l, \\ \delta' &= p\delta, \\ h' &= h - q\delta = l + \delta p. \end{aligned} \tag{5.1}$$

If a 1 is encoded then

$$\begin{aligned} l' &= l + p\delta, \\ \delta' &= q\delta, \\ h' &= h. \end{aligned} \tag{5.2}$$

Let the input sequence be $S = s_1 s_2 \cdots s_i \cdots s_n$, where $s_i \in \{0, 1\}$. Define

$$\begin{aligned} a_i &= \begin{cases} 1 & \text{if } s_i = 0 \\ 0 & \text{otherwise} \end{cases}, \\ b_i &= \begin{cases} 1 & \text{if } s_i = 1 \\ 0 & \text{otherwise} \end{cases}, \\ A_j &= \sum_{i=1}^{j} a_i, \\ B_j &= \sum_{i=1}^{j} b_i. \end{aligned}$$

That is, A_j and B_j count the number of 0s and 1s respectively in the input up to and including the jth symbol.

Let the characteristic polynomials for l, h, and δ after encoding i input symbols be l_i, h_i, and δ_i respectively. From Equations 5.1 and 5.2

$$\delta_i = p^{A_i} q^{B_i} = p^{A_i}(1-p)^{B_i}.$$

The lower bound only changes when a 1 is sent, thus

$$l_i = \sum_{j=1}^{i} b_j p \delta_{j-1}.$$

Likewise the upper bound only changes when an 0 is sent, thus

$$h_i = 1 - \sum_{j=1}^{i} a_j q \delta_{j-1} = 1 - \sum_{j=1}^{i} a_j (1-p)\delta_{j-1}.$$

On occasion we will have need of the derivatives of the characteristic polynomials. The derivatives are easily calculated to be:

$$\frac{\mathrm{d}}{\mathrm{d}p}\delta_i = p^{A_i - 1}(1-p)^{B_i - 1}[(1-p)A_i - pB_i], \tag{5.3}$$

$$\frac{\mathrm{d}}{\mathrm{d}p} l_i = \sum_{j=1}^{i} b_j p^{A_{j-1}}(1-p)^{B_{j-1}-1}[(A_{j-1}+1)(1-p) - B_{j-1}p],$$

$$\frac{\mathrm{d}}{\mathrm{d}p} h_i = \sum_{j=1}^{i} a_j p^{A_{j-1}-1}(1-p)^{B_{j-1}}[p(B_{j-1}+1) - A_{j-1}(1-p)]. \tag{5.4}$$

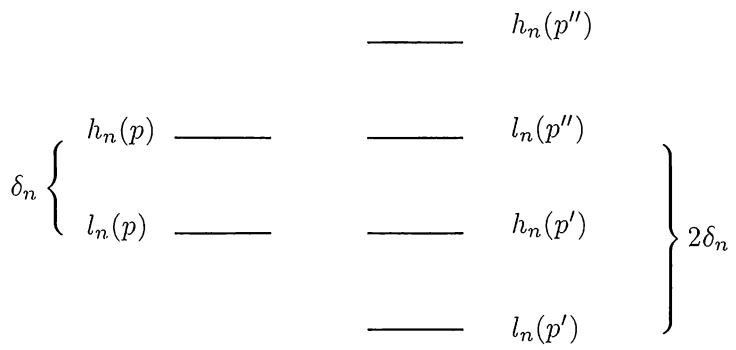


Figure 5.2: Range of decoding estimates. $\Delta p = p'' - p'$.

5.3 Inferring p

In this section bounds on the number of symbols necessary to determine the probability p are obtained. It will be shown that a $w + 2$ symbol chosen plaintext is sufficient to uniquely determine p. This is a simplified version of the general problem. In practice there may be many unknowns the attacker needs to determine besides the single probability p. Comments about the more general problem are made in Section 5.6.

The interval left at the end of encoding the sequence S of length n using probabilities p and q will be $\delta_n = h_n - l_n$. Let $\hat{p}$ be an estimate of p. If we are going to succeed in decoding all n symbols using $\hat{p}$ we require an overlap between the coding interval which resulted when S was encoded with p and the coding interval which would result if S were encoded using $\hat{p}$. Let the width of values of p that decode correctly be Δp. Then the range of either l_n or h_n can be twice the width of the interval; that is,

$$\Delta p \frac{\mathrm{d}}{\mathrm{d}p} l_n = 2\delta_n,$$

or equivalently,

$$\Delta p \frac{\mathrm{d}}{\mathrm{d}p} h_n = 2\delta_n. \tag{5.5}$$

In obtaining these equations we have made the assumption that the width of the coding interval resulting from encoding S does not change significantly over the range Δp, as shown in Figure 5.2. The assumption is reasonable because the derivative $\left|\frac{\mathrm{d}}{\mathrm{d}p}\delta_n\right|$ will be very small for most sequences. Later we verify this assumption more rigorously.

If we are going to succeed in determining p, then we require Δp to be as small as possible. This can be achieved by making δ_n small and $\frac{\mathrm{d}}{\mathrm{d}p}h_n$ large (or equivalently $\frac{\mathrm{d}}{\mathrm{d}p}l_n$ large). In particular, for a w-bit arithmetic encoder we require $\Delta p \leq 2^{-w}$ to uniquely determine p. The following theorems show that $\frac{\mathrm{d}}{\mathrm{d}p}h_n$ is indeed large and that the attack will succeed for sequences beginning with 101 or 010.

THEOREM 5.1: For any sequence beginning with 101,

$$\frac{\mathrm{d}}{\mathrm{d}p}h_n \geq \begin{cases} 1 & \text{for } 0 < p \leq \frac{1}{2} \\ 2(1-p) & \text{for } \frac{1}{2} < p < 1 \end{cases} \tag{5.6}$$

PROOF: Using Equation 5.4 we have

$$\frac{\mathrm{d}}{\mathrm{d}p}h_n = 2 - 2p + \sum_{j=3}^{n} a_j p^{A_{j-1}-1}(1-p)^{B_{j-1}}[p(B_{j-1}+1) - A_{j-1}(1-p)].$$

Terms in the sum will be negative when

$$A_{j-1} > \frac{p}{1-p}(B_{j-1}+1).$$

Next note $p^{A_{j-1}-1}(1-p)^{B_{j-1}}$ is decreasing for increasing j, and consequently the interval is narrowed between subsequent 0s. Sending 1s cannot cause expansion. Therefore the derivative will be a minimum when the sequence is 101 or 1010^{∞}. If the sequence is 101 then the derivative is $2-2p$ and the result follows. It remains to show that if the sequence is 1010^{∞} then the derivative is at least 1 for $0 < p \leq \frac{1}{2}$ and at least $2(1-p)$ for $\frac{1}{2} < p < 1$.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}p}h_n(1010^{n-3}) &= 2 - 2p + (1-p)^2 \sum_{j=4}^{n} p^{j-4}[jp - j + 3] \\
&= 2 - 2p + (1-p)^2 \left[\sum_{j=4}^{n} jp^{j-3} - \sum_{j=4}^{n} jp^{j-4} + 3\sum_{j=4}^{n} p^{j-4} \right] \\
&= 2 - 2p + (1-p)^2 \left[np^{n-3} - 4 - \sum_{j=5}^{n} p^{j-4} + 3\sum_{j=4}^{n} p^{j-4} \right] \\
&= 2 - 2p + (1-p)^2 \left[np^{n-3} - 4 + 3 + 2\sum_{j=5}^{n} p^{j-4} \right] \\
&= 2 - 2p + (1-p)^2 \left[np^{n-3} - 1 + 2\frac{p(1-p^{n-4})}{1-p} \right] \\
&= (1-p)^2 \left[np^{n-3} - 1 + \frac{2p - 2p^{n-3} + 2}{1-p} \right] \\
&= (1-p)\left[1 + 3p + (n-2)p^{n-3} - np^{n-2} \right]
\end{aligned}$$

If $0 < p \leq \frac{1}{2}$,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}p}h_n(1010^{n-3}) &= 1 + 2p + (n-2)p^{n-3} - (2n-2)p^{n-2} + np^{n-1} - 3p^2 \\
&\geq 1 + p[2 + (n-2)p^{n-4} - (n-2)p^{n-4} + np^{n-2} - 3p] \\
&\geq 1 + p[2 - 3p + np^{n-2}] \\
&\geq 1
\end{aligned}$$

To show,

$$\frac{\mathrm{d}}{\mathrm{d}p}h_n(1010^{n-3}) \geq 2(1-p)$$

for $\frac{1}{2} < p < 1$, it suffices to show that $3p + (n-2)p^{n-3} - np^{n-2} \geq 1$. For the case $n = 3$ it is easy to see that $3p + (n-2)p^{n-3} - np^{n-2} = 1$.

Now, $3p + (n-2)p^{n-3} - np^{n-2} \geq 1$ if and only if $n \geq \frac{(1-3p)/p^{n-3}+2}{1-p}$. The right-hand side of this last inequality cannot exceed $2/(1-p)$, since $1 - 3p/p^{n-3} \leq 0$ for all cases under consideration. It follows that $n \geq 4$ satisfies the inequality.

Thus, $\frac{\mathrm{d}}{\mathrm{d}p}h_n(1010^{n-3}) \geq 2(1-p)$, for $\frac{1}{2} < p < 1$ and $n \geq 3$, which is what we wanted to prove.

■

THEOREM 5.2: For any sequence beginning with 010,

$$\frac{\mathrm{d}}{\mathrm{d}p}h_n \geq \begin{cases} 2p & \text{for } 0 < p \leq \frac{1}{2} \\ 1 & \text{for } \frac{1}{2} < p < 1 \end{cases}$$

PROOF: By symmetry. Apply Theorem 5.1 to $1-p$ and $\overline{S}$.

■

THEOREM 5.3: For the plaintext $S = (10)^{n/2}$, $w+2$ symbols are sufficient to determine a w-bit probability.

PROOF: First let us check the magnitude of $\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})$. Using Equation 5.3,

$$\left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})\right| = \left|\frac{n}{2}p^{n/2-1}(1-p)^{n/2-1}(1-2p)\right|$$

which is a maximum when,

$$p = \frac{1}{2} \pm \frac{1}{2\sqrt{n-1}}.$$

It follows that (especially as n gets large)

$$\left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})\right| \ll 1.$$

To show that

$$\left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})\right| \ll 2(1-p)$$

it suffices to show that

$$\phi = \left|\frac{n}{2}p^{n/2-1}(1-p)^{n/2-2}(1-2p)\right| \ll 2.$$

ϕ is a maximum when

$$p = \frac{1}{2}\frac{2n-3+\sqrt{4n-7}}{2n-4}.$$

Thus,

$$\left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})\right| \ll 1 \text{ and } \left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}((10)^{n/2})\right| \ll 2(1-p)$$

except when $p \to 1$, in which case replace the $\ll$ by $\leq$.

If $p \leq \frac{1}{2}$ then $\frac{\mathrm{d}}{\mathrm{d}p}h_n \geq 1$ and $\delta_n \leq \left(\frac{1}{2}\right)^n$ (with equality occurring when $p = \frac{1}{2}$). So from Equation 5.5:

$$\Delta p \leq 2\left(\frac{1}{2}\right)^n = \left(\frac{1}{2}\right)^{n-1}$$

If $p > \frac{1}{2}$ then $\frac{\mathrm{d}}{\mathrm{d}p}h_n \geq 2(1-p)$. So from Equation 5.5:

$$\begin{aligned}
\Delta p &\leq \frac{2p^{n/2}(1-p)^{n/2}}{2(1-p)} \\
&\leq p^{n/2}(1-p)^{n/2-1} \\
&\leq [p(1-p)]^{n/2-1} \\
&\leq \left(\frac{1}{4}\right)^{n/2-1} \\
&= \left(\frac{1}{2}\right)^{n-2}
\end{aligned}$$

Therefore a sequence of $w+2$ symbols is sufficient to uniquely determine the state of a w-bit register.

■

Other sequences work as well. Since $\delta_n = p^{A_n}(1-p)^{B_n}$, it is desirable to send more 0s when p is near zero and more 1s when p is near one to ensure δ_n is small.

In practice, a few more symbols than suggested may be needed since actual implementations of arithmetic coders have small errors due to the use of finite-precision arithmetic. Figure 5.3 gives experimental evidence to support the theory developed above (when an arithmetic coder adapted from [WNC87] is used). In all cases, at most $w-1$ symbols were needed—slightly better than the bound achieved above. We did not allow probabilities $p = 0$ and $p = 1$ since these would cause infinite length outputs, but every other possible w-bit probability was tried. The number of bits used in the arithmetic was 17 (the implementation required this so that overflow and underflow did not occur).

5.4 Chosen Plaintext Attack

We have seen that a sequence of $w+2$ symbols is sufficient to uniquely determine p in a w-bit arithmetic coder. This section shows how an attacker can mount a chosen plaintext attack to efficiently determine the value of p.

Since the attacker knows the plaintext, S, the attacker can determine the characteristic polynomials $h(S)$ and $l(S)$, the bounds for the coding interval. By examining the output of the arithmetic coder, the attacker can determine the result of encoding S, call this value

w	maximum symbols needed
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14

Figure 5.3: Experimental results on the number of symbols needed to uniquely determine the state of an arithmetic coder for the sequence $(ab)^*$.

γ. Since γ will lie in the interval bounded by l_n and h_n, we have $l(S) \leq \gamma \leq h(S)$. By solving these polynomial inequalities the attacker can bound p.

For example, suppose we encode the plaintext $S = 10101010$ using an 8-bit arithmetic coder with $p = 94/256 \approx 0.3672$. The output of an ideal arithmetic coder could be 011110101 which corresponds to $\gamma \approx 0.4785$.

The characteristic polynomials for 10101010 are

$$h(10101010) = p + p^2 - 6p^5 + 9p^6 - 5p^7 + p^8,$$

and

$$l(10101010) = p + p^2 - p^4 - 2p^5 + 3p^6 - p^7.$$

The following inequalities hold:

$$p + p^2 - p^4 - 2p^5 + 3p^6 - p^7 \leq 0.4785 \leq p + p^2 - 6p^5 + 9p^6 - 5p^7 + p^8.$$

By solving these polynomials, we get the following estimates for p: from $l(S)$, $p \approx 0.3683$; from $h(S)$, $p \approx 0.3663$. As expected, $p = 94/256 \approx 0.3672$ falls between these two constraints and further 93/256 and 95/256 lie outside these constraints. So we have successfully uniquely identified p to be 94/256.

The value of γ will vary slightly depending on the implementation. The arithmetic coder we used for the experiments above actually gave 0111101011 which corresponds to $\gamma \approx 0.4794$. Using this value gives the interval $0.3669 \leq p \leq 0.3689$; and again $p = 94/256$ is the only number of the form $r/256$ ($r \in \mathbb{Z}$) that falls within this interval.

5.5 Known Plaintext Attack

It is possible to generalize the results of Section 5.3 to a known plaintext attack. In practice, it is easier for an attacker to use (or guess) known plaintext than to use a chosen

plaintext. Obviously if the known plaintext starts with 010 or 101 then we can immediately apply our previous results.

By generalizing Theorem 5.3 to cover other possible sequences we can bound the number of symbols needed for a known plaintext attack. Theorem 5.4 shows that many short sequences can be successfully used to determine p, and gives a bound for each possible sequence. If a particular sequence is known then tighter bounds for that particular sequence can sometimes be found.

THEOREM 5.4: To determine a w-bit probability:

1. The sequences 1^n and 0^n require $n = p2^{w+1}$ symbols.

2. Many sequences of the form 1^m0^{n-m} or 0^m1^{n-m} with $m \geq 1$ and $n \geq m+1$ require $n = w + m + 1 + O(\lg m)$ symbols.

3. Sequences beginning with 1^m01^l or 0^m10^l with $m \geq 1$ and $l \geq 1$ require $n = m + 1 + O\left(\frac{w + 1 - \lg(m+1)}{-\lg(1-p)}\right)$ symbols.

4. Sequences beginning with 1^m0^l1, $m \geq 1$, $l \geq 2$ require $n = l + 1 + O(2^w p^l)$ provided $p \leq \frac{1}{2}$. The same bound holds for sequences beginning with 0^m1^l0 when $p \geq \frac{1}{2}$.

PROOF: We prove the theorem for the sequences starting with 1. The proofs are symmetric for the sequences starting with 0.

1. The sequence $S = 1^n$ requires $n = p2^{w+1}$.

   Since $l_n(1^n) = p^n$, the derivative is easily seen to be np^{n-1}. It follows that

   $$\left(\frac{1}{2}\right)^w = \frac{2p^n}{np^{n-1}},$$

   which readily reduces to $n = p2^{w+1}$. Further, $\left|\frac{\mathrm{d}\delta_n}{\mathrm{d}p}\right| = n(1-p)^{n-1}$ which is much less than np^{n-1} except when $p \to 0$.

2. Sequences of the form $S = 1^m0^{n-m}$ with $m \geq 1$ and $n \geq m+1$. Using Equation 5.4,

   $$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^{n-m}) &= (1-p)^m \sum_{j=m+1}^{n} p^{j-m-2}[jp - j + m + 1] \\ &= (1-p)^m \left[m\frac{(1-p^{n-m-1})}{1-p} + np^{n-m-1}\right].\end{aligned}$$

   Using Equation 5.3,

   $$\left|\frac{\mathrm{d}}{\mathrm{d}p}\delta_n(1^m0^{n-m})\right| = p^{n-m-1}(1-p)^{m-1}|n - m - np|.$$

Before we can use Equation 5.5, we must verify our assumption about the interval; that is, we require

$$\left|\frac{\mathrm{d}}{\mathrm{d}p}\delta_n(1^m0^{n-m})\right| \ll \frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^{n-m}).$$

Now,

$$\begin{aligned}
\frac{\frac{\mathrm{d}}{\mathrm{d}p}h_n(b^m a^{n-m})}{\left|\frac{\mathrm{d}}{\mathrm{d}p}\delta_n(b^m a^{n-m})\right|} &= \frac{(1-p)^m\left[\frac{m(1-p^{n-m-1})}{1-p}+np^{n-m-1}\right]}{p^{n-m-1}(1-p)^{m-1}|n-m-np|} \\
&= \frac{m(1-p^{n-m-1})+(1-p)np^{n-m-1}}{p^{n-m-1}|n-m-np|} \\
&= \frac{m}{p^{n-m-1}|n-m-np|}+\frac{n-m-np}{|n-m-np|} \\
&= \frac{m}{p^{n-m-1}|n-m-np|}\pm 1 \\
&\gg 1.
\end{aligned}$$

So $\left|\frac{\mathrm{d}}{\mathrm{d}p}\delta_n(1^m0^{n-m})\right| \ll \frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^{n-m})$ for all values except $p \to 1$.
Now using Equation 5.5,

$$\begin{aligned}
\frac{2\delta_n(1^m0^{n-m})}{\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^{n-m})} &= \frac{2p^{n-m}(1-p)^m}{(1-p)^m\left[\frac{m(1-p^{n-m-1})}{1-p}+np^{n-m-1}\right]} \\
&= \frac{2p^{n-m}(1-p)}{m-mp^{n-m-1}+np^{n-m-1}-np^{n-m}}.
\end{aligned}$$

In the interval $0 < p \le \frac{1}{2}$, this has a maximum when $p = \frac{1}{2}$, and has the value $m2^{n-m-1}+n$. Therefore,

$$\begin{aligned}
2^w &\ge m2^{n-m-1}+n \\
2^w &\ge m2^{n-m-1} \\
n &\le w+m+1-\lg m.
\end{aligned}$$

For $p > \frac{1}{2}$, the situation is much more complex, and no general bound is available.

3. If $S = 1^m01^l$ then $\delta_n(1^m01^l) = p(1-p)^{m+l}$ and $\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m01^l) = (m+1)(1-p)^m$. We require

$$\begin{aligned}
\left(\frac{1}{2}\right)^w &\le \frac{2\delta_n(1^m01^l)}{\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m01^l)} = \frac{p(1-p)^{m+l}}{(m+1)(1-p)^m} \\
&= \frac{2p(1-p)^l}{m+1} \\
&\le \frac{2(1-p)^l}{m+1}.
\end{aligned}$$

The larger l is, the more accurately Equation 5.5 is satisfied. This can be seen by comparing $\left|\frac{\mathrm{d}\delta_n(1^m01^l)}{\mathrm{d}p}\right| = (1-p)^{m+l-1}(1-p-mp-lp)$ to $\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m01^l)$.

Therefore, since $n = m+l+1$,

$$n \leq m+1+\frac{w+1-\lg(m+1)}{-\lg(1-p)}.$$

Clearly, as $p \to 0$ an infinite number of symbols are required. Otherwise $m+1+O\left(\frac{w+1-\lg(m+1)}{-\lg(1-p)}\right)$ symbols are required.

4. If $S = 1^m0^l1$ then

$$\delta_n(1^m0^l1) = p^l(1-p)^{m+1} \text{ and } \frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^l1) = [lp^{l-1}(1-p)+(1-p^l)m](1-p)^{m+1}.$$

We require

$$\left(\frac{1}{2}\right)^w \leq \frac{2\delta_n(1^m0^l1)}{\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^l1)} = \frac{2p^l(1-p)^2}{lp^{l-1}(1-p)+(1-p^l)m}.$$

Solving for m we get

$$m \leq \frac{2^{w+1}p^l - lp^{l-1}(1-p)}{(1-p^l)}.$$

Provided $p \leq \frac{1}{2}$ and since $n = m+l+1$,

$$n \leq l+1+O(2^w p^l).$$

The cases where

$$\left|\frac{\mathrm{d}\delta_n(1^m0^l1)}{\mathrm{d}p}\right| = p^{l-1}(1-p)^m|l-lp-mp-p|$$

is much less than $\frac{\mathrm{d}}{\mathrm{d}p}h_n(1^m0^l1)$ are precisely those where $p \leq \frac{1}{2}$. These sequences cannot be used when $p \to 1$.

■

5.6 Extensions

The problem we have attacked above is a very simplified version of what happens in a real adaptive compressor. In such a situation there may be many more parts of the system that will be unknown besides the single probability. For example: the initial state of the arithmetic coder—the upper and lower bound—may be unknown. In terms of the analysis that we have done above this introduces two new variables into the polynomial characterizing the output. Thus with a single attacking text there is no way to uniquely solve the polynomial which now has three unknowns. However, if it is assumed that the encoder can be restarted to the same initial state three times, and each time attacked with

a different input sequence, then the three different polynomials produced can be solved to obtain all the unknowns in the system. We have not done a detailed analysis of this situation but if the m unknowns are stored to a precision of w bits then m sequences of approximately w symbols each, for a total of $O(mw)$ symbols, seems to be enough to crack the system.

Many extensions can be modelled in this way by extending the polynomial to more unknowns. For example an alphabet of k symbols has $k-1$ unknown probabilities. The model may have many states each with its own set of $k-1$ probabilities, and so on.

It is unclear how reasonable it is to assume that the encoder can be repeatedly reset and retried to get the different independent polynomials. In situations where a communications link is unreliable and error correction and retry are not possible, then it is necessary to periodically reset the state of the system and re-establish contact. This would be the case if the encrypted message were being broadcast to a number of recipients some of who were not receiving at one time but were later. Thus the need to synchronize makes the system vulnerable to attack.

However, in a point-to-point communication or in applications where text is being stored, it is reasonable to assume that resetting is never needed. The proposed attack then fails for any system with more than one unknown and it is an open question as to whether the extensions mentioned above are vulnerable to attack.

This discussion has also ignored the question of the complexity of finding solutions for n polynomials in n unknowns. In particular, the complexity of finding solutions as discrete approximations 'near' the exact solutions needs to be explored. It is not clear what the complexity of this calculation is; keeping in mind that the polynomials have a special form, so general results [Mat70, MR75, MA78, FY80, Kar72] about the complexity of solving polynomials may not apply here.

Another interesting and relevant situation arises if the probabilities adapt over time. If we restrict ourselves to the binary case and assume that the initial probability of a 0 is f/F, where f and F are both integers, and the initial probability of a 1 to be $g/F = (F-f)/F$, then the following recurrences are obtained:

If a 0 is encoded then

$$\begin{aligned} l' &\leftarrow l \\ \delta' &\leftarrow f\delta/F \\ h' &\leftarrow h - g\delta/F \\ f' &\leftarrow f+1 \\ F' &\leftarrow F+1. \end{aligned}$$

If a 1 is encoded then

$$\begin{aligned} l' &\leftarrow l + f\delta/F \\ \delta' &\leftarrow g\delta/F \\ h' &\leftarrow h \\ f' &\leftarrow f \\ F' &\leftarrow F + 1. \end{aligned}$$

Clearly at each stage we have $g + f = F$ and $0 \leq f/F \leq 1$. Using the same notation as before, the following characteristic equations can be obtained:

$$\begin{aligned} \delta_n &= \prod_{i=1}^{n} \left[a_i \frac{f + A_{i-1}}{F + i - 1} + b_i \frac{g + B_{i-1}}{F + i - 1} \right] \\ &= \frac{(F-1)!}{(F+n-1)!} \prod_{i=1}^{n} [a_i(f + A_{i-1}) + (1 - a_i)(F + i - 1 - f - A_{i-1})] \\ l_n &= \sum_{i=1}^{n} (1 - a_i) \frac{f + A_{i-1}}{F + i - 1} \delta_{i-1} \\ &= \sum_{i=1}^{n} (1 - a_i) \frac{f + A_{i-1}(F-1)!}{(F+i-1)!} \prod_{j=1}^{i-1} [a_j(f + A_{j-1} + (1 - a_j)(F + j - 1 - f - A_{j-1})] \\ h_n &= 1 - \sum_{i=1}^{n} a_i \frac{g + B_{i-1}}{F + i - 1} \delta_{i-1} \\ &= 1 - \sum_{i=1}^{n} a_i \frac{(g + B_{i-1})(F-1)!}{(F+1-1)!} \prod_{j=1}^{i-1} [a_j(f + A_{j-1} + (1 - a_j)(F + j - 1 - f - A_{j-1})]. \end{aligned}$$

While these expressions are obviously much more complex than those for the static case, it is true that once a particular sequence is selected they can be expanded to give polynomials in f which can then be solved.

5.7 The Subset Sum Problem

Sections 5.1–6 dealt with the difficulty of determining the state of an unknown arithmetic coder. Standard arithmetic coding was considered and no attempt was made to enhance the security of the system by including a key. That is, we took as our key part of the state of the arithmetic coder. We now discuss how a key mechanism can be incorporated into arithmetic coding and show that several schemes have a close affinity to the subset sum problem.

The ***subset sum problem*** is a special case of the ***knapsack problem*** and in the cryptology literature is often referred to as the knapsack problem. The subset sum problem is hard, its decision problem was shown to be **NP**-complete by Karp [Kar72]. It will be

shown that the related search problem of actually finding a solution, *even when a solution is known to exist*, is at least as hard as any **NP**-complete problem.

DEFINITION 5.1: Let $K = \{\kappa_1, \kappa_2, \ldots, \kappa_k\}$ be a finite set and t be a positive integer called the ***target***. Associated with each $\kappa_i \in K$ is a positive integer $s(\kappa_i)$ called its ***size***. The ***subset sum decision problem*** asks the question: is there any subset C of K such that the sum of the sizes of the elements in C is precisely t; that is, does there exists a C such that

$$C \subseteq K \text{ and } \sum_{j \in C} s(\kappa_j) = t?$$

The problem can also be stated as a $\{0,1\}$-integer programming problem. Does there exist a solution to

$$\sum_{i=1}^{k} s(\kappa_i) x_i = t \quad \text{where } x_i = 0 \text{ or } x_i = 1?$$

In addition to the decision problem, there are the related search problems of finding a particular solution and of finding all solutions.

As defined, the sizes can only be positive integers, but the problem can be generalized to allow for arbitrary integers. Any element with size zero does not affect the decision problem, but if a solution does exist then the presence of size zero elements implies the existence of additional solutions. Allowing negative sizes cannot result in a simpler problem since an algorithm that could solve this problem over all the integers could definitely solve the problem over the positive integers.

THEOREM 5.5: The subset sum problem is **NP**-complete [Kar72].

THEOREM 5.6: Finding a solution to subset sum problems, even when a solution is known to exist, cannot be done in polynomial-time unless **P=NP**.

PROOF: The existence of a polynomial-time algorithm for this problem is postulated; such an algorithm could be used to solve the subset sum decision problem in polynomial time. Thus the postulated algorithm cannot exist unless **P=NP**.

In detail, let $\mathcal{A}$ be a polynomial-time algorithm that computes a solution to a subset sum problem known to have at least one solution. Since $\mathcal{A}$ is a polynomial-time algorithm, there must exist a polynomial upper bound p in the cardinality of the set.

Given an arbitrary subset sum problem, S, run $\mathcal{A}$ on S. If $\mathcal{A}$ halts within the time p, check the output. Checking takes polynomial-time since the subset sum problem is in **NP**. If the output is not a solution then S does not have a solution. Alternatively, if $\mathcal{A}$ fails to halt within the time p then S does not have a solution. In either case we have decided in polynomial-time whether or not S has a solution.

Since the subset sum decision problem is **NP**-complete it follows that if $\mathcal{A}$ exists then **P=NP**.

■

The subset sum problem has previously been suggested for use in public-key cryptosystems [Hel79, MH78, Sha78]. These first proposals were broken because of special structure present in the subset sums used [Sha79a, Sha82]. These results were subsequently summarized in [Pat87, Pfl89]. Since the original system was broken many variants have been proposed. Many of these have also been analyzed and broken. Overviews of these systems and their cryptanalysis can be found in [BO88, BO91, Bri88, BS83, Des88]. The Chor-Rivest knapsack [CR85] is currently the most secure knapsack system, but is computationally expensive and is not immune from attack [SH95].

Theorem 5.3 is a worst case bound. It proves there are many subset sum problems which are hard to solve. Such a result is *not* sufficient grounds on which to rest security claims. Security requires nearly every case to be hard. However, complexity theory has not yet evolved far enough to obtain these stronger results. The best alternative is to show a system does not produce cases known to be solvable in polynomial time. We emphasize again the absence of such cases does not guarantee security.

Two classes of subset sums having polynomial-time algorithms are now examined: superincreasing subset sums (solvable with a simple greedy algorithm in linear time) and low-density subset sums (solvable with the short-vector algorithm in polynomial-time).

5.7.1 Superincreasing Subset Sums

In a ***superincreasing subset sum*** $s(\kappa_1), \ldots, s(\kappa_k)$, each term $s(\kappa_i)$ is bigger than the sum of all the preceding terms; that is,

$$s(\kappa_i) > \sum_{j=1}^{i-1} s(\kappa_j).$$

For example $1, 2, 4, 8, 16$ is a superincreasing subset sum, as is $1, 3, 9, 30, 102$ whereas $1, 3, 9, 30, 40$ is not. Superincreasing subset sums can be solved by a simple greedy algorithm (Figure 5.4) in linear time. The algorithm starts with the largest number in the sum. If this number is smaller than the current target t then it is in the sum, otherwise it is not. Reduce the target by the amount of this element if it is in the sum, otherwise don't change the target. Iterate on the next smallest element. If the target is reduced to zero on any iteration then the subset sum has a solution.

Superincreasing knapsacks were used as the basis of the first knapsack cryptosystems, with the superincreasing structure hidden by the transformation $s'(\kappa_i) = s(\kappa_i)n \bmod m$ for integers n and m, where $m > \max_i\{s(\kappa_i)\}$ and $\gcd(n, s(\kappa_i)) = 1$ for all i. However, this transformation proved insufficient to hide the superincreasing structure and the system was broken [Sha79a, Sha82].

```
superincreasing-solve (s, k, t)

vector s;          [k-dimensional vector of sizes]
int k;             [number of elements in the vector]
int t;             [target for sum]

begin
        for (i ← k; i > 0; i − −) begin
                if (s_i < t) begin
                        t ← t − s_i;
                        output s_i is in the knapsack;
                end;
        end;
        if (t = 0)
                output solution complete;
        else
                output no solution;
        end;
end;
```

Figure 5.4: Pseudocode for a greedy algorithm to solve superincreasing subset sum problems.

5.7.2 Low-Density Subset Sums

DEFINITION 5.2: The ***density*** D of a subset sum with sizes $s(\kappa_1), \ldots, s(\kappa_k)$ is given by

$$D = \frac{k}{\lg(\max_i\{s(\kappa_i)\})}. \tag{5.7}$$

If the density is low enough ($D < 0.645$ or $D < (2-\epsilon)[\lg(4/3)]^{-1}/k$ for any fixed $\epsilon > 0$) then the ***short-vector algorithm*** [LO85] will almost surely find a solution.

The short-vector algorithm relies heavily on being able to find the shortest vector in a $(k+1)$-dimensional lattice. No algorithm exists which has been proven to do this in polynomial time. Such a vector is typically found by first finding a reduced basis for the lattice. The L^3-algorithm [LLL82] is a popular choice for finding the reduced basis and has complexity $O(k^6[\log(\max_i\{s(\kappa_i)\})]^3)$. When more advanced techniques are used for the arithmetic operations, the complexity of finding a reduced basis with the L^3-algorithm can be reduced to $O(k[\log(\max_i\{s(\kappa_i)\})]^3)$.

5.8 A Simple Way of Using Key Bits

There are several ways in which a key can be added to the coding process with only a small loss of compression. The following is a simple method, we call ***interval narrowing***,

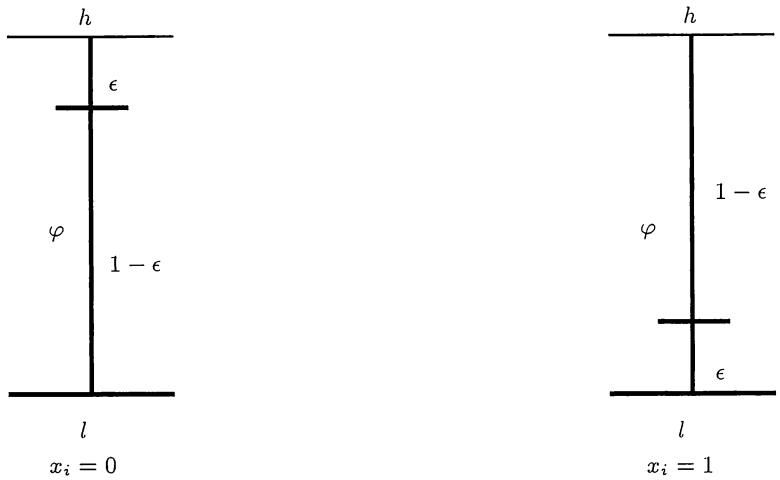


Figure 5.5: A simple model for using the key bits which results in polynomials having only linear terms in the key bits.

of making the result of arithmetic coding dependent on k key bits $x_1, \ldots, x_k$. Although insecure, this simplistic model provides insight into more complicated models. Let n denote the length of a message chosen from the alphabet $\{0, 1\}$.

Prior to encoding each symbol the current interval will be narrowed. The narrowing will be accomplished by either increasing the value of the lower bound l or decreasing the value of the upper bound h by an amount $\epsilon\delta$, where $0 < \epsilon < 1$. To minimize compression loss ϵ should be small. Which of these operations is applied depends on a key bit, as illustrated in Figure 5.5. After narrowing the current interval the actual symbol will be encoded according to the compression model. This narrowing can also be considered as a two step encoding process, where we first encode a cryptosymbol φ using a key model and then the actual message symbol using the compression model. A similar technique can be used to add controlled redundancy to the output of an arithmetic coder to facilitate error detection [BCIRW97].

The following recursive relationships for encoding with this model are obtained (expressions for the upper bound h are not given, as the results for the upper bound are symmetric with those for the lower bound):

For encoding 'φa':

$$\begin{aligned} \delta' &= (1-\epsilon)p\delta \\ l' &= l + x_i\epsilon\delta \end{aligned}$$

For encoding 'φb':

$$\begin{aligned} \delta' &= (1-\epsilon)(1-p)\delta \\ l' &= l + x_i\epsilon\delta + p(1-\epsilon)\delta \end{aligned}$$

(By letting $\epsilon \to 0$, these recurrences degenerate to those for keyless arithmetic coding.)

For the purposes of the analysis we will assume that p is a constant. If p is adaptive, the results will not be greatly affected, since by assumption the attacker is aware of the value of p even if it is changing. An important aspect of this cryptosystem is that the width of the current interval is independent of the key bits. From an analytic point of view this is desirable because the characteristic polynomial for l contains only linear terms in the key bits:

$$\begin{aligned} \delta_n &= (1-\epsilon)^n p^{A_n} (1-p)^{n-A_n} \\ l_n &= \sum_{i=1}^{n} [x_i \epsilon \delta_{i-1} + (1-a_i) p (1-\epsilon) \delta_{i-1}]. \end{aligned}$$

In any implementation of this model, ϵ and p will be approximated by rational numbers. The equation for l_n can be normalized to integers, since if $\epsilon = e/E$ and $p = f/F$ where $e, E, f, F \in \mathbb{Z}^+$ then

$$E^n F^n l_n = \sum_{i=1}^{n} (EF)^{n-i} (E-e)^{i-1} f^{A_{i-1}} (F-f)^{i-1-A_{i-1}} [Fex_i + (1-a_i) f(E-e)].$$

Since each term in the sum is an integer, it follows that the sum itself is an integer, and therefore $E^n F^n l_n$ is an integer. By expanding the sum on the right, the coefficients α_i for the key bits x_i, $0 \le i \le k$ can be determined. Any constant terms on the right can be moved to the left hand side and we can write:

$$L_n = \sum_{i=1}^{n} \alpha_i x_i,$$

where $L_n = E^n F^n l_n - C$ for some integer constant C and integer coefficients α_i. This is precisely the subset sum problem discussed in Section 5.7.

Given a finite key $x_1, x_2, \ldots, x_k$ and a message of more than k symbols, it is necessary to reuse the key information, otherwise the attacks presented in Sections 5.1–6 can be applied. That is, if the key bits are not reused then it is possible to break into the transmission after the key bits have been exhausted.

Recall that the density of a subset sum is defined by Equation 5.7. For interval narrowing, the maximum size will be $O(E^n F^n)$, and the density will be given by $D \approx k/n \lg(EF)$. Since the length of the key is fixed, this density can be made arbitrarily small by sending a long enough message (large n). It follows that the short-vector algorithm [LO85] can be used to determine the key bits.

Interval narrowing also suffers from chosen-plaintext attacks of the form 1^n or 0^n. In keyless arithmetic coding l_n does not change when the sequence 1^n is used, but l_n changes in this system whenever $x_i = 1$, and this extra information can be used to determine the key. In an extreme case, consider the result when $p = \frac{1}{2}$, $n = 5$, and $\epsilon = \frac{1}{2}$ with the message 11111. Then

$$2^9 l_5 = 2^8 + 2^6 + 2^4 + 2^2 + 1 + 2^9 x_1 + 2^7 x_2 + 2^5 x_3 + 2^3 x_4 + 2 x_5$$

and it is possible to directly find the key bits from the odd numbered bits in $2^9 l_5$. Note: the coefficients here form a superincreasing sequence and the density is $0.\dot{5}$.

More generally if the sequence 1^n is used, then

$$E^n F^n l_n = \sum_{i=1}^{n} E^{n-i} F^{n-i} (E-e)^{i-1} (F-f)^{i-1} F e x_i.$$

Ignoring the value of the key bits and considering the ratio of two consecutive terms of this sum we find

$$\frac{(EF)^{n-i}(E-e)^{i-1}(F-f)^{i-1}Fe}{E^{n-i+1}F^{n-i+1}(E-e)^{i}(F-f)^{i}Fex_i} = \frac{1}{EF(E-e)(F-f)}.$$

Since $E > e \geq 1$ and $F > f \geq 1$ we have $EF(E-e)(F-f) \geq 4$. This is a sufficient condition for the coefficients of the sum to form a superincreasing sequence.

5.9 Generalizations and Discussion

It is easy to construct more complicated arithmetic coders in which the width of the coding interval as well as the upper and lower bounds depend on the key bits (for example, by narrowing from both ends when $x_i = 1$, and doing nothing when $x_i = 0$). Characteristic polynomials can also be produced for these arithmetic coders. They are more general in the sense that each term of the polynomial can contain arbitrary products of the key bits $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_k^{\alpha_k} \in \{0,1\}$, rather than just linear terms. Analysis in these situations is considerably more complex.

It is even possible to construct systems which depend on a key but which do not cause loss of compression. For instance, the key can be used to decide which of two permutations will be used for encoding the next symbol, as shown in Figure 5.6. This system results in the characteristic polynomials:

$$\begin{aligned} \delta_n &= p^{A_n}(1-p)^{n-A_n} \\ l_n &= \sum_{i=1}^{n} [a_i p - a_i x_i - x_i p + x_i] \delta_{i-1}. \end{aligned}$$

Unfortunately this system has the same density problems as the interval narrowing approach.

Alternatively, the output of several arithmetic coders could be mingled. Such a coder might be a natural choice for multidimensional data. The key could be used to decide which arithmetic coder to use next.

These generalizations may lead to problems which are harder than the subset sum problem. The products can take on a wide variety of forms—although, not every polynomial is possible. Which polynomials can be produced depends on the way that the key bits are used.

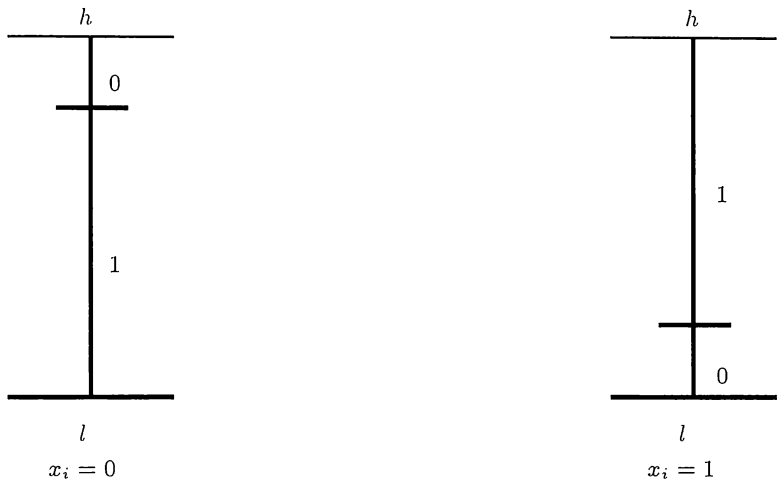


Figure 5.6: A permutation model for keyed arithmetic coding resulting in no loss of compression.

Several people have looked at the solvability of polynomials. In a seminal paper, Matijasevič [Mat70] showed that solvability of Diophantine equations in several variables is in general undecidable, thus answering Hilbert's tenth problem [Hil00]. Later it was shown [MR75] that Diophantine equations with only thirteen unknowns are undecidable. Further, solving one quadratic equation in two variables over $\mathbb{N}$ is **NP**-complete [MA78]. Other solvability results include [FY80, Kar72]. Linear equations can be solved in deterministic polynomial time. These results are worst case complexities and only recently has there been much interest in the average complexity of this type of problem.

However, none of these results apply here because a solution is known to exist (namely, the result of the encoding) and the indeterminates only have two values, 0 or 1, whereas the more general problems are over infinite sets. Further, in the case where there is more than one solution, the attacker must find all (or at least the majority) of the solutions to determine which solutions lead to likely plaintexts.

In the analysis we were very lenient on the attacker, and assumed the only unknown was the key. But in practice, it is extremely unlikely that an attacker would know exactly the state of an adaptive compression model.

Despite the apparent increase in the complexity that is gained by allowing more general polynomials in the coding intervals, it is difficult to see how this extra complexity could translate into greater security.

The theorems in this chapter have shown that the static binary arithmetic coder can be broken with a chosen plaintext of length linear in the number of bits of state in the arithmetic coder. Many known plaintexts also suffice. Although detailed analysis of more complex situations has not been carried out, it seems reasonable to conjecture that if there are m unknowns each stored to w bits of precision, and the coder can be restarted, then $O(mw)$ symbols are sufficient to crack the system. The situation for a coder which cannot be restarted remains an open question.

Further, we have shown that several methods of adding an explicit key to arithmetic coding lead to the subset sum problem. Unfortunately, the cases produced fall into a class of sums solvable in polynomial time.

Collectively, these results strongly suggest that if compression is going to be used for encryption then the security must come from the model rather than the arithmetic coder. To this end, Chapters 6, 7, and 8 investigate the security of various types of model used with arithmetic coding and various data compression methods which do not rely on arithmetic coding.

Chapter 6

Huffman Coding and Splaying

6.1 Introduction

This chapter deals with two compression algorithms from a cryptologic standpoint: the semisplay prefix tree compression algorithm of Jones [Jon88], and Huffman coding [Huf52]. The study of these algorithms is pertinent as such systems are now being used in the commercial sector. Indeed, the semisplay system discussed is used by Lotus Development Corporation in their word processor Ami Pro for encryption. Huffman coding has been used as a security device in a CD-ROM text retrieval system [KBD89]. The security of Huffman coding appears to have been first discussed by Rubin [Rub79b].

In a Huffman code [Huf52], each symbol is represented by a binary string of variable length. To obtain compression the most frequent symbols are given the shortest codes while rare symbols are given longer codes. To ensure decoding is unambiguous no code can be a prefix for another code. A Huffman code is easily visualized as a prefix tree with the leaves denoting the symbols of the alphabet. Figure 6.1 depicts such a tree for a seven symbol alphabet. Throughout this chapter, a left branch corresponds to a 0 output and a right branch corresponds to a 1 output, and codes are formed top down from the root.

A simple greedy algorithm exists to construct an optimal Huffman tree once the symbol frequencies are known (see Section 3.3.1). Unfortunately, obtaining the frequencies requires an additional pass over the message to be compressed, hindering on-line operation. Several authors [Gal78, Knu85, Vit86] have given adaptive Huffman algorithms that are near optimal and do not require the extra pass. Jones [Jon88] suggested using splay trees [TS85] as an alternative to adaptive Huffman codes.

Neither Huffman trees nor splay trees give optimal results in the information-theoretic sense. Except in contrived cases, they waste code space because they always output at least one bit per symbol encoded.

In Section 6.2 a brief overview of splaying and semisplaying is given, followed, in Section 6.3, by some comments on ciphertext only attacks and exhaustive searches. In

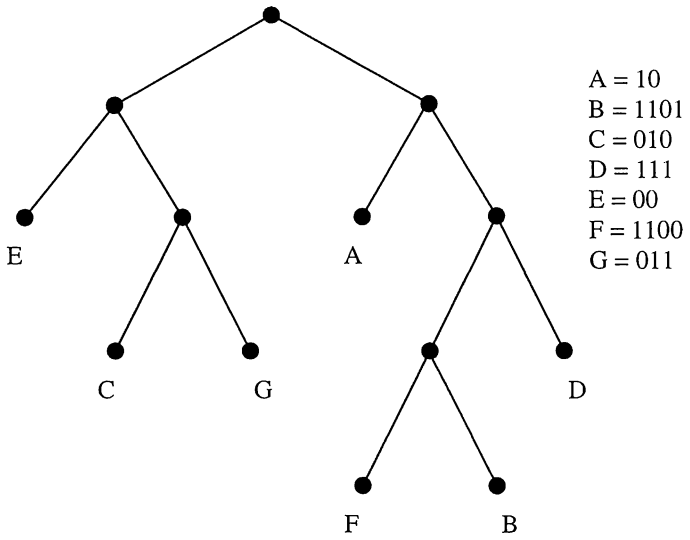


Figure 6.1: Example of a Huffman code for a seven symbol alphabet.

Section 6.4 a general strategy for attacking tree-based cryptosystems is presented. Section 6.5 contains a known plaintext attack for the splay compressor and Section 6.6 a chosen plaintext attack. In Section 6.7 the chosen plaintext attack is applied to Huffman coding. Section 6.8 discusses the applicability of the attack to other tree-based compression systems.

6.2 Splay Trees

Splay trees [TS85] were originally developed as a form of self-balancing binary search tree but have been found to have other applications, such as the implementation of priority queues and data compression [Jon88]. Whenever a node is accessed in a splay tree, the entire tree is rotated so that the accessed node becomes the root while maintaining the lexicographic ordering of the tree.[1] Sleator and Tarjan [TS85] proved that if nodes are accessed according to a static probability distribution, the amortized splay tree access time is within a constant factor of the optimal static tree for the distribution.

A variant of splaying called ***semisplaying*** is applicable to trees having data stored in the leaves. In semisplaying the accessed node is not moved to the root but rather the path to the accessed node is shortened by a factor of two. Semisplaying has the same access bounds as splaying to within a constant factor [TS85].

Since semisplay trees adapt to an arbitrary input distribution they are an ideal alternative to adaptive Huffman coding. Splay tree systems achieve compression in much the same way as Huffman codes by giving frequently occurring symbols shorter codes.

[1] Splaying must be about an internal node of the tree since it is impossible to have a leaf node at the root. Thus splaying is only applicable to trees with information stored on internal nodes.

Whenever a symbol occurs it is moved closer to the root, so if it occurs often it will always be near the root and thus receive short codes.

Since it is not necessary to preserve the lexicographic ordering of the nodes in a prefix code, the semisplaying operation can be implemented using semirotations. Figure 6.2 illustrates a semirotation operation. Semirotations can be implemented efficiently using pointers since only two links need to be modified per semirotation.

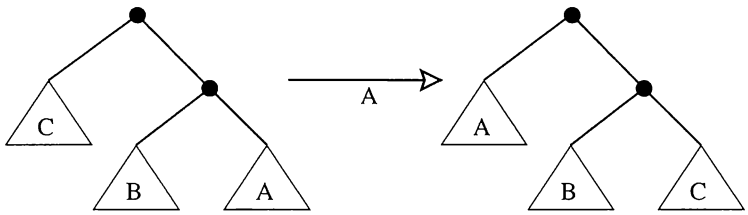


Figure 6.2: A semirotation about A.

While semisplay compression methods do not rival the compression performance of the better adaptive methods like PPM [CW84b] and Ziv-Lempel coding [ZL77, ZL78], they are very fast and require only modest amounts of memory. A semisplay system using only the symbols of the alphabet (order-0 semisplaying), can reduce book1 (750 kilobytes) to 500 kilobytes, compared with 216 kilobytes for PPMD [Tea95] (a modern adaptive compression system).

The following theorems are used in the cryptanalysis.

THEOREM 6.1: Any symbol of the alphabet can be made a child of the root by semisplaying about it $\lceil \lg n \rceil$ times in succession, where n is the number of leaves (or equivalently, the size of the alphabet).

PROOF: Since the initial depth of any symbol cannot exceed $n - 1$, the result follows trivially from the fact that each time a symbol is sent it is moved approximately half-way towards the root.

■

THEOREM 6.2: Any node not on the path to the node being semisplayed has its depth in the tree increased by at most one during the semisplaying operation.

PROOF: Consider an arbitrary subtree not on the path to the splay node. This subtree can be involved in at most one semirotation and thus its depth can increase by at most one. Therefore, any node in the subtree has its depth increased by at most one. Since every node not on the path is in a subtree not on the path, it follows that every node not on the path has its depth increased by at most one.

■

THEOREM 6.3: A binary prefix tree with m internal nodes has $2m + 1$ nodes in total.

PROOF: Since each internal node has two children, there are $2m$ children. Including the root node, this gives $2m + 1$ nodes in total.

■

The tree used by the semisplayer can be initialized to any one of a large number of possibilities. The initial state of the tree can thus be considered as a key. One way to implement this would be to encode a small segment of text, considered as the key, prior to sending the actual message.

The problem for the cryptanalyst, then, is to reconstruct the splay tree model given ciphertext and plaintext. Once the model is reconstructed, the attacker's model will remain synchronized with the legitimate recipient and all subsequent text will be decoded. The (weak) assumption that the cryptanalyst is aware in advance of how many symbols are in the plaintext alphabet is also made.

6.3 Exhaustive Search

The number of distinct prefix trees with n-leaves (alphabet size n) is given by

$$\frac{(2n-2)!}{(n-1)!}$$

Figure 6.3 tabulates this number for various values of n. Although a small number of these trees cannot arise by any amount of splaying (notably completely balanced cases), it quickly becomes infeasible to carry out an exhaustive search. Even a single symbol splay tree compression system will have 256 leaves (or 257 leaves if an end of message symbol is included). The number of trees in this case easily exceeds the number of particles in the known universe. When $n > 14$, the number of initial states exceeds the 56-bit key of the Data Encryption Standard [NBS77].

The output of the single symbol semisplaying data compression algorithm fails many statistical tests of randomness indicating redundancy is still present (Chapter 4). However, it is unclear whether the redundancy remaining is sufficient for a ciphertext only attack.

6.4 Backtracking Attack for Semisplaying

The security offered by the semisplaying data compression algorithm relies heavily on the assumption that the division between the codewords or the ***blocking*** of the output is unknown. This assumption is reasonable since the operating system and underlying hardware will almost certainly buffer the output and buffering could definitely be incorporated into the algorithm if necessary.

If an attacker could determine the blocking of the output, a known plaintext in which every symbol (except possibly one) occurs is sufficient to uniquely determine the tree. This is because each block gives the path in the tree to the corresponding plaintext symbol.

n	trees	n	trees
2	2	14	6.48×10^{16}
3	12	15	3.50×10^{18}
4	120	16	2.03×10^{20}
5	1680	26	1.96×10^{39}
6	30240	27	2.00×10^{41}
7	665280	35	8.40×10^{57}
8	17297280	50	1.55×10^{91}
9	518918400	128	4.36×10^{288}
10	1.76×10^{10}	256	3.97×10^{656}
11	6.70×10^{11}	257	4.05×10^{660}
12	2.82×10^{13}	1024	7.54×10^{3251}
13	1.30×10^{15}		

Figure 6.3: The number of prefix trees compared to the number of symbols in the alphabet.

For example, consider the message BADBED drawn from the five-symbol alphabet {A, B, C, D, E}. Let the output be 10|10|111|111|1101|001 where the vertical bar indicates the blocking. It is shown how to reconstruct the semisplay tree from this information (Figure 6.4). At each step a triangle denotes a branch of the tree not yet resolved by the cryptanalyst.

The first block, 10, is the path to B in the tree, thus Figure 6.4(a) was the situation prior to the transmission of the first B and Figure 6.4(b) is the result of semisplaying about B. The next 10 in the output corresponds to the plaintext A, and this new information can be added to the tree (Figure 6.4(c)), followed by a semisplay about A (Figure 6.4(d)).[2] Output 111 corresponds to the first D, resulting in Figure 6.4(e), and is followed by a semisplay about D (Figure 6.4(f)). The next 111 is for B and is consistent with the tree so far; semisplaying about B gives Figure 6.4(g). Output 1101 corresponds to the plaintext E. Add E to the tree (Figure 6.4(h)) and semisplay about E to get Figure 6.4(i). Since the only symbol unaccounted for is C, the last triangle in Figure 6.4(i) must be C (Figure 6.4(j)). The last block of output, 001, is for the last D and Figure 6.4(k) is the final result.

The structure of the tree has been completely determined and by working backwards the initial tree can be determined. Thus any other messages encoded with the same initial tree (or equivalently, the same key) can also be decoded. Alternatively, any future transmission continuing on from where the known plaintext ends can be decoded.

The only problem to overcome in order to have a general attack is the determination of the blocking of the ciphertext.

In the absence of any blocking information simply guessing the divisions is a sensible

[2]Observe an effect of splaying: although the first two symbols are distinct they have resulted in the same output 10.

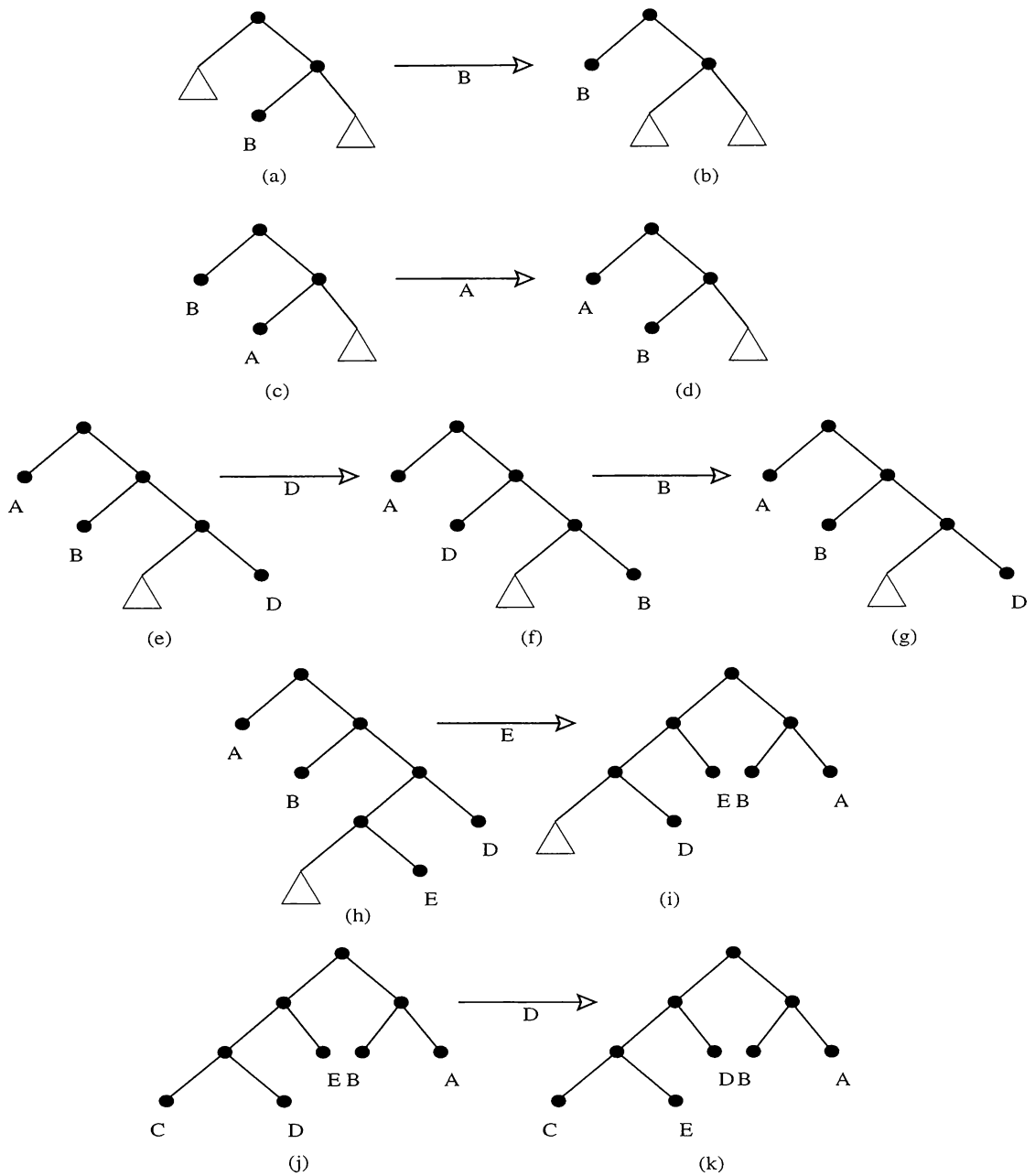


Figure 6.4: Known-plaintext attack when the blocking is known in advance. A $\triangle$ denotes a piece of the tree which must exist but whose structure and labels are not yet known to the cryptanalyst.

approach. The size of the alphabet constrains the length of blocks, for an n symbol alphabet, a block has length at most $n-1$. Further, the mean block size will be approximately $\lg n$.

The attack presented guesses divisions in an incremental manner starting with the code for the first plaintext symbol. If at any stage the partially constructed tree is found to be inconsistent with known information, the algorithm returns to an earlier symbol and makes a different guess. This approach would be infeasible if it were difficult to detect when an incorrect guess had been made. However, repetitions of symbols quickly cause inconsistencies when an incorrect guess is made and in practice the method is fast for many sequences. For some chosen sequences, the complexity is linear in the size of the alphabet, as proven in Section 6.6.

Consider a plaintext symbol α and the string $b \in \{0,1\}^{n-1}$ of the $n-1$ next ciphertext bits in relation to a partially determined tree. The following situations can arise.

- If α is already in the tree, then the path to α should be a prefix of the string b. If this is not the case then an earlier guess must have been incorrect and we have to back up to the previous symbol. Otherwise move on to the next symbol.
- If α is not in the tree, then create a leaf node for it. Consider the $n-1$ nonempty prefixes of b (including b itself). If a prefix corresponds to an internal node or a node for a symbol other than α, then that prefix cannot be the code for α. If none of the prefixes of b are possible codes for α, then back up to the previous symbol.
- Starting with the shortest, consider each prefix passing the previous test in turn. If using that prefix would result in a tree containing too many nodes, then it is rejected (this situation can only occur when an error has been made previously, since possibilities longer than the correct solution are not tried). Likewise, if the prefix results in a tree with insufficient space for the addition of remaining symbols, it is rejected (this only occurs when the tree is nearly full). Otherwise, the prefix may be the code for α, so insert α at the corresponding position and continue with the next symbol.

The above is easily embodied in a simple backtracking algorithm. It is possible to implement the algorithm so that the execution time is proportional to the number of insertions made in the tree.

6.5 Known Plaintext Attack

The security of semisplaying is examined from the encoder's point of view. This means the plaintext consists of a sequence of symbols from the alphabet and the ciphertext is a binary string.

To simulate a known plaintext attack, random plaintexts were considered. In practice the plaintexts might be of a specific form, like English prose, and thus may lead to easier attacks. Figure 6.5 shows some empirical results for this situation. It is observed that the mean number of insertions grows exponentially in the size of the alphabet. This appears to indicate that a known plaintext attack using this approach is infeasible.

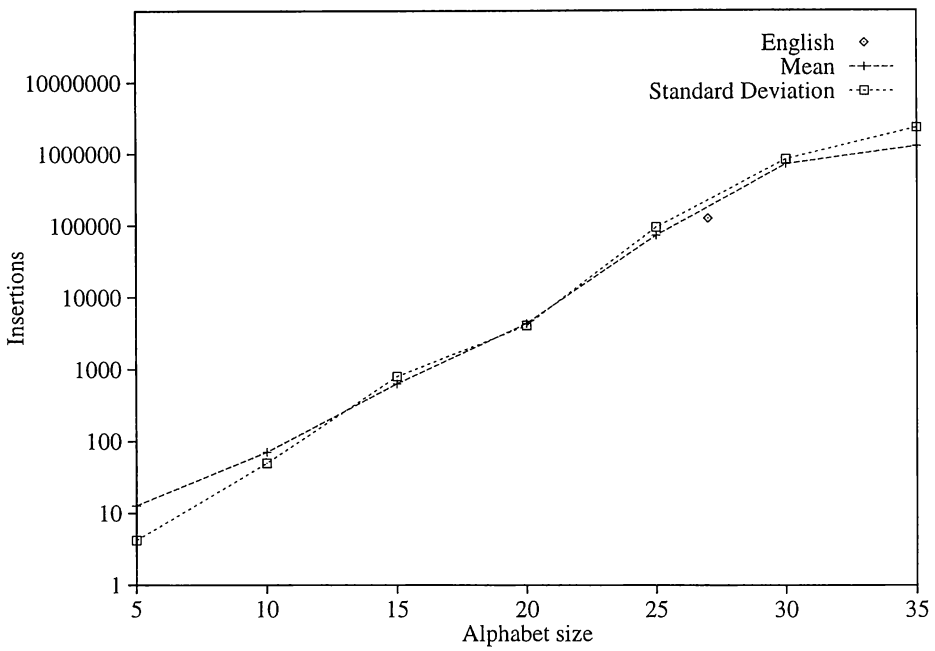


Figure 6.5: This graph shows that the mean time taken for the known-plaintext attack grows exponentially in the size of the alphabet. However, the standard deviation also grows exponentially and thus only a few attempts are required before an easy plaintext is found.

It was found that only two types of error were significant in this situation. Errors occurring because the path ended at an internal node or incorrect leaf accounted for 58% of all errors, and creating trees with too many nodes accounted for 42% of errors. The error of not reaching the correct node when the node's position was already determined was very rare. This is because in a larger alphabet the expected interval between identical symbols is quite large; therefore a given symbol will nearly always be inserted in the correct position prior to the next occurrence of the same symbol.

Despite the mean time being exponential, the attack is still feasible since the majority of random plaintexts require relatively few insertions. Indeed, the observed mode is $2n-2$, where n is the alphabet size. Except in fortuitous circumstances, $2n-2$ is the minimum number of insertions which can be expected since there are $2n-2$ unknown nodes in the tree at the outset. Figure 6.6 is the observed distribution for $n = 12$. In general, the probability of making an error when inserting a symbol is a complicated quantity, depending on the number of symbols in the alphabet, the entire state of the splay tree, and whether or not any previous errors have occurred.

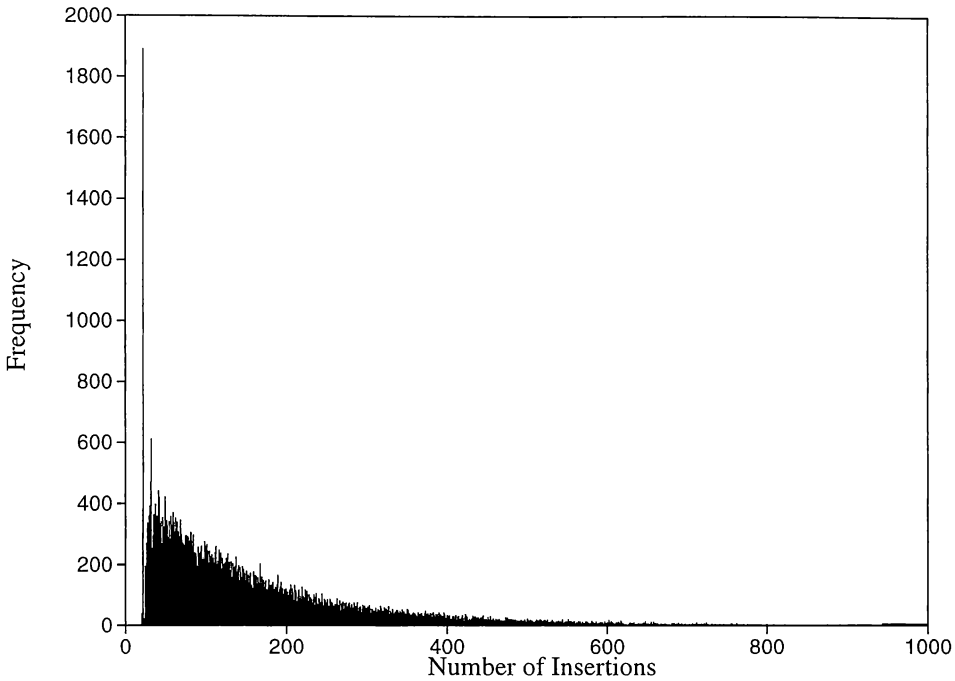


Figure 6.6: The distribution of number of insertions for a random plaintext attack with a twelve symbol alphabet.

Since it is obvious that the backtracking attack is likely to make a large number of incorrect insertions it is best to consider the consequences of an incorrect insertion; in particular how long will we expect to wait (in terms of symbols processed) before the incorrect insertion is detected and rectified.

THEOREM 6.4: Given an incorrect insertion has been made, there is at least a 50% chance that the incorrect insertion will be detected when the next plaintext symbol is considered.

PROOF: There are four ways in which an incorrect insertion can be detected:

1. A subsequent occurrence of the symbol fails to reach the purported position of the symbol in the tree.
2. An occurrence of another symbol whose position in the tree is already known fails to reach its known position.
3. An occurrence of a symbol whose position is not currently known ends up at a different previously inserted symbol.
4. The plaintext is exhausted before the ciphertext.

The probability of an incorrect insertion decreases as the number of correctly inserted symbols increases because there are less and less unresolved branches in the tree. In the

analysis, the contribution of the last condition is ignored since it only occurs near the end of the attack when incorrect insertions are unlikely.

Let us assume that c symbols have been correctly inserted into the tree. What are the consequences of inserting the next symbol incorrectly?

The scenario now is a partially resolved tree, a plaintext symbol, s, and some ciphertext bits which are not the true code bits for symbol s (because of the incorrect insertion the inferred ciphertext boundary will be incorrect). Consider the probability that the error will be detected when the symbol s is encoded. There are three situations which can arise according to the first three cases mentioned above: (1) s is a subsequent occurrence of the incorrectly inserted symbol, (2) s is one of the c correctly inserted symbols, or (3) s is a new symbol whose location in the tree is unknown. Since the plaintext is random, the probabilities of s being in each case are $1/n$, c/n, and $(n-c-1)/n$, respectively. (In fact the following analysis will hold if these probabilities are replaced with any three numbers which sum to 1, and thus there is no real requirement that the plaintext be random.)

Case (1) is relatively rare, at least for large alphabets. In order for the incorrect insertion of s to go undetected the next $\lceil d/2 \rceil$ ciphertext bits must match the incorrect path, where d is the depth where s was incorrectly inserted. The probability of this happening is about $2^{-d/2}$. On average d will be $O(\lg n)$, but in the worst case $d = 1$. Thus the probability that the error goes undetected is at most $1/(2n)$.

In case (2) the probability that the next ciphertext bits will correctly code for the next plaintext symbol is needed. Again in the worst case this plaintext symbol could be a child of the root and so there is at most a 50% chance that the ciphertext will correctly lead to the symbol. Consequently, the probability the error goes undetected is at most $c/(2n)$.

In case (3) the probability that the next ciphertext bits lead to an unresolved branch of the tree is needed. This is by far the most difficult case and events of this type can easily lead to further incorrect insertions. Case (3) is most relevant at the start of the attack when the positions of most symbols are unknown. However, the probability can be weakly bounded by 1/2 by noting that at some point on the path prescribed by the ciphertext there is a choice between a known symbol position (since at least one other symbol has been inserted) and an unresolved branch. Therefore the probability that an error goes undetected is at most $(n-c-1)/(2n)$.

Adding the contributions of all three cases gives

$$\frac{1}{2n} + \frac{c}{2n} + \frac{n-c-1}{2n} = \frac{1}{2}.$$

There is therefore at least a 50% chance of an incorrect insertion being detected when the next plaintext symbol is considered.

■

By the theorem, the probability of remaining undetected after considering b further symbols subsequent to the error is at most 2^{-b}, which is vanishingly small in b. It follows

that the expected number of symbols required to detect an error is bounded above by

$$E = \sum_{i=1}^{\infty} i2^{-i} = 2.$$

It follows that if every symbol were incorrectly inserted (an unlikely event), then on average $3n$ insertions will be required. This is consistent with the observed mode of $2n - 2$ insertions.

6.6 Chosen Plaintext Attack

For certain chosen sequences the attack can be proven to run in linear time. The following theorem establishes this for the sequence in which each symbol of the alphabet is repeated n times, where n is the size of the alphabet.

THEOREM 6.5: The sequence $\coprod_{i=1}^{n} a_i^n$ of length n^2 breaks the splay-tree algorithm with $O(n)$ insertions.

PROOF: By Theorem 6.1 the subsequence $a_i^{\lceil \lg n \rceil}$ will make symbol a_i a child of the root. The remaining $n - \lceil \lg n \rceil$ occurrences of symbol a_i serve to ensure that the boundary in the ciphertext of the next symbol can be detected unambiguously.

Assume that $j - 1$ symbols have been correctly inserted in the tree. The path to the next symbol, a_j, is some prefix of the next $n - 1$ ciphertext bits. The backtracking approach tries each of these in turn, starting with the shortest, until the correct path is found. It follows that the only possibilities tried in the attack correspond to nodes in the semisplay tree. Further, whenever a symbol is actually inserted into the tree, all the nodes on its path are created and once they are created, they are no longer eligible as insertion points. Finally, since there are $2n - 1$ nodes in the tree, and the root position is known from the start, it follows that at most $2n - 2$ insertions are required.

■

Shorter sequences do not in general result in linear time but many still form feasible attacks. Experiments indicate that the sequence $\coprod a_i^{\lceil \lg n \rceil}$ of length $n \lg n$ is a good compromise between length and execution time. The graph in Figure 6.7 compares the number of insertions required for this sequence, the one in the theorem, and for the sequence $\coprod a_i^2$. There is no loss in performance from using $\coprod a_i^{\lceil \lg n \rceil}$ instead of $\coprod a_i^n$ although for higher values of n the two curves should diverge. The $\coprod a_i^2$ sequence clearly takes considerably longer and would become impractical for large alphabets.

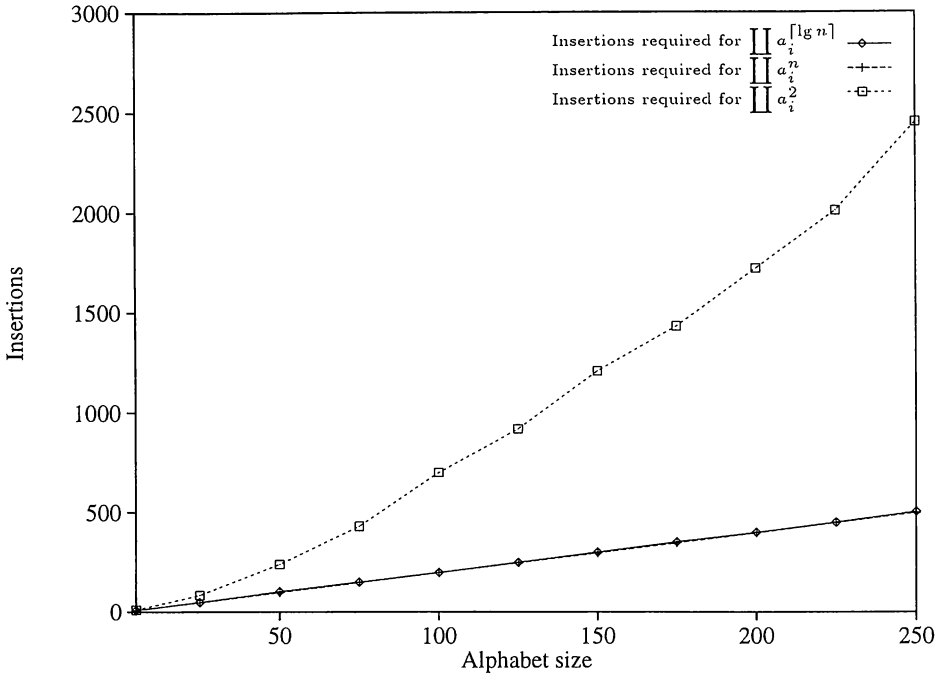


Figure 6.7: The number of insertions required for the test sequences.

6.7 Huffman Coding

A similar backtracking attack can be applied to a static Huffman code. Again the problem for the cryptanalyst is to reconstruct the state of an unknown tree. The attack is simpler since no adaptation of the tree occurs. This means that the verification of symbol placement is simpler. A chosen plaintext of the form $\coprod a_i^k$ for $k \geq 2$ is sufficient. For large k, the repeating patterns are easily detectable.

For example, the tree in Figure 6.1 is trivially determined using the plaintext $\coprod a_i^2$. Using this plaintext the ciphertext

$$10101101110101001011111100001100110001101 1$$

will be observed. Now pretend the tree is unknown and the task is to determine the code. Since the alphabet has size seven the maximum path length is six, thus we must look for repeating patterns with length at most six. Examination of the ciphertext reveals that 10 is the first repeat and nothing longer repeats, thus the code for A is 10. The next repeat is either just 1 or else 1101. However, assuming the code for B is 1 then it is discovered that no suitable repeat for C can be found, consequently the code for B must have been 1101. Continuing in this way the codes for the five remaining symbols can be identified.

This attack will only be feasible if the number of errors made and the consequences of making an error are found to be small. By an error we mean the incorrect assignment of a node in the Huffman tree to a given symbol. When such an error is detected (which is guaranteed to happen eventually, provided the plaintext contains at least one occurrence

of each possible symbol) the backtracking attack will try the next longest possibility, and eventually the correct code will be established. By analysing the backtracking attack for plaintexts of the form $\coprod_{i=1}^{n} a_i^k$ where k is a constant, the probability of an error occurring can be bounded. The probability of making an error is found to be small, and for k sufficiently large any error is immediately detected.

DEFINITION 6.1: Let $\phi(x) = |\{y : \gcd(x, y) = 1 \text{ and } y \in \mathbb{Z}^+ \text{ and } 1 \leq x < y\}|$ be ***Euler's totient function***; the number of integers between 1 and x relatively prime to x.

In the analysis the probability of being in error after sending the sequence a_i^k is considered.

The Huffman encoding of a_i^k results in the output c_i^k consisting of the Huffman code for symbol a_i repeated k times. Since there are n symbols in the alphabet, the longest possible code is $n-1$ bits. Let l denote the length of the code c_i; so, $l = |c_i| \leq n-1$. The cryptanalyst, who is (initially) unaware of where the boundaries in the ciphertext occur, must attempt to deduce the Huffman code for each symbol of the alphabet.

The probability of making an incorrect insertion during the backtracking cryptanalysis is approximately the probability of finding a sequence s^k which is some proper prefix of c_i^k. Further, $|s| < l$ since the backtracking method will not search beyond the correct answer. This probability is exact for the very first symbol inserted, but inexact thereafter, because the presence of known nodes in the Huffman tree can eliminate some prefixes of the required type. Therefore the probability of making an error can actually be smaller than but not exceed the upper bound obtained below.

We proceed by calculating the expected number of prefixes of c_i^k having the form s^k. Because the backtracking approach starts with the shortest such prefix and stops when the correct code is found, we need only consider repeated prefixes of length 1 through $l-1$. Let m denote the length of the repeated unit, $1 \leq m \leq l-1$; that is $km = |s^k|$.

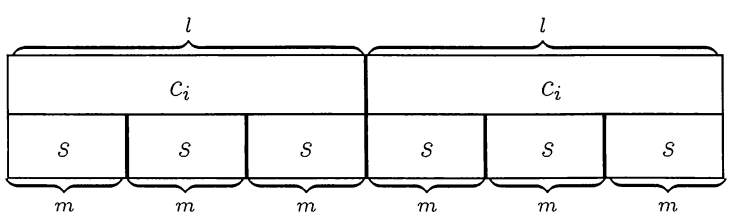


Figure 6.8: A case where m divides l.

Consider the case where $m|l$, as illustrated in Figure 6.8. In this situation the boundaries between consecutive c_i's coincide with boundaries between the s's. Thus only a single c_i need be considered because whatever is true for the first c_i will follow automatically in subsequent instances. By counting the number of pairs of bits that must match, the probability $2^{-(l-m)}$ is obtained for this case. The total contribution for these cases is

therefore

$$\sum_{m|l} 2^{-(l-m)}.$$

If $\nu(r)$ denotes the number of positive divisors of r, then this sum is bounded above by $(\nu(l)-1)2^{-l/2}$ by considering the largest non-trivial divisor of l and excluding the case $m = l$.

Slightly more general is the case where $\gcd(m,l) \neq 1$. This includes the situation just discussed as a special case, and other possibilities such as the one shown in Figure 6.9.

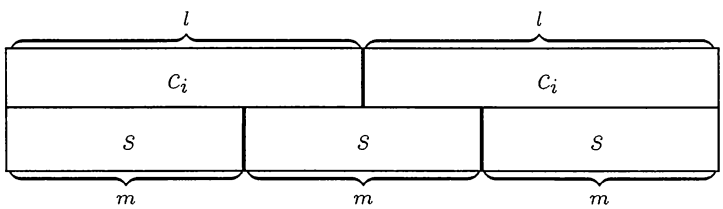


Figure 6.9: A case where $\gcd(m,l) \neq 1$.

In this case, an s-boundary will coincide with a c_i-boundary periodically, with the first coincidence occurring at bit position $\text{lcm}(m,l)$ (provided $kl \geq \text{lcm}(m,l)$ otherwise no such boundary will occur); where lcm denotes the least common multiple. Again we determine the number of pairs of bits that must match. Let $[b]$ denote the set of positive j for which bit b must equal bit j where $1 \leq b \leq m$. Two bits j and b will be aligned if there exist two positive integers q and r such that $j + ql = b + rm$. Thus,

$$\begin{aligned}
[b] &= \{j : j + ql = b + rm, 1 \leq j \leq l, q, r \in \mathbb{Z}^+\} \\
&= \{j : j + ql - rm = b, 1 \leq j \leq l, q, r \in \mathbb{Z}^+\} \\
&= \{j : j + d(ql' - rm') = b, 1 \leq j \leq l, q, r \in \mathbb{Z}^+\}
\end{aligned}$$

where $d = \gcd(l,m)$, $l' = l/d \in \mathbb{Z}^+$, and $m' = m/d \in \mathbb{Z}^+$. Let $t = ql' - rm'$ then

$$\begin{aligned}
[b] &= \{j : j + dt = b, 1 \leq j \leq l, q, r, t \in \mathbb{Z}^+\} \\
&= \{j : j = b (\text{mod}\, d), 1 \leq j \leq l, q, r, t \in \mathbb{Z}^+\}.
\end{aligned}$$

Therefore, there are precisely d distinct sets $[b]$ and each has cardinality $l' = l/d$. Since each set contains the trivial member $b \in [b]$, this leaves $l - d$ pairs of bits which must match. Hence the resulting probability is $2^{-(l-\gcd(l,m))}$. Note this is consistent with the cases where $m|l$. This situation will occur in $l - \phi(l)$ ways.

The remaining $\phi(n)$ cases are those where $\gcd(l,m) = 1$. In this situation all the bits of c_i must be the same, thus $l-1$ pairs must match, and this is consistent with $l - \gcd(l,m)$. (There is nothing in the previous case which fails when $\gcd(l,m) = 1$.) Thus in every case the probability is $2^{-(l-\gcd(l,m))}$.

Since the largest divisor of l (excluding l itself) is at most $l/2$ we can bound the expected number of suitable prefixes as

$$\begin{aligned}\sum_{m=1}^{l-1} 2^{-(l-\gcd(l,m))} &\leq \sum_{m=1}^{l-1} 2^{-(l-l/2)} \\ &= \sum_{m=1}^{l-1} 2^{-l/2} \\ &= (l-1)2^{-l/2}.\end{aligned}$$

This equation is dominated by the exponential term, and the $l-1$ factor quickly becomes irrelevant as l increases.

It is somewhat surprising that this result appears to be independent of k (the number of repeats); until the assumption that $kl \geq \text{lcm}(m,l)$ for all m and l is recalled. Since m can be as large as $l-1$ and l can be as large as $n-1$, the maximum value of $\text{lcm}(m,l)$ is $(n-1)(n-2)$. Therefore $k(n-1) \geq (n-1)(n-2)$ or $k \geq n-2$. Notice this is essentially the same result as that for the splay tree.

Another observation is that the probability of getting an error actually decreases when l, the length of the code word, increases!

It is not immediately obvious what happens after an error occurs. After all the total time required by the backtracking attack depends heavily on how quickly errors are detected. In fact, all such errors will be recognized as soon as a subsequent symbol is transmitted (provided $k \geq n-1$). To see this, suppose s is incorrectly used instead of c_i; then on decoding the next symbol we will in fact still be consuming bits from c_i rather than the correct code word. This immediate detection of errors is guaranteed provided $|s^k| \geq |c_i|$, which is always the case if $k \geq n-1$. These bits will be precisely the code s, and clearly they or any prefix of them cannot be the code for the next symbol, and thus the error is detected. Therefore, if $k \geq n-1$, at most $2n$ insertions will be required to determine the tree.

In some circumstances a particular ordering of the symbols in $\coprod a_i^k$ will lead to a faster attack. Assume now that some prior information is given about the probabilities of the symbols of the source. Then the attack is best made by sending the rarest symbol first. Such symbols are likely to have long codes, and therefore a smaller probability of causing an error, and the correct placement of such symbols determines a large number of nodes in the Huffman tree. This in turn makes the placement of subsequent symbols less prone to errors.

6.8 Conclusions

A known plaintext attack against the semisplay tree data compression algorithm [Jon88] has been presented. The attack has expected time complexity of $3n$ insertions. This

data compression system has been used in at least one commercial product as a security device. While the attack is, in general, exponential, it has been found that relatively few plaintexts need be tried before an easy one is found. Further, for a broad class of chosen plaintexts, the attack is linear in the size of the alphabet. The same method of attack is also applicable to Huffman coding.

Since splaying can be viewed as a form of adaptive Huffman coding, it seems reasonable that other forms of adaptive Huffman coding would also be susceptible to the attack described. In particular, if the details of the adaptation mechanism are publicly known (more precisely, available to the cryptanalyst), then the attack could be modified to accommodate those details. The exact time taken by the backtracking algorithm may change, possibly making the attack infeasible. However, it seems unlikely that any of the usual forms of adaptive Huffman codes [Gal78, Knu85, Vit86] contain any tricks likely to make the attack particularly difficult.

The use of any tree-based data compressor whose tree structure is static or adapted in a similar vein to the splay tree cannot be recommended as a security device.

Chapter 7

The Security of PPM

In this chapter, the cryptanalytic properties of the prediction by partial matching (PPM) compressor are studied [CW84b, BCW90]. This compressor currently holds the record for text compression and has been shown to rival the predictive abilities of human subjects [TC96, Tea95].

After giving a reasonably detailed description of the PPM method we introduce a new approach to cryptanalysis. In particular, it is shown how a PPM model can be used to automatically break a simple substitution cipher. The results of this approach are comparable to other methods for the automated solution of such ciphers, but it has fewer limitations and can solve a wider variety of problems. In the next chapter, the approach is extended, and a PPM model used to automate the cryptanalysis of Ziv-Lempel systems.

Starting in Section 7.3, the use of PPM as a cryptosystem is discussed. An existing chosen-plaintext attack [BH92, BH93] is extended to higher order models and the general implications of this kind of attack are discussed. The technique of forgetting is suggested as a means of avoiding such attacks. Some comments about the security of other forms of PPM, such as the unbounded PPM* compressor [CTW95], and a PPM compressor primed with a large static model of English, are also made.

Of all the compression techniques examined, the PPM method offers the strongest challenge to cryptanalysts and there are many security aspects of this system which are still open to debate. It is hoped that by exposing known results others will take up the challenge and answer some of the remaining questions.

7.1 The PPM Compressor

An explanation of PPM was presented in Chapter 3. What follows is an expanded form of that description.

In the ***prediction by partial matching*** (PPM) compressor, the previously transmitted symbols are used to condition the probability of the next symbol. There are many variants

of the basic approach depending on how many symbols are used to predict the next, whether or not multiple predictions are used, and how shorter context models are used when necessary. The predictions are based on simple frequency counts of the transmission so far. The primary decision to be made is the context length modelled. Better predictions can be made when long contexts are used: it is much easier to guess what comes after motorcycl than after cl. However, this means most contexts will never be seen, but all must be assigned some probability. Alternatively, the context can be made very short, but then compression will be poor in the long term as little structure will be learnt.

The ***order*** of a model is the maximum context length used to predict the next symbol. In practice, an order-o model will sometimes base its prediction on less than o symbols. By convention the order-(-1) model predicts each symbol with equal probability and the order-0 model predicts each symbol with probability proportional to the number of times it has occurred previously.

The PPM method can either be ***bounded***, if some maximum order is specified in advance, or ***unbounded*** [Tea95], if contexts are allowed to be arbitrarily long. A tree representation is suitable for both approaches.

The general PPM method requires the predictions of all orders be blended together to give an overall probability for the next symbol. The most general approach is to form a weighted sum of the individual probabilities:

$$\Pr(s_j = \phi) = \sum_{i=-1}^{o} w_i p_i(\phi),$$

where the w_i are a set of weights normalized to sum to 1, s_j is the next symbol, and $p_i(\phi)$ is the probability of symbol ϕ according to the order-i model. Calculating the sum is computationally expensive and there is no single 'right' way to determine the weights to use. The probabilities $p_i(\phi)$ are rational and based on the frequency counts:

$$p_i(\phi) = \frac{f_i(\phi)}{F_i} \qquad \text{where} \qquad F_i = \sum_{\phi} f_i(\phi), \qquad f_i(\phi) \in \mathbb{Z}.$$

However, this formula for $p_i(\phi)$ leads to the ***zero-frequency problem*** since symbols not previously encountered are given zero probability [WB91]. This is a problem because it is always possible that such symbols might occur. In practice all symbols must be given some probability of occurring. The formula for $p_i(\phi)$ must be modified so that symbols not previously encountered in a given context can be represented. Once again, there is no theoretical basis for preferring any particular method of modifying the probabilities.

If the context itself has never been seen before, then all of $f_i(\phi)$ and F_i will be zero. However, we are guaranteed that some shorter context has been seen before; in the worst case the order-(-1) model can be used. The order-(-1) model always predicts every symbol.

In practice full blending is not used because it is computationally expensive; instead each context assigns a probability, called an ***escape probability***, to a novel symbol occurring. The PPM variants differ in the way escape probabilities are assigned. When a novel symbol is seen the escape probability is sent, followed by the prediction of the next shorter context. Several escapes may be made before a context is reached which predicts the symbol. In the worst case the order-(-1) model makes the prediction. The escape mechanism has an equivalent blending mechanism as follows. Denoting the probability of an escape at level o by e_o, equivalent weights can be calculated from the escape probabilities by

$$\begin{aligned} w_o &= (1-e_o)\prod_{i=o+1}^{l} e_i, \qquad -1 \le o < l \\ w_l &= 1-e_l \end{aligned}$$

where l is the highest order context making a nonnull prediction. In this formula, the weight of each successively lower order is reduced by the escape probability from one order to the next. The weights will be normalized (all positive and summing to one) provided only that the escape probabilities are between 0 and 1 and that $e_{-1} = 0$. Escape probabilities are easier to visualize than weights and are more practical in implementation.

If $p_o(\phi)$ is the probability assigned to the symbol ϕ by the order-o model, the weighted contribution of the model to the blended probability of ϕ is

$$w_o p_o(\phi) = (1-e_o)p_o(\phi)\prod_{i=o+1}^{l} e_i.$$

In other words, it is the probability of decreasing to an order-o model, and not going any further, and selecting ϕ at that level. These weighted probabilities can then be summed over all values of o to determine the blended probability for ϕ. Specifying an escape mechanism amounts to choosing values for e_o and p_o. We now explain how the probabilities are determined in some variants.

- **PPMA** [CW84b]: This method uses the plausible assumption that novel symbols are more likely when the context has only been seen a few times. The probability of the escape symbol is $e_i = 1/(F_i+1)$ and the probabilities for the other symbols becomes
$$p_i(\phi) = \frac{f_i(\phi)}{F_i+1}.$$
- **PPMB** [CW84b]: In this method no prediction is made unless the symbol has occurred more than once in the current context. This is done by subtracting one from all counts, with the subtracted counts being combined to give the escape probability. The idea is to filter anomalous events. Thus $e_i = q_i/F_i$ where q_i is the number of different symbols seen in the given context. The probabilities for the remaining symbols become
$$p_i(\phi) = \frac{f_i(\phi)-1}{F_i}.$$

- **PPMC** [Mof90, Mof88]: This is a popular method similar to PPMB except symbols are predicted immediately. Thus

$$e_i = \frac{q_i}{F_i + q_i} \qquad \text{and} \qquad p_i(\phi) = \frac{f_i(\phi)}{F_i + q_i}.$$

- **PPMD** [How93]: This is a small modification to PPMC where each count is incremented by a 1/2. It sets $e_i = q_i/2F_i$ and

$$p_i(\phi) = \frac{2f_i(\phi) - 1}{2F_i}.$$

  The use of ***deterministic scaling*** [Tea95] also leads to better performance. In deterministic scaling a context that makes only a single prediction has its count temporarily increased. Experimental evidence indicated a scale factor of about 3 works best.

- **PPMP** [WB91]: This is a variant employing escape probabilities based on a Poisson process model. The escape probability is

$$e_i = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{q_i(k)}{F_i^k},$$

  where $q_i(k)$ is the number of types that have been seen exactly k times so far; thus $q_i = q_i(1)$.

- **PPMX** [WB91]: This is an approximation to PPMP using the first term of the summation, $e_i = q_i/F_i$.

7.1.1 Exclusions

In a fully blended model, the probability of a symbol includes predictions from contexts of many different orders, which makes it very time consuming to calculate. Moreover, an arithmetic coder requires cumulative probabilities from the model. These are slow to evaluate. Full blending is not a practical technique for finite-context modelling.

The escape mechanism can be used as the basis of an approximate blending technique called ***exclusion***, which decomposes a symbol's probability into several simpler predictions. It works as follows. When coding the symbol ϕ using context models with a maximum order m, the order-m model is first consulted. If it predicts ϕ with a nonzero probability, it is used to code ϕ. Otherwise the escape code is transmitted, and the order-$(m-1)$ model attempts to predict ϕ. Coding proceeds by escaping to smaller contexts until ϕ is predicted. The order-(-1) model guarantees that this will eventually happen.

The exclusion method is so named because it excludes lower-order predictions from the final probability of a symbol. Consequently, all other symbols encountered in higher-order contexts can be safely excluded from subsequent probability calculations because they will

never be coded by a lower-order model. This can be accomplished by modifying counts in lower-order models by (temporarily) setting the count associated with a symbol to zero if it has been predicted by a higher-order model. Thus the probability of a symbol is taken only from the highest-order context that predicts it.

There is a small flaw in each of methods A, B, and C when they are used with exclusions. If all the symbols of the alphabet have been encountered in a particular context the escape probability continues to be predicted with a small but nonzero probability. This can be important if a small alphabet is used. The algorithm could be modified to reduce the escape probability when the number of symbols approaches the maximum possible.

A further simplification of the blending technique is ***lazy exclusions***, which uses the escape mechanism in the same way as exclusions to identify the longest context that predicts the symbol to be coded; but does not exclude the counts of symbols predicted by longer contexts when making the probability estimate. This will always give worse compression (by about 5%) because such symbols will be predicted in the lower-order contexts, so the code space allocated to them is completely wasted [BCW90]. However, it is faster and there is no need to keep track of the symbols that need to be excluded.

In a fully blended model, it is natural to update counts in all the models of order $0, 1, \ldots, m$ after each symbol is coded, since all contexts were involved in the prediction. However, when exclusions are used, only one context is used to predict the symbol. This suggests a modification, called ***update exclusions***, where the count for the predicted symbol is not incremented if it is already predicted by a higher-order context. It is counted only in the context used to predict it. This improves compression by about 2% [BCW90].

7.1.2 Scaling Counts

While the models are being constructed, the counts associated with each context, F_o and $f_o(\phi)$, are being incremented. Typically frequency counts used by an arithmetic coder are narrow (14-bits for example) and it is convenient to store the model counts in the same width. The counts can then overflow.[1] An easy way of preventing this is to halve F_o and all of the $f_o(\phi)$ whenever F_o threatens to exceed the maximum allowed integer. It is important to ensure that none of the $f_o(\phi)$ are left at 0, which can be done either by rounding up or by deleting the reference to ϕ when $f_o(\phi) = 0$.

Retaining frequencies to low precision does little harm because small errors in predicted frequencies have almost no effect on the average code length. Consider a set of symbols with true probabilities p_i and predicted probabilities $p_i + \varepsilon_i$. The expected code length will then be:

$$-\sum p_i \lg(p_i + \varepsilon_i) = -\sum p_i \lg p_i - \sum \varepsilon_i + \sum \frac{\varepsilon_i^2}{2p_i} - \sum \frac{\varepsilon_i^3}{3p_i^2} + \cdots .$$

[1] Implementations of arithmetic coding using wider registers, say 32-bits, also exist and are naturally less prone to overflow.

The predicted probabilities sum to 1, $\sum \varepsilon_i = 0$, so, assuming that the errors are smaller than the probabilities p_i, the increase over entropy is approximately

$$\sum \frac{\varepsilon_i^2}{2p_i}.$$

If the maximum allowed count is C, the errors will be of order $1/C$ and the increase in the average code length of order $1/C^2$.

Scaling counts can improve compression. This is because the statistics in a text are usually different at the beginning than they are at the end. Hence recent statistics are often a better guide to what will be next and scaling reduces the effect of older statistics.

7.2 Automated Cryptanalysis of Simple Substitutions

In this section a PPM data compression model is used to automatically break simple substitution ciphers. In Chapter 8 this strategy is extended and PPM is used in a ciphertext only attack against Ziv-Lempel coding.

The idea of using text compression to break substitution ciphers stems from the observation that the best PPM models can predict language about as well as expert human subjects [TC96]. Although the attack is illustrated for English text, it is equally applicable to any natural language and to other redundant sources like programming languages.

The ciphertext only cryptanalysis of simple cryptosystems relies heavily on the statistical properties of the source language. Getting computers to perform this analysis is not a trivial matter. Although computers have been routinely used for a variety of tasks in cryptanalysis since their invention, the automatic recognition of valid decryptions has remained a taxing problem. For example, while several designs for machines to exhaustively search the keyspace of the DES have been published [Wie93, Sch94] they all assume at least known plaintext because of the difficulty of quickly recognizing a valid decrypt in a ciphertext only attack.

Previous approaches used in the solution of simple substitution ciphers are discussed, concentrating on previous methods for automated cryptanalysis. Then a description of the language model used in our approach is given, followed by some results.

7.2.1 Previous Approaches

Human subjects experienced in cryptanalysis can typically break a sentence-long cryptogram in a few minutes. A number of expositions of strategies for hand analysis have been given [Bal60, Fri76, Gai56, Wil59]; however, none of these descriptions are explicit enough to be called an algorithm. Generally, hand approaches are a combination of frequency analysis (usually only zeroth order), pattern matching, and word recognition. The use of special clues like apostrophes also tends to be highlighted.

One approach to automatic cryptanalysis [CM86] uses an expert system which attempts to capture the knowledge of the expert cryptanalyst. In a similar vein, another approach [Sch77, And89a, And89b] attempts to first determine the vowels. Then using the vowels and special clues (like apostrophes), a heuristic search of a specially constructed dictionary is performed.

But the majority of proposals for automated cryptanalysis fall into two classes: probabilistic methods [PR79, HM83] and pattern matching methods [Har94, Luc88]. In general, pattern matching methods seem to work better (in that they can solve shorter cryptograms). However, pattern matching methods cannot solve cryptograms which do not contain any words from the set of possible patterns. Although our results are comparable to those of the pattern matching method; our approach is a probabilistic method.

All previous approaches [PR79, Har94, Luc88] deal with twenty-six letter English and assume the spacing between words has already been identified or was not encrypted. However, the relaxation algorithm [PR79] can be applied when spaces are concealed [HM83]. The relaxation approach can also be used to automatically solve polyalphabetic substitution ciphers [Kin94]. Our approach is described for twenty-seven letter English (the twenty-six letters and a space symbol). That is, the space is allowed to be encrypted as well. In practice the distinction is unimportant because the space symbol is readily determined. In addition, the approach continues to perform well, without modification, on cryptograms from which spacing has been removed.

In the relaxation method, breaking the code is considered as a probabilistic labelling problem. Every code letter is assigned probabilities of representing each plaintext letter. By examining joint letter probabilities, the initial estimates are updated in a Bayesian manner; and hopefully after a number of iterations the probabilities for the correct mapping will converge to 1. The joint letter probabilities used to update the symbol probabilities are based on trigram frequencies (in [PR79] these were derived from *Wuthering Heights* [Bro73]). From the examples given in [PR79], it appears that the method only works for medium length cryptograms. They give two examples, a paragraph from a technical report (996 symbols) and Lincoln's Gettysburg Address (1149 symbols), for which the method is successful but make no comments as to the limitations of the approach.

In contrast, the pattern matching methods [Har94, Luc88] work well on short cryptograms but cannot handle trivial variations, such as examples with spacing removed. In the pattern matching approach words in the cryptogram are compared structurally with a lexicon. The time taken and accuracy of the approach is a trade-off on the size of the lexicon. Initially it may appear infeasible, since even an average size dictionary will list 100 000 distinct words; and it is likely that about 400 000 distinct words are in use at any particular time. If names of chemical compounds and taxonomic names are included this figure jumps to several million words. However, word frequencies obey Zipf's law [Zip39], and consequently any randomly selected word from free running English text has greater than 50% chance of being one of the following 135 words (starting with the

most common) [KF67]:

> the, of, and, to, a, in, that, is, was, he, for, it, with, as, his, on, be, at, by, I, this, had, not, are, but, from, or, have, an, they, which, one, you, were, her, all, she, there, would, their, we, him, been, has, when, who, will, more, no, if, out, so, said, what, up, its, about, into, than, them, can, only, other, new, some, could, time, these, two, may, then, do, first, any, my, now, such, like, our, over, man, me, even, most, made, after, also, did, many, before, must, through, back, years, where, much, your, way, well, down, should, because, each, just, those, people, Mr, how, too, little, state, good, very, make, world, still, own, see, men, work, long, get, here, between, both, life, being, under, never, day, same, another, know, while, last.

In fact, the words 'the' and 'of' alone account for more than 10% of all written words.

7.2.2 Language Model

All methods for automated cryptanalysis contain a model of the source whether this is made explicit or not. In the pattern matching methods the model is a lexicon of valid English words. In the relaxation algorithm [PR79] trigram frequencies are used.

Our approach uses a PPM model. Twenty novels and the Brown Corpus [FK82], as detailed in Figure 7.1, were used as training text in building the model. The lengths given are before reduction to twenty-seven letter English. The resulting zeroth order counts are given in Figure 7.2. After training the model it is written to disk and then remains static during cryptanalysis.

Most cryptograms were solved using an order-3 model. For a few difficult cases an order-4 model was tried. Due to memory constraints and insufficient training text, higher order models and unbounded models were not considered. For higher orders it would be desirable to have more training text to ensure a comprehensive and fair representation of the language. Further improvement would probably occur if a word model was incorporated.

For speed, the models are stored in a linear array as opposed to the more usual trie representation [BCW90]. The mapping $0 \leftrightarrow \epsilon$, $1 \leftrightarrow a$, $2 \leftrightarrow b$, ..., $26 \leftrightarrow z$, and $27 \leftrightarrow \sqcup$ is used. Then (for order-3) the frequency for context $\alpha\beta\gamma\delta$ is stored at location $\alpha 28^3 + \beta 28^2 + \gamma 28 + \delta$ of the array. The shorter context $\gamma\delta$ is stored at $28\gamma + \delta$ (equivalent to taking α and β as null).

It is possible to calculate an entropy for the entire model by considering a weighted average of the entropy of each possible context. For order-3 this is:

Book	Length (bytes)
The Adventures of Huckleberry Finn [Twa23]	527187
The Alhambra [Irv73]	692436
The Brown Corpus [FK82]	5998543
The Call of the Wild [Lon83]	177738
The Critique of Pure Reason [Kan55]	1260768
Descent of Man [Dar72]	1615141
Don Quixote [Cer91]	2274232
Far from the Madding Crowd [Har78] (book1)	765722
From the Earth to the Moon [Ver60]	242613
Gulliver's Travels [Swi47]	570937
The Hacker Crackdown [Ste92]	680109
The Hobbit [Tol66] (book3)	512179
Jane Eyre [Bro43]	1020769
The Jungle Book [Kip88]	173908
Lady Chatterley's Lover [Law59]	645423
The Phantom of the Opera [Ler88]	473543
Principles of Computer Speech [Wit82] (book2)	597853
Robinson Crusoe [Def62]	623409
Sense and Sensibility [Aus68]	676667
Treasure Island [Ste60]	364759
War and Peace (Book I) [Tol68]	273247
Total	20167183

Figure 7.1: Novels used to build the language model.

$$
\begin{aligned}
H_3 \;=\; & -q_3 \sum_{x \in A^3} p(x) \sum_{a \in A} p(a|x) \lg p(a|x) \\
& -(1-q_3) q_2 \sum_{x \in A^2} p(x) \sum_{a \in A} p(a|x) \lg p(a|x) \\
& -(1-q_3)(1-q_2) q_1 \sum_{x \in A} p(x) \sum_{a \in A} p(a|x) \lg p(a|x) \\
& -(1-q_3)(1-q_2)(1-q_1) q_0 \sum_{a \in A} p(a|\epsilon) \lg p(a|\epsilon) \\
& +(1-q_3)(1-q_2)(1-q_1)(1-q_0) \lg |A|,
\end{aligned}
$$

where q_i is the proportion of contexts that make nonzero predictions at order-i; and where all the conditional probabilities include escape probabilities when appropriate.

After training according to Figure 7.1, the value $H_3 \approx 2.25$ bits per symbol is obtained. A similar calculation for the order-4 model gives $H_4 \approx 1.95$ bits per symbol.

Letter	Count	Letter	Count
␣	3510082	M	401811
E	1970819	F	368319
T	1449120	W	326625
A	1261904	G	306670
O	1211410	P	290182
I	1120686	Y	288342
N	1114926	B	245222
S	1015683	V	154547
H	931843	K	106237
R	926052	X	33132
D	655938	J	22264
L	633046	Q	19957
C	444396	Z	12108
U	437382	Total	19258703

Figure 7.2: The zeroth order counts in the resulting model.

7.2.3 The Method

Zeroth order statistics of the ciphertext are used to obtain an initial permutation of the alphabet. In practice only a few symbols are correctly positioned in this way, but it does enable us to generate an order for making changes in the permutation. For example, if it turns out that the most frequent symbol in a particular cryptogram is not the space, then it is most likely to be our choice for E or T. The ***uncertainty*** of a permutation for a cryptogram is the ratio of the length of the cryptogram, in bits, when compressed using the language model to the length of the cryptogram in symbols. The smaller the uncertainty, the more closely the cryptogram resembles the model.

The basic approach is to repeatedly alter the permutation, recomputing the uncertainty at each stage. Whenever the uncertainty decreases, this is taken as evidence of an improved solution. However, swapping a single pair of symbols at each step does not work very well because the procedure is easily trapped in local minima. Therefore, several symbols are permuted at each stage.

Initially some random swaps are carried out to get the permutation headed in the right direction. This is arranged as a simulated annealing process [KGV83]. Starting at a temperature t we make t random swaps before recomputing the uncertainty. As time passes the value of t is reduced and the number of swaps made at that temperature is increased. Experiments indicate that it is best to do the annealing first at a relatively low temperature ($t = 5$) and then a subsequent step at much higher temperature ($t = 20$). This two stage approach allows the maximum benefit from the zeroth order permutation; whereas starting at a high temperature would cause this information to be lost. In fact, most cryptograms can be solved without recourse to annealing at all, but the use of

annealing, despite being an extra step, often results in faster convergence to a solution.

The annealing is followed by an exhaustive search over all 4! possible permutations of each group of four symbols. This has proved adequate for the solution of most ciphers, but if necessary, it would be possible to iterate over the attack several times.

7.2.4 Example Decryptions

The algorithm described has been implemented in C on a UNIX workstation. Although the program does not execute as quickly as the pattern matching approaches, it often produces a readable decryption in about two minutes.[2] The order-4 program requires 71 megabytes of memory, whereas the order-3 program requires only 3 megabytes. Using the same data structure an order-5 model would require in excess of 1500 megabytes of memory. However, by replacing our arrays with trie structures, higher orders could be considered. The method has been tried on a variety of cryptograms and in almost all cases it found a reasonable solution. For our purposes less than four errors in the final permutation is considered a successful decrypt. This is rather harsh since when five or sometimes even six errors occur the correct plaintext is still usually apparent. Figure 7.3 shows the results for a number of cryptograms from the literature [Wei96]. By our criteria a 83% success rate was achieved. Further, 60% of cryptograms were solved with no errors. This compares with 60% reported for a pattern matching approach (unfortunately the criteria for success are not specified) [Luc88]. Some particularly illustrative examples are given here.

Errors	Frequency
0	76
1	13
2	7
3	5
4	3
> 4	22

Figure 7.3: Summary of results. Each error represents an incorrect determination of a symbol in the permutation. Eleven examples which had more than four errors were solved when the order-4 model was used. In addition seventeen of the failures were for lists of words on particular topics and therefore did not form proper sentences.

In a similar manner to other approaches, very short cryptograms do not always lead to the 'correct' decryption. There are two reasons why the method can fail. Other possible decryptions can be deemed more likely by the language model or the procedure can get trapped in a local minimum and fail to find the optimal solution. However, a better model

[2] On certain cryptograms the pattern matching approach can produce answers in a matter of seconds [Har94].

of English would not allow shorter cryptograms to be solved because short cryptograms are inherently ambiguous. This is because unicity distance for a simple substitution is about 26,

$$U = \frac{H(K)}{R} = \frac{\lg 27!}{\lg 27 - 1.2} = 26.2.$$

But since the entropy of the order-3 model is about $H_3 = 2.25$, we cannot consistently expect to solve cryptograms shorter than

$$U = \frac{\lg 27!}{\lg 27 - H_3} = 37.2$$

symbols. For the order-4 model, $U = 33.2$. This is illustrated on the cryptogram *to be, or not to be: that is the question*, where our approach gives the answer `to be, of not to be: that is the prestion`, with a uncertainty of 1.811 whereas the 'correct' answer has a uncertainty of 1.909. If the order-4 model is used instead of the order-3 model, `to be, of not to be: that is the question` is obtained with a uncertainty of 1.611; whereas the 'correct' answer has uncertainty 1.656. The pattern matching approach [Har94] does not obtain a unique solution either, but finds the six possibilities given in Figure 7.4.

```
TO BE, OF NOT TO BE: THAT IS THE QUESTION
TO WE, OF NOT TO WE: THAT IS THE QUESTION
TO ME, OF NOT TO ME: THAT IS THE QUESTION
TO BE, OR NOT TO BE: THAT IS THE QUESTION
TO WE, OR NOT TO WE: THAT IS THE QUESTION
TO ME, OR NOT TO ME: THAT IS THE QUESTION
```

Figure 7.4: Solutions found by the pattern matching approach.

A second example, Figure 7.5, illustrates the robustness of the compression based approach. The ciphertext is a 129 letter sentence from *Gadsby* [Wri39] (a 50000 word novel in which the letter `e` never appears): *Upon this basis I am going to show you how a bunch of bright young folks did find a champion; a man with boys and girls of his own.* Hart [Har94] has claimed that no probabilistic method could solve this cryptogram because of its unusual letter distribution. However, we can completely solve the example (Figure 7.5), improving on the pattern matching method which failed to determine two letters (`c` and `k`). The space symbol and the single letter word `i` are quickly determined, and at the end of the annealing at least ten words have been correctly identified. But it is not until near the end of attack that the program determines that improvement occurs when the `e` is dropped altogether from the permutation.

A third example also from Hart [Har94]: *Loco Hobo once had Frau who put Hoi Polloi into a coma with an aria made famous by boy soprano*, is also successfully solved by our model, Figure 7.6. This sentence is unlikely according to our model (rated at 3.654 bits

```
Doing close annealing attack [4].
7.552 rgea dtio unoio i nc seias de oteh mer teh n urayt el uwistd meras lepko fif liaf n ytncgiea
7.517 rgeo dsia unaia i nc teiot de aseh mer seh n uroys ef uwitsd merot fepka lil fiol n ysncgieo
7.111 rgeo dsia uhaia i hl teiot de asen mer sen h urobs ef ucitsd merot fepka wiw fiow h bshlgieo
6.906 tgeo dais fhsis i hl reior de saen met aen h ftoba eu fcirad metor uepks wiw uiow h bahlgieo
6.531 wgeo dais uhsis i hl reior de saen few aen h uwoba em ucirad fewor mekps tit miot h bahlgieo
6.445 wgeo dais thsis i hl reior de saen few aen h twoba em tcirad fewor mekps uiu miou h bahlgieo
6.349 wgeo dais thsis i hl reior de saen few aen h twopa em tcirad fewor mekbs uiu miou h pahlgieo
6.294 wgeo dair thrir i hl seios de raen few aen h twopa em tcisad fewos mekbr uiu miou h pahlgieo
6.200 wgeo lair thrir i hu seios le raen few aen h twopa ed tcisal fewos dekbr mim diom h pahugieo
6.187 wgeo lais thsis i hu reior le saen few aen h twopa ed tciral fewor dekbs mim diom h pahugieo
6.181 wgeo lais thsis i hu reior le saen mew aen h twopa ed tciral mewor dekbs fif diof h pahugieo
6.117 wgeo lais thsis i hu reior le saen mew aen h twopa ed tciral mewor dekys fif diof h pahugieo
6.076 wpeo lais thsis i hu reior le saen mew aen h twoga ed tciral mewor dekys fif diof h gahupieo
6.066 lpeo rais thsis i hu weiow re saen mel aen h tloga ed tciwar melow dekys fif diof h gahupieo
5.955 npeo wais thsis i hu reior we sael men ael h tnoga ed tciraw menor dekys fif diof h gahupieo
5.857 fpei whos tasos o ak reoir we shel mef hel a tfigh ed tcorhw mefir deuys non doin a ghakpoei
5.734 fkeo whis tasis i an reior we shel mef hel a tfogh ed tcirhw mefor deuys pip diop a ghankieo
5.452 rkeo whis tasis i an feiof we shel der hel a troch em tgifhw derof meuys pip miop a chankieo
5.218 rkeo this wasis i an feiof te shel der hel a wroch em wgifht derof meuys pip miop a chankieo
5.077 rkeo this wasis i an feiof te shel per hel a wroch em wgifht perof meuys did miod a chankieo
5.022 rkeo this pasis i an feiof te shel wer hel a proch em pgifht werof meuys did miod a chankieo
4.987 rkeo this pasis i an feiof te shel wer hel a proch em pgifht werof meubs did miod a chankieo
4.906 rkeo this pasis i an feiof te shel wer hel a proch em pgifht werof meuxs did miod a chankieo
Doing close annealing attack [20].
4.860 aker this mosis i on leirl te shep wea hep o marbh ef mcilht wearl feuxs did fird o bhonkier
4.849 amer this cosis i on leirl te shep wea hep o carkh ef cvilht wearl feuxs did fird o khonmier
4.810 ador this kesis i en moirm to shop woa hop e karch of kyimht woarm fouxs lil firl e chendior
4.557 knor this easis i al moirm to show pok how a ekrch of eyimht pokrm fouxs did fird a chalnior
4.446 knor this casis i al moirm to show pok how a ckrgh of cyimht pokrm fouxs did fird a ghalnior
4.403 klor this casis i an moirm to show pok how a ckrgh of ceimht pokrm fouxs did fird a ghanlior
4.400 knor this casis i al moirm to show gok how a ckrph of ceimht gokrm fouxs did fird a phalnior
4.294 enor this fasis i ag moirm to show loe how a ferph ok fcimht loerm kouxs did kird a phagnior
4.223 enor this fasis i ag doird to show coe how a ferph ok flidht coerd kouxs mim kirm a phagnior
4.159 enor this casis i ag doird to shom woe hom a cerph of clidht woerd fouxs kik firk a phagnior
4.136 egor this casis i an doird to shok woe hok a cerph of clidht woerd fouxs mim firm a phangior
4.050 egor this casis i an doird to show moe how a cerph of clidht moerd fouxs kik firk a phangior
3.946 egor this casis i an doird to show koe how a cerph of clidht koerd fouxs mim firm a phangior
3.935 enor this casis i ag doird to show koe how a cerph of clidht koerd fouxs mim firm a phagnior
3.921 enor this casis i ag moirm to show koe how a cerph of climht koerm fouxs did fird a phagnior
3.902 enor this casis i ag moirm to show doe how a cerph of climht doerm fouxs kik firk a phagnior
Doing order 4 attack
3.893 enor this casis i am goirg to show doe how a cerph of clight doerg fouxs kik firk a phamnior
3.858 enor this casis i am poirp to show doe how a cergh of clipht doerp fouxs kik firk a ghamnior
3.854 enor this casis i am poirp to show doe how a cergh of clipht doerp fouxs bib firb a ghamnior
3.816 enor this casis i am poirp to show boe how a cergh of clipht boerp fouxs did fird a ghamnior
3.770 enor this basis i am poirp to show coe how a bergh of blipht coerp fouxs did fird a ghamnior
3.761 enor this basis i am poirp to show goe how a berch of blipht goerp fouxs did fird a chamnior
3.556 eron this basis i am poinp to show goe how a bench of blipht goenp fouxs did find a chamrion
3.483 elon this basis i am poinp to show goe how a bench of bripht goenp fouxs did find a chamlion
2.953 elon this basis i am going to show poe how a bench of bright poeng fouxs did find a chamlion
2.943 ulon this basis i am going to show pou how a bunch of bright poung foexs did find a chamlion
2.768 upon this basis i am going to show lou how a bunch of bright loung foexs did find a champion
2.580 upon this basis i am going to show you how a bunch of bright young foexs did find a champion
2.555 upon this basis i am going to show you how a bunch of blight young foexs did find a champion
2.534 upon this basis i am going to show you how a bunch of blight young foevs did find a champion
2.531 upon this basis i am going to show you how a bunch of blight young foers did find a champion
2.359 upon this basis i am going to show you how a bunch of bright young foles did find a champion
2.320 upon this basis i am going to show you how a bunch of bright young folks did find a champion
```

Figure 7.5: Example cryptogram of 129 symbols with unusual statistics. The space symbol is immediately determined because it is the most common symbol in the cryptogram. As `o` is the next most common symbol it is initially classified as `e`, and so on. The numbers on the left are the uncertainty in bits/symbol.

per character) and our attack finds incorrect solutions which are considered to be more likely. However, the correct result is passed over and the final result is still better than the result obtained in [Har94].

The best answer to *sleepy mandrill eyes mandolin player singing pop songs* [Luc88] using the order-3 model is `sleepy contrill eyes contalin ployer singing pap sangs` which has three errors, `c`↔`m`, `o`↔`a`, and `t`↔`d`. But by using the order-4 model (Figure 7.7) this reduces to a single error, `b`↔`m`.

It is interesting to consider the success as a function of cryptogram length. Figure 7.8 plots this information for cryptograms of length up to 200 (every example longer than this was successfully solved with no errors).

Our last example, an extract from *The Moonstone* [Col82], illustrates the robustness of our approach by showing how a cryptogram can still be solved even when the spaces have been removed. To be fair, a considerably longer cryptogram is needed to succeed in such a difficult situation. Notice, however, that if a model were built specifically for this situation, much shorter examples could be solved. The execution trace is shown in Figure 7.9. There are only two errors in the final answer `y`↔`x` and ␣ ↔`y`. The false classification of a symbol as space is to be expected to occur in any such example; but this affects at most one symbol and provided the remaining symbols are correctly determined there is no problem identifying the true symbol.

7.3 Using PPM as a Cryptosystem

The use of PPM as a cryptosystem is now considered. Circumstantial evidence given in Chapter 4 suggests PPM may be immune from ciphertext only attack. However, resistance from ciphertext only attacks does not imply resistance to more powerful attacks. In the next section a vulnerability to chosen plaintext attack will be demonstrated.

Before PPM can be used as a cryptosystem, a key mechanism must be introduced. This mechanism should satisfy the desiderata of Chapter 3, namely: it should be simple, protect all the compressed output, not significantly degrade compression, and not call for keys significantly longer than conventional cryptosystems.

The extra complexity of PPM when compared to other compressors allows more scope in the selection of a key mechanism. The following five possibilities represents a few obvious choices:

- The initial state of the arithmetic coder (as discussed in Chapter 5). The available key size is linear in the register width.

- The initial structure of the model. The key size can be made arbitrarily large by choosing a suitable model.

```
Doing close annealing attack [4].
7.153 rece neue etcp niy ghio fne los nea lerrea atse i cedi fasn it ihai diyp gideow um uem welhi
6.940 uece tere encp tiy ghio fte sol tea seuuea anle i cedi falt in ihai diyp gideow rm rem weshi
6.488 hece tere encp tiy guio fte sol tea sehhea anle i cedi falt in iuai diyp gideow rm rem wesui
6.032 cehe teme enhp tiy guio wte sol tea seccea anle i hedi walt in iuai diyp gideof mr mer fesui
5.998 rese tece ensk tiy guio wte hol tea herrea anle i sedi walt in iuai diyk gideof cm cem fehui
5.960 rese tece eusk tiy gnio wte hol tea herrea aule i sedi walt iu inai diyk gideof cm cem fehni
5.923 hese tece eusk tif gnio wte rol tea rehhea aule i sedi walt iu inai difk gideoy cm cem yerni
5.713 hese tece eusk tip gnio wte rol tea rehhea aule i sedi walt iu inai dipk gideoy cm cem yerni
5.621 here tece eurk tip gnio wte sol tea sehhea aule i redi walt iu inai dipk gideoy cm cem yesni
5.456 here tece eurk tip gnio wte los tea lehhea ause i redi wast iu inai dipk gideoy cm cem yelni
5.379 here teme eurk tip gnio wte dos tea dehhea ause i reli wast iu inai lipk gileoy mc mec yedni
5.364 here teme enrk tip guio wte dos tea dehhea anse i reli wast in iuai lipk gileoy mc mec yedui
5.348 here teme enrc tik puio wte dos tea dehhea anse i reli wast in iuai likc pileoy mg meg yedui
5.327 here teme enrc tik puis wte dso tea dehhea anoe i reli waot in iuai likc pilesy mg meg yedui
5.175 here teme enry tik guis wte dso tea dehhea anoe i reli waot in iuai liky gilesc mp mep cedui
5.119 here teme enry tic guis wte dso tea dehhea anoe i reli waot in iuai licy gilesv mp mep vedui
5.033 mere tehe enry tic guio wte dos tea demmea anse i reli wast in iuai licy gileov hp hep vedui
4.960 here tepe enry tic guio wte dos tea dehhea anse i reli wast in iuai licy gileov pm pem vedui
4.935 here teme enry tic guio wte los tea lehhea anse i redi wast in iuai dicy gideov mp mep velui
4.908 here teme enry tip guio wte los tea lehhea anse i redi wast in iuai dipy gideoc mv mev celui
4.756 here teme enry tic guio wte dos tea dehhea anse i reli wast in iuai licy gileop mv mev pedui
4.712 heve teme envy tic guio wte dos tea dehhea anse i veli wast in iuai licy gileop mr mer pedui
4.644 hele teme enly tic guio wte dos tea dehhea anse i levi wast in iuai vicy giveop mr mer pedui
Doing close annealing attack [20].
4.641 pele teme enly tic guio wte dos tea deppea anse i levi wast in iuai vicy giveoh mr mer hedui
Doing order 4 attack
4.627 pele teme enly tic guio wte dos tea deppea anse i levi wast in iuai vicy giveof mr mer fedui
4.580 hele teme enly tic guio wte dos tea dehhea anse i levi wast in iuai vicy giveof mr mer fedui
4.558 hele seme enly sit guio wse doc sea dehhea ance i levi wacs in iuai vity giveof mr mer fedui
4.489 sele heme enly hit guio whe doc hea dessea ance i levi wach in iuai vity giveof mr mer fedui
4.458 seve heme envy hit guio whe doc hea dessea ance i veli wach in iuai lity gileof mr mer fedui
4.434 seve heme envy hit guio phe doc hea dessea ance i veli pach in iuai lity gileof mr mer fedui
4.408 seve heme envy hit gwio phe doc hea dessea ance i veli pach in iwai lity gileof mr mer fedwi
4.403 sele heme enly hit gwio phe doc hea dessea ance i levi pach in iwai vity giveof mr mer fedwi
4.401 sede heme endy hit glio phe woc hea wessea ance i devi pach in ilai vity giveof mr mer fewli
4.370 sede heme endy hit glio phe boc hea bessea ance i devi pach in ilai vity giveof mr mer febli
4.324 sere heme enry hit glio phe boc hea bessea ance i revi pach in ilai vity giveof md med febli
4.280 sele heme enly hit grio phe boc hea bessea ance i levi pach in irai vity giveof md med febri
4.243 hele seme enly sat grao pse boc sei behhei ince a leva pics an aria vaty gaveof md med febra
4.078 hele seme enly sac grao pse bot sei behhei inte a leva pits an aria vacy gaveof md med febra
4.034 here seme enry sac glao pse bot sei behhei inte a reva pits an alia vacy gaveof md med febla
4.032 here seme enry sac glao pse bot sei behhei inte a reva pits an alia vacy gaveow md med webla
4.000 vere hese enry had glao phe bot hei bevvei inte a rema pith an alia mady gameof sx sex febla
3.983 vere hese enry had mlao phe bot hei bevvei inte a rega pith an alia gady mageof sx sex febla
3.967 vere hese enry had clao phe bot hei bevvei inte a rega pith an alia gady cageof sx sex febla
3.896 vere hese enry had clao whe bot hei bevvei inte a rega with an alia gady cageof sx sex febla
3.879 vere hese enry had plao whe bot hei bevvei inte a rega with an alia gady pageof sx sex febla
3.827 leve hese envy had prao whe bot hei bellei inte a vega with an aria gady pageof sx sex febra
3.795 loco hogo once had frau who but hoi bolloi into a coma with an aria made famous gp gop sobra
3.752 loco homo once hag frau who but hoi bolloi into a cova with an aria vage favous my moy sobra
3.661 loco homo once had frau who but hoi bolloi into a cova with an aria vade favous my moy sobra
3.654 loco hobo once had frau who put hoi polloi into a coma with an aria made famous by boy sopra
3.646 loso hobo onse had frau who cut hoi colloi into a soma with an aria made famoup by boy pocra
3.626 logo hobo onge had frau who cut hoi colloi into a goma with an aria made famous by boy socra
```

Figure 7.6: Example cryptogram of 95 symbols.

```
Doing close annealing attack [4].
7.581 aiooth frelunii ohoa frelsine tirhou anedned tst aseda
7.486 aiooth flerunii ohoa flersine tilhou anedned tst aseda
7.257 aittoh flerunii thta flersine oilhtu anedned oso aseda
7.066 aittos flehunii tsta flehrine oilstu anedned oro areda
6.706 aittos flehurii tsta flehnire oilstu aredred ono aneda
6.165 aittos flemurii tsta flemnire oilstu aredred ono aneda
6.084 ainnso flemurii nona flemtire silonu aredred sts ateda
5.912 sinnao flemurii nons flemtire ailonu sredred ata steds
5.795 sinnto flemurii nons flemaire tilonu sredred tat saeds
Doing close annealing attack [20].
5.764 tinnsa flemurii nant flemoire silanu trecrec sos toect
5.322 tinnsa flemurii nant flemoire silanu tredred sos toedt
5.178 tinnsa clemurii nant clemoire silanu tredred sos toedt
5.140 tinnpa clemurii nant clemoire pilanu tresres pop toest
5.058 tinnpa clemurii nant clemoire pilanu tredred pop toedt
Doing order 4 attack
4.921 tinnga clemurii nant clemoire gilanu tredred gog toedt
4.904 tissmu clenarii sust clenoire milusa tredred mom toedt
4.816 tissmu glenarii sust glenoire milusa tredred mom toedt
4.800 tillsu wcenarii lult wcenoire sicula tredred sos toedt
4.693 tillbo wcenarii lolt wcenuire bicola tresres bub tuest
4.634 sellbo frigatee lols frigueti berola stictic bub suics
4.615 sellbo whitanee lols whitueni behola snicnic bub suics
4.609 sellbo rhindtee lols rhinueti behold stictic bub suics
4.511 tellbo chandree lolt chaniera behold tramram bib tiamt
4.439 tenndo placgree nont placiera delong tramram did tiamt
4.332 sleepy darknoll eyes darkilor playen sortort pip sirts
4.328 sleepy gardnoll eyes gardilor playen sortort pip sirts
4.109 sleepy gandroll eyes gandilon player sontont pip sints
3.780 sleepy gantroll eyes gantilon player sondond pip sinds
3.628 sleepy fantroll eyes fantilon player sondond pip sinds
3.435 sleepy condrill eyes condalin ployer singing pap sangs
3.331 sleepy contrill eyes contalin ployer singing pap sangs
3.165 sleepy candrill eyes candolin player singing pop songs
3.020 sleepy bandrill eyes bandolin player singing pop songs
```

Figure 7.7: Example cryptogram of 55 symbols (solved with order-4 model).

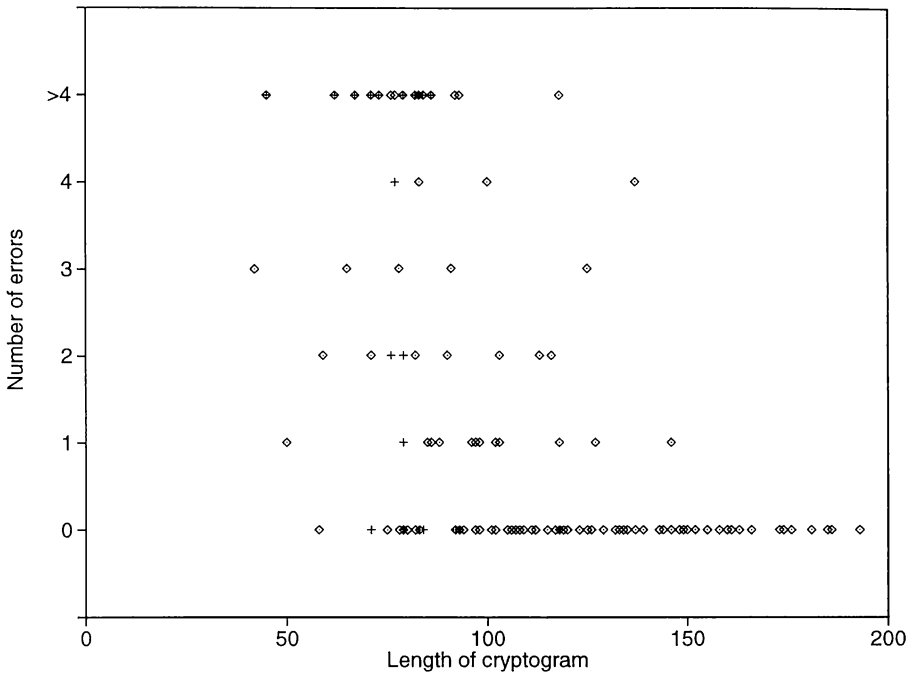


Figure 7.8: Length of cryptograms versus successful decryption. The diamonds represent order-3 cryptanalysis and the pluses order-4 cryptanalysis. The order-4 model was applied only to those cases with more than four errors when the order-3 model was used.

- The initial position within some large static publicly available model. The key size is equivalent to the number of contexts in the model.
- The initial counts of the contexts in the model. Again the key size is directly dependent on the size of the model.
- The permutation of the symbols in the model and/or the arithmetic coder. If n is the alphabet size and w the register width, this permits key sizes of the order $n!w$.

The cryptanalysis of a PPM system can be viewed as the problem of determining the model. Because of the intricacies of PPM, it is likely that most of the model must be determined before any useful decryption can be performed. An exhaustive search over all possible models is infeasible as the number of models grows exponentially in the amount of memory made available to the model.

7.4 Chosen Plaintext Attack Against PPM

The security of fixed order memory bounded PPM is discussed. A chosen plaintext attack requiring a lengthy chosen sequence is given. The attack exploits the count scaling mechanism of PPM and illustrates a general weakness of the adaptive strategy, in that, the models are designed to place less significance on information learned early on in the

```
Doing close annealing attack [4].
7.539 th stonreafols neafimitir vtniwt dlnorilirecumim neiddn ssth s cea sgnett aeaeadeitorun cite
7.519 th stonreafods neafiwitir vtnimt ldnoridirecuwiw neilln ssth s cea sknett aeaealeitorun cite
7.182 thestonr apodsen apikitirevtnimteldnoridir cukiken illnessthesec aeswn ttea a al itorunecit
7.145 thestond aforsen afikitidextnimtelrnodirid cukiken illnessthesec aeswn ttea a al itodunecit
6.969 thestorn afodser afikitinextrimteldronidin cukiker illressthesec aeswr ttea a al itonurecit
6.943 thestorn afodser afikitinextribteldronidin mukiker illressthesem aeswr ttea a al itonuremit
6.911 thestorn afodser afikitinextribteldronidin mukiker illressthesem aeswr ttea a al itonuremit
6.693 thestorna codsera cikitinextribteldronidinamukikeraillressthesema eswratte a a laitonuremita
6.669 thestorna kodsera kimitinextribteldronidinacumimeraillressthesеca eswratte a a laitonurecita
6.646 thestorna kodsera kimitineytribteldronidinacumimeraillressthesеca eswratte a a laitonurecita
6.479 thentorsa kodnera kimitiseytricteldrosidisabumimeraillrenntheneba enwratte a a laitosurebita
6.380 thentorsa kodnera kimitiseptricteldrosidisabumimeraillrenntheneba enwratte a a laitosurebita
6.365 thentorsa kodnera kimitisextricteldrosidisabumimeraillrenntheneba enwratte a a laitosurebita
6.341 thentorsa kodnera kimitisextricteldrosidisabumimeraillrenntheneba enwratte a a laitosurebita
6.301 thestorna kodsera kimitinextricteldronidinabumimeraillresstheseba eswratte a a laitonurebita
6.292 thestorna kodsera kimitinextricteldronidinabumimeraillressthesеba espratte a a laitonurebita
6.284 thestorna kodsera kimitinextricteldronidinabumimeraillressthesеba espratte a a laitonurebita
6.222 thestorna kodsera kimitinextricteldronidinabumimeraillressthesеba espratte a a laitonurebita
6.181 thestorna kodsera kimitinextricteldronidinabumimeraillressthesеba espratte a a laitonurebita
Doing close annealing attack [20].
6.158 thestorna podsera pimitinextricteldronidinavumimeraillressthesеva esgratte a a laitonurevita
6.142 thestorna podsera pimitinextricteldronidinavumimeraillressthesеva esgratte a a laitonurevita
6.071 thestorn apodser apivitinextricteldronidin muviver illressthesem aesgr ttea a al itonuremit
6.046 thestorn apodser apivitinextricteldronidin muviver illressthesem aesgr ttea a al itonuremit
Doing order 4 attack
5.992 thestorm apodser apivitimextrictendromidim luviver innressthesel aesgr ttea a an itomurelit
5.883 thestormanpodseranpivitimextricte dromidimaluviverai resstheselanesgrattenanan aitomurelita
5.707 thestorminpodserinpavatamextracte dromadamiluvaveria ressthеselinesgritteninin iatomurelati
5.684 thestorminpofserinpavatamextracte fromafamiluvaveria ressthеselinesgritteninin iatomurelati
5.624 thestorminpofserinpavatamextracte fromafamilyvaveria ressthеselinesgritteninin iatomyrelati
5.447 thestorminpofserinpavatamextractedfromafamil vaveriaddressthеselinesgritteninindiatom relati
5.240 thestorminvofserinvapatamextractedfromafamil paperiaddressthеselinesgritteninindiatom relati
5.213 thestormingofseringapatamextractedfromafamil paperiaddressthеselinesvritteninindiatom relati
4.884 thestormingofseringapatamextractedfromafamil paperiaddressthеselineskritteninindiatom relati
4.846 thestormingofseringapatamextractedfromafamil paperiaddressthеselinesyritteninindiatom relati
4.652 thestormingofseringapatamextractedfromafamil paperiaddressthеselineswritteninindiatom relati
4.643 thestormingofseringapatameytractedfromafamil paperiaddressthеselineswritteninindiatom relati
      vesinenglandm objectistoeyplainthemotivewhichhasinducedmetorefusetherighthandoffriendshiptom
       cousinjohnherncastlethereservewhichihavehithertomaintainedinthismatterhasbeenmisinterpreted
      b membersofm famil whosegoodopinionicannotconsenttoforfeit
```

Figure 7.9: Example cryptogram of 334 symbols and with no spaces.

compression process which does not recur later in the input. By sending suitable plaintext the unknown model can be driven into a known state. This attack is very long and would be difficult to carry out in practice. To guard against this attack the technique of forgetting is introduced.

This type of chosen attack was first introduced by Bergen and Hogan [BH92, BH93]. They applied it against a fixed-model [BH92], and later against a simple adaptive model (essentially an order-0 PPM model) [BH93]. We generalize the method to work against any adaptive fixed-order model with fixed width counts. The same technique cannot be used to break unbounded versions of PPM such as PPM* [CTW95] because the attack requires that the model have a fixed and finite number of nodes. It is assumed the order of the PPM model is known before the attack commences.

Because the counts are of fixed width, it is periodically necessary for the model to halve the counts. Notice, however, the wider the counts the less often they need to be scaled. We assume scaling occurs whenever they threaten to exceed the maximum allowable value.

The attack works by constructing a model $\hat{M}$ which is an estimate of the true (and initially unknown) model M. At each stage symbols are sent designed to improve the closeness of the estimate to the real model. The attack is complicated by the fact that M changes as the sequence is sent. Essentially, a type of moving target search is performed over a very large space. At the end of the attack, it is guaranteed that the model $\hat{M}$ will be identical to the model M; and the cryptanalyst can use $\hat{M}$ to decode any further transmissions on the channel. Because an unknown number of count halvings may have occurred during the attack, it remains impossible to reconstruct the state of the model M before the attack commenced. This means that if the model is periodically reset to some secret state then the cryptanalyst will have to perform the attack again. Since the sequence transmitted forces the models to synchronize, it is not necessary for the cryptanalyst to examine the encoded output.

Let n denote the size of the alphabet $A = \{a_1, \ldots, a_n\}$, o the order of the model, and μ the maximum count allowed. Then the chosen plaintext required is

$$S = \Upsilon S(o) \tag{7.1}$$

where

$$S(r) = \coprod_{i_1=1}^{n} \coprod_{i_2=1}^{n} \cdots \coprod_{i_r=1}^{n} (a_{i_1} a_{i_2} \cdots a_{i_r} a_1)^{\mu \lg \mu} (a_{i_1} a_{i_2} \cdots a_{i_r} a_0)^{\mu \lg \mu},$$

and Υ is any sequence in which every context of length o occurs at least once. The length of this sequence is

$$|S| = |\Upsilon| + 2(o+1)(\mu \lg \mu) n^o.$$

The obvious way to form the sequence Υ is to simply concatenate all the possible subsets of A^{o+1}. So for example, if $A = \{0, 1\}$, then for $o = 2$ we would concatenate the triples 000, 001, 010, 011, 100, 101, 110, and 111 to obtain 000001010011100101110111.

The length of Υ would then be $(o+1)n^{o+1}$. However, all the triples occur at least twice in this sequence, and this suggests that some improvement should be possible. The following theorem shows that we can do much better, in that it is always possible to produce a closed sequence in which every tuple occurs exactly once.

THEOREM 7.1: [Goo46] Given a finite alphabet A of size n, there is a closed n-ary sequence of length n^r in which each r-tuple in A^r occurs exactly once. (Here closed means the end of the sequence is considered contiguous with its beginning.)

PROOF: [Goo46] Suppose $r > 1$ (the case $r = 1$ is trivial). Let any sequence of $r-1$ symbols drawn from A be called a vertex. If u and v are two vertices of the forms $a_1a_2\cdots a_{r-1}$ and $a_2a_3\cdots a_r$, then the sequence $a_1a_2\cdots a_r$ is called an edge from u to v. This system of vertices and edges is a directed graph containing n^{r-1} vertices. Any two vertices $a_1a_2\cdots a_{r-1}$ and $b_1b_2\cdots b_{r-1}$ are connected by the edges corresponding to sequences of length r in the sequence of symbols $a_1a_2\cdots a_{r-1}b_1b_2\cdots b_{r-1}$. Finally, there are n edges leading into and n leading out of each vertex. (Note the vertices $a_ia_i\cdots a_i$ have a loop which counts as one in and one out.) This is a sufficient condition for the directed graph to have a Euler cycle [BCL79]. Taking in turn the first symbol in each edge of such a cycle, we obtain a sequence of n^r symbols which satisfies the requirements of the theorem.

■

Figure 7.10 illustrates the theorem for the case $r = n = 3$ on alphabet {A,B,C}. One Euler cycle in this figure is AAABBBAACCCABABCCBBCACBCBAC. Such sequences are called ***deBruijn sequences*** [deB46]. Alternative ways of constructing such sequences have been given [Ree46, Gol67, Mar34]. The more general question of the existence of a closed sequence of k n-ary symbols such that k subsequences of r consecutive symbols are all different has also been addressed [Lem71].

Although in our application we do not have closed sequences, one can simply copy the first o symbols of the sequence onto the end of the sequence, thereby forming a sequence of length $n^{o+1} + o$ in which every $(o+1)$-tuple occurs at least once. For the example we then have

$$\Upsilon = \text{AAABBBAACCCABABCCBBCACBCBACAA.}$$

Thus an upper bound on the length of the chosen plaintext 7.1 is

$$n^{o+1} + o + 2(o+1)(\mu \lg \mu)n^o.$$

Therefore for a fixed order o, the length of the chosen plaintext sequence is at most polynomial in the size of the alphabet and the maximum count. However, it is more usual in these circumstances to consider the complexity in terms of the number of bits required to express μ rather than μ itself. So if μ is expressed using w bits (that is, $\mu \leq 2^w$) we have

$$n^{o+1} + o + w(o+1)2^{w+1}n^o,$$

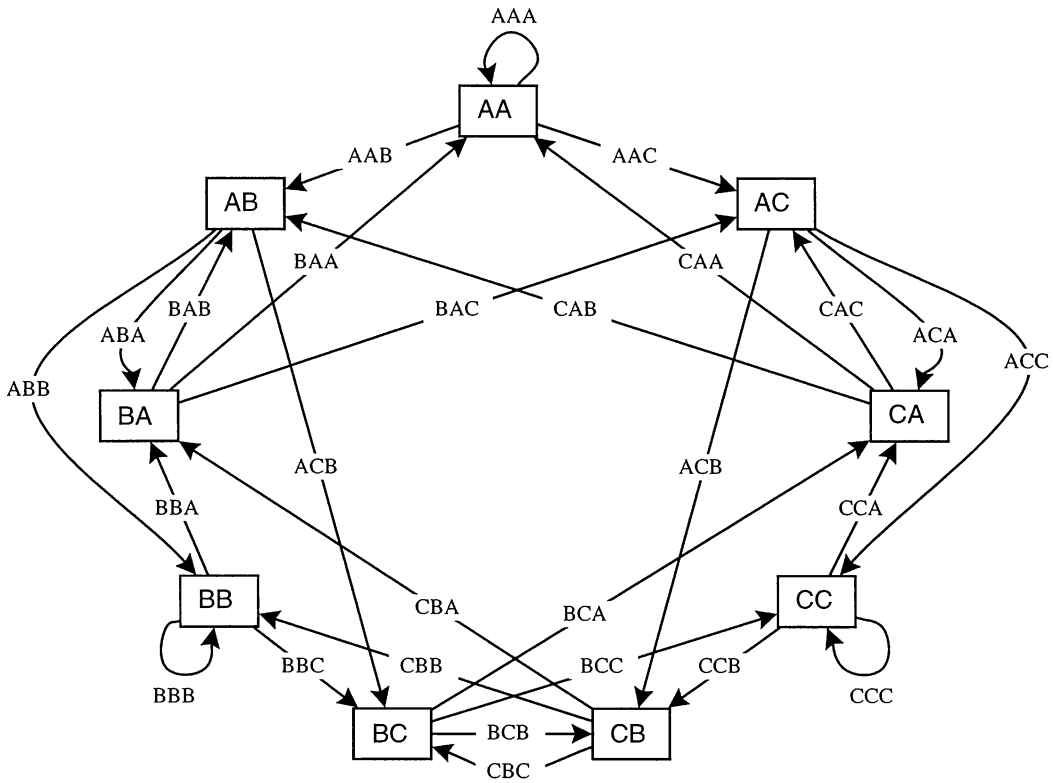


Figure 7.10: The deBruijn sequence construction for $n = r = 3$.

and the attack is exponential in the width of the counts and order of the model.

How does the sequence S drive both models into the same state? After sending Υ, every context of order o or less will have occurred at least once. This ensures that both models, M and $\hat{M}$, have the same number of nodes. However, the actual counts for the symbols in M remain unknown. Now the sequence $S(o)$ is sent, which will synchronize the counts in all orders in the two models. This occurs because each context $a_{i_1} a_{i_2} \cdots a_1$ is sent enough times that $\lg \mu$ count halvings occur in that context. Then all counts in that context, except $a_{i_1} a_{i_2} \cdots a_1$, will be 1 in both M and $\hat{M}$. To synchronize the a_1 value a second context is sent sufficient times to reduce the count for a_1 to 1 in both models.

At the end of the attack, both models will have identical counts for every context in the model. However, the models still may not be identical because of the ordering of the symbols. Further symbols may thus be required to force the same ordering in both models. For the scheme in [BCW90], sending the sequence Υ once more ensures the ordering becomes the same in both models.

EXAMPLE: Consider the attack for an order-2 model on alphabet {A,B,C}. Take a_0 = A, a_1 = B, and $n = 8$. The state of the models at the end of step 2 is given in Figure 7.11, where the sequence AAABBBAACCCABABCCBBCACBCBACAA has been sent. This is followed by the sequence $S(3)$ which is guaranteed to force all the counts to the same values in both models, as shown in Figure 7.12. The complete sequence $\Upsilon S(3)$

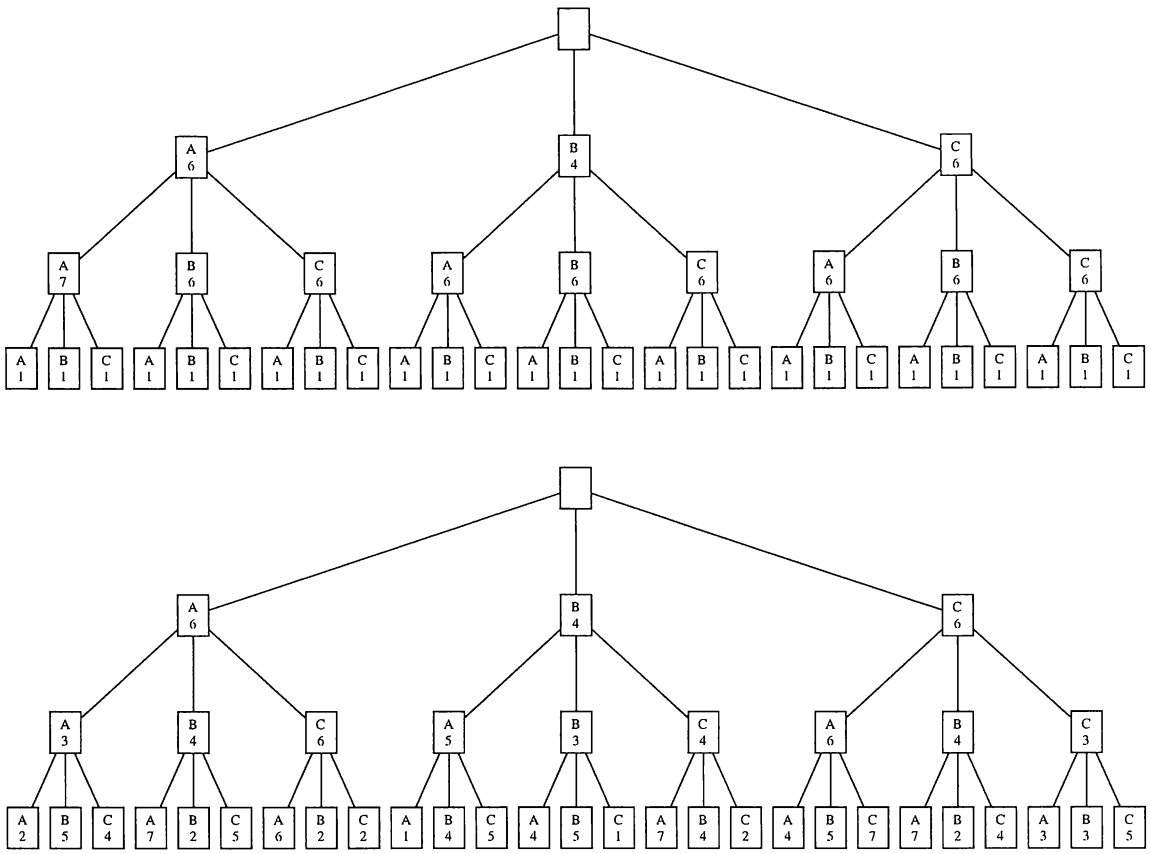


Figure 7.11: The state of the two models after sending the sequence AAABBBAACCCABABCCBBCACBCBACAA. Recall that the maximum count is eight, and scaling occurs whenever a count reaches eight. The top tree is the estimate $\hat{M}$ while the lower tree is the real model M (which at this stage is unknown to the cryptanalyst). In the estimate $\hat{M}$, the count for the singleton B is less than those of A and C because B occurs less often in the sequence.

is given in Figure 7.13.

7.4.1 Forgetting

One simple idea to make it harder for the cryptanalyst to follow the adaptation is to selectively prune parts of the model, say using a pseudorandom number generator. We call this idea ***random forgetting***.

Unfortunately, a true source of random bits cannot be assumed because of the constraint that the encoder and decoder must have identical information. Clearly, this can only be achieved if the sequence of random numbers used is the same for both the encoder and the decoder. Further, the most common ways of generating random bits (such as linear congruential generators [Knu81] and shift registers [Gol67]) are cryptographically insecure [Boy89]—although it is not clear how these results might apply in this situation.

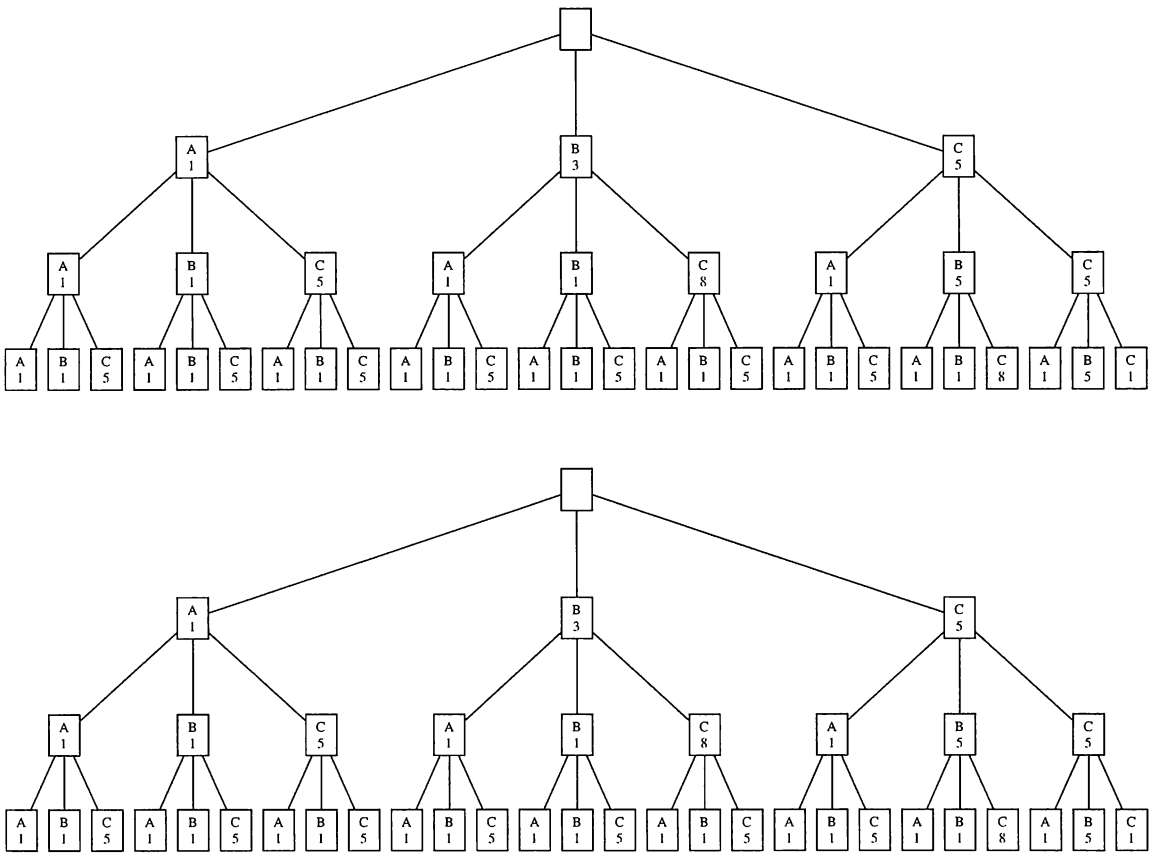


Figure 7.12: The state of the two models after sending the complete attack sequence. Both models are now the same and the cryptanalysts will be able to decode all future communications.

```
AAABBBAACCCABABCCBBCACBCBACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAABAABAABAABAABAABAABAABAABAABAABAABAABAA
BAABAABAABAABAABAABAABAABAABAABABAABAABAABAABAABAABAABAABAABAABAABAABAABAA
BAABAABAABAABAABAABAABAABAABAABBABBABBABBABBABBABBABBABBABBABBABBABBABBABB
ABBABBABBABBABBABBABBABBABBABBABBACAACAACAACAACAACAACAACAACAACAACAACAACAAC
AACAACAACAACAACAACAACAACAACAACAACAACBACBACBACBACBACBACBACBACBACBACBACBACBA
CBACBACBACBACBACBACBACBACBACBACBACBBAABAABAABAABAABAABAABAABAABAABAABAABAA
BAABAABAABAABAABAABAABAABAABAABAABAABABBABBABBABBABBABBABBABBABBABBABBABBA
BBABBABBABBABBABBABBABBABBABBABBABBABBBABBABBABBABBABBABBABBABBABBABBABBAB
BABBABBABBABBABBABBABBABBABBABBABBABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCABCABCABCABCABCABCABCABCABCABCABC
ABCABCABCABCABCABCABCABCABCABCABCABCABCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCB
CBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBCAACAACAACAACAACAACAACAACAACAACAA
CAACAACAACAACAACAACAACAACAACAACAACAACAACAACABCABCABCABCABCABCABCABCABCABCA
BCABCABCABCABCABCABCABCABCABCABCABCABCABCABCBACBACBACBACBACBACBACBACBACBAC
BACBACBACBACBACBACBACBACBACBACBACBACBACBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCB
CBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCBBCCACCACCACCACCACCACCACCACC
ACCACCACCACCACCACCACCACCACCACCACCACCACCACCACCACCBCCBCCBCCBCCBCCBCCBCCBCCBC
CBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCBCCB
```

Figure 7.13: The sequence $\Upsilon S(3)$.

Those generators which are considered secure (for example the BBS generator [BBS86]) are in fact derivatives of existing cryptosystems and have high computational complexity, or require a lengthy initialization and therefore detract from our overall goal of having a simpler communication system.

The actual forgetting process involves choosing a leaf in the trie at random and deleting that leaf from the trie. Depending on the order, this might happen several times for each symbol encoded or decoded, but usually only one or two deletions will be necessary.

The performance of PPM augmented with a random forgetting mechanism does not significantly degrade compression.

Forgetting confounds the chosen plaintext attack because that attack relies on all contexts existing in the trie. In particular the purpose of the sequence Υ was to ensure that this was the case. But when forgetting is used, certain contexts will be pruned in an unpredictable manner and the estimate $\hat{M}$ will not be close enough to M for these prunings to be traced.

The security of PPM used in this manner is still an open question.

7.4.2 Discussion

The chosen plaintext attack just discussed illustrates a potential weakness of adaptive models in general, but it is not a very practical attack. It is not clear whether there are any better attacks when the key is the unknown state of the model. It would seem that part of this security comes from the difficulty of inverting arithmetic coding in this general

situation. But even if the arithmetic coding were invertible, it remains unclear how the probabilities passed to the arithmetic coder could be helpful in determining the unknown model. Further, such attacks cannot be applied usefully to PPM models with unbounded context lengths.

7.5 Primed PPM

The compression achieved by a PPM model can be enhanced by priming the model [TC96]. For English this typically means training the model on a representative sample of text. Intuitively, such training should increase the ciphertext only resistance of PPM because the model will be closely matched to the source from the outset. But what influence does this have on the stronger known and chosen plaintext attacks?

Assume a PPM model has been initialized using a very large static model, known to everyone. In particular, assume the model is known to the cryptanalyst.

A possible key in such a system is the starting position in the tree. This is equivalent to prepending a small piece of text to the front of the message transmitted. However, even with such a large tree, an exhaustive search would remain a feasible option. That is, a cryptanalyst could try decoding intercepted text from each possible starting position. The only possible problem for the cryptanalyst is that the accuracy of PPM models could cause several decryptions of short messages to form valid sentences in English. Thus, this key mechanism would actually make PPM vulnerable to a ciphertext only attack.

The question is more interesting if a longer piece of text is prepended to the message. In this case the model will no longer be exactly the same as the public model. However, it will not be significantly different and an exhaustive search might still suffice. In particular, unless a message happens to start in a part of the tree where adaptation has occurred, the first part of the message will be readable. When continued decoding leads to garbage output the cryptanalyst can identify where the tree has been modified and instigate an exhaustive search over likely modifications.

Simple variations on this idea, like blending a larger static model with a local model, are also vulnerable to an exhaustive search.

All of these cases highlight how a seemingly secure algorithm can be made totally insecure with tinkering. A similar situation occurs when the S-boxes of DES are altered. Almost all such variations lead to a weaker system [BS93].

7.6 Conclusions

PPM models are currently the best computer models of English and they now rival the predictive ability of human experts [TC96]. Since ciphertext only attacks rely on statistical leverage, it seems such an attack is unlikely against PPM unless better models of English

can be developed. This view is further reinforced by the results of Chapter 4 which indicate the amount of redundancy remaining after compression by PPM is comparable to that of the DES.

Currently the best attacks against standard PPM are lengthy chosen plaintexts, which drive a PPM model into a known state by taking advantage of the count scaling mechanism. The length of the attack is exponential in the order of the model, polynomial in the maximum count and the alphabet size. This makes the attack impractical for all but the smallest PPM models. For instance, an order-3 model for twenty-seven symbol English using 32-bit arithmetic coding requires a chosen plaintext of length 2.164×10^{16} symbols. Therefore, it would appear that the security of PPM is easily improved by increasing the order of the model and the width of the arithmetic coding registers. Further, the security of PPM benefits from the difficulty of inverting arithmetic coding in this general situation. However, even if it were possible to invert arithmetic coding here, it is not clear how the resulting information could be used in an attack.

The most interesting unresolved case is that of PPM* [CTW95]. Since PPM* is currently the best computer model of English [Bun97], it is reasonable to suppose that there is no easy ciphertext only attack against PPM*. Arguments to support this claim are given in Chapter 9. Further, it is, in general, impossible to force a randomly initialized PPM* model into a given state using any sequence. In particular, a PPM* model always adds one new node to the model for each symbol transmitted, and the counts on other nodes may also be affected. However, the attack of Section 7.4 requires that the model has at most a fixed number of nodes; and so clearly it cannot be applied to PPM*.

There remains the possibility that there may be some other method of inferring the model. It would seem that any such method must require access to the result of encoding the chosen plaintext. This in turn means that it will be necessary to invert arithmetic coding in the general adaptive case. To date there is no known way of doing this. Therefore, at this time, we know of no way to break a PPM* model using 32-bit arithmetic coding with the key consisting of an unknown initial model.

We have shown how PPM models can be used to allow automatic solution of simple substitution ciphers. In the next chapter this powerful approach is extended, and PPM models are used to break Ziv-Lempel coding. It would be interesting to investigate what other cryptosystems can be attacked in this manner.

Chapter 8

Cryptanalysis of Ziv-Lempel Schemes

The Ziv-Lempel compressors [ZL77, ZL78] are widely used. They are the basis of a large number of archiving tools and other compression programs. Two particular examples are the Stacker program [WGI91] and the UNIX `compress` program. While Ziv-Lempel approaches do not compress as well as PPM, they are easily implemented and offer extremely rapid decompression. They are particularly suited to applications where a file is to be compressed once but decompressed many times.

Ziv-Lempel compressors partition the plaintext into a series of ***phrases*** called the ***dictionary***. Compression is achieved by storing pointers to phrases rather than the phrases themselves. Exactly how the plaintext is partitioned into phrases and which phrases are eligible for pointer references depends on the exact variant used as discussed in Chapter 3.

In fact there are two main styles of Ziv-Lempel coding, derived from two distinct proposals [ZL77, ZL78]. In this chapter cryptanalyses of the LZR [RPE81] and LZ78 [ZL78] variants of Ziv-Lempel coding are given. Since LZR is based on the earlier LZ77 [ZL77] method, these two essentially cover both styles of Ziv-Lempel coding and it is reasonable to assume that the attacks presented can be extended to cover other variants.

In each case, both known plaintext and ciphertext only attacks are given. Section 8.2 contains the cryptanalysis of LZR and Section 8.3 the corresponding cryptanalysis for LZ78. Section 8.1 makes some general comments on using these compressors as cryptosystems.

8.1 Using Ziv-Lempel Compressors as Cryptosystems

Before a Ziv-Lempel compressor can be used as a cryptosystem it is necessary to introduce a key. In light of the desiderata given in Chapter 3, one simple approach is to use the initial dictionary as the key. The greatest security is then afforded when the statistics of

the key closely resemble those of intended messages. In this situation the message text is likely to have long matches in the key text maximizing the uncertainty in the ciphertext only attack described below, but making a known plaintext attack easier. If the entropy of the source is s bits/symbol, then a key of l symbols offers an effective key size of sl bits. For example, the entropy of English is about 1.2 bits/symbol and so 394 symbols (about 88 English words) are required to achieve an effective key size of 512 bits.

There is a subtle danger here. If the cryptanalyst is aware of the source (e.g. English, or maybe English used in commerce) then this information modifies the prior probabilities of the keys. The more detailed the cryptanalyst's prior information the smaller the class of probable keys. In the extreme it may become feasible to attempt an exhaustive search over a particular class of key.

The alternative is to initialize the dictionary to be a random sequence. This maximizes the prior uncertainty of the key. However, the key material will be used infrequently in the construction of phrases because it is not closely matched to the source being encoded. This means the cryptanalyst may only determine a few bits of the key, but a redundant source will be easily decoded because there will only be a few (and short) references into the key

Another simple approach is to exclusive-or the first segment of output with a random key, relying on the complexity of the Ziv-Lempel model to make it impossible to decode the remainder of the message. That is, we assume that it is difficult to begin decoding a Ziv-Lempel message part way into the output stream.

A third approach is to remove the first segment of the compressed output and transmit it encrypted with a public-key cryptosystem such as RSA [RSA78], and then transmit the remainder of the compressed output without modification. Because RSA runs rather slowly, it is desirable that the segment size be keep small. This approach allows the rapid transmission of long messages between two parties who have not previously communicated and without the need to generate a session key as needed in other similar protocols [Sch94]. Another advantage of this approach is the key depends on both the parties communicating and the message itself.

All three of these approaches are shown to be insecure, especially when used with redundant sources. But such sources are precisely the situation where compression is most beneficial. Therefore, we conclude that the Ziv-Lempel compression methods should not be used as security devices. It is shown that it is possible to decompress Ziv-Lempel output when an initial portion of the compressed output has been removed. In some cases it is even possible to reconstruct the contents of the removed portion. All three of the approaches outlined above can be considered as forms of this model. From now on the removed portion will be considered synonymous with the key and the initial dictionary.

There is another danger here which is not directly addressed in our cryptanalysis. In particular, the first part of the plaintext tends to compress only slightly because the compression model has not yet gathered sufficient information. This problem can be

reduced by priming the compressor with a large publicly available model. However, such models must be assumed known to the cryptanalyst and do not in themselves enhance security.

8.2 Cryptanalysis of LZR

A brief description of the LZR algorithm is given; explaining in addition how it differs from the original proposal LZ77. A trivial known plaintext attack is presented followed by a ciphertext only attack applicable to redundant sources. The attacks indicate that LZR and LZ77 should not be used as security devices.

8.2.1 Description of LZR

In the LZ77 form of Ziv-Lempel coding [ZL77], pointers denote phrases in a fixed-size window that precedes the coding position. The content of the window constitutes phrases of the dictionary. Matches may overlap with the text, although in the analysis this possibility is ignored. The longest match (up to some maximum length) is coded as a triple (p, l, s) where p is the offset to the longest match from the start of the window, l is the length of the match in symbols, and s is the first symbol which failed to match. The window size is typically about eight kilobytes. The pointer p is 0 if there is no match in the window, in which case the length l is obviously not required.

The LZR approach is the same as LZ77 except pointers denote any position in the already encoded text. Hence the window grows linearly as more and more text is compressed. The pointer p and length l are encoded with a variable length code, since they can grow to arbitrary sizes. The LZR method tends to be slow during compression since the search time and memory requirements increase as more and more text is processed. Like all Ziv-Lempel approaches, it remains fast for decompression.

For example, LZR encodes the message ♠♣♣♣♣◇♣◇♣♡ drawn from the alphabet $\{♠, ♣, ◇, ♡\}$ using the triples

$$(0, 0, ♠), (0, 0, ♣), (2, 3, ◇), (5, 3, ♡).$$

Let $[\![\cdot, \cdot]\!]$ be some standard bijective pairing function. In practice the triples must be further encoded using a bijective pairing function $(p, l, s) = [\![p, l, s]\!] = [\![[\![p, l]\!], r]\!]$ or other form of self-delimiting code. Entire triples must also be self delimiting, and the entire message is encoded as $[\![[\![p_1, l_1, s_1]\!], [\![p_2, l_2, s_2]\!] \cdots]\!]$.

However, this difficulty of coding the triples does little to enhance the security of the overall system because the decoder must be able to unambiguously interpret the output. Thus, during the cryptanalysis we work directly with the triples (p_i, l_i, s_i) and assume the cryptanalyst has already inverted the output coding. A slight difficulty arises if the front of the compressed file is removed, since in this situation the file might not start on a triple

boundary. However, it is feasible to search over various offsets into the compressed output in an attempt to identify a triple boundary.

8.2.2 Known Plaintext Attack

Consider the use of LZR as a cryptosystem where the key consists of extra text prepended to the plaintext or equivalently an initialized window. This arrangement is trivially broken by a known plaintext attack.

The central observation made in the attack is that each triple conveys information about the contents of the window. This information can be used in conjunction with a known plaintext to reconstruct unknown portions of the window. Assume that the length of the initial window (the length of the key) is known to the cryptanalyst.

Even without knowledge of the key length the attack described is still possible; although additional computation may be incurred. For once the index of any symbol in the known plaintext is identified, the length of the key can be immediately determined. For example, when the plaintext contains repeats like ...quuxz...quuxy..., the repetition will be encoded by a triple $(i, 4, \mathtt{y})$ from which the index of the first q is i.

To see how the triples convey the necessary information, let ♠♣♣♣♣♢♣♢♣♡ be a secret prepended text of length 10, and let the plaintext message be ♡♡♢♣♢♡♡♣♠♠. Then the cryptanalyst will observe the output triples

$$(10, 2, \diamondsuit), (5, 2, \heartsuit), (10, 1, \clubsuit), (1, 1, \spadesuit),$$

from which symbols 1, 5, 6, and 10, of the key are immediately determined. To see how this is possible consider the first triple $(10, 2, \diamondsuit)$ which must match the first two symbols of the plaintext. Thus, symbol 10 is ♡, and symbol 11 is ♡. But, the key was only of length 10, hence symbol 11 actually corresponds to the first ♡ of the plaintext. This match is therefore one of those unusual cases where the replacement text is determined partially by the replacement text itself. Some information about subsequent symbols is also obtained since for each triple we learn one symbol that it cannot be; for binary alphabets this makes the attack much easier. Thus, from the first triple we learn that symbol 12 is not ♢ (otherwise the match would have been longer). In fact, in this case symbol 12 is ♡, since it is actually part of the plaintext. The example plaintext does not contain sufficient information to determine any other symbols of the key.

While the mechanics of this attack are obvious, it is not at all apparent how long the known plaintext needs to be in order to recover a useful percentage of the key. This is dependent on the kind of key chosen. As the plaintext gets longer a greater percentage of the matches will occur in the plaintext itself, rather than the key. This fact is exploited in the ciphertext only attack discussed below, but it hinders reconstruction of the key in the known plaintext attack.

Some keys are very hard to completely determine. For instance, consider the key sequence JJ consisting of the sequence J repeated twice. Since the Ziv-Lempel compressors return only the index of the first match, it is no easy matter to gain any information about the second J. Indeed the only way in which anything can be learnt about it is if the plaintext itself contains the sequence J. However, in practice, lack of knowledge about the second J would not be a hinderance to cryptanalysis, since references to it will not normally occur in intercepted ciphertext.

8.2.3 Expected Length of Matches

In order to get some idea of the expected proportion of the key determined with a known plaintext, the expected length of matches for a uniform random source is calculated.[1] Intuitively, the expected match length for a redundant source will be at least that of the random source. The probability distribution for a match of length k is given. This distribution is used to estimate the expected length of the longest match, L.

Let two sequences $A = a_0 a_1 \cdots a_m$ and $B = b_0 b_1 \cdots$ be drawn randomly from the same alphabet $S = \{0, 1, \ldots, n-1\}$ of n symbols. The sequence A is finite and the sequence B is finite or unbounded. The longest contiguous match of the sequence $b_0 b_1 \cdots$ is to be found anywhere in the sequence A. In terms of the compression model, A is the current contents of the window and B is the text to be compressed.

Let X_k denote the number of matches of length k in the sequence A. The probability of two uniform random sequences of length k matching is n^{-k} and the number of possible positions for a match is $m - k + 1$. Thus, the expected number of matches of length k is

$$E(X_k) = (m - k + 1)n^{-k}.$$

By examining the situation where $E(X_k) = 1$, a rough estimate, $k \approx \log_n m$, is obtained for the expected length of the longest match. It is easy to see why this estimate is reasonable. For each length k, there are n^k possible sequences of that length and we would expect the longest match to be roughly independent of the sequence. Therefore, for A of length m, we have the constraint $n^k \approx m$, from which $k \approx \log_n m$.

To prove that the expected length of the longest match, L, is within a constant of $\log_n m$ it is shown that

$$\Pr(L \geq \log_n m + \delta(m)) \to 0 \text{ if } \delta(m) \to \infty,$$

and

$$\Pr(L \geq \log_n m + \delta(m)) \to 1 \text{ if } \delta(m) \to -\infty.$$

The first bound is apparent by substituting $\log_n m + \delta(m)$ for k in $E(X_k)$:

$$E(X_k) = \frac{1}{m}(m - \lg m + \delta(m) + 1)n^{-\delta(m)}$$

[1] I thank Dr Brendan McKay for showing me how to obtain the results of this section.

$$\begin{aligned} &\to 0 && \text{as } \delta(m) \to \infty \\ &\to \infty && \text{as } \delta(m) \to -\infty \end{aligned}$$

To show the second bound is more difficult. We proceed by calculating the second moment

$$E(X_k^2) = E(X_k(X_k - 1) + X_k) = E(X_k(X_k - 1)) + E(X_k).$$

The quantity $X_k(X_k - 1)$ is the number of ways of choosing two distinct but possibly overlapping matches of length k in A. The two matches are considered to be ordered. This can be done in $(m - k + 1)(m - k)$ ways. The probability of each match is simply n^{-2k} since there are always exactly $2k$ independent pairs of symbols which must match. This calculation is expedited because this probability is independent of the amount of overlap (if any) between the two matches. Thus,

$$E(X_k(X_k - 1)) = (m - k + 1)(m - k)n^{-2k}$$

and

$$E(X_k^2) = (m - k + 1)(m - k + n^k)n^{-2k}.$$

Now the variance can be found to be

$$\begin{aligned} \text{Var}(X_k) &= E(X_k^2) - E(X_k)^2 \\ &= (m - k + n^k)(m - k + 1)n^{-2k} - (m - k + 1)^2 n^{-2k} \\ &= (m - k + 1)(n^{-k} - n^{-2k}). \end{aligned}$$

Then,

$$\begin{aligned} \frac{\text{Var}(X_k)}{E(X_k^2)} &= \frac{(m - k + 1)(n^k - 1)n^{-2k}}{(m - k + 1)(m - k + n^k)n^{-2k}} \\ &= \frac{n^k - 1}{m - k + n^k} \\ &= \frac{m + n^{\delta(m)} - 1}{2m - \log_n m - \delta(m) + n^{\delta(m)}} \\ &\to 0 \qquad \text{as } \delta(m) \to -\infty. \end{aligned}$$

Putting it together, $E(X_k) \to \infty$ and $\text{Var}(X_k)/E(X_k^2) \to 0$ as $\delta(m) \to -\infty$. Intuitively this indicates the distribution of X_k is closer to $E(X_k)$ in units of $\text{Var}(X_k)$ as $\delta(m)$ becomes large and negative. Since $\text{Var}(X_k) = E(X_k^2) - E(X_k)^2$ we have

$$\frac{\text{Var}(X_k)}{E(X_k^2)} = 1 - \frac{E(X_k)^2}{E(X_k^2)}$$

and since $\text{Var}(X_k)/E(X_k^2) \to 0$ this implies $E(X_k)^2/E(X_k^2) \to 1$, thus $E(X_k^2) \approx E(X_k)^2$ and

$$\frac{\text{Var}(X_k)}{E(X_k)^2} \to 0 \quad \text{or} \quad \frac{\sqrt{\text{Var}(X_k)}}{E(X_k)} \to 0.$$

Applying Chebyshev's inequality [KS77],

$$\Pr(X \geq \epsilon) \leq \frac{E(X)}{\epsilon}.$$

With $\epsilon = \sqrt{\mathrm{Var}(X_k)}/E(X_k)$ this gives

$$\Pr\left(X_k < \frac{\sqrt{\mathrm{Var}(X_k)}}{E(X_k)}\right) \geq 1 - \frac{E(X_k)^2}{\sqrt{\mathrm{Var}(X_k)}}.$$

Thus

$$\Pr(X_k < 0) \to 0 \qquad \text{as } \delta(m) \to -\infty$$

and consequently

$$\Pr(X_k > 0) \to 1 \qquad \text{as } \delta(m) \to -\infty.$$

Finally,

$$\Pr(L \geq \log_n m + \delta(m)) \to 1 \qquad \text{if } \delta(m) \to -\infty,$$

and the second bound holds. This shows the longest match is within a constant of $\log_n m$.

It is now shown that the contribution of overlapping matches is negligible compared to the nonoverlapping matches for all moments. This will be used to show that X_k has the Poisson distribution. Let

$$(\alpha)_t = \alpha(\alpha - 1) \cdots (\alpha - t + 1)$$

denote the falling factorial of α and let X_k denote the number of matches of length $k = \log_n m + \delta$ as before. In calculating $E((X_k)_t)$ the nonoverlapping and overlapping cases are considered separately. The contribution of nonoverlapping matches is determined as follows. Place t blocks of length k each in a sequence of length m. This can be done in $\binom{m-tk+t}{m-tk}$ ways ignoring the order of the blocks (this problem is equivalent to the number of positive integer solutions of $x_1 + \cdots + x_t + x_{t+1} = m - tk$). There are $t!$ orderings of the blocks. Thus

$$\begin{aligned} t!\binom{m-tk+t}{m-tk} n^{-kt} &= (m-tk+t)(m-tk+t-1)\cdots(m-tk)n^{-kt} \\ &= (m - O(k))^t n^{-kt} \\ &= m^t n^{-kt}(1 + O(1/m)) \end{aligned}$$

where the third equality follows since both k and t are constants. Since the matches are nonoverlapping they are independent, each with probability n^{-k}. Thus the total contribution to $E((X_k)_t)$ by nonoverlapping matches is

$$m^t n^{-kt}(1 + O(1/m)),$$

or in terms of δ,

$$n^{-\delta t}(1 + O(1/m)).$$

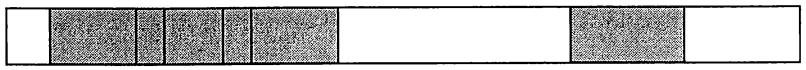


Figure 8.1: A situation with four matches but only two components.

Consider now the contribution to $E((X_k)_t)$ by overlapping matches. Let a ***component*** be a contiguous block of matches. Figure 8.1 shows a case where $t = 4$ but with only two components. Let c denote the number of components, then $1 < c < t$. The case $c = t$ corresponds to the nonoverlapping case just discussed. Each component can be placed in at most $m - O(k)$ positions, and there is at most $k^t t!$ different ways of forming the components. Thus the number of positions is at most

$$(m - O(k))^c k^t t! \leq m^c k^t t!.$$

Matching the first component containing at least two blocks has probability at least n^{-2k}. The probability for placing each of the remaining $c - 1$ components is at least n^{-k} each. Thus the total probability is $n^{-2k} n^{-(c-1)k} = n^{-(c+1)k}$, so the total contribution of overlapping matches with c components is at most

$$n^{-(c+1)k} m^c k^t t!,$$

or, in terms of δ,

$$\frac{1}{m} n^{-(c+1)\delta} (\log_n m + \delta)^t t!.$$

Summing over all possible numbers of components gives

$$\begin{aligned}
\sum_{c=1}^{t-1} \frac{1}{m} n^{-(c+1)\delta} (\log_n m + \delta)^t t! &= \frac{1}{m} (\log_n m + \delta)^t t! \sum_{c=1}^{t-1} n^{-(c+1)\delta} \\
&= \frac{1}{m} (\log_n m + \delta)^t t! \frac{n^{-\delta}(n^{-\delta t} - n^{-\delta})}{n^{-\delta} - 1} \\
&= O\left(\frac{(\log_n m + \delta)^t}{m} \right).
\end{aligned}$$

For fixed t, this tends to zero as m increases and its contribution is therefore negligible compared to the nonoverlapping cases. Consequently,

$$E((X_k)_t) \to n^{-\delta t} \text{ as } m \to \infty \text{ for each fixed } t.$$

The convergence of these factorial moments is sufficient to guarantee that X_k has the Poisson distribution with mean $n^{-\delta}$ [KS77]:

$$\Pr(X_k = r) = e^{-n^{-\delta}} n^{-\delta r} / r!.$$

For example, the probability that there are no matches at all is

$$\Pr(X_k = 0) = e^{-n^{-\delta}}.$$

Although the calculation was strictly for a fixed value of δ it can be reconciled with intuition. Namely, if $\delta \to \infty$ then $\Pr(X_k = 0) \to 0$ and if $\delta \to -\infty$ then $\Pr(X_k = 0) \to 1$.

Letting L again denote the longest match

$$\Pr(L \geq k) = \Pr(X_k > 0) \to 1 - e^{-2^{-\delta}},$$

and thus

$$\Pr(L = k) = \Pr(L \geq k) - \Pr(L \geq k+1) \to e^{-2^{-\delta-1}} - e^{-2^{-\delta}}.$$

8.2.4 Multiple Plaintexts

The known plaintext attack can now be studied in more detail. Given N plaintexts and corresponding ciphertexts, encoded using the same key, the expected proportion of key text determined is investigated. Only the first match of each plaintext is considered. In reality we would expect to do better than this because many matches from each plaintext will determine bits of the key.

The case $k = 1$ where only one symbol of the key is matched per plaintext is an important limiting situation. This leads to a reasonable lower bound, since it is logical to suppose that every message would depend on at least one key bit. This is guaranteed to be the case if the key contains at least one occurrence of each symbol of the alphabet. This situation is analyzed by considering $p(N, \beta)$, the probability that N randomly selected plaintexts determine β bits of the key. Let m denote the length of the key. For large m (in particular $m \gg N$) we expect

$$p(N, \beta) \approx \begin{cases} 1 & \text{if } N = \beta \\ 0 & \text{if } N \neq \beta \end{cases}, \qquad m \gg N.$$

But it is the cases where m is small ($m \leq 1024$ say) that are relevant to the cryptanalysis. Clearly, we will still have $p(1,1) = 1$. But $p(N,N) < 1$ for $N > 1$ because of the possibility that some plaintexts determine the same symbols of the key.

In general, the recurrence

$$p(N, \beta) = \frac{\beta}{m} p(N-1, \beta) + \frac{m - \beta + 1}{m} p(N-1, \beta - 1) \tag{8.1}$$

holds, which arises by considering the two ways in which the $N-1$ case can be extended to the N case. Namely, by choosing a case where β symbols are already determined and then selecting one of those symbols or by choosing a case with $\beta - 1$ symbols determined and then selecting one of the remaining $m - \beta + 1$ symbols. Then $p(N, \beta)$ will be a proper (normalized) probability distribution in β for each positive integer N. Further, $p(0, \beta) = 0$ and $p(N, 0) = 0$ are additional boundary conditions.

Examining the recurrence, notice a factor of $1/m$ is produced on each downward iteration of N, hence this term can be removed from the recurrence by defining $f(N, \beta) = m^{N-1} p(N, \beta)$ to obtain a simpler recurrence

$$f(N, \beta) = \beta f(N-1, \beta) + (m - \beta + 1) f(N-1, \beta - 1)$$

with $f(1,1) = 1$. From this recurrence a structure resembling Pascal's triangle can be constructed (where $(\mu)_t$ is used to denote $(m-1)(m-2)\cdots(m-t)$):

$N=1$					1					
$N=2$				1		$(\mu)_1$				
$N=3$			1		$3(\mu)_1$		$(\mu)_2$			
$N=4$		1		$7(\mu)_1$		$6(\mu)_2$		$(\mu)_3$		
$N=5$	1		$15(\mu)_1$		$25(\mu)_2$		$10(\mu)_3$		$(\mu)_4$	
$N=6$	1	$31(\mu)_1$		$90(\mu)_2$		$65(\mu)_3$		$15(\mu)_4$		$(\mu)_5$

The coefficients in front of the μ terms are Stirling numbers of the second kind. Hence

$$f(N,\beta) = \left\{ N \atop \beta \right\} \frac{(m-1)!}{(m-\beta)!},$$

so that

$$p(N,\beta) = \frac{1}{m^{N-1}} \left\{ N \atop \beta \right\} \frac{(m-1)!}{(m-\beta)!} = \frac{m!}{m^N} \left\{ N \atop \beta \right\} \frac{1}{(m-\beta)!}.$$

Now the expected number of symbols determined by the N plaintexts when only the first symbol of the first match is considered is

$$\begin{aligned}
E(N) &= \sum_{\beta=1}^{N} \beta \frac{m!}{m^N} \left\{ N \atop \beta \right\} \frac{1}{(m-\beta)!} \\
&= \frac{1}{m^N} \sum_{\beta=1}^{N} \beta \binom{m}{\beta} \left\{ N \atop \beta \right\} \beta! \\
&= \frac{1}{m^N} \sum_{\beta=1}^{N} \binom{m}{\beta} \left[\left\{ N+1 \atop \beta \right\} - \left\{ N \atop \beta-1 \right\} \right] \beta! \\
&= \frac{1}{m^N} \sum_{\beta=1}^{N} \binom{m}{\beta} \left\{ N+1 \atop \beta \right\} \beta! - \frac{m}{m^N} \sum_{\beta=1}^{N} \binom{m-1}{\beta-1} \left\{ N \atop \beta-1 \right\} (\beta-1)! \\
&= \frac{m^N}{m^{N-1}} - \frac{(m-1)^N}{m^{N-1}} \\
&= m(1-(1-1/m)^N).
\end{aligned}$$

From this result the correctness of the limiting cases is apparent; for if $m \to \infty$, then $E(N) \to N$; and if $N \to \infty$, then $E(N) \to m$, as intuition suggests they must. Thus for $k = 1$ the proportion of the m symbol buffer determined is given by $1-(1-1/m)^N$. This function is graphed in Figure 8.2 for various sizes of key. To a crude approximation m ciphertexts are sufficient to determine half the symbols of a key of size m. Notice, however, that determining half the symbols of key for a redundant source means that nearly all the message is readable because the first symbols determined are likely to be those that occur most often in actual messages.

Unfortunately, there does not appear to be any easy way of extending this analysis to longer matches, $k > 1$. In particular, it is difficult to generalize the recurrence,

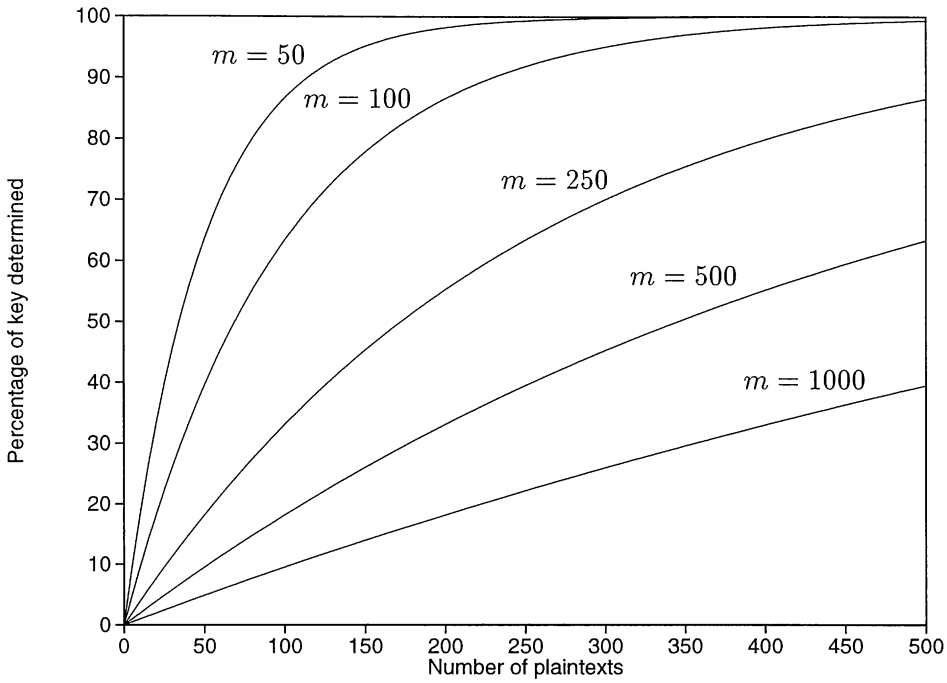


Figure 8.2: The function $E(N) = 1 - (1 - 1/m)^N$.

Equation 8.1, because of overlapping effects. The ideal result would be a general formula applicable for any N and k, but it does not seem such a result will be possible with this approach. However, by fixing the number N of plaintexts, it is possible to use combinatorial arguments to calculate the expected value for small N and arbitrary k.

Let $Q = m - k + 1$. Consider the case where there is exactly $N = 2$ plaintexts and where the match lengths are some constant k. This includes the expected case where $k = \log_n m$. The calculation is split into three parts depending on the positioning of the two matches:

- *Non-overlapping cases.* Say the first block occurs with offset i from the start of the window; then the number of non-overlapping and subsequent positions for the second block is $m - 2k - i + 1$. In addition, there are two ways of selecting the first block. Hence the number of non-overlapping combinations is given by

$$\begin{aligned}
\alpha_1 &= 2 \sum_{i=0}^{m-2k} (m - 2k + i + 1) \\
&= 2(m - 2k + 1)(m - 2k + 1) - 2 \sum_{i=0}^{m-2k} i \\
&= 2(m - 2k + 1)^2 - (m - 2k)(m - 2k + 1) \\
&= (m - 2k + 1)(m - 2k + 2)
\end{aligned}$$

- *Partially overlapping cases.* Let the two blocks overlap by $0 < q < k$. Then the number of positions in which to place this block, of total length $2k-q$, is $m-2k+q+1$.

Further, there are two ways to pick the first component of the block. Hence:

$$\begin{aligned}\alpha_2 &= 2\sum_{q=1}^{k-1}(m-2k+q+1) \\ &= 2(m-2k+1)(k-1)+2\sum_{q=1}^{k-1}q \\ &= 2(k-1)(m-2k+1)+k(k-1) \\ &= (k-1)(2m-3k+2).\end{aligned}$$

- *Fully overlapping cases.* When the blocks are fully coincident ($q = k$), it does not matter in which order the blocks are chosen. This can occur in

$$\alpha_3 = m-k+1$$

distinct ways.

Summing the contributions α_1, α_2, and α_3 gives $\alpha = \alpha_1 + \alpha_2 + \alpha_3 = Q^2$ which is the total number of ways of placing the two blocks.

These calculations help to determine the expected number of symbols of keytext determined by two plaintexts:

$$E(2) = \frac{1}{\alpha}\sum_{q=0}^{k}(2k-q)f(q)$$

where $f(q)$ is the number of ways in which the blocks can overlap by q symbols. Hence:

$$\begin{aligned}E(2) &= \frac{1}{\alpha}\left[(m-2k+1)(m-2k+2)k+2Qk+2\sum_{q=1}^{k-1}(m-2k+q+1)(2k-q)\right] \\ &= \frac{1}{\alpha}\left[(m-2k+1)(m-2k+2)2k+Qk+(k-1)(9m-14k+10)\frac{k}{3}\right] \\ &= \frac{1}{3}\frac{k}{Q^2}[6m^2-15mk+12m+10k^2-15k+5] \\ &= \frac{1}{3}\frac{k}{Q^2}[5(m-2k+1)(m-k+1)+m(m+2)]\end{aligned}$$

This result is consistent with our result for the case where $k = 1$.

The analysis of the $N = 3$ case is more difficult than the $N = 2$ case because the number of ways the blocks can be placed is greatly increased. The total number of combinations is trivially $\alpha = Q^3$ but determining the number of cases with each possible overlap $0 \leq q \leq 2k$ is nontrivial. The set of all solutions is divided according to the number of components in a placement, and then finer divisions as shown in Figure 8.3.

In each case, the binomial coefficient arises from considering the number of ways of placing a given matching in the buffer. The other term counts relevant arrangements of

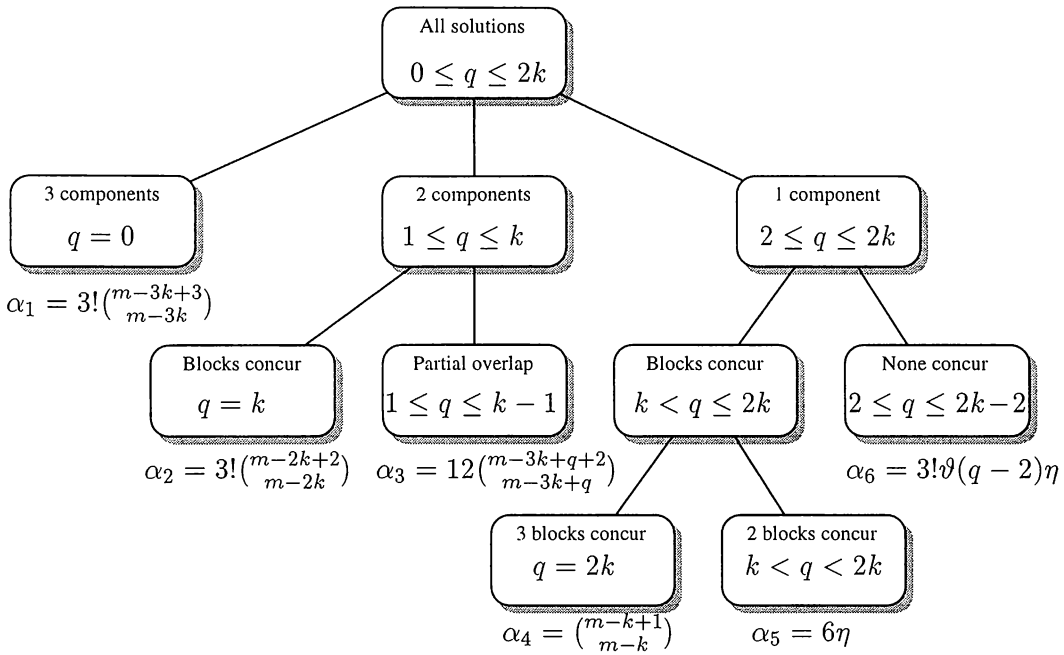


Figure 8.3: Solution tree for the $N = 3$ case, where $\eta = \binom{m-3k+q+1}{m-3k+q}$.

blocks within the match and other symmetries. The only difficult case is α_6, where $\vartheta(q)$ is the coefficient of x^q in $(1 + x + x^2 + \cdots + x^{k-2})^2$ which is

$$\vartheta(q) = \begin{cases} q+1 & \text{if } q \le k-2 \\ 2k-q-3 & \text{if } q > k-2 \end{cases}$$

This coefficient arises by considering the generating function for the number of integer solutions to

$$x_1 + x_2 = q, \qquad 0 \le x_i \le k-2$$

which encapsulates the number of ways of generating an overlap of q with each part at most $k-1$.

It is an easy, but tedious, calculation to verify that every possible placement has been accounted for; that is, to check that

$$\alpha = \alpha_1 + \alpha_2 + \alpha_4 + \sum_{q=1}^{k-1} \alpha_3 + \sum_{q=k+1}^{2k-1} \alpha_5 + \sum_{q=2}^{2k-2} \alpha_6.$$

In verifying this sum the following identity is useful

$$\sum_{q=a}^{b} \binom{c+q+d}{c+q} = \frac{1}{d+1}\left[\binom{c+b+1+d}{c+b} - \binom{c+a+d}{c+a-1}\right].$$

We now proceed to calculate the expected number of symbols of keytext determined by three plaintexts as done in the $N = 2$ case:

$$E(3) \;=\; \frac{1}{\alpha}\sum_{q=0}^{2k}(3k-q)f(q)$$

$$= \frac{1}{\alpha}\left[3!(3k)\binom{m-3k+3}{m-3k} + 3!(2k)\binom{m-2k+2}{m-2k} + k\binom{m-k+1}{m-k} + \right.$$

$$+12\sum_{q=1}^{k-1}(3k-q)\binom{m-3k+q+2}{m-3k+q}$$

$$+3!\sum_{q=k+1}^{2k-1}(3k-q)\binom{m-3k+q+1}{m-3k+q}$$

$$+3!\sum_{k=2}^{k}(3k-q)(q-1)\binom{m-3k+q+1}{m-3k+q}$$

$$\left.+3!\sum_{q=k+1}^{2k-2}(3k-q)(2k-q-1)\binom{m-3k+q+1}{m-3k+q}\right].$$

The four sums are respectively:

$$\begin{aligned}
S_1 &= 15k^2m^2 - 15km^2 - 76k^3m + 123k^2m - 47km + \frac{195}{2}k^4 - 219k^3 + \frac{309}{2}k^2 - 33k \\
S_2 &= 9k^2m - 9km - 14k^3 + 24k^2 - 10k \\
S_3 &= 7k^3m - 9k^2m + 2km - \frac{33}{2}k^4 + 33k^3 - \frac{39}{2}k^2 + 3k \\
S_4 &= 5k^3m - 15k^2m + 10km - \frac{17}{2}k^4 + 31k^3 - \frac{67}{2}k^2 + 11k
\end{aligned}$$

Using these results it is found, after some algebra, that

$$E(3) = \frac{1}{2}\frac{k}{Q^3}[6m^3 - 24km^2 + 18m^2 + 34k^2m - 48km + 16m - 17k^3 + 34k^2 - 21k + 4]$$

where the polynomial in the brackets has no factorization over the integers. By a more protracted and laborious calculation a similar result is obtained for the case of four plaintexts:

$$E(4) = \frac{1}{15}\frac{k}{Q^4}[60m^4 - 330km^3 + 240m^3 + 720k^2m^2 - 990km^2 + 330m^2 -$$

$$-735k^3m + 1440k^2m - 900km + 180m + 294k^4 - 735k^3 + 650k^2 - 240k + 31].$$

From the results for $N = 2, 3, 4$, it appears that if $k \ll \log_n m$, then

$$E(N) = O\left(\frac{kNm^N}{Q^N}\right)$$

since the m^N will dominate in each case. But then $Q^N \approx m^N$, so $E(N) = O(kN)$ and the expected number of key symbols determined by N plaintexts is at least linear in the number of plaintexts.

It would be nice to have a formula for any N and k so that the average case could be characterized. It seems feasible that such a formula might be obtained by decomposing the problem according to the number of components as done in the $N = 3$ and $N = 4$

cases. However, many of the symmetries are non-trivial to count and the lack of nice factorizations for the $N = 3$ and $N = 4$ cases would seem to preclude any simple closed form.

In the analysis, multiple plaintexts making single matches have been considered. But in practice the scenario is more likely to be a single plaintext making multiple matches. But these situations are roughly equivalent since independent uniform plaintext was assumed. So taking N as the number of matches into the key, the same analysis holds. However, the length of the plaintext is not so easy to determine because matches will also occur into the plaintext itself in rough proportion to the ratio of keytext to plaintext.

8.2.5 Ciphertext Only Attack

When encoding from a stationery ergodic source the rate of LZR is known to be asymptotically optimal [WZ94]. This suggests that LZR should be immune from ciphertext only attack. However, the convergence of LZR to the optimum is known to be slow (this is perhaps best illustrated by noting that other compressors, such as PPM, can easily outperform LZR). This lack of convergence was demonstrated in another way in Chapter 4 where tests of randomness revealed regularities in the compressed output. Indeed, those tests indicated that the amount of redundancy remaining exceeded the redundancy of the `crypt` program for which there are attacks [RW84]. This evidence suggests that Ziv-Lempel approaches may be susceptible to ciphertext only attack. Before such attacks could occur, a better model of English must be found. Such models already exist. A person who can read English probably has a good enough model. Alternatively, the models used by the PPM compressor are comparable. It is shown that either of these models is sufficient to break LZR in a ciphertext only attack.

Given a message M drawn from a source S, consider the problem of decoding the LZR encoding of M when the initial L symbols have been removed (that is, the first L symbols are taken as the key). The complication of starting between phrase boundaries is ignored.

Then it is possible to decode the given ciphertext and just mark unresolved symbols in some way. To do this the ciphertext is decoded in the normal manner. The only difficulty arises when a triple references symbols in the L unknown ones. Since initially these symbols are unknown to the cryptanalyst, we denote them as (i) where i is the position the symbol occurs, $1 \leq i \leq L$. When a reference occurs to multiple symbols, it can be expanded as a series of singletons. Thus the triple (10,3,`a`) can be decompressed as the sequence `(10)(11)(12)a`.

For a first example, Figure 8.4, the source will be English, the key has length $L = 64$, and the model is a human expert. By inspecting this output it is a relatively trivial matter to determine many of the missing symbols. Spaces or newlines are often the easiest symbols to identify. In particular, symbols occurring after punctuation or between apparent words are most likely to be spaces. In the example, symbols 3, 16, 21, 29, 33, 40, 48 are spaces.

(0)(1)(2)r(2)(3)w(58)(59)(60)(61)m(30)(31)y(3)p(6)t(1)s(44)(45)(46)a(19)␣(11)e(15)
(16)u(27)␣(18)n(19)o(44)(45)(46)o(13)(14)␣moun(19)a(18)ns(28)(29)(30)(31)(32)(33)
m(30)(31)y(3)p(6)s(13)(14)s(61)(62)v(58)(59)↩(45)(46)(47)m.(33)(34)u(19)␣m(12)
(13)t(61)(62)f(44)(45)(46)(47)(48)p(6)t(1)s(44)w(58)(59)(60)(61)c(1)(2)a(19)s(29)
(30)(31)(32)(33)d(2)c(2)p(19)i(62)(63)s(29)(30)(31)(32)(33)l(14)(15)(16)n(12)w(1)(2)
r(2)(3)o(5)␣(19)o↩(37)a(15)(16)e(31)(32)s;(29)(30)(31)(32)(33)mo(13)t(61)(62)f(44)
(45)(46)(47)(48)p(6)s(13)(14)s(61)w(58)(59)(60)(61)i(25)f(54)(55)t(14)(15)(16)(17)
(18)(19)(20)(21)e(53)i(11)␣(19)(20)i(25)gs(29)(30)(31)(32)(33)dr(2)adfu(11)↩(15)a
(25)ge(5)s.(33)T(46)(47)(48)(49)(50)(51)(52)(53)(54)(55)(56)a(42)(43)(44)(45)(46)
(47)(48)h(12)b(37)i(19),(48)he(11)p(14)(15)(16)by(3)t(1)(2)(3)w(18)s(2)(3)a(15)v
(18)c(60)(61)(62)f(3)E(11)r(62)(63)d(29)(30)(31)(32)↩(45)(46)(47)(48)kn(12)wl(14)
(15)g(2)(3)an(32)(33)memory(3)of(44)G(30)(31)(32)a(11)f(28)(29)t(12)o(8)(9)t(1)(2)
(3)r(18)g(1)t(3)roa(15)(16)to(44)(45)(46)e(3)r(18)g(1)t(3)p(6)s(13).

L(62)(63)g(48)(49)ays(21)(22)ft(14)r(44)(45)(46)(47)y(48)ha(15)(16)c(11)imb(14)(15)
(16)ou(19)␣of(44)(45)(46)(47)(48)va(11)l(47)y(48)a(31)(32)(33)l(14)f(19)␣t(1)(2)(3)L
(6)st(3)H(12)me(11)y↩Hous(60)(61)mi(11)es(16)be(20)i(25)d(28)(29)th(47)y(48)w
(58)(59)(60)(61)s(19)il(11)␣g(12)i(63)g(48)u(27)␣a(31)(32)(33)u(27)␣a(31)(32)(33)u
(27).(3)I(19)␣w(6)s␣(22)(23)h(51)(52)d(3)p(6)t(1)↩(30)(31)(32)(33)a(48)(49)ange(5)o
us␣p(6)t(1),(21)(22)(23)cr(62)o(8)e(15)(16)(17)ay(48)a(31)(32)(33)a␣(11)o(25)e(11)y
␣a(31)(32)(33)a␣(11)o(25)g.(33)N(12)w␣(45)(46)(47)y(48)coul(32)↩l(12)o(8)(9)b(6)(7)
(8)(9)ov(58)(59)␣(45)(46)(47)(48)l(30)(31)(32)s(44)(45)(46)(47)y(48)ha(15)(16)l(14)f
(19),(48)l(30)i(15)(16)ou(19)␣be(20)i(25)d␣(45)(46)(47)m␣fa(5)␣be(11)o(17).(3)F(51)
(52),↩fa(5)␣a(17)ay(48)i(25)␣(45)(46)(47)(48)W(54)(55)t,(16)(17)h(58)(59)(60)(61)t
(20)i(25)gs(29)w(58)(59)(60)(61)b(11)u(2)(3)an(32)(33)fa(18)nt(28)(29)B(35)(36)(37)
(38)␣knew␣(45)(46)(47)r(47)(48)l(30)y↩h(18)s␣(12)wn(48)coun(19)ry(3)of(44)s(22)fe
(29)(30)(31)(32)(33)c(12)mfo(5)t(6)b(11)e␣(19)(20)i(25)gs,(29)(30)(31)(32)(33)h(18)s
␣l(18)(19)t(11)e␣h(12)b(37)i(19)-(20)il(11).(3)He↩s(20)iv(58)(59)(60)d.(3)I(19)␣w(6)s
␣g(2)t(19)ing(48)b(18)(19)te(5)␣c(12)l(15)(16)u(27)␣h(58)(59)(60),(40)(41)(42)(43)
(44)(45)(46)(47)(48)wi(25)d␣c(6)m(60)(61)shr(18)l(11)␣among↩t(1)(2)(3)ro(7)(8)s.
(33)(34)oul(32)e(5)s,(29)t(12)o,(21)(22)t(44)(45)imes(28)(29)c(6)m(60)(61)ga(11)lo
(27)i(63)g(48)(49)own(48)t(1)(2)(3)moun(19)a(18)n-(13)id(2)s,(48)le(19)↩l(12)os(60)
(61)by(61)mid-day(48)sun␣u(27)o(25)␣(45)(46)(47)(48)sn(12)w,(29)(30)(31)(32)(33)p
(6)s(13)(14)d␣among␣(45)(46)(47)m␣(w(1)ic(1)␣w(6)s␣luc(8)y)(28)(29)o(59)↩ov(58)
(59)␣(45)(46)(47)i(5)␣h(2)ads␣(w(1)ic(1)␣w(6)s␣al(30)rming).(33)T(46)(47)(48)n(18)
g(1)ts(44)w(58)(59)(60)(61)comfo(5)tl(54)(55)s(29)(30)(31)(32)(33)chil(11),↩(30)(31)
(32)(33)t(46)(47)y(48)di(15)(16)n(12)t(48)(49)ar(60)(61)to(23)(24)i(63)g(48)o(5)␣
(19)a(11)k(29)t(12)o␣(11)(12)ud(28)(29)fo(5)␣(19)h(2)(3)ec(1)o(54)(55)(56)(57)(58)
(59)(60)(61)u(25)c(30)(31)ny,↩(30)(31)(32)(33)t(46)(47)␣(13)ile(31)c(60)(61)se(47)m
e(15)(16)to(44)d(18)sl(18)k(60)(61)being(48)bro(8)en---exc(2)p(19)␣by(3)t(1)(2)(3)n
(12)is(60)(61)(62)f(3)w(6)te(59)↩a(42)(43)(44)(45)(46)(47)(48)wai(11)␣of(3)wi(63)d
(29)(30)(31)(32)␣(45)(46)(47)(48)c(5)(6)(7)(8)(9)of(44)sto(25)e.↩

Figure 8.4: The decompression of a Ziv-Lempel file with the first 64 symbols removed. Numbers in parentheses represent unknown symbols to be determined by the cryptanalyst.

Good guesses can now be made for a variety of other symbols. In particular, the text `bro(8)en` suggests `broken`, `sn(12)w` suggests `snow`, `moun(19)a(18)n` suggests `mountain`, and so forth. Thus, determining the symbols is similar to ciphertext only attacks against homophonic simple substitution ciphers. Each symbol determined provides additional clues for the resolution of the remaining symbols.

After making the substitutions mentioned so far and `h` for 1, `a` for 6, and `a` for 30, the text in Figure 8.5 is obtained. It is then a relatively trivial matter to fill in the remaining symbols and obtain the plaintext given in Figure 8.6. Figure 8.7 lists the key as determined. Notice that the key symbols 4, 10, 26, and 39 must be guessed solely from the context of the surrounding key, since references to these symbols do not occur in the ciphertext.

When a random key is used it is impossible for a cryptanalyst to determine those symbols of the key not used in a given plaintext. However, the ciphertext is actually easier to decode since there are less matches into the key, and the matches when they do occur tend to be short. Figure 8.8 shows our example plaintext when encoded with a random key. When compared with Figure 8.4, it is immediately obvious that the cryptanalyst's task has been simplified.

It may happen that the ciphertext contains relatively few or, in an extreme case, no known symbols. The extreme case occurs when the message being transmitted is in fact the key, whereupon the compressed output consists of the sole triple $(0, l, \odot)$ where l is the length of the key and $\odot$ is the terminating symbol. In such a case, techniques similar to those used in breaking homophonic simple substitutions can be applied. By counting the number of times each unknown symbol occurs, a ranked list of unknowns may be produced. It is then likely that the most frequent unknowns correspond to spaces, `es`, and `ts`. The most common group of three $(a)(b)(c)$ is likely to be `the`, and so on. The chief difficulty is that any particular symbol may be coded by several different phrases.

8.2.6 Automated Cryptanalysis

The similarity of the cryptanalysis of the preceding section to that of simple substitution ciphers suggests the attack could be automated, in the manner of Section 7.2. Although the constraints are less restrictive (there can be several different unknowns for a single plaintext symbol), it was found that an order-3 PPM model could successfully be used to reconstruct the text. An order-4 model gave marginally better performance. For example with 200 unknowns and a ciphertext of length 5000 the order-3 model made 91 incorrect assignments whereas the order-4 model made 81; with a ciphertext of length 100 000 the order-3 model had 22 errors whereas the order-4 model had 18. Models larger than order-4 were not tried because of memory constraints.

The experiments were initially restricted to twenty-seven letter English. *The Moonstone* [Col82] was compressed using LZR with the window initialized with a key of length

(0)h(2)r(2)␣w(58)(59)(60)(61)ma(31)y␣paths(44)(45)(46)at␣(11)e(15)␣u(27)␣into(44)(45)
(46)o(13)(14)␣mountains(28)␣a(31)(32)␣ma(31)y␣pas(13)(14)s(61)(62)v(58)(59)↩(45)(46)
(47)m.␣(34)ut␣mo(13)t(61)(62)f(44)(45)(46)(47)␣paths(44)w(58)(59)(60)(61)ch(2)ats␣a
(31)(32)␣d(2)c(2)pti(62)(63)s␣a(31)(32)␣l(14)(15)␣nowh(2)r(2)␣o(5)␣to↩(37)a(15)␣e(31)
(32)s;␣a(31)(32)␣mo(13)t(61)(62)f(44)(45)(46)(47)␣pas(13)(14)s(61)w(58)(59)(60)(61)i
(25)f(54)(55)t(14)(15)␣(17)it(20)␣e(53)i(11)␣t(20)i(25)gs␣a(31)(32)␣dr(2)adfu(11)↩
(15)a(25)ge(5)s.␣T(46)(47)␣(49)(50)(51)(52)(53)(54)(55)(56)a(42)(43)(44)(45)(46)(47)␣
hob(37)it,␣he(11)p(14)(15)␣by␣th(2)␣wis(2)␣a(15)vic(60)(61)(62)f␣E(11)r(62)(63)d␣a(31)
(32)↩(45)(46)(47)␣knowl(14)(15)g(2)␣an(32)␣memory␣of(44)Ga(31)(32)a(11)f(28)␣took(9)t
h(2)␣right␣roa(15)␣to(44)(45)(46)e␣right␣pas(13).

L(62)(63)g␣(49)ays␣(22)ft(14)r(44)(45)(46)(47)y␣ha(15)␣c(11)imb(14)(15)␣out␣of(44)(45)
(46)(47)␣va(11)l(47)y␣a(31)(32)␣l(14)ft␣th(2)␣Last␣Home(11)y↩Hous(60)(61)mi(11)es␣be
(20)i(25)d(28)␣th(47)y␣w(58)(59)(60)(61)stil(11)␣goi(63)g␣u(27)␣a(31)(32)␣u(27)␣a(31)
(32)␣u(27).␣It␣was␣(22)(23)h(51)(52)d␣path↩a(31)(32)␣a␣(49)ange(5)ous␣path,␣(22)(23)
cr(62)oke(15)␣(17)ay␣a(31)(32)␣a␣(11)o(25)e(11)y␣a(31)(32)␣a␣(11)o(25)g.␣Now␣(45)(46)
(47)y␣coul(32)↩look(9)ba(7)k(9)ov(58)(59)␣(45)(46)(47)␣la(31)(32)s(44)(45)(46)(47)y␣h
a(15)␣l(14)ft,␣lai(15)␣out␣be(20)i(25)d␣(45)(46)(47)m␣fa(5)␣be(11)o(17).␣F(51)(52),↩
fa(5)␣a(17)ay␣i(25)␣(45)(46)(47)␣W(54)(55)t,␣(17)h(58)(59)(60)(61)t(20)i(25)gs␣w(58)
(59)(60)(61)b(11)u(2)␣an(32)␣faint(28)␣B(35)(36)(37)(38)␣knew␣(45)(46)(47)r(47)␣lay↩
his␣own␣country␣of(44)s(22)fe␣a(31)(32)␣comfo(5)tab(11)e␣t(20)i(25)gs,␣a(31)(32)␣his␣
litt(11)e␣hob(37)it-(20)il(11).␣He↩s(20)iv(58)(59)(60)d.␣It␣was␣g(2)tting␣bitte(5)␣
col(15)␣u(27)␣h(58)(59)(60),␣(41)(42)(43)(44)(45)(46)(47)␣wi(25)d␣cam(60)(61)shril(11)
␣among↩th(2)␣ro(7)ks.␣(34)oul(32)e(5)s,␣too,␣(22)t(44)(45)imes(28)␣cam(60)(61)ga(11)l
o(27)i(63)g␣(49)own␣th(2)␣mountain-(13)id(2)s,␣let↩loos(60)(61)by(61)mid-day␣sun␣u(27)
o(25)␣(45)(46)(47)␣snow,␣a(31)(32)␣pas(13)(14)d␣among␣(45)(46)(47)m␣(which␣was␣lucky)
(28)␣o(59)↩ov(58)(59)␣(45)(46)(47)i(5)␣h(2)ads␣(which␣was␣alarming).␣T(46)(47)␣nights
(44)w(58)(59)(60)(61)comfo(5)tl(54)(55)s␣a(31)(32)␣chil(11),↩a(31)(32)␣t(46)(47)y␣di
(15)␣not␣(49)ar(60)(61)to(23)(24)i(63)g␣o(5)␣ta(11)k␣too␣(11)oud(28)␣fo(5)␣th(2)␣echo
(54)(55)(56)(57)(58)(59)(60)(61)u(25)ca(31)ny,↩a(31)(32)␣t(46)(47)␣(13)ile(31)c(60)
(61)se(47)me(15)␣to(44)dislik(60)(61)being␣broken---exc(2)pt␣by␣th(2)␣nois(60)(61)(62)f
␣wate(59)↩a(42)(43)(44)(45)(46)(47)␣wai(11)␣of␣wi(63)d␣a(31)(32)␣(45)(46)(47)␣c(5)a(7)
k(9)of(44)sto(25)e.

Figure 8.5: Part way through the cryptanalysis.

There were many paths that led up into those mountains, and many passes over them. But most of the paths were cheats and deceptions and led nowhere or to bad ends; and most of the passes were infested with evil things and dreadful dangers. The dwarves and the hobbit, helped by the wise advice of Elrond and the knowledge and memory of Gandalf, took the right road to the right pass.

Long days after they had climbed out of the valley and left the Last Homely House miles behind, they were still going up and up and up. It was a hard path and a dangerous path, a crooked way and a lonely and a long. Now they could look back over the lands they had left, laid out behind them far below. Far, far away in the West, where things were blue and faint, Bilbo knew there lay his own country of safe and comfortable things, and his little hobbit-hill. He shivered. It was getting bitter cold up here, and the wind came shrill among the rocks. Boulders, too, at times, came galloping down the mountain-sides, let loose by mid-day sun upon the snow, and passed among them (which was lucky), or over their heads (which was alarming). The nights were comfortless and chill, and they did not dare to sing or talk too loud, for the echoes were uncanny, and the silence seemed to dislike being broken---except by the noise of water and the wail of wind and the crack of stone.

— from *The Hobbit* by J. R. R. Tolkien.

Figure 8.6: The completed cryptanalysis.

0	T	16	␣	32	d	48	␣
1	h	17	w	33	␣	49	d
2	e	18	i	34	B	50	w
3	␣	19	t	35	i	51	a
4	**c**	20	h	36	l	52	r
5	r	21	␣	37	b	53	v
6	a	22	a	38	o	54	e
7	c	23	␣	39	,	55	s
8	k	24	s	40	␣	56	␣
9	␣	24	n	41	a	57	w
10	**c**	26	**a**	42	n	58	e
11	l	27	p	43	d	59	r
12	o	28	,	44	␣	60	e
13	s	29	␣	45	t	61	␣
14	e	30	a	46	h	62	o
15	d	31	n	47	e	63	n

Figure 8.7: The key used in the example. The symbols for 4, 10, 26, and 39, have been solely determined by the context of the key itself.

```
T(20)er(5)␣wer(5)␣m(21)n(41)␣p(21)t(20)s␣t(20)(21)t␣l(5)d␣up␣i(54)to␣t(20)ose␣mo(39)n
tai(54)s,␣a(54)d␣m(21)n(41)␣p(21)sses␣o(28)er↩t(20)em.␣B(39)t␣most␣of␣t(20)e␣p(21)t
(20)s␣wer(5)␣c(20)eats␣a(54)d␣d(5)c(5)ptions␣a(54)d␣l(5)d␣now(20)er(5)␣or␣to↩b(21)d␣
e(54)ds(51)␣a(54)d␣most␣of␣t(20)e␣p(21)sses␣wer(5)␣i(54)fest(5)d␣wit(20)␣(5)vil␣t(20)
i(54)gs␣a(54)d␣dreadfu(16)↩da(54)gers.␣T(20)e␣dwarves␣a(54)d␣t(20)e␣hobbit,␣he(16)p
(5)d␣b(41)␣t(20)e␣wis(5)␣advic(5)␣of␣E(16)rond␣a(54)d↩t(20)e␣knowl(5)dg(5)␣and␣memor
(41)␣of␣Ga(54)da(16)f,␣too(1)␣t(20)e␣rig(20)t␣ro(21)d␣to␣t(20)e␣rig(20)t␣p(21)ss.↩

Long␣da(41)s␣aft(5)r␣t(20)ey␣had␣c(16)imb(5)d␣o(39)t␣of␣t(20)e␣va(16)ley␣a(54)d␣l(5)ft
␣t(20)e␣L(21)st␣Home(16)y↩Ho(39)s(5)␣mil(5)s␣be(20)i(54)d,␣they␣wer(5)␣still␣going␣up
␣a(54)d␣up␣a(54)d␣up.␣It␣w(21)s␣(21)␣(20)(21)rd␣p(21)t(20)↩a(54)d␣a␣dangero(39)s␣p
(21)t(20),␣(21)␣croo(1)ed␣way␣a(54)d␣a␣lo(54)e(16)y␣a(54)d␣a␣lo(54)g.␣Now␣t(20)ey␣co
(39)ld↩loo(1)␣b(21)c(1)␣o(28)er␣t(20)e␣la(54)ds␣t(20)ey␣had␣l(5)ft,␣laid␣o(39)t␣be
(20)i(54)d␣t(20)em␣far␣be(16)ow.␣Far,↩far␣away␣i(54)␣t(20)e␣West,␣wher(5)␣t(20)i(54)g
s␣wer(5)␣b(16)u(5)␣and␣fai(54)t,␣Bilbo␣k(54)ew␣t(20)ere␣lay↩his␣own␣co(39)ntr(41)␣of
␣safe␣a(54)d␣comfort(21)bl(5)␣t(20)i(54)gs,␣a(54)d␣his␣littl(5)␣hobbit-(20)il(16).␣He
↩s(20)iver(5)d.␣It␣w(21)s␣g(5)tting␣bitter␣cold␣up␣her(5),␣a(54)d␣t(20)e␣wi(54)d␣c
(21)m(5)␣shrill␣among↩t(20)e␣roc(1)s.␣Bo(39)lders,␣too,␣at␣times,␣c(21)m(5)␣ga(16)lop
ing␣down␣t(20)e␣mo(39)ntai(54)-sid(5)s,␣let↩loos(5)␣by␣mid(63)day␣s(39)n␣upo(54)␣t
(20)e␣snow,␣a(54)d␣p(21)ssed␣among␣t(20)em␣(w(20)ic(20)␣w(21)s␣l(39)c(1)y(19),␣or↩o
(28)er␣t(20)eir␣heads␣(w(20)ic(20)␣w(21)s␣alarming).␣T(20)e␣nig(20)ts␣wer(5)␣comfortle
ss␣a(54)d␣chill,↩a(54)d␣t(20)ey␣did␣not␣dar(5)␣to␣sing␣or␣ta(16)k␣too␣loud,␣for␣th(5)
␣ec(20)oes␣wer(5)␣u(54)c(21)nn(41),↩a(54)d␣t(20)e␣sile(54)c(5)␣seemed␣to␣dislik(5)␣be
ing␣bro(1)en(63)-(63)exc(5)pt␣b(41)␣t(20)e␣nois(5)␣of␣w(21)ter↩a(54)d␣t(20)e␣wail␣of␣
wind␣a(54)d␣t(20)e␣crac(1)␣of␣sto(54)e.↩
```

Figure 8.8: The example plaintext encoded with a key consisting of random printable ASCII characters. Note the missing symbols do not correspond with those given in the previous figures.

```
yes after the lapse of eight centuries the moonstone looks forth once more over the
walls of the sacred city in which its story first began how it has found its way back
to its wild native land by wha
```

Figure 8.9: The key used in the automated experiments.

```
(39)(40)(41)(42)(43)(44)(45)(46)(47)(48)(49)(50)(51)(52)a(3)r(20)m(137)(138)c(12)(13)
b(195)(196)(197)i(14)k(35)(36)␣(29)o(86)(87)i(46)(47)␣(16)ro(86)o(25)u(12)(13)t(94)(95)
(96)(97)t(60)(61)m(109)(110)g(19)(20)(21)(22)s(7)(8)i(110)ga(16)a(6)a(43)␣(1)x(6)r(98)
(99)t(101)(102)(103)fr(20)m␣a(3)fa(43)i(14)y␣(16)a(16)e(8)(9)i(3)(4)d(102)r(1)(2)s(9)
(10)(11)(12)s(12)(13)(14)i(50)(51)s(83)(84)r(105)(106)t(30)(31)␣(109)(110)(111)i(154)
(155)ia(3)t(171)(172)m(107)(108)r(1)l(183)(184)(185)(186)(187)s(108)(109)(110)(111)e
(110)gl(190)(191)(192)(193)m(107)(108)o(134)j(1)c(27)(28)i(37)(38)(39)o␣(1)xp(14)(15)i
(110)(111)t(40)(41)(42)(43)(44)t(185)(186)(187)(188)w(113)(114)(115)(116)(117)h(148)
(149)(150)i(154)(155)u(29)(30)d(42)(43)e(169)(170)(171)(172)r(1)f(33)s(12)(13)t(94)(95)
(96)r(24)(25)(26)(27)(28)h(190)(191)(192)(193)o(21)(22)f(34)(35)(36)n(102)s(113)(114)p
(3)t(171)(172)m(107)(108)c(152)(153)s(109)(110)(111)j(20)h(138)(139)(140)e(8)n(29)a(47)
(48)l(12)(13)t(94)(95)(96)res(7)(8)v(187)(188)w(113)(114)(115)(116)(117)i(146)(147)
(148)v(12)(13)h(105)(106)h(7)(8)t(171)(172)ma(109)(110)t(15)i(110)e(155)(156)(157)n(9)
(10)(11)i(2)(3)m(183)(184)t(7)(8)(9)h(148)(149)(150)b(1)e(110)(111)mi(37)i(31)(32)e(8)
p(72)(73)t(101)(102)(103)b(107)(108)m(1)m(134)(135)r(88)(89)(90)(91)(92)m(126)(127)
```

Figure 8.10: Sample output from *The Moonstone*.

200 taken from the last paragraph of the book. The key is shown in Figure 8.9. The key was chosen to closely match the remainder of the text, or in effect to make the ciphertext only cryptanalysis as difficult as possible by maximizing the number of matches into the key. Any other key would with very high probability result in a simpler task for the cryptanalyst. A sample of the output upon decompression appears in Figure 8.10.

In the automated attack all the unknowns are initially set to q, to give a very improbable plaintext. We use q rather than the rarer z because qq is rarer than zz. But in fact, experiments indicate that the initial assignment makes little difference to the final result. The unknowns are ranked according to how many times they occur in the decompressed text. Then, starting with the most common unknown, it is replaced with the symbol that gives the best improvement in compression when the PPM model is used, as described in Section 7.2. This is done for each symbol in turn. This replacement strategy is iterated until no further improvement occurs. Figure 8.11 shows a sample output compared to the original when an order-3 PPM model is used.

In this example, no use is made of the statistics of the key itself. This means that the attack is completely independent of any information contained in the key. However, there

ORIGINAL:

the moonstone a romance by wilkie collins prologue the storming of seringapatam extracted from a family paper i address these lines written in india to my relatives in england my object is to explain the motive which has induced me to refuse the right hand of friendship to my cousin john herncastle the reserve which i have hitherto maintained in this matter has been misinterpreted by members of my family whose good opinion i cannot consent to forfeit i request them to suspend their decision until they have read my narrative and i declare on my word of honour that what i am now about to write is strictly and literally the truth the private difference between my cousin and me took its rise in a great public event in which we were both concerned the storming of seringapatam under general baird on the th of may in order that the circumstances may be clearly understood

RECONSTRUCTION:

two moonstone a romance by wilkie collins prolonue tou storming of seringapatam extracted from a family paper i address these lines written in india to my relatives in england my object is to explain two motive where has induced me to refuse tou rinst hand of friendshep to my cousin john herncastle tou reserve where i have hitherto maintained in this matter has been misinterpreted by members of my family whose good opinion i cannot consent to forfeit i request them to suspend their decision until they have read my narrative and i declare on my word of honour that what i am now about to write is strictly and literally the truth the private difference between my cousin and me took its rise in a great publer event in where we were both concerned the storming of seringapatam under general baird on the th of may in order that the circumstances may be clearly understood

Figure 8.11: Sample reconstruction from *The Moonstone*.

is no harm in including the key in the analysis, and this is likely to be beneficial if the key is drawn from a similar source to the message itself or is closely matched to the model used in cryptanalysis. In the remainder of the experiments the key was also used in the attack.

As with most ciphertext only attacks, the longer the plaintext, the easier the attack. This is because as the Ziv-Lempel model expands there is less and less reliance on the key in the formation of phrases, consequently when the key is used there will be long contexts on either side of the unknown phrase. Figure 8.12 shows the performance of the attack as a function of the length of ciphertext. The gap between the two curves is roughly constant, indicating that the technique scales well. Even for very long ciphertexts, many symbols must be guessed using very scant information. Indeed, in the example, symbol 175 does not occur at all in the ciphertext and one symbol does not occur until position 260789 in the file. Several symbols occur less than twenty times in the entire file. In contrast, the most common symbol occurs 7114 times. Understandably the symbols that occur few times are the ones consistently incorrectly determined in the attack. Further, the most common symbols naturally correspond to common symbols like spaces, `es` and `ts`. When very short plaintexts are used, some key bits are correctly determined even though they have not occurred. This occurs because some key bits are common symbols which the algorithm gets right by chance. Figure 8.13 shows the frequency of occurrence for each symbol in the key.

The entropy of the original plaintext with respect to the model is about 2.37 bits per symbol. This compares with 2.41 bits per symbol for the best solution found by the automated cryptanalysis. This indicates that the model should be adequate for obtaining better solutions, but that the method currently employed to determine the symbols is not optimal. By maintaining probability distributions for each symbol and updating them in a Bayesian manner, it is expected that the attack could be improved.

Most compressed texts will not be restricted to twenty-seven symbol English. In principle, a larger PPM model could be built to include all 256 of the ASCII characters and the attack applied as described. However, it is interesting to consider what would happen if an attempt is made to decode full ASCII text using only the twenty-seven symbol model. If this proves possible, it means that twenty-seven symbol models would suffice to reconstruct full ASCII test. This is desirable because twenty-seven symbol models are considerably smaller than full ASCII models of the same order. Clearly, portions of the key which correspond to symbols not in twenty-seven letter English will not be determined correctly. But there are good reasons to suspect that upper-case letters will be determined as their lower-case counterparts. Figure 8.14 shows the result of such an experiment. Again the key consisted of 200 symbols and the length of the ciphertext was 50000 symbols.

It is interesting that many of the punctuation symbols have been classified as `s`. This is not surprising because by adding `s` many words become plural and still form valid English. It seems then that a model that included the lower-case letters, digits, a single punctuation

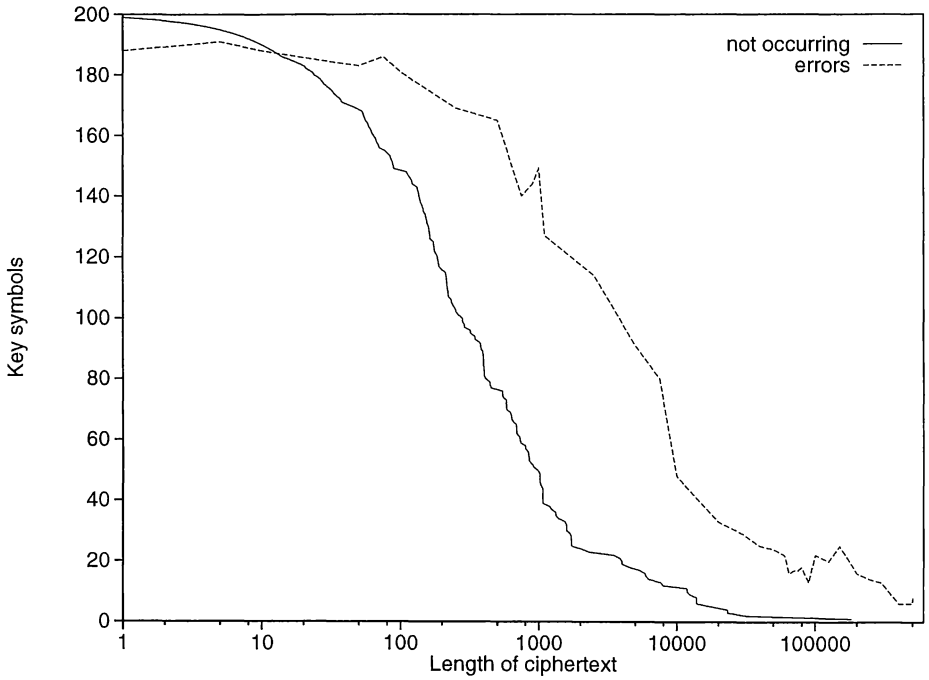


Figure 8.12: The number of errors made versus the length of the ciphertext. The `not occurring` line represents the number of key symbols which have not yet appeared in the ciphertext and it is unreasonable to expect that such key symbols will be determined. The `error` line represents the number of key symbols incorrectly or not determined at all by the automated attack.

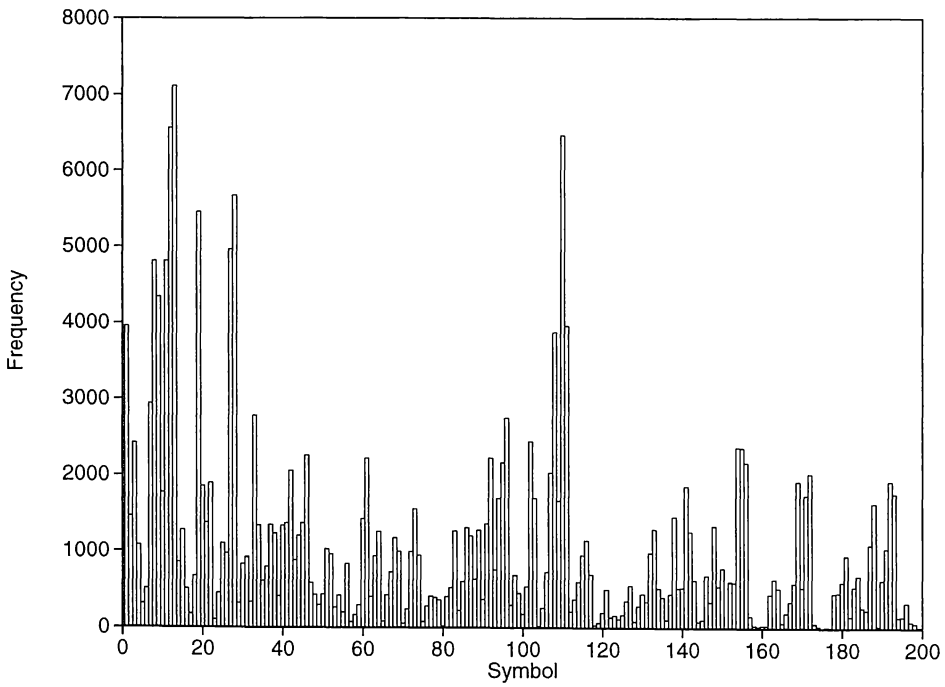


Figure 8.13: Symbol position versus frequency of occurrence.

symbol, and a space symbol would be sufficient to get a readable rendition in this more general setting.

8.3 Cryptanalysis of LZ78

We now focus on the second form of Ziv-Lempel coding, LZ78 [ZL78]. It is shown that attacks similar to those against LZR can be applied to LZ78. Although the cryptanalysis is for LZ78 the attacks presented could be modified for other similar coders, such as LZW, LZC, and LZT [Wel84, Tis87].

A brief description of LZ78 is given, followed by an explanation of the two attacks. We conclude that LZ78 and its derivatives should not be used as security devices.

8.3.1 Description of LZ78

Given a message $x_1x_2\cdots x_n$ drawn from a finite alphabet, the LZ78 approach is to code the message as a series of phrases, where each phrase is the longest matching phrase seen previously, plus the first nonmatching symbol. The output thus consists of a series of pairs (p_i, s_i), where p_i is an index to phrase i and s_i is the first nonmatching symbol. When there is no previous match, phrase number zero is used. In the analysis, we are somewhat sloppy and use p_i to denote both the index to phrase i and phrase i itself. This is acceptable because one can always be obtained from the other. We use $|p_i|$ to denote the length in symbols of phrase p_i.

For example, the message ♠♣♣♣♣◇♣◇♣♡ drawn from the alphabet {♠, ♣, ◇, ♡} would produce the pairs

$$(0, ♠), (0, ♣), (2, ♣), (2, ◇), (4, ♣), (0, ♡)$$

where the phrases are $0 = \Lambda$ (the empty sequence), $1 =$ ♠, $2 =$ ♣, $3 =$ ♣♣, $4 =$ ♣◇, $5 =$ ♣◇♣, and $6 =$ ♡.

As the number of phrases can grow unboundedly as $n \to \infty$, it is necessary for any implementation to be able to encode arbitrarily large integers unambiguously. To ensure good compression an economical encoding must be used. A variety of techniques exist for doing this [BCW90]. Further, the pairs (p_i, s_i) must be encoded so that p_i and s_i and the pairs themselves are self delimiting.

In practice the algorithm cannot be implemented exactly as described above, since the number of phrases grows without bound. The usual variants invoke heuristics to periodically prune the number of phrases. The simplest heuristic is to rebuild a new model from scratch once the number of phrases exceeds some predetermined maximum. Since such heuristics are applied deterministically they cannot foil the attack described; although the attack must be modified to include the pruning.

KEY:

```
White was that ship and long was it a-building, and long it awaited the end of which
C\'\i rdan had spoken. But when all these things were done, and the Heir of Isildur
had taken up the lordship of Men,
```

RECONSTRUCTION:

```
white was the sthip and long was it a buildings j t long i wawaited the end of which
tilierrmsn had spokens but when all these the as were dones how her h ir of isindur
few taurn up the lordshow of mo
```

ORIGINAL TEXT:

```
There was Eru, the One, who in Arda is called Il\'uvatar; and he made first the Ainur,
the Holy Ones, that were the offspring of his thought, and they were with him before
aught else was made. And he spoke to them, propounding to them themes of music; and
they sang before him, and he was glad. But for a long while they sang only each alone,
or but few together, while the rest hearkened; for each comprehended only that part
of the mind of Il\'uvatar from which he came, and in the understanding of their brethen
they grew but slowly. Yet ever as they listened they came to deeper understanding,
and increased in unison and harmony.
```

RECONSTRUCTION:

```
there was erus the ones who in arms is called illuve ar  and he made firs sthe ainur
her holy oness that were the offspring of his thought how hery were with him before
aught else was made and he spoke to thems propounding to them themes of music how
hery sang before him and he was glad but for s long while hery sang only each alones
or but few together while her res shearkened for each comprehended only the spart
of the mind of illuve ar from which he cames how in the unders anding of their brethen
hery grew but slowly yet ever as they listened hery came to deeper unders anding
how increased in unison and harmonys
```

Figure 8.14: Trying to decode full ASCII using a twenty-seven symbol model.

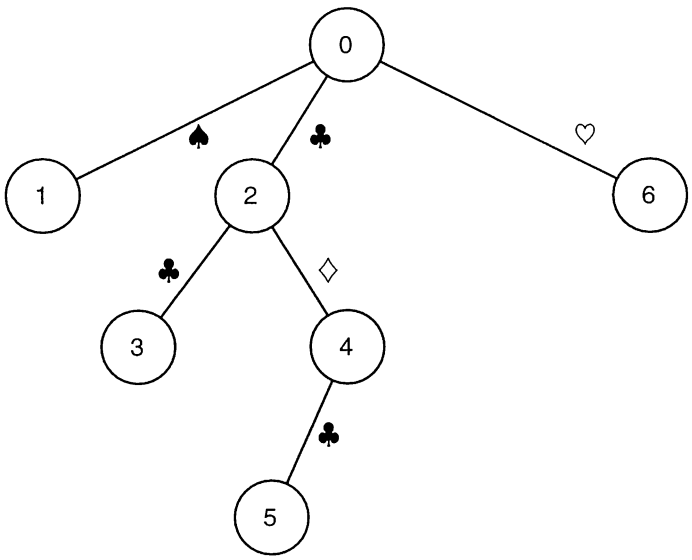


Figure 8.15: The trie resulting from the encoding of ♠♣♣♣♣♢♣♢♣♡ using LZ78.

A popular approach to storing the phrases is to use a trie. The trie for the above example is shown in Figure 8.15. The root, corresponding to the empty phrase, is allocated phrase number zero; thereafter, phrase numbers are allocated in an incremental manner. More generally, the phrase numbers can be assigned by any invertible computable function. But, no matter how the labelling is done, a function can be constructed which maps the phrase numbers to the numerical order in which they are created. Thus, we shall assume, without loss of generality, that phrases are labelled in the order they are inserted into the trie. One important consequence is that the path from a leaf to the root will consist of a strictly decreasing sequence of phrase numbers.

It is possible to use LZ78 in all the key modes discussed in Section 8.1. The security of LZ78 is examined when the key consists of a portion of keytext encoded prior to the message, or, equivalently, when the key consists of a trie preloaded with some phrases.

8.3.2 Known Plaintext Attack

To simplify the analysis, assume the cryptanalyst is aware of the highest phrase number, L, currently in use. If this is not the case, L can easily be determined by repeatedly sending the same symbol until two output pairs have been produced, at which time $L-1$ would be the phrase number of the second pair. Unfortunately, having to resort to this would require a chosen plaintext attack.

A small example is given to illustrate the attack. Let the earlier example constitute the key; that is, the phrase trie is initialized to Figure 8.15 prior to encoding any message, and consider encoding the message ♡♡♢♣♢♡♡♣♠♠. The cryptanalyst will observe the output pairs

$$(6,\heartsuit),(0,\diamondsuit),(4,\heartsuit),(6,\clubsuit),(1,\spadesuit).$$

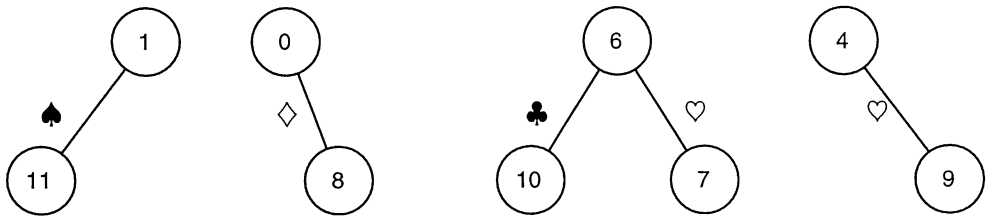


Figure 8.16: Partially determined trie.

This information and the fact that initially $L = 6$ is sufficient to determine large portions of the trie. By considering the output pairs alone we can immediately determine that:

phrase 7 is the ♡-child of phrase 6,
phrase 8 is the ♢-child of phrase 0,
phrase 9 is the ♡-child of phrase 4,
phrase 10 is the ♣-child of phrase 6,
phrase 11 is the ♠-child of phrase 1.

Using only this information, the cryptanalyst can produce the forest in Figure 8.16. In this way all additions to the unknown initial trie are trivially determined. So far, only the ciphertext has been used.

With access to the plaintext, further progress is possible. For example, phrase 7 must end in ♡. Since $L = 6$ and $p_6 \neq \epsilon$ we know that $2 \leq |p_7| \leq 6$. Therefore, phrase 7 is either ♡♡ or ♡♡♢♣♢♡. However, the latter is seen to be inconsistent with the second pair, as $(0, ♢)$ requires phrase 8 to be the singleton ♢, because $p_0 = \epsilon$. Hence, $p_7 =$ ♡♡ and $p_8 =$ ♢. This implies that phrase 6 is the ♡-child of the root, $p_6 =$ ♡.

Now consider the third pair $(4, ♡)$. The candidate phrases are ♣♢♡ and ♣♢♡♡. Once again, by considering the next pair $(6, ♣)$, we see that ♣ is not a child of the root and must be preceded by at least one symbol. This rules out ♣♢♡♡ for phrase 9, consequently $p_9 =$ ♣♢♡, and $p_4 =$ ♣♢. This in turn means that $p_{10} =$ ♡♣, so that $p_6 =$ ♡. Finally, the last pair $(1, ♠)$ must match the remaining plaintext, so $p_{11} =$ ♠♠ and $p_1 =$ ♠. Collecting all this information together, the trie in Figure 8.17 is obtained.

The sequence was insufficient to completely determine the trie. Trie positions for phrases 2, 3, and 5 remain unknown. The unlabelled node in Figure 8.17 must either be phrase 2 or 3; it cannot be 5 since 4 is below it. Phrase 5 could be a child of nodes 1 or 4 (in fact it is a child of 4). If 2 is the unlabeled node, then 3 is a child of 1 or 2. If 3 is the unlabelled node, then 2 must be a child of 1. All of these possibilities could be distinguished by additional plaintext, such as ♣♣♣.

It is interesting to note that the attack is apparently more efficient for larger alphabets. This is because the number of candidates for each pair is reduces as the alphabet size increases.

In the stronger chosen plaintext attack the cryptanalyst can continue to send plaintext until all of the key has been determined. When the sequence must be chosen beforehand,

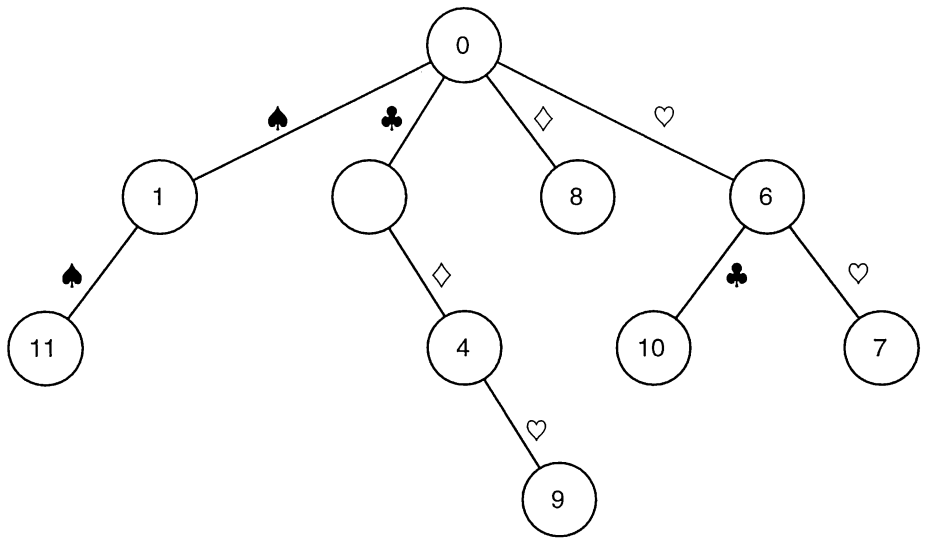


Figure 8.17: Further determination.

Champerknowne's sequence [Cha33]

$$\coprod_{i=1}^{k} a_i \coprod_{j=1}^{k} \coprod_{i=1}^{k} a_j a_i \coprod_{l=1}^{k} \coprod_{j=1}^{k} \coprod_{i=1}^{k} a_l a_j a_i \cdots$$

is a sensible choice. Roughly speaking, this sequence will try every possible phrase in turn, starting with the shortest. In an adaptive chosen-plaintext attack, portions of the sequence can be curtailed as the corresponding portions of the trie are determined.

Any phrases having all their children present cannot be labelled by the attack. However, this is no deficiency in the attack since these phrases will never occur in the output of any transmission.

Again we are faced with the problem of quantifying how much plaintext is required to determine a given proportion of the key.

In [JS95] it is proven after much analysis that M_m, the number of phrases constructed from a sequence of length m for a memoryless source with unequal probabilities of symbols generation, has expectation and variance

$$E(M_m) \sim \frac{mH}{\lg m}$$

$$\mathrm{Var}(M_m) \sim \frac{(H_2 - H^2)m}{\lg^2 m},$$

where $H = -\sum p_i \lg p_i$, is the entropy of the alphabet; and,

$$H_2 = \sum p_i \lg^2 p_i.$$

In the case where each symbol occurs with equal probability, the variance becomes:

$$\mathrm{Var}(M_n) \sim \frac{(C + \delta(\lg m))m}{\lg^3 m},$$

where $C = 0.26600\ldots$ is a constant and $\delta(x)$ is a periodic continuous function of period 1, mean 0, and very small amplitude ($< 10^{-6}$). It follows that the mean phrase length is

$$\frac{m}{E(M)} = \frac{\lg m}{H}.$$

For example, in the symmetric case, $H = \lg n$, where n is the size of the alphabet. Hence

$$\frac{m}{E(M)} = \frac{\lg n}{\lg m} = \log_n m.$$

Thus for a random source the expected length of a randomly selected phrase is $\log_n m$, as in the LZ77 case.

The number of phrases in a m symbol window should be about $m/\log_n m$. If it is assumed that each plaintext matches one of these phrases (they are not much good as a key if this is not the case) then the same analysis as before can be used. Thus roughly half the phrases can be determined by $m/\log_n m$ plaintexts.

8.3.3 Ciphertext Only Attack

It is possible to obtain the plaintext solely from the compressed text for a redundant source. Again, all that is required is that the cryptanalyst have a better model of the source than the Ziv-Lempel model. An example of this attack is provided for an English text.

The last chapter of book3 was compressed and the first 200 phrases deleted (this represents 5.6% of the total number of phrases in the chapter). An attempt is made to decompress the remaining text (the last portion of it appears in Figure 8.18). By examining the decompressed output, it is easy to determine what some of the missing phrases are. Each phrase determined gives additional clues to the identity of phrases remaining. Eventually the entire list of phrases can be reconstructed. Figure 8.19 lists the phrases for this example.

This example is particularly easy as only a modest proportion of the total phrases are unknown. However, note that these 200 phrases correspond to 477 text characters or a 3816 bit key, already far greater than the key size used in many conventional cryptosystems. In fact we cheated slightly in the example; in particular we were unable to determine phrases 173 and 113. Our guesses of `t,` and `e,` represent only minor deviations from the text. Phrases 21 and 63 would also have been hard for someone unfamiliar with the book to deduce. A larger example where 5000 phrases were removed was also solvable by hand.

If a larger percentage of the file is unknown, much more guesswork is necessary by the cryptanalyst, and better and better models of English are required.

Notice that the list of phrases in Figure 8.19 corresponds to the initial text which was removed. Indeed, the entries shown in boldface were determined solely from this context since they are not used in any of the subsequent text.

```
'T(53)n␣the␣(87)rop(53)c(89)s(45)the␣o(31)d␣so(83)s␣(162)ve␣(2)urn(167)␣ou(33)
o␣b(92)r(70)e,(29)f(2)er␣a(78)fa(6)hi(88)!'(55)aid␣B(142)bo.(78)
'O(125)cou(16)se!'␣sa(15)d␣G(5)ndalf.␣'An(108)wh(100)s(12)oul(163)ot␣(198)␣p
(16)ov(92)ru(13)?(3)Su(54)l(100)yo(70)
(49)on't(132)is(26)eli(81)e␣(179)p(16)op(53)ci(149),␣b(97)au(71)␣y(22)u␣had(29)
␣h(5)nd␣in(199)ri(153)ng(20)m(29)b(22)ut
(44)our(71)lf?␣Yo(70)␣d(88)'(30)r(145)lly(55)up(87)os(13),␣d(34)y(22)u,(18)at␣
a(99)␣yo(200)␣a(49)ve(8)tu(126)␣(5)nd
e(6)c(5)pe(90)ere(3)mana(74)ed␣(26)y␣(183)er(171)u(105)k,(3)ju(17)␣f(62)␣y(22)
ur␣s(22)le␣(26)en(172)i(2)?(3)You␣(106)e␣a␣(41)ery
f(155)e␣(87)er(176)n(84)␣M(86)Bagg(155)s,(174)␣I(29)m(3)ve(16)y␣f(88)d␣o(125)y
(22)u;(199)ut␣y(22)u␣a(54)␣o(8)l(100)q(70)ite␣(5)
(31)it(2)le(3)fell(133)␣i(76)a␣(4)id(25)wor(31)d␣af(2)er␣al(31)!'

'Th(5)nk␣(74)ood(8)ess!'␣sai(108)Bil(143)␣(191)u(74)hi(83),␣a(8)d␣ha(8)ded(144)
im␣(179)to(110)cc(22)-j(106).(78)
```

Figure 8.18: The decompression of a Ziv-Lempel file with the first 200 phrases removed. Numbers in parentheses represent unknown phrases to be determined by the cryptanalyst.

Even when an entire text cannot be reconstructed, a sufficient amount may be recovered so that its intention is clear. The attack works particularly well for long texts: as more and more text is compressed there is less and less reliance on the first few phrases.

Again it is worth considering how this process could be automated. This system is more complex in that each unknown consists of an entire phrase and not just a single symbol. It is, in general, impossible to work out in advance exactly how long the phrase is going to be. But in principle there is no reason why such phrases could not be guessed. Indeed it is easy to generate a list of likely phrases (see Figure 2.5). The effect of various phrases on the uncertainty can be measured by a PPM model just as readily as individual symbols. Indeed this system is actually more constrained than LZR because it is guaranteed that each phrase is unique. Further, if phrase p_i has length $l > 1$, then each proper prefix of p_i must also be a phrase and have a phrase number less than i. For example, say phrase 10 is `the`, then the phrase `th` must occur beforehand (say as number 5), and the phrase `t` must occur before that (say as number 2).

It would appear that a possible strategy for automatic attack would be to first determine phrase 1 (which must be a single symbol) and then phrase 2 which is either a single symbol or phrase 1 followed by a single symbol and so on. However, it would then be critical that these low phrase numbers are identified correctly.

1	I	51	l,	101	the	151	ill
2	t	52	␣w	102	␣on	152	␣si
3	␣	53	he	103	e␣th	153	ngi
4	w	54	re	104	at␣	154	ng␣
5	a	55	␣s	105	c	155	in
6	s	56	to	106	ar	156	␣the␣t
7	␣o	57	od	107	rie	157	ree
8	n	58	␣the␣	108	d␣	158	s,
9	␣m	59	L	109	the↩	159	␣as
10	ay	60	ast	110	ba	160	␣**if**
11	␣t	61	␣(	111	gg	161	␣they␣
12	h	62	or	112	ag	162	ha
13	e	63	␣**the␣F**	113	e;	163	d␣n
14	␣**F**	64	ir	114	␣an	164	ot
15	i	65	**st)**	115	d␣t	165	␣sto
16	r	66	␣H	116	hey	166	pp
17	st	67	om	117	␣al	167	ed
18	␣th	68	**ely**	118	l␣	168	↩s
19	at	69	␣Ho	119	f	169	inc
20	␣the	70	u	120	elt	170	e␣h
21	␣**tw**	71	se	121	␣i	171	e␣l
22	o	72	.	122	**n␣n**	172	ef
23	␣c	73	␣A	123	ee	173	**t;**
24	am	74	g	124	d␣o	174	␣and
25	e␣	75	ai	125	f␣	175	␣as␣
26	b	76	n␣	126	res	176	so
27	**ac**	77	iy	127	**t.**	177	on␣
28	k	78	↩	128	␣**As**	178	as␣
29	␣a	79	wa	129	␣they	179	the␣
30	t␣	80	s␣	130	␣r	180	**rid**
31	l	81	ev	131	**ode**	181	ers
32	**as**	82	eni	132	␣d	182	␣ca
33	t␣t	83	ng	133	ow	183	m
34	o␣	84	,	134	n␣t	184	e␣d
35	th	85	␣thei	135	he␣	185	own
36	e␣b	86	r␣	136	ste	186	␣in
37	ri	87	p	137	ep	187	to␣
38	nk	88	on	138	␣p	188	the␣l
39	␣of	89	ie	139	**ath**	189	owe
40	␣the↩	90	s␣w	140	,↩	190	**r␣g**
41	v	91	er	141	B	191	la
42	al	92	e␣t	142	il	192	de
43	le	93	ire	143	bo	193	s␣o
44	y	94	d,	144	␣h	194	f␣t
45	␣of␣	95	␣e	145	ea	195	he↩
46	R	96	sp	146	rd	196	wo
47	iv	97	ec	147	␣the␣e	197	**od␣**
48	**en**	98	**ia**	148	lv	198	they
49	d	99	ll	149	es	199	␣b
50	el	100	y␣	150	␣st	200	ur

Figure 8.19: The first 200 phrases in the last chapter of book3. Entries shown in bold face did not occur in the remaining text and were deduced from their context in the list of phrases.

8.4 Conclusion

It has been explicitly shown that LZR and LZ78 are vulnerable to known plaintext attacks, and in the case of a redundant source vulnerable also to ciphertext only attacks. Further, the ciphertext only attack against LZR can be automated using a PPM model of the source. The reasons for these weaknesses are twofold. Firstly, the tuples or triples output by the Ziv-Lempel compressors explicitly and directly give the state adaptation they have caused on the overall model. This enables a record to be maintained of all changes made to the model. Secondly, although the Ziv-Lempel compressors are asymptotically optimal, they are slow to converge to this optimum, and are therefore not as secure from ciphertext only attacks as one might initially suppose.

Part of the reason that the PPM compressor appears more secure than Ziv-Lempel approaches is that it does not directly list the state change with each output, and hence this type of known plaintext attack will not succeed.

It is clear that other Ziv-Lempel variants would be susceptible to the attacks presented. In particular, any variant employing a sliding window is inherently insecure since after a finite number of symbols have been processed, the window will have progressively less dependance on the key.

The attacks presented may find application in recovery of data from corrupt archives. It is easy to see how it could be applied if an initial segment of a compressed file was lost. The technique could also be used if there was some way of knowing which phrases had been lost. One way would be to periodically encode the current phrase number in the compressed output. This would have a negligible effect on compression but might allow decompression to continue in spite of errors.

Chapter 9

Concluding Remarks

Currently many electronic communications are separately compressed and encrypted prior to transmission. It is recognized that compression prior to transmission not only reduces the cost of transmission but also affords greater security. This thesis investigated the possibility of combining these two activities with the intention of simplifying the overall communication system.

Compression is achieved by removing redundancy from the message being compressed, and a number of techniques are available for doing this [BCW90]. The best compressors are adaptive and as a message is compressed the compression model becomes more closely attuned to the source.

Removal of redundancy helps the cryptographer in two main ways:

- It increases resistance to ciphertext only attacks because the frequency distribution of the ciphertext symbols is essentially flat. This deprives the cryptanalyst of the usual statistical leverage used in a ciphertext only attack.
- Since for an ideal compressor each possible output is equally likely, a data compressor can be used to safeguard against dictionary style attacks. Such attacks are only possible when the prior distribution of possible messages is not uniform, so the cryptanalyst is able to reduce the number of likely messages (or parts of messages) to a manageable level. Thus, compressing messages prior to encryption represents a viable alternative to inventing new cryptosystems with unwieldy block sizes. Thus by removing redundancy, the unicity distance of even the simplest cryptosystems is increased. For example, the only ciphertext only attack against a simple substitution of a uniform source (one in which all messages of the same length are equally likely) is an exhaustive search. But in this case, even an exhaustive search will produce no extra information, because each possible decryption represents an equally plausible message.

However, real compressors are not optimal except for certain artificial sources. Consider now the more difficult problem of natural languages, English say. Previously, it was

usual practice to determine how well a compressor performed by comparing its predictive ability with that of human experts. But modern compressors now rival human predictive ability [TC96] and there is no theoretical or even practical reason why a computer based model of English could not be superior to a human model.

It has been estimated that the average person will hear between 35 and 500 million words by the age of thirty [TC96]. Some models are now trained on an amount of text comparable to this [BDDLM92]. Much larger volumes of text might be used for training in the future. For instance, it is estimated that there are at least fifteen billion words accessible on the World Wide Web [TC97] as of November 1996, and this is a potential source of training text. Further, there are a number of specific projects, such as Project Gutenberg [Har92], which aim to vastly increase the amount of literary material available online.

A great advantage of models like PPM is that it is actually not necessary to store the training text itself, but only frequency counts for the various contexts. This is more economical than having to store the entire training text. Figure 9.1 [1] shows the model size for an order-5 model as a function of amount of training text. The memory required is closely proportional to $x^{0.4}$ where x is the size of compressed text. If this relationship continues, then roughly 560 megabytes would be sufficient to model fifteen billion words. The $x^{0.4}$ relationship is comparable with previous results indicating that the size of the vocabulary is proportional to $x^{0.5}$ to $x^{0.6}$ [Sal88].

Therefore it is not unreasonable to suppose that computer models of English will eventually surpass those of the average person. This has two important implications for cryptology. First, as already mentioned, a compressor based on such a model will automatically have a very high resistance to ciphertext only attack. That is, the compressor can be followed by a simple substitution and the best ciphertext only attack will be exhaustive search. Second, the ability to break future systems (particularly with ciphertext only attacks) will depend on having a good model of the source. This was demonstrated in Chapters 7 and 8 where a PPM model was successfully used to break simple substitution ciphers and Ziv-Lempel compression.

It seems that the development of more powerful models for natural languages is a sensible avenue of research for cryptologists. It also seems reasonable to suppose that government intelligence agencies are likely to maintain large language models in the future (if they do not do so already). Larger accurate models of natural languages are likely to be of considerable interest to language researchers. Teahan and Cleary [TC97] have shown how methods based on our PPM code breaking examples can be used for language identification, authorship attribution, spelling correction, and speech recognition.

At this time, it appears that the greatest potential contribution of data compression technology to cryptology is in furnishing accurate models of natural languages. The ques-

[1] Figure 9.1 was kindly supplied by Bill Teahan

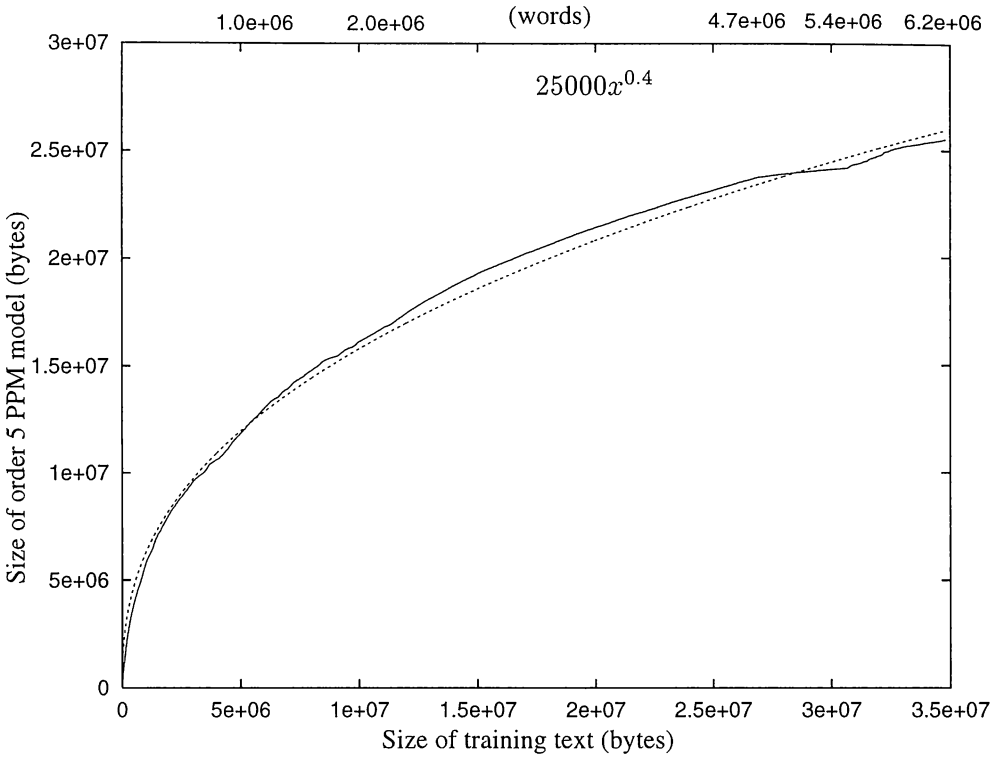


Figure 9.1: Order-5 PPM model implemented as a trie.

tion of whether data compressors can make viable cryptosystems is of importance because of the ciphertext only resistance they offer.

The difficulty here is that resistance to ciphertext only attack has very little implication to wider security issues, like resistance to the more powerful known plaintext and chosen plaintext attacks. This was without doubt our biggest oversight at the outset, and it was some time before we recognized the importance of this point. To the uninitiated, these more powerful attacks may seem relatively unlikely; but it is well known that most commercial information crime is perpetrated by insiders [WW90]. Further, many of the more spectacular breaks of historical ciphers were of this type [Kah66].

At the outset of the investigation, we had already formed some intuitive opinions about the security properties of various data compressors, and it is interesting to contrast these with the results of our analyses. In particular, it seemed reasonable to suppose that PPM and arithmetic coding could offer security. But since all the systems examined were designed solely for data compression, they contain no methodology deliberately designed to make unauthorized decoding difficult. Indeed, such difficulty is something which people working in the compression field try to avoid. It is easy to see why this is the case: simple methods are easier to explain, easier to implement, are typically faster, and occupy less memory. There is, for instance, no reason to add diffusion steps in a compressor for the purpose of compression; indeed diffusion is contrary to compression. Of course we could always go ahead and build a compressor that did include such operations; but the end

result will be just as slow (and more difficult to analyse) than using a standard compressor followed by a standard cryptosystem. We maintain a certain degree of belief in the security of PPM* because none of the attacks developed in the thesis are applicable to this system. In particular, PPM* cannot be broken by forcing the model into a known state. Further, an attack against PPM* would almost certainly require an ability to invert arithmetic coding in the general adaptive case. To date, there is no known way to invert arithmetic coding in this situation, unless restarting of the coder is permitted.

9.1 Summary of Results

Overall our scepticism has increased as we have learnt more about the tough requirements for security and as we have discovered a variety of weaknesses in different compressors. While there is reason to suspect that the better compressors are immune to ciphertext only attack with current models, we cannot offer any good arguments as to why they should be resistant to the more powerful attacks.

9.1.1 Arithmetic Coding

Our high initial belief in the security of arithmetic coding stemmed mainly from the observation that it is perfect in the information theoretic sense. The discovery that arithmetic coding could be used for error detection [BCIRW97] enhanced our belief in arithmetic coding as a truly universal coder which could be used for compression, error recovery, and security.

The warning highlighted above is borne out by arithmetic coding where we find no ciphertext only attack but for which there is a chosen-plaintext attack. Strictly speaking, only an explicit attack against a static arithmetic coder has been given, along with an indication of how this attack might be extended to more general situations.

Arithmetic coding also highlights the difficulty of adding a key mechanism to an existing algorithm. Several possible ways of including a key were considered in Chapter 5 leading to a known **NP**-complete problem. But the cases that arose fell into a subclass known to be solvable in polynomial time.

Finally, the information-theoretic optimality of arithmetic coding is only with respect to the driving model. So while ciphertext only security is strong for well defined sources, it is far from clear with the more usual sources such as English text. Even when the model driving the arithmetic coder is adaptive, there are serious problems with the rate of convergence.

9.1.2 Huffman coding

Huffman coding is the other significant coding technique that is widely used. Its popularity no doubt arose from its simplicity and speed. Huffman coding is not perfect in the information theoretic sense (for example, at least one bit is output for each symbol encoded). This, coupled with the intrinsic simplicity of the algorithm greatly reduced our initial belief that Huffman coding could be secure. But by adding bits to a Huffman stream it has been shown that decoding the stream becomes **NP**-complete [FK94]. Further, Huffman coding is provably secure from a ciphertext only attack in certain special situations [GMR96]. While these results are positive they are both relatively weak. For instance, a problem can be **NP**-complete and yet still have polynomial time average case [Wil84, BW85].

Again, these positive results are for ciphertext only attacks, and in Chapter 6 a backtracking search was used in a known plaintext attack.

9.1.3 Splaying

In Chapter 6 the semisplaying data compressor was attacked using a backtracking search. Since this compressor has already been used in at least one commercial product as a security device, this attack is important. Initially we expected splaying to be more challenging than Huffman coding because of the tree adaptation. However, the chosen plaintext attack using the backtracking search has the same complexity in both cases.

It appears that with slight modifications the backtracking search would be applicable to adaptive Huffman coders (for example [Gal78, Knu85, Vit86]). Assuming the cryptanalyst is aware of the detailed system in use, none of these adaptive approaches contain any operations likely to foil the backtracking strategy.

9.1.4 Ziv-Lempel

The attacks given in Chapter 8 against the Ziv-Lempel family of compressors are interesting results. The attacks are important because these compressors are widely used (for example in the Stacker program [WGI91], the UNIX `compress` program and a variety of other commercial compression programs). On the face of it, the ciphertext only attacks seem to contradict the asymptotic optimality of Ziv-Lempel compression [ZL77, ZL78, WZ94]. However, Ziv-Lempel compressors are slow in converging to this limit [BCW90] when compared with other compressors. This is evidenced by the fact that PPM compresses better than Ziv-Lempel for a variety of finite sources.

The ciphertext only attack demonstrates that a proof of asymptotic optimality is not sufficient protection against ciphertext only attack. To be secure, such a compressor will need to converge much faster than the Ziv-Lempel approaches.

It is unclear whether the PPM compressor converges fast enough. The ease with which an order-3 PPM model can reconstruct text from a Ziv-Lempel coder indicates that PPM models can capture a source more accurately than Ziv-Lempel approaches.

Because the Ziv-Lempel algorithms are relatively well understood [JS95, WZ94], it is easier to derive analytical results for these coders. The proven vulnerability of the Ziv-Lempel approaches to known plaintext attacks is mainly a result of information in the compressed output. The pairs or triples output by a Ziv-Lempel coder explicitly indicate the adaptation taking place in the model. In contrast, there is no known way that the output of a PPM compressor can be inverted to give the same or equivalent information.

Although we gave explicit attacks only for the LZR and LZ77 variants of Ziv-Lempel coding (being one representative from each of the two styles), there is no reason to suppose that slight modifications of the attack would not be applicable to the ever increasing number of variants of this family.

9.1.5 PPM

Currently the best attack against PPM is to flood the model with a very long string. We were aware of this attack [BH92, BH93] before commencing the investigation, but this did not prevent us from maintaining a high degree of belief in the security of PPM. Although the attack of Section 7.3 highlights a potential weakness of adaptive methods, it is not very practical because the required sequence is long and must be transmitted during normal operation. Further, there are a number of simple strategies to guard against such an attack. One approach is to 'randomly' forget portions of the tree. Alternatively, an unbounded context length can be used.

There are several reasons why PPM is currently the most likely candidate to lead to a secure compressor. Firstly, it apparently converges to sources, such as English text, much faster than other compressors (although this cannot be proven). Secondly, the best PPM models of English now rival the predictive ability of human subjects. This implies a greater resistance to ciphertext only attacks. In particular, it would appear that while PPM remains the best compression technique, it would be impossible to devise an automated ciphertext only attack. This follows because it is reasonable to assume that any such attack must be using a better model of the source. Such a model (if it existed) would lead naturally to a compressor which outperformed PPM.

The model adaptation in PPM is nontrivial, and more importantly cannot easily be determined by looking at the output alone. Even if the bold assumption is made that arithmetic coding can be inverted in this general situation (which is still an open question), so that we know the probabilities that were encoded, the problem of reconstructing the unknown model seems to remain intractable.

PPM deserves further scrutiny, but our results for other compressors make us cautious about making any claims of security for this system.

9.2 Small Points

The ability to reconstruct an initial missing segment of a Ziv-Lempel file for a redundant source suggests that it ought to be possible to recover most of the information in a corrupt archive. However, our method relied on the cryptanalyst knowing how many and which phrases had been deleted. If it is only the first portion of the file which is damaged then a guessing strategy may suffice. It is not obvious how the more general problem could be tackled. One approach might be to periodically insert the current phrase number in the compressed output (say every thousand phrases). Under normal operation, decompression proceeds as usual and the phrase number would be verified at the designated positions. Then when a discrepancy is found an error recovery routine could be invoked. Of course, there is still potential for disaster if an inserted number is corrupted. It is not clear whether this is worthy of further investigation.

In Chapter 4 it was shown that the better compressors passed a number of well known tests of randomness. This suggests that it might be possible to use a compressor as a random number generator. Indeed, one such proposal already exists in the literature, based on Ziv-Lempel coding [JB90]. Before this happens it would be desirable to carry out a theoretical analysis of what happens in practice when random data is compressed. Such an analysis might also lead to new insights into the operation of current compressors and to practical improvements. Current compressors expand a random input by at least 10%, but complexity results imply an increase of only $O(\log n)$ need be incurred, where n is the length of the random data. So there appears to be considerable room for improvement. Note that some commercial compressors check for poor compression and use a single bit to flag that the source has just been copied verbatim.

9.3 Recommendations

Although it is clearly impossible to disprove the thesis by presenting any number of specific examples where it fails, such failures do count as strong evidence against the thesis. It has been shown that arithmetic coding, Huffman coding, prediction by partial matching, semisplaying, and Ziv-Lempel coding all contain security flaws of one form or another to the extent that *none of these should be used as security devices.* While we have not explicitly covered every compressor available, the main styles of lossless compression algorithms and their associated coders have been examined.

Books on cryptology rarely make mention of data compression. This is perhaps because professional cryptologists are skeptical about the use of compression. Our results show such skepticism is warranted, even though the origins of such skepticism may have been for other reasons (such as the difficulty of obtaining precise analytic security results for compression systems).

While we cannot recommend the isolated use of any compressor as a security device,

compression remains an important adjunct to encryption, and all critical transmissions should be compressed prior to encryption. This is important to avoid dictionary attacks and more generally reduce the chance of ciphertext only attack. In high security applications using one-time pads, compression does not increase security (and strictly speaking is unnecessary). However, compression does allow the most effective use of the key material. This is desirable because the generation of a one-time pad is both a delicate and expensive business.

9.4 Objections

Because not every possible compressor has been examined it could be objected that there exists a secure compressor yet to be discovered. This is an easy objection to make and we have made it ourselves on several occasions. But in each case where an initially promising system was examined in detail a security weakness was found. It is likely that the techniques used in obtaining our results will be applicable to other systems not yet examined.

There is of course the large class of lossy compressors which were not investigated. Such compressors are used primarily for images, motion video, and sound compression where exact reconstruction of the source is not necessary. Security for video and speech has in the past been achieved from a slightly different technological viewpoint. For instance secure audio has generally been achieved by ***scrambling***, which involves distributing the signal over various frequency bands [BP85]. Often this is considered more a problem of physics, since it is physical laws that limit the kinds of transformations possible [BP85]. This was the natural way of handling these objects when they were transmitted by predominantly analogue means. But since telephone networks are becoming increasingly digitized, it is perhaps not surprising that we are in the midst of a paradigm shift and that video and speech security are now being drawn under the umbrella of encryption. Ironically, all of this is occurring just as physical laws are being harnessed in the development of novel quantum mechanical cryptosystems [BBE92]. Thus it seems that an investigation of compression systems like the quantizers lying at the heart of the JPEG [Wal91] and MPEG [Gal91] applications is warranted. Note that some older image compression standards such as the Graphical Interchange Format (GIF) are actually based on Ziv-Lempel systems.

A second potential objection is that not all reasonable key mechanisms have been examined. This objection has merit. Human inventiveness being what it is, it is entirely possible that someone might conceive of a more secure key mechanism for any of the systems discussed. But, despite appearances, it is actually quite difficult to invent a key mechanism that does not rely on some other cryptographic primitive. For instance, one seemingly innocuous idea is to simply flip a few random bits in the compressed output. The main difficulty here is in producing such a random sequence. If the cryptanalyst knows in advance which bits are flipped then no security is gained. Thus, to produce the random bits one is forced to used a cryptographically strong bit generator. But such

generators are computationally expensive and the end result is that no saving is made in the cost of communication. More generally, any key system should obey the desiderata given in Chapter 3. Failure to meet one of those requirements will result in either an insecure system, or one in which no savings have been made.

As compression models improve, they will naturally become less vulnerable to ciphertext only attacks. The main focus of a cryptographer interested in using a compressor as a cryptosystem should therefore be in increasing resistance to known plaintext and chosen plaintext attacks. Most of our attacks have been successful because the compressor leaked too much information about its state. Ideally, large amounts of internal state should be updated in an unpredictable manner. However, the need to maintain the synchronicity of the encoder and decoder and the desire for speed while maintaining good compression preclude many otherwise feasible ideas.

9.5 Further Work

A number of suggestions for future research have already been made, but they are collected here so that a clearer picture of what remains to be done emerges.

The most important goal would appear to be the development of better language models. While such improvements are probably only of minor importance to compression (in that only minor improvement in compression ratio will occur), they are likely to lead to compressors with higher resistance to ciphertext only and dictionary attacks. Language identification and speech recognition are also useful to cryptanalysis and the use of language models in these areas should be further explored.

Any improvement in modelling should lead to improved attacks against simple ciphers. This will of course be somewhat dependent on sufficient training text being available. It would be nice to investigate the extent to which the automatic attack given in Chapter 7 can be applied to other systems. There would appear to be no practical difficulties in extending the attack to Beaufort, Vigenère, Playfair, polyalphabetic substitution, and some homophonic ciphers [Kah66].

It is more interesting, however, to consider how such models might be combined with a dictionary attack to perform automated ciphertext only attacks against public-key cryptosystems. Such a model could be used to select likely plaintext segments from whatever information is currently available. In this manner it may be possible to obtain ciphertext only attacks against a variety of public-key cryptosystems.

Of the compressors examined, only arithmetic coding and PPM seem worthy of further study. While arithmetic coding is not secure in some simple cases (Chapter 5), it is still unclear how difficult it is to invert it in more general (and more important) situations, such as when it it used in conjunction with PPM. More analysis of PPM is needed: the chosen plaintext attack discussed in Chapter 7 is impractical for modest alphabets and it

would be interesting to see if any shorter attacks could be devised.

In particular an analysis of PPM* would be most valuable. None of the techniques we have developed can be successfully applied to this system. At this time there is no known way to break a message encoded with PPM* using 32-bit counts and an initial model as the key.

The use of compressors in random number generation and in error recovery has been addressed earlier. The first of these would lend itself to both theoretical and practical investigation. It is likely that the results of a theoretical investigation in this area would provide clues for the further improvement of language modelling.

Bibliography

[ABA79] AMERICAN BANKERS ASSOCIATION, 'Management and use of personal information numbers', ABA Bank Card Standard, Aids from ABA, Catalogue number 207213, American Bankers Association, 1979.

[Abr63] N. ABRAMSON, *Information Theory and Coding*, McGraw-Hill, NY, 1963.

[ACGS88] W. ALEXI, B.-Z. CHOR, O. GOLDREICH, & C. P. SCHNORR, 'RSA and Rabin functions: certain parts are as hard as the whole', *SIAM J on Computing* **17**, 2, pp. 194–209, 1988.

[AFK89] M. ABADI, J. FEIGENBAUM, & J. KILIAN, 'On hiding information from an oracle', *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pp. 195–203, 1987.

[AGLL94] D. ATKINS, M. GRAFF, A. K. LENSTRA, & P. C. LEYLAND, 'The magic words are squeamish ossifrage', *Fourth Workshop on the Theory and Applications of Cryptology*, Wollongong, Australia, pp. 219–229, 1994.

[And89a] ROLAND ANDERSON, 'Finding vowels in simple substitutions ciphers by computers', *Cryptology: Machines, History & Methods*, Artech House, 1989.

[And89b] ROLAND ANDERSON, 'Improving the machine recognition of vowels in simple substitution ciphers', *Cryptology: Machines, History & Methods*, Artech House, 1989.

[And94] ROSS J. ANDERSON, 'Why cryptosystems fail', *Communications of the ACM* **37**, 11, pp. 32–40, 1994.

[ANSI25] ANSI X3.92, 'American national standard for data encryption algorithm (DEA)', American National Standards Institute, 1981.

[Aus68] JANE AUSTEN, *Sense and Sensibility*, Heron Books, London, 1968.

[Bal60] W. W. R. BALL, *Mathematical Recreations and Essays*, Macmillan, NY, 1960.

[Bar55] G. A. BARNARD, 'Statistical calculation of word entropies for four Western languages', *IEEE Transactions on Information Theory* **1**, 1, pp. 49–53, 1955.

[BB88] GILES BRASSARD & PAUL BRATLEY, *Algorithmics: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[BB89] C. H. BENNETT & G. BRASSARD, 'The dawn of a new era for quantum cryptography: the experimental prototype is working!', *SIGACT News* **20**, 4, pp. 78–82, 1989.

[BBBSS91] C. H. BENNETT, F. BESSETTE, G. BRASSARD, L. SALVAIL, & J. SMOLIN, 'Experimental quantum cryptography', *Advances in Cryptology: EUROCRYPT'90*, Lecture Notes in Computer Science (473), Springer-Verlag, Berlin, pp. 253–265, 1991.

[BBE92] C. H. BENNETT, G. BRASSARD, & A. K. EKERT, 'Quantum cryptography', *Scientific American* **267**, 4, pp. 50–57, 1992.

[BBS86] L. BLUM, M. BLUM, & M. SHUB, 'A simple unpredictable pseudo-random number generator', *SIAM J on Computing* **15**, 2, pp. 364–383, 1986.

[BCD88] G. BRASSARD, C. CRÉPEAU, & D. CHAUM, 'Minimum disclosure proofs of knowledge', *J of Computer and System Sciences* **37**, 2, pp. 156–189, 1988.

[BCIRW97] COLIN BOYD, JOHN G. CLEARY, SEAN A. IRVINE, INGRID RINSMA-MELCHERT, & IAN H. WITTEN, 'Integrating error detection into arithmetic coding', *IEEE Transactions on Communications*, January 1997.

[BCL79] M. BEHZAD, G. CHARTRAND, & L. LESNIAK-FOSTER, *Graphs and Digraphs*, Wadsworth International Group, California, 1979.

[BCW90] TIMOTHY C. BELL, JOHN G. CLEARY, & IAN H. WITTEN, *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[BDDLM92] P. F. BROWN, S. A. DELLA PIETRA, V. J. DELLA PIETRA, J. C. LAI, & R. L. MERCER, 'An estimate of an upper bound for the entropy of English', *Computational Linguistics* **18**, 1, pp. 31–40, 1992.

[Bel86] T. C. BELL, 'Better OPM/L text compression', *IEEE Transactions on Communications* **34**, 12, pp. 1176–1182, 1986.

[Bel87] T. C. BELL, *A Unifying Theory and Improvements for Existing Approaches to Text Compression*, PhD Thesis, Dept. of Computer Science, University of Canterbury, Christchurch, New Zealand, 1987.

[BH92] HELEN A. BERGEN & JAMES M. HOGAN, 'Data security in a fixed-model arithmetic coding compression algorithm', *Computers & Security* **11**, pp. 445–461, 1992.

[BH93] HELEN A. BERGEN & JAMES M. HOGAN, 'A chosen plaintext attack on an adaptive arithmetic coding compression algorithm', *Computers & Security* **12**, pp. 157–167, 1993.

[Bih92] E. BIHAM, 'New types of cryptanalytic attacks using related keys', Technical Report #753, Computer Science Dept., Technion, Israel Institute of Technology, 1992.

[Bla79] G. R. BLAKELY, 'Safeguarding cryptographic keys', *Proceedings of the National Computer Conference 1979*, American Federation of Information Processing Societies **48**, pp. 242–268, 1979.

[BLMG94] J. BRASSIL, S. LOW, N. MAXEMCHUK, & L. O'GORMAN, 'Electronic marking and identification techniques to discourage document copying', *IEEE Infocom 94*, pp. 1278–1287, 1994.

[Blu82] M. BLUM, 'Coin flipping by telephone: a protocol for solving impossible problems', *Proceedings of the 24th IEEE Computer Conference (CompCon)*, pp. 133–137, 1982.

[BM84] M. BLUM & S. MICALI, 'How to generate cryptographically strong sequences of pseudo-random bits', *SIAM J on Computing* **13**, 4, pp. 850–864, 1984.

[BM89] T. C. BELL & A. M. MOFFAT, 'A note on the DMC data compression scheme', *Computer Journal* **32**, 1, pp. 16–20, 1989.

[BO88] E. F. BRICKELL & A. M. ODLYZKO, 'Cryptanalysis: a survey of recent results', *Proceedings of the IEEE* **76**, 5, pp. 578–593, 1988.

[BO91] E. F. BRICKELL & A. M. ODLYZKO, 'Cryptanalysis: a survey of recent results', *Contemporary Cryptology: The Science of Information Integrity*, G. Simmons (ed.), IEEE Press, pp. 501–540, 1991.

[Boy89] JOAN BOYAR, 'Inferring sequences produced by pseudo-random number generators', *J ACM* **36**, 1, pp. 129–141, 1989.

[Boy90] COLIN BOYD, 'Enhancing secrecy by data compression: theoretical and practical aspects', *Advances in Cryptology: EUROCRYPT'91*, Lecture Notes in Computer Science (547), pp. 266–280, 1990.

[BP82] H. BEKER & F. PIPER, *Cipher Systems*, Wiley, NY, 1982.

[BP85] HENRY J. BEKER & FRED C. PIPER, *Secure Speech Communications*, Academic Press, Orlando, FL, 1985.

[Bre87] R. P. BRENT, 'A linear algorithm for data compression', *Australian Computer Journal* **19**, 2, pp. 64–68, 1987.

[Bri88] E. F. BRICKELL, 'The cryptanalysis of knapsack cryptosystems', *Applications of Discrete Mathematics*, Society for Industrial and Applied Mathematics, pp. 3–23, 1988.

[Bro43] CHARLOTTE BRONTË, *Jane Eyre*, Random House, NY, 1943.

[Bro73] EMILE BRONTË, *Wuthering Heights*, AMS Press, NY, 1973.

[BS83] E. F. BRICKELL & G. J. SIMMONS, 'A status report on knapsack based public-key cryptosystems', *Congressus Numerantium* **7**, pp. 3–72, 1983.

[BS93] ELI BIHAM & ADI SHAMIR, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, NY, 1993.

[Bun97] S. BUNTON, 'Semantically motivated improvements for PPM variants', *The British Computer J, Special Data Compression Issue*, to appear, June 1997.

[BW85] E. BENDER & H. S. WILF, 'A theoretical analysis of backtracking in the graph-colouring problem', *J of Algorithms* **6**, pp. 275–282, 1985.

[BW94] M. BURROWS & D. J. WHEELER, 'A block-sorting lossless data compression algorithm', Digital Systems Research Center Research Report, SRC-124, Digital Systems Reseach Center, 10 May 1994.

[CA90] D. CHAUM & H. VAN ANTWERPEN, 'Undeniable signatures', *Advances in Cryptology: CRYPTO'89*, Lecture Notes in Computer Science (435), Springer-Verlag, Berlin, pp. 212–216, 1990.

[Cer91] MIGUEL DE CERVANTES SAAVEDRA, *Don Quixote*, Knopf, NY, 1991.

[CFM90] D. CHAUM, A. FIAT, & M. NAOR, 'Untraceable electronic cash', *Advances in Cryptology: CRYPTO'88*, Lecture Notes in Computer Science (403), Springer-Verlag, Berlin, pp. 319–327, 1990.

[CH87] G. V. CORMACK & R. N. HORSPOOL, 'Data compression using dynamic Markov modelling', *Computer Journal* **30**, 6, pp. 541–550, 1987.

[Cha33] D. G. CHAMPERNOWNE, 'The construction of decimals normal in the scale of ten', *J London Mathematical Society* **8**, pp. 254–260, 1933.

[Cha85] D. CHAUM, 'Security without identification: transaction systems to make big brother obsolete', *Communications of the ACM* **28**, 10, pp. 1030–1044, 1985.

[Cha88] D. CHAUM, 'The dining cryptographers problem: unconditional sender and receiver untraceability', *J of Cryptology* **1**, 1, pp. 65–75, 1988.

[Cha91] D. CHAUM, 'Group signatures', *Advances in Cryptology: EUROCRYPT'93*, Lecture Notes in Computer Science (765), Springer-Verlag, pp. 257–265, 1991.

[CIR95] J. CLEARY, S. IRVINE, & I. RINSMA-MELCHERT, 'On the insecurity of arithmetic coding', *Computers & Security*, **14**, pp. 167–180, 1995.

[CK50] T. M. COVER & R. C. KING, 'A convergent gambling estimate of the entropy of English', *IEEE Transactions on Information Theory* **24**, 4, pp. 413–421, 1950.

[CLR90] THOMAS H. CORMEN, CHARLES E. LEISERSON, & Ronald L. Rivest, *Introduction to Algorithms*, MIT Press, McGraw-Hill, Cambridge, MA, 1990.

[CM86] JOHN H. CARROLL & STEVE MARTIN, 'The automated cryptanalysis of substitution ciphers', *Cryptologia* **10**, 4, pp. 193–209, 1986.

[Coh88] EARL T. COHEN, `random(3)`, *UNIX Programmer's Manual*, 1988.

[Col82] WILKIE COLLINS, *The Moonstone*, Oxford University Press, Oxford, 1982.

[CR85] B. CHOR & R. L. RIVEST, 'A knapsack type public-key cryptosystem based on arithmetic in finite fields', *Advances in Cryptology: CRYPTO'84*, Lecture Notes in Computer Science (196), Springer-Verlag, Berlin, pp. 54–65, 1985.

[CR89] JOHN M. CARROLL & LYNDA E. ROBBINS, 'Using binary derivatives to test an enhancement of DES', *Cryptology: Machines, History & Methods*, Artech House, 1989.

[CTW95] JOHN G. CLEARY, W. J. TEAHAN, & I. H. WITTEN, 'Unbounded length contexts for PPM', *Proceedings Data Compression Conference*, Snowbird, Utah, IEEE Computer Society Press, 1995.

[CW84a] J. G. CLEARY & I. H. WITTEN, 'A comparison of enumerative and adaptive codes', *IEEE Transactions on Information Theory* **30**, 2, pp. 306–315, 1984.

[CW84b] J. G. CLEARY & I. H. WITTEN, 'Data compression using adaptive coding and partial string matching', *IEEE Transactions on Communications* **32**, 4, pp. 396–402, 1984.

[Dar72] CHARLES DARWIN, *The Descent of Man, and Selection in Relation to Sex*, AMS Press, NY, 1972.

[deB46] N. G. DE BRUIJN, 'A combinatorial problem', *Proceedings Nederlandse Akademie van Wetenschappen* **49**, pp. 758–764, *Indagationes Math* **8**, pp. 461–467, 1946.

[Def62] DANIEL DEFOE, *Robinson Crusoe*, Macmillan, NY, 1962.

[Den82] DOROTHY E. R. DENNING, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.

[Des88] Y. DESMEDT, 'What happened with knapsack cryptographic schemes', *Performance Limits in Communication, Theory and Practice*, NATO ASI Series E: Applied Sciences, **142**, Kluwer Academic Publishers, pp. 113–134, 1988.

[Des90] Y. DESMEDT, 'Abuses in cryptography and how to fight them', *Advances in Cryptology: CRYPTO'88*, Lecture Notes in Computer Science (403), Springer-Verlag, Berlin, pp. 375–389, 1990.

[DH76a] W. DIFFIE & M. E. HELLMAN, 'Multiuser cryptographic techniques', *Proceedings of AFIPS National Computer Conference*, pp. 109–112, 1976.

[DH76b] W. DIFFIE & M. E. HELLMAN, 'New directions in cryptography', *IEEE Transactions on Information Theory* **22**, 6, pp. 644–654, 1976.

[Dif77] W. DIFFIE, lecture at IEEE Information Theory Workshop, Ithaca, NY, 1977.

[DoT84] DEPARTMENT OF THE TREASURY, 'Electronic funds and securities transfer policy', *Department of the Treasury Directives Manual* Section 80, Department of the Treasury, 1984.

[DP84] D. W. DAVIES & W. L. PRICE, *Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer*, Wiley, NY, 1984.

[Ebe93] H. EBERLE, 'A high-speed DES implementation for network applications', *Advances in Cryptology: CRYPTO'93*, Lecture Notes in Computer Science (773), Springer-Verlag, Berlin, pp. 527–545, 1993.

[EGL85] S. EVEN, O. GOLDREICH, & A. LEMPEL, 'A randomizing protocol for signing contracts', *Communications of the ACM* **28**, 6, pp. 637–647, 1985.

[ElG85] T. ELGAMAL, 'A public-key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Transactions on Information Theory* **31**, 4, pp. 469–472, 1985.

[Fel88] FRANK A. FELDMAN, 'Fast spectral tests for measuring nonrandomness and the DES', *Advances in Cryptology: CRYPTO'87*, Lecture Notes in Computer Science (293), Springer-Verlag, Berlin, pp. 243–253, 1988.

[Fen96] PETER M. FENWICK, 'The Burrows-Wheeler transform for block sorting text compression: principles and improvements', *The Computer J* **39**, 9, pp. 731–740, 1996.

[FG89] E. R. FIALA & D. H. GREENE, 'Data compression with finite windows', *Communications of the ACM* **32**, 4, pp. 490–505, 1989.

[FK82] W. N. FRANCIS & H. KUČERA, *Frequency Analysis of English Usage: Lexicon and Grammar*, Houghton Mifflin, Boston, 1982.

[FK94] A. S. FRAENKEL & S. T. KLEIN, 'Complexity aspects of guessing prefix codes', *Algorithmica* **12**, pp. 409–419, 1994.

[FLW92] ALAN M. FERRENBERG, D. P. LANDAU, & Y. JOANNA WONG, 'Monte Carlo simulations: hidden errors from "good" random number generators', *Physical Review Letters* **69**, 23, pp. 3382–3384, 1992.

[Fri76] W. F. FRIEDMAN, *Elements of Cryptanalysis*, Aegean Park Press, Laguna Hills, CA, 1976.

[FY80] AVIEZRI S. FRAENKEL & YACOV YESHA, 'Complexity of solving algebraic equations', *Information Processing Letters* **10**, 4/5, pp. 178–179, 1980.

[Gai56] H. F. GAINES, *Cryptanalysis*, Dover, NY, 1956.

[Gai77] J. GAIT, 'A new nonlinear pseudorandom number generator', *IEEE Transactions on Software Engineering* **SE-3**, 5, pp. 359–363, 1977.

[Gal78] R. G. GALLAGER, 'Variations on a theme by Huffman', *IEEE Transactions on Information Theory* **24**, 6, pp. 668–674, November 1978.

[Gal91] DIDIER LE GALL, 'MPEG: A video compression standard for multimedia applications', *Communications of the ACM* **34**, 4, pp. 46–58, 1991.

[GJ79] MICHAEL GAREY & DAVID S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco, 1979.

[GM82] S. GOLDWASSER & S. MICALI, 'Probabilistic encryption and how to play mental poker keeping secret all partial information', *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pp. 270–299, 1982.

[GMR89] S. GOLDWASSER, S. MICALI, & C. RACKOFF, 'The knowledge complexity of interactive proof systems', *SIAM J on Computing* **18**, 1, pp. 186–208, 1989.

[GMR96] DAVID W. GILLMAN, MOJDEH MOHTASHEMI, RONALD L. RIVEST, 'On breaking a Huffman code', *IEEE Transactions on Information Theory* **42**, 3, pp. 972–976, 1996.

[GMW86] O. GOLDREICH, S. MICALI, & A. WIGDERSON, 'Proofs that yield nothing but their validity and a methodology of cryptographic protocol design', *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, pp. 174–187, 1986.

[Gol67] S. W. GOLOMB, *Shift Register Sequences*, Holden Day, San Francisco, 1967.

[Goo46] I. J. GOOD, 'Normal recurring decimals', *J London Mathematical Society* **21**, pp. 167–169, 1946.

[Hag94] B. C. W. HAGELIN, 'The story of the Hagelin cryptos', *Cryptologia* **18**, 3, pp. 204–242, 1994.

[Har78] T. HARDY, *Far from the Madding Crowd*, Penguin, NY, 1978.

[Har92] MICHAEL HART, 'History and philosophy of Project Gutenberg', ⟨URL: `http://wuarchive.wustl.edu/doc/gutenberg/pg/history.html`⟩, 1992 (accessed 21 January 1997).

[Har94] GEORGE W. HART, 'To decode short cryptograms', *Communications of the ACM* **37**, 9, pp. 102–108, 1994.

[Has86] J. HASTAD, 'On using RSA with low exponent in a public-key network', *Advances in Cryptology: CRYPTO'85*, Lecture Notes in Computer Science (218), Springer-Verlag, Berlin, pp. 403–408, 1986.

[Hay90] B. HAYES, 'Anonymous one time signatures and flexible untraceable electronic cash', *Advances in Cryptology: AUSCRYPT'90*, Springer-Verlag, Berlin, pp. 294–305, 1990.

[HC86] R. N. HORSPOOL & G. V. CORMACK, 'Dynamic Markov modelling: a prediction technique', *Proceedings International Conference on the Systems Sciences*, Honolulu, HA, 1986.

[Hel79] M. E. HELLMAN, 'The mathematics of public-key cryptography', *Scientific American* **241**, 8, pp. 146–157, 1979.

[Hil00] D. HILBERT, 'Mathematische Probleme: Vortrag gehalten auf dem internationalen Mathematiker-Kongress zu Paris, 1970', *Nachrichten der Akademie der Wissenschaften in Göttingen. II. Mathematisch-Physikalische Klasse*, pp. 253–297, 1900.

[HM83] D. G. N. HUNTER & A. R. MCKENZIE, 'Experiments with relaxation algorithms for breaking simple substitution ciphers', *Computer Journal* **26**, 1, pp. 68–71, 1983.

[How93] PAUL GLOR HOWARD, *The Design and Analysis of Efficient Lossless Data Compression Systems*, PhD Thesis, Dept. of Computer Science, Brown University, Providence, RI 02912–1910, 1993.

[HS91] S. HABER & W. S. STORNETTA, 'How to time-stamp a digital document', *J of Cryptology* **3**, 2, pp. 99–112, 1991.

[Huf52] D. A. HUFFMAN, 'A method for the construction of minimum-redundancy codes', *Proceedings Institute of Electrical and Radio Engineers* **40**, 9, pp. 1098–1101, 1952.

[ICR95] SEAN A. IRVINE, JOHN G. CLEARY, & INGRID RINSMA-MELCHERT, 'The subset sum problem and arithmetic coding', *New Zealand Computer Science Research Students Conference Proceedings*, University of Waikato, Hamilton, NZ, 1995.

[Irv73] WASHINGTON IRVING, *The Alhambra*, AMS Press, NY, 1973.

[Jak85] M. JAKOBSSON, 'Compression of character strings by an adaptive dictionary', *BIT* **25**, 4, pp. 593–603, 1985.

[JB90] CEES J. A. JANSEN & DICK E. BOEKEE, 'A binary sequence generator based on Ziv-Lempel source coding', *Advances in Cryptology: AUSCRYPT'90*, Lecture Notes in Computer Science (453), Springer-Verlag, Berlin, 1990.

[JJ68] D. JAMISON & K. JAMISON, 'A note on the entropy of partially-known languages', *Information and Control* **12**, pp. 164–167, 1968.

[Jon88] D. W. JONES, 'Application of splay trees to data compression', *Communications of the ACM* **31**, 8, pp. 996–1007, 1988.

[JS95] PHILIPPE JACQUET & WOJCIECH SZPANKOWSKI, 'Asymptotic behaviour of the Lempel-Ziv parsing scheme and digital search trees', *Theoretical Computer Science* **144**, 1995.

[Kah66] DAVID KAHN, *The Codebreakers: The Story of Secret Writing*, Weidenfeld and Nicolson, London, 1966.

[Kan55] IMMANUEL KANT, *The Critique of Pure Reason*, translated by J. M. D. Meiklejohn, *Encyclopaedia Britannica*, Chicago, 1955.

[Kar72] RICHARD M. KARP, 'Reducibility among combinatorial problems', *Complexity of Computer Computations*, Plenum Press, NY, 1972.

[KBD89] SHMUEL T. KLEIN, ABRAHAM BOOKSTEIN, & SCOTT DEERWESTER, 'Storing text retrieval systems on CD-ROM: compression and encryption considerations', *ACM Transactions on Information Systems* **7**, 3, pp. 230–245, 1989.

[KF67] H. Kučera & W. N. Francis, *Computational Analysis of Present-Day American English*, Brown University Press, Providence, RI, 1967.

[KGV83] S. Kirkpatrick, C. D. Gelatt, & M. P. Vecchi, 'Optimization by simulated annealing', *Science* **220**, pp. 671–680, 1983.

[Kil90] J. Kilian, *Uses of Randomness in Algorithms and Protocols*, MIT Press, Cambridge, 1990.

[Kin94] John C. King, 'An algorithm for the complete automated cryptanalysis of periodic polyalphabetic substitution ciphers', *Cryptologia* **18**, 4, pp. 332–355, 1994.

[Kip88] Rudyard Kipling, *The Jungle Book*, World Book, Chicago, 1988.

[Kle90] D. V. Klein, "Foiling the cracker': a survey of, and implications to, password security', *Proceedings of the USENIX UNIX Security Workshop*, pp. 5–14, 1990.

[Knu69] Donald E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1969.

[Knu81] Donald E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, (2nd edition), Addison-Wesley, Reading, MA, 1981.

[Knu85] Donald E. Knuth, 'Dynamic Huffman coding', *J of Algorithms* **6**, 2, pp. 163–180, 1985.

[Kra92] H. Krawczyk, 'How to predict congruential generators', *J of Algorithms* **13**, 4, pp. 527–545, 1992.

[KS77] Maurice Kendall & A. Stuart, *The Advanced Theory of Statistics, Volume 1, Distribution Theory* (4th edition), Charles Griffin and Company Ltd., London, 1977.

[KZ95] E. Koch & J. Zhao, 'Embedding robust labels into images for copyright protection', *Proceedings of 1995 IEEE Workshop on Nonlinear Signal and Image Processing*, 1995.

[Lan84] G. G. Langdon, 'An introduction to arithmetic coding', *IBM J Research and Development* **28**, 2, pp. 135–149, 1984.

[Law59] D. H. Lawrence, *Lady Chatterley's Lover*, Grove Press, NY, 1959.

[Lem71] Abraham Lempel, 'm-ary closed sequences', *J Combinatorial Theory* **10**, pp. 253–258, 1971.

[Ler88] Gaston Leroux, *The Phantom of the Opera*, Perennial Library, NY, 1988.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., & L. Lovász, 'Factoring polynomials with rational coefficients', *Mathematische Annalen* **261**, pp. 515–534, 1982.

[LLMP90] A. K. Lenstra, H. W. Lentstra, Jr., M. S. Manasse, & J. M. Pollard, 'The number field sieve', *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, pp. 564–572, 1990.

[LO85] J. C. LAGARIAS & A. M. ODLYZKO, 'Solving low-density subset sum problems', *J ACM* **32**, 1, pp. 229–246, 1985.

[Lon83] JACK LONDON, *The Call of the Wild*, Chatham River Press, NY, 1983.

[Luc88] MICHAEL LUCKS, 'A constraint satisfaction algorithm for the automated decryption of simple substitution ciphers', *Advances in Cryptology: CRYPTO'88*, Lecture Notes in Computer Science (403), Springer-Verlag, Berlin, 1988.

[LV93] MING LI & PAUL VITÁNYI, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, NY, 1993.

[MA78] KENNETH L. MANDERS & LEONARD ADLEMAN, 'NP-complete decision problems for binary quadratics', *J of Computer and System Sciences* **16**, pp. 168–184, 1978.

[Mar34] M. H. MARTIN, 'A problem in arrangements', *Bulletin American Mathematical Soc.* **40**, pp. 859–864, 1934.

[Mat70] JU. V. MATIJASEVIČ, 'Enumerable sets are Diophantine', *Dokl. Akad. Nauk SSSR* **191**, pp. 279–282 (in Russian), *Soviet Mathematical Doklady* **11**, pp. 354–357 (in English), 1970.

[Mau91] U. M. MAURER, 'A universal statistical test for random bit generators', *Advances in Cryptolog: CRYPTO'90*, Lecture Notes in Computer Science (537), Springer-Verlag, Berlin, pp. 409–420, 1991.

[Mau92] U. M. MAURER, 'A universal statistical test for random bit generators', *J of Cryptology* **5**, 2, pp. 89–106, 1992.

[Max94] N. F. MAXEMCHUK, 'Electronic document distribution', *AT&T Technical J* **73**, 5, pp. 73–80, 1994.

[MH78] R. C. MERKLE & M. E. HELLMAN, 'Hiding information and signatures in trap-door knapsacks', *IEEE Transactions on Information Theory* **24**, pp. 525–530, 1978.

[Mof87] A. MOFFAT, 'Word based text compression', Research Report, Dept. of Computer Science, University of Melbourne, Parkville, Victoria, Australia, 1987.

[Mof88] A. MOFFAT, 'A note on the PPM data compression algorithm', Research Report 88/7, Dept. of Computer Science, University of Melbourne, Parkville, Victoria, Australia, 1988.

[Mof90] A. MOFFAT, 'Implementing the PPM data compression scheme', *IEEE Transactions on Communications* **38**, pp. 1917–1921, 1990.

[MR75] JU. V. MATIJASEVIČ & JULIA ROBINSON, 'Reduction of an arbitrary Diophantine equation to one in 13 unknowns', *Acta Arithmetica* **27**, pp. 521–553, 1975.

[MT79] C. H. MEYER & W. L. TUCHMAN, 'Design considerations for cryptography', *Proceedings of the NCC* **42**, Montvale, AFIPS Press, NJ, pp. 594–597, 1979.

[MW84] V. S. MILLER & M. N. WEGMAN, 'Variations on a theme by Ziv and Lempel', *Combinatorial algorithms on words*, NATO-ASI Series, F12, Springer-Verlag, Berlin, pp. 131–140, 1984.

[NBS77] NATIONAL BUREAU OF STANDARDS, 'Data encryption standard', *Federal Information Processing Standard*, US Dept. of Commerce, FIPS PUB 46, Washington DC, 1977.

[NBS80] NATIONAL BUREAU OF STANDARDS, 'DES modes of operation', *Federal Information Processing Standard*, US Dept. of Commerce, FIPS PUB 81, Washington DC, 1980.

[NWM94] CRAIG G. NEVILL-MANNING, IAN H. WITTEN, & DAVID L. MAULSBY, 'Compression by induction of hierarchical grammars', *Proceedings Data Compression Conference 1994*, Snowbird, Utah, 1994.

[NZPSA93] NEW ZEALAND PUBLIC SERVICES ASSOCIATION, *The Privacy Act 1993*, 1993

[Pap94] CHRISTOS H. PAPADIMITRIOU, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[Pat87] W. PATTERSON, *Mathematical Cryptology for Computer Scientists and Mathematicians*, Rowman and Littlefield, NJ, 1987.

[Pet92] I. PETERSON, 'Monte Carlo physics: a cautionary lesson', *Science News* **142**, 25, p. 422, 1992.

[Pfl89] C. P. PFLEEGER, *Security in Computing*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[PM88] STEPHEN K. PARK & KEITH W. MILLER, 'Random number generators: good ones are hard to find', *Communications of the ACM* **31**, 10, pp. 1192–1201, 1988.

[PR79] SHUMEL PELEG & AZRIEL ROSENFELD, 'Breaking substitution ciphers using a relaxation algorithm', *Communications of the ACM* **22**, 11, 1979.

[PW90] B. PFITZMANN & M. WAIDNER, 'Formal aspects of fail-stop signatures', Fakultät für Informatik, University Karlsruhe, Deutschland, Report 22/90, 1990.

[Ree46] D. REES, 'Note on a paper by I. J. Good', *J London Mathematical Society* **21**, pp. 169–172, 1946.

[Ris76] J. J. RISSANEN, 'Generalized Kraft inequality and arithmetic coding', *IBM J Research and Development* **20**, pp. 198–203, 1976.

[Ris79] J. J. RISSANEN, 'Arithmetic codings as number representations', *Acta Polytech. Scand. Math.* **31**, pp. 44–51, 1979.

[RPE81] M. RODEH, V. R. PRATT, & S. EVEN, 'Linear algorithm for data compression via string matching', *J ACM* **28**, 1, pp. 16–24, 1981.

[RSA78] R. RIVEST, A. SHAMIR, & L. M. ADLEMAN, 'A method for obtaining digital signatures and public-key cryptosystems', *Communications of the ACM* **21**, 2, pp. 120–126, 1978.

[Rub79b] F. RUBIN, 'Cryptographic aspects of data compression codes', *Cryptologia* **3**, 4, pp. 202–205, 1979.

[Rue92] R. A. RUEPPEL, 'Stream ciphers', *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, pp. 65–134, 1992.

[RW84] J. A. REEDS & B. J. WEINBERGER, 'File security and the UNIX crypt command', *AT&T Technical J* **63**, 8, pp. 1673–1683, 1984.

[Sal88] G. SALTON, *Automatic Text Processing*, Addison-Wesley, NY, 1988.

[Sal90] A. SALOMAA, *Public-Key Cryptography*, Springer-Verlag: Berlin, 1990.

[Sch77] BRUCE R. SCHATZ, 'Automated analysis of cryptograms', *Cryptologia* **1**, 2, pp. 116–142, 1977.

[Sch94] BRUCE SCHNEIER, *Applied Cryptography*, Wiley, NY, 1994.

[Sed88] ROBERT SEDGEWICK, *Algorithms* (Second Edition), Addison-Wesley, Reading, MA, 1988.

[SH95] C. P. SCHNORR & H. H. HÖRNER, 'Attacking the Chor-Rivest cryptosystem by improved lattice reduction', *Advances in Cryptology: EUROCRYPT'95*, Lecture Notes in Computer Science (921), Springer-Verlag, Berlin, pp. 1–12, 1995.

[Sha48] C. E. SHANNON, 'A mathematical theory of communication', *Bell System Technical J* **27**, pp. 379–423; 623–656, 1948.

[Sha49] C. E. SHANNON, 'Communication theory of secrecy systems', *Bell System Technical J* **28**, pp. 656–715, 1949.

[Sha51] C. E. SHANNON, 'Prediction and entropy of printed English', *Bell System Technical J* **30**, pp. 50–64, 1951.

[Sha78] A. SHAMIR, 'A fast signature scheme', MIT Library for Computer Science, Technical Memorandum MIT/LCS/TM-107, 1978.

[Sha79a] A. SHAMIR, 'On the cryptocomplexity of knapsack systems', *Proceedings of the 11th ACM Symposium on Theory of Computing*, pp. 118–129, 1979.

[Sha79b] A. SHAMIR, 'How to share a secret', *Communications of the ACM* **24**, 11, pp. 612–613, 1979.

[Sha81] A. SHAMIR, 'On the generation of cryptographically strong pseudo-random sequence', *Eighth International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science (62), Springer-Verlag, Berlin, 1981.

[Sha82] A. SHAMIR, 'A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem', *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*, pp. 145–152, 1982.

[Sho94] PETER W. SHOR, 'Algorithms for quantum computation: discrete logarithms and factoring', *Proceedings 35th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, pp. 124–134, 1994.

[Sil87] R. D. SILVERMAN, 'The multiple polynomial quadratic sieve', *Mathematics of Computation* **48**, 177, pp. 329–339, 1987.

[Sim84] G. J. SIMMONS, 'The prisoner's problem and the subliminal channels', *Advances in Cryptology: CRYPTO'83*, Plenum Press, NY, pp. 51–67, 1984.

[Sim85] G. J. SIMMONS, 'The subliminal channel and digital signatures', *Advances in Cryptology: EUROCRYPT'84*, Lecture Notes in Computer Science (209), Springer-Verlag, Berlin, pp. 364–378, 1985.

[Sin66] A. SINKOV, *Elementary Cryptanalysis*, Mathematics Association of America, Random House, NY, 1966.

[Ste60] ROBERT LOUIS STEVENSON, *Treasure Island*, Doubleday, NY, 1960.

[Ste92] BRUCE STERLING, *The Hacker Crackdown*, Bantam Books, NY, 1992.

[Sud93] TONY SUDBERY, 'Instant teleportation', *Nature* **362**, pp. 586–587, 15 April 1993.

[Sun88] SUN MICROSYSTEMS, *SunOS Reference Manual*, Part number 800–1751–10, 1988.

[Swi47] JONATHAN SWIFT, *Gulliver's Travels*, Grosset & Dunlap, NY, 1947.

[TC96] W. J. TEAHAN & JOHN G. CLEARY, 'The entropy of English using PPM-based models', *Proceedings Data Compression Conference*, Snowbird, Utah, pp. 53–62, 1996.

[TC97] W. J. TEAHAN & JOHN G. CLEARY, 'Models of English text', Data Compression Conference 97, Snowbird, Utah, 1997.

[Tea95] W. J. TEAHAN, 'Probability estimation for PPM', *New Zealand Computer Science Research Students Conference Proceedings*, University of Waikato, Hamilton, NZ, pp. 267–274, 1995.

[Til88] HENK C. A. VAN TILBORG, *An Introduction to Cryptology*, Kluwer Academic Press, Boston, 1988.

[Tis87] P. TISCHER, 'A modified Lempel-Ziv-Welch data compression scheme', *Australian Computer Science Communications* **9**, 1, pp. 262–272, 1987.

[Tol66] J. R. R. TOLKIEN, *The Hobbit*, Ballantine Books, NY, 1966.

[Tol68] LEO TOLSTOY, *War and Peace*, New American Library, 1968.

[TS85] R. E. TARJAN & D. D. SLEATOR, 'Self-adjusting binary search trees', *J ACM* **32**, 3, pp. 652–686, 1985.

[Twa23] MARK TWAIN, *The Adventures of Huckleberry Finn*, Gabriel Wells, NY, 1923.

[Ver60] JULES VERNE, *From the Earth to the Moon*, Dover, NY, 1960.

[Vit86] J. S. VITTER, 'Two papers on dynamic Huffman codes', Technical Report CS–83–13, Brown University Computer Science, Providence, RI (revised), 1986.

[Wal91] GREGORY K. WALLACE, 'The JPEG still picture compression standard', *Communications of the ACM* **34**, 4, pp. 30–44, 1991.

[WB91] IAN H. WITTEN & TIMOTHY C. BELL, 'The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression', *IEEE Transactions on Information Theory* **37**, 4, pp. 1085–1094, 1991.

[WC88] IAN H. WITTEN & JOHN G. CLEARY, 'On the privacy afforded by adaptive text compression', *Computers & Security* **7**, (4), pp. 397–408, 1988.

[Wei96] JANIS WEINER, *Cryptograms*, Stavrolex Publications, Fort Washington, PA, Fall 1996.

[Wel84] T.A. WELCH, 'A technique for high-performance data compression', *IEEE Computer* **17**, 6, pp. 8–19, June 1984.

[WGI91] D. WHITING, G. GEORGE, & G. IVEY, 'Data compression apparatus and method', US Patent #5016009, 1991.

[Wie93] M. J. WIENER, 'Efficient DES key search', *Advances in Cryptology: CRYPTO'93*, Lecture Notes in Computer Science (773), Springer-Verlag, 1993.

[Wil59] E. A. WILLIAMS, *An Invitation to Cryptograms*, Simon and Schuster, NY, 1959.

[Wil84] H. S. WILF, 'Backtrack: An $O(1)$ average time algorithm for the graph colouring problem', *Information Processing Letters* **18**, pp. 119–122, 1984.

[Wil94] WILLIAM J. WILSON, 'Chinks in the armor of public key cryptosystems', University of Waikato, Technical Report 94/3, March 1994.

[Wit82] IAN H. WITTEN, *Principles of Computer Speech*, Academic Press, London, 1982.

[WNC87] IAN H. WITTEN, RADFORD M. NEAL, & JOHN G. CLEARY, 'Arithmetic coding for data compression', *Communications of the ACM* **30**, 6, pp. 520–540, 1987.

[Wri39] E. V. WRIGHT, *Gadsby*, Kassel Books, 1939.

[WW90] K. WONG & S. WATT, *Managing Information Security*, Elsevier Science Publishers, Barking, 1990.

[WZ94] A. D. WYNER & J. ZIV, 'The sliding-window Lempel-Ziv algorithm is asymptotically optimal', *Proceedings of the IEEE* **82**, 6, pp. 872–877, 1994.

[Zip39] G. K. ZIPF, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, 1939.

[ZL77] J. ZIV & A. LEMPEL, 'A universal algorithm for sequential data compression', *IEEE Transactions on Information Theory* **23**, 3, pp. 337–343, 1977.

[ZL78] J. ZIV & A. LEMPEL, 'Compression of individual sequences via variable-rate coding', *IEEE Transactions on Information Theory* **24**, 5, pp. 530–536, 1978.

Index