The Cryptocomplexity of Four Difficult Problems

by

Luke James OConnor

A thesis

presented to the University of Manitoba

in partial fulfillment of the

requirements for the degree of

Master of Science

in

Computer Science.

Winnipeg, Manitoba, January 1988.

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-44114-3

THE CRYPTOCOMPLEXITY OF FOUR DIFFICULT PROBLEMS

BY

LUKE JAMES OCONNOR

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

© 1988

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

I hereby declare that I am the sole author of this thesis.

I authorise the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Luke James OConnor.

I further authorise the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Luke James OConnor.

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below and give address and date.

ACKNOWLEDGEMENTS

I wish to acknowledge the efforts of my supervisor Mr. H. C. Williams and Mr. D. R. Stinson, whose patient persistence brought the first draft of this thesis to its present from. In particular, Hugh assisted in the sections that were concerned with number theory problems, and in general, he administered the adroit use of the comma. Doug's assistance was primarily in the sections concerned with algorithms and complexity. His examination was very thorough, and he also inspired many enlightening discussions. Also I would like to thank Mr. Michael Rogers for always having the time to fix faulty terminals and patch up so called reliable software.

Preface

This thesis is concerned with the nature of so called hard or intractable problems and their relevance to cryptography. We have restricted our attention to four such problems, namely: factoring, the discrete logarithm problem, the knapsack problem and determining quadratic residuosity. The most noteworthy cryptosystems are derived from one or a combination of these problems. In this thesis we will examine how well these problems adapt to cryptography, and how the notion of intractability models the security of a public key cryptosystem.

The thesis is divided into 5 chapters. The first chapter, consisting of five sections, is introductory. Most of the notation and fundamentals of complexity theory needed for reading the thesis are explained in these sections. In §1 we define a hard cryptographic transformation and show how an intractable function will fulfill these requirements. In §2 we introduce one way functions and public key cryptosystems. Algorithm complexity is explained in §3, along with the running time of an algorithm. Section 4 deals with complexity classes and the general cryptographic problem. In §5 we define a reduction, which will be used throughout the thesis to compare the difficulty of problems.

In chapter 2, section 6 formally defines the problems that will be studied, and what we expect to learn from such a study. In §7 we begin to analyse the complexity of public key cryptography specifically. The security ratio of a system is described. In §8 we examine why certain problems are considered to be hard, and how the designer can select a hard instance of a hard problem.

In chapter 3, a section is devoted to each of the intractable problems previously mentioned. First, the complexity of the general problem is considered, an example of a general algorithm, and then we look at some cryptosystems based on a specific instance of the general problem. Section 9 is on factoring, §10 on quadratic residuosity, §11 addresses the discrete logarithm problem and §12 the knapsack problem.

In chapter 4 we deal consider how well a worst case analysis models the security of a cryptosystem. In §13 we consider the power of random algorithms. How successful is probabilistic computation? In §14 we attempt to give an analysis of the general cryptographic problem, how difficult it is expected to be and how difficult it is compared to other problems. Is the problem **NP**–hard? Or **NP**–complete? We also consider partial solutions to the general cryptanalytic problem and bit security. In §15 we consider one way functions more closely and the information they conceal. Lastly, in chapter 5, we show that some suitably restricted general problems are equivalent in difficulty to breaking certain cryptosystems.

CONTENTS

CHAPTER 1 COMPLEXITY AND HARD TRANSFORMATIONS 1

Section 1 *Cryptography and Hard Problems* 1

Section 2 *Intractable functions* 9

Section 3 *Complexity of Algorithms* 11

Section 4 *The Theory of Hard Problems* 17

Section 5 *Reductions and Equivalent Problems* 24

CHAPTER 2 CRYPTOCOMPLEXITY 27

Section 6 *Some Hard Cryptographic Problems* 27

Section 7 *Public Key Complexity* 31

Section 8 *Search Problems and Worst Case Analysis* 38

CHAPTER 3 SOME POTENTIAL ONE WAY FUNCTIONS 42

Section 9 *The Problem of Factoring* 42

9.1 *The Continued Fraction method* 45
9.2 *Factoring and Security* 56

Section 10 *Quadratic Residuosity* 59

Section 11 *The Discrete Logarithm Problem* 70

11.1 *Algorithms for the Discrete Logarithm Problem* 70
11.2 *The Pohlig–Hellman Algorithm* 77
11.3 *Cryptosystems Based on the Discrete Logarithm Problem* 81

Section 12 *The Knapsack Problem* 85

12.1 *The general problem* 85
12.2 *Cryptosystems based on the knapsack problem* 88
12.3 *The Chor–Rivest Knapsack* 93

CHAPTER 4 THE THEORY AND PRACTICE OF SECURE SYSTEMS 95

Section 13 *Random Algorithms* 95

13.1 *How accurate is Worst Case analysis?* 95
13.2 *Average Case Analysis and Probabilistic Computation* 100
13.3 *Random Reductions* 106

Section 14 *Decision Problems and Reductions* 109

14.1 *Hard Decision Problems* 109
14.2 *Partial Solutions to the GCP* 110
14.3 *Bit Security* 112
14.4 *Reductions Amongst the Basic Problems* 119

Section 15 *One Way Functions* 126

CHAPTER 5 SPECIFIC PROBLEMS 133

16.1 *More on Quadratic Residuosity* 136
16.2 *RSA and Factoring* 141
16.3 *Factoring and the Discrete Logarithm Problem* 144

Section 17 CONCLUSION 146

References 153

CHAPTER 1 Complexity and Hard Transformations

SECTION 1 Cryptography and difficult problems

Cryptography is concerned with various methods that can be employed to establish and maintain secure communications between two parties. An encryption algorithm is a finite set of instructions that operate on a plaintext message to transform it into ciphertext, which should appear to be nothing more than a random collection of symbols. How random and unstructured the ciphertext appears will depend upon the particular encryption algorithm used. An encryption algorithm will define a family of transformations which will form an indexed set. Each member of the set is selected or referenced by a *key*, selected from the keyspace K. A cryptosystem consists of a general encryption algorithm E and its key k_1, and a general decryption algorithm D and its key k_2. If $k_1 = k_2$ or knowledge of one key easily leads to the other then the cryptosystem is said to be *symmetric*, otherwise it is said to *asymmetric*. Associated with each cryptosystem is a message space M and a cipherspace C such that

$$\forall m \in M, \exists c \in C \qquad E_{k1}(m) = c$$

$$\forall c \in C, \text{ and in the range of } E, \exists m \in M \qquad D_{k2}(c) = D_{k2}(E_{k1}(m)) = m$$

Decryption is the inverse operation of encryption. The flow of information is shown in Diagram 1. In some systems $C = M$ in which case E_{k1} is a permutation.

An opponent or eavesdropper can intercept the ciphertext c and is assumed to be familiar with the details of the general encryption algorithm used to produce the ciphertext. The opponent is assumed not to have access to the keying information of the cryptosystem, indeed it is knowledge of this keying information alone that distinguishes the system designer from an opponent.

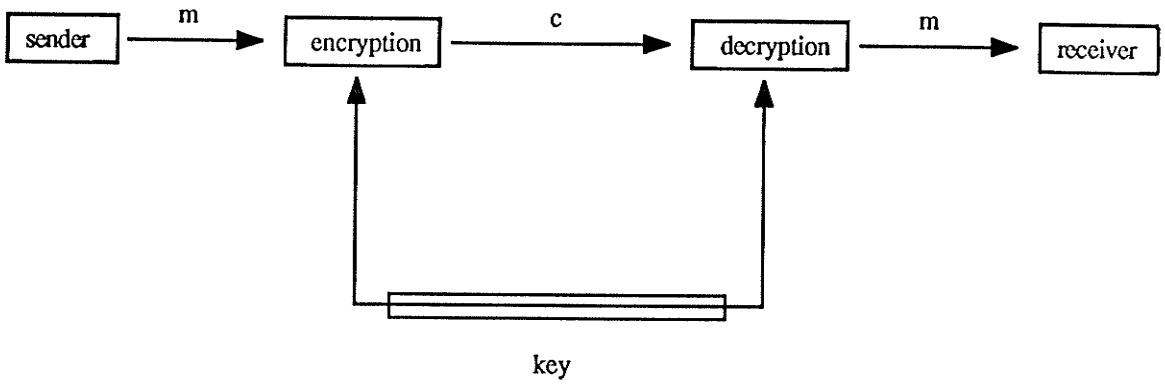


Diagram 1.1

Example 1.1. Hill encipherment. Let the message to be encrypted be $\mathbf{m} = (m_1, m_2, \ldots, m_n)$, and then divide the message into 3-grams, ie. ((m_1, m_2, m_3), $(m_4, m_5\, m_6)$ ... (m_i, m_{i+1}, m_{i+2})) adding characters if necessary to pad out a block of three characters. Next encode each letter into a numerical representation, where A= 0, B=1, ... , Y=24, Z=25. For example, suppose the message is

NO ONE GETS OUT OF HERE ALIVE

which when grouped into 3-grams and represented numerically becomes (13, 14, 14), (13, 4 , 6), (4, 19, 18), (14, 20, 19), (14, 5, 7), (4, 17, 4), (10, 11, 8), (22, 4, 25). The letter Z has been appended to complete the last 3-gram. The blanks were not encoded. Let T and T^{-1} be inverse transformations, defined by

$$T = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix} \quad \text{and} \quad T^{-1} = \begin{bmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{bmatrix}$$

where $t_{i,j}$, $1 \le i, j \le 3$ are the entries of T and $t^{-1}{}_{i,j}$, $1 \le i, j \le 3$ are the entries of T^{-1}. The message is encrypted by the rule

$$\mathbf{C} = \mathbf{m}T$$

where each 3-gram (m_k, m_{k+1}, m_{k+2}) is transformed to (c_k, c_{k+1}, c_{k+2}) by

$$c_{k+i} \equiv \sum_{j=1}^{3} m_{k+i} \cdot t_{i+1\,j} \mod 26 \qquad i = 0, 1, 2$$

and the encipherment of the example is (9, 13, 8), (7, 3, 18), (13, 24, 24), (23, 13, 13), (20, 11, 15), (24, 21, 9), (7, 4, 12), (21, 19, 7), which decodes to

JN IHD SNYY XNN UL PYVJ HEMVT

with the original spacing and the last Z discarded. Each 3-gram message is transformed into a corresponding 3-gram ciphertext. Decipherment of (c_k, c_{k+1}, c_{k+2}) is achieved through T^{-1},

$$m_{k+i} \equiv \sum_{j=1}^{3} c_{k+i} \cdot t^{-1}_{i+1\,j} \mod 26 \qquad i = 0, 1, 2$$

and the original message is recovered. •

A cryptosystem is said to be cryptanalysed if it is possible for an opponent to unambiguously decrypt an intercepted ciphertext. We define the general cryptanalytic problem (*GCP*) as

INSTANCE: *general cryptosystem E, D, K, M, C and* $c = E_{k1}(m)$;

PROBLEM: *determine the message m that was encrypted to give ciphertext c.*

The cryptanalyst knows everything except m and $k1$. The *GCP* does not require that the key be determined since it may not be necessary to recover the key to determine the plaintext message. There are two broad approaches to solving the *GCP*. On the one hand, if c is only expected to decrypt to one meaningful message, then a cryptanalyst can search the keyspace until $D_{k2}(c)$ gives the message. On the other hand, the cryptanalyst may attempt to use side information to facilitate an

illegal decryption. Side information is a general term for additional information an opponent can deduce beyond the description of the system. For example, an opponent may use the statistical properties of the plaintext language to his advantage, or use the context of the transmission to guess words that are expected to appear in the message (probable words). Also the format of a message and the protocol used to transmit messages may give the opponent some side information.

The adequacy of a cryptosystem can be partially certified by considering the security of the system in various scenarios that assume the cryptanalyst is in possession of a certain amount of information. In the *ciphertext only attack* the cryptanalyst has intercepted several ciphertexts but knows very little otherwise. This is the weakest threat to the system. A *known plaintext attack* is where the opponent can deduce some probable words in the ciphertext or may be able to acquire some actual plaintext–ciphertext pairs (m_i, c_i). A system that resists a known plaintext attack is considered to be secure (DES was accepted on this basis). Lastly, a *chosen plaintext attack* is where the opponent may submit an infinite amount of plaintext and obtain the corresponding ciphertexts. A system that resists this attack is very strong. Even so, these attacks are a very ad hoc approach of defining security. If a system is secure against a known plaintext attack, then this empirical analysis indicates that the work necessary to break the system exceeds the value of the information it protects.

Most cryptosystems cannot resist a cryptanalysis when the keyspace of the cryptosystem is subjected to a direct search. Hence the difficulty of the *GCP* cannot exceed the time required to search the keyspace. The cardinality of the keyspace may be finite but still very large. The designer can choose the size of the key such that a naive generate–and–test algorithm to search the keyspace will take thousands to billions of years to complete. An algorithm for a problem is said to *computationally infeasible* if the amount of resources (generally time and space) required to compute the algorithm is unrealistic or unreasonable. For example, a computation that would take several million years to execute while requiring half the energy of the sun at its disposal is certainly an unrealistic undertaking. On a less grand scale, even a few hundred years on a parallel machine is

considered an unreasonable amount of time to wait for a computation. The keyspace of a 20-bit key can be searched by a sequential machine with a microsecond instruction cycle in a day or so [KNU1]. A cryptanalyst could expend a modest amount of resources to solve the *GCP* for a system with a 20-bit key. It is the task of the designer to choose the size of the key appropriately so a brute force approach is computationally infeasible. The DES system uses a 56-bit key but there has been some protest that this provides inadequate security against a special purpose machine [HELL1]. In §7 the practical limits of computation are discussed again.

Over the centuries, designers have not been particularly troubled by the task of constructing cryptosystems for which the possible number of key settings is astronomically large. By the same token, cryptanalysts have taken great delight in demonstrating that a large keyspace is a necessary but not suffucient requirement for a cryptosystem to be secure.

Guaranteeing that there is an "insignificant" correlation between the plaintext and encrypted forms of a message has proved harder than expected. If the correlation between the two forms of a message is "significant" then an illegal decryption can be effected without requiring the keyspace to be searched. Ideally the encryption algorithm should obscure the meaning of the plaintext message totally. There are three interpretations of the effect of the encryption algorithm;

(1.1) given E and $c = E_{k1}(m)$ an opponent cannot unambiguously determine m ;

(1.2) given E and $c = E_{k1}(m)$ there is no way to determine m ; other than by searching the key space.

(1.3) given E, $c = E_{k1}(m)$ and k_1 it is computationally infeasible to determine m and/or k_2;

A cryptosystem for which (1.1) is true is called *unconditionally secure*, while a cryptosystem for which (1.3) is true is called *computationally secure*. An unconditionally secure system, as the name suggests, remains secure even in the presence of an opponent with an unlimited amount of computing resources. The one time pad, devised by Claude Shannon [SHAN1] [KAHN], is an example of an unconditionally secure system.

The cryptosystems that appeal to (1.2) as a basis of security are called classical or conventional systems. Both type (1.1) and type (1.2) systems are symmetric and type (1.3) systems are asymmetric, though they could be adjusted for use as symmetric system.

This thesis will concern itself with the task of finding encryption functions that are computationally difficult to invert. We are necessarily restricting ourselves to public key (asymmetric) systems. We observe that this is not the only notion of difficulty. Intuitively we may consider a problem difficult if there is no apparent way to solve it, even if we know the solution exists. This notion of difficulty seems reasonable, though vague, and is close to the colloquial usage of the word "difficulty". A test where the average mark is 52% is considered difficult, at least by those who wrote it. Alternatively, consider deeming a problem difficult if the solution is obvious in the sense that we know how to find this solution, yet the time required to find it is very large, say several lifetimes. For example, imagine that we were asked to determine the number of individual sediment particles on the floor of the Great Lakes using the algorithm that you count them one by one. Though this problem is easily stated and we know how to find the solution and that the solution must exist, it is tedious and laborious to solve. This is the abstract definition of difficulty employed by complexity theoreticians. For the better part it is a quantitative idea rather than qualitative one, since we are interested in the operational limits of computation, though both the intuitive and abstract definitions of difficulty state that a *difficult problem cannot be solved rapidly*. With the abstract definition of difficulty, exactly how rapidly a problem can be solved is measurable and commensurable with other difficult problems. For example, a 125 digit number could be factored in approximately 100 years using existing algorithms and machines.

Concerning the intuitive notion of difficulty, far less can be said. How difficult is it to invert the DES encryption function? There is no known method other than searching the keyspace. But that could change tommorrow, due to an insight, just as easily as an efficient algorithm for factoring could "materialise". More often than not, the symmetric systems that use the intuitive notion of difficulty have been broken. Such systems are hard because they have complicated and

intricate encryption algorithms that are difficult to analyse and invert. It is hoped that the attacks of the cryptanalyst will be fruitless and he will conclude that the only obvious method of attack is to search the keyspace. There are several ways of producing complicated encryption algorithms [SHAN] [DES], and they are generally based on non-commutative and/or non-linear operations. Shannon first suggested using substitutions and transpositions successively on a message to produce a good "mixing transformation". The designers of DES based their system on this mixing transformation.

An invertible linear transformation T from an n–bit message $\mathbf{m} = (m_1, m_2, \ldots, m_n)$ to an n–bit ciphertext $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ is a non–singular matrix

$$T = \begin{bmatrix} t_{1,1} & t_{1,2} & \cdot & \cdot & \cdot & t_{1,n} \\ t_{2,1} & t_{2,2} & \cdot & \cdot & \cdot & t_{2,n} \\ & & & & & \\ \cdot & & \cdot & \cdot & & \\ \cdot & & & \cdot & & \\ \cdot & & & & \cdot & \\ & & & & & \cdot \\ t_{n,1} & t_{n,2} & \cdot & \cdot & \cdot & t_{n,n} \end{bmatrix}$$

where

$$c_i \equiv m_1 t_{i,1} + m_2 t_{i,2} + \ldots + m_{n-1} t_{i,n-1} + m_n t_{i,n} \mod 2$$

Further for two messages m_1 and m_2, and a scalar u,

$$T(m_1 + m_2) = T(m_1) + T(m_2) \text{ and } T(um_1) = uT(m_1) \tag{1.1}$$

All other transformations that do not satisfy eq.(1.1) are non–linear. If the task of inverting a system can be reduced to the problem of solving n equations in n unknowns then the transformation is very weak. Linear systems, such as in example 1.1, are usually easily solved.

Individually, transpositions and substitutions are very weak, but they can alternated to form a powerful non–linear transformation. Most secure conventional encryption functions are based on non–linear transformations. We give an example of a non–linear ransformation in example 2.1.

Example 2.1 Table look-up; let $f(x)$ be the table look-up transformation and let f map from n-bit strings to n-bit strings. ie. f is a permutation. Let $n = c \times d$ so that x may be considered as consisting of precisely c lots of d-byte words (eg. if $n = 8 = 2 \times 4$ then 10110110 is made up of the 2 4-bytes 1011 and 0110). Given a random table with 2^d entries, let the transformation substitute each of the d-bytes by the table entry that corresponds to its binary representation, ie. replace 1011 with the 11th entry in the table. If the result is to be one-to-one then the 2^d table entries must include each integer in the range $[0, 2^{d-1}]$. This technique is used in the DES algorithm. •

See Evans *et al.* [EKW] for a further discussion of linear and non-linear systems. The designer must rely exlusively on the inherent strength of the non-linear function the system is based on, as the designer may have little idea of how the encryption might be inverted. The real security of these systems is certified by the length of time they resist concerted cryptanalysis.

For these non-linear systems the problem of inverting their encryption functions is ill-defined. What is meant by this is that there is no clearly stated problem whose solution will yield a cryptanalysis of a non-linear system. If no such clearly stated problem can be found then an opponent has no alternative but to search the keyspace. However an opponent will never be sure that *all* the possible methods of attack have been exhausted. The security of all systems reside in the difficulty of inverting the encryption function. On the one hand, the non-linear systems are designed so that inverting the encryption function is a *difficult problem to define*, while on the other, for a computationally secure system the problem of inverting the encryption function is precisely defined but *difficult to solve*. Here we see how the varying notions of difficulty have been applied.

Computationally secure systems are based on problems that are computationally infeasible. If the best known or most efficient algorithm for a problem is computationally infeasible, then the problem is considered to be *intractable*. Intractable problems are accepted as problems that are difficult to compute. For public key cryptography, the designers will be searching for problems that can be incorporated into the construction of a cryptosystem such that inverting the encryption function, without the keying information (or solving the respective *GCP*), is a difficult, and hopefully, intractable task.

SECTION 2 Intractable Functions

In the seminal paper *New Directions in Cryptography*, Diffie and Hellman introduce the concept of functions that are computationally infeasible to invert as a basis for secret exchanges. Basing secure communications on intractable problems was an extension of the password problem that came to attention some years before. Imagine that users are required to identify themselves by giving a password that authorises the use of their computer account. Somewhere in the system there must be a file that contains the passwords of all users that is searched when a user logs on. Any user who gains access to the password file can use this information to impersonate any other user.

One solution to this problem is to restrict access to the password file. But various administrators and other keepers of the system, due to necessity and their position in the user heirachy, will still require access to the protected password file[1]. Another approach is to encrypt the contents of the password file so that the passwords are not stored explicitly. Let each user i possess a password x_i and consider storing $f(x_i)$ in the password file where f is an easily computed function but difficult to invert. The function f is said to be *one way function* if

[1]The general problem of ensuring that a user has no access rights to an object, say a file or a command, is undecidable. Harrison, Ruzzo and Ullman [HRU] demonstrate that the Halting Problem [HaU] is reducible to the problem of determining the access rights of a user with a finite set of commands. The problem is decidable if the command set is suitably restricted.

(2.1) $f(x)$ is easily computed for all x in the domain of f;

(2.2) given almost any y such that $y = f(x)$ for some x, then it is computationally infeasible to find x.

Evans *et al.* [EKW] have presented an intricate non-linear algorithm as a candidate one way function. Purdy [PUR] has also suggested that $f(x)$ may be a large polynomial modulo a prime. Diffie and Hellman believe that discrete exponentiation may also be a one way function.

A one way function conforms to the notion of an encryption algorithm that is difficult to invert. If the cryptosystem is to be useful as well as secure then the intended reciver of a transmission must be able to easily decrypt the ciphertext (ie. invert the encryption algorithm). We say that the function f is a *trapdoor one way function* if

(2.3) $f(x)$ is easily computed for all x in the domain of f;

(2.4) given almost any y such that $y = f(x)$ for some x, it is computationally infeasible to find x unless certain privileged information about the construction of f is known. If this information is known then x is easily found.

The use of trapdoors is the extension that Diffie and Hellman introduced so that one way functions could be used to construct cryptosystems. Let C be the cipherspace and M the message space. The keyspace K consists of ordered pairs $(k1, k2)$. A public key cryptosystem (PKC) is defined as

$$E_{k1} : \{M\} \longrightarrow \{C\}$$
$$D_{k2} : \{C\} \longrightarrow \{M\}$$

(2.5) for all $(k1, k2) \in K : D_{k2}$ is the inverse of E_{k1};

(2.6) for all $(k1, k2) \in K$ and $m \in M$: $c = E_{k1}(m)$ and $D_{k2}(c)$ are easily computed;

(2.7) for almost all $(k1, k2) \in K$: it is computationally infeasible to derive E_{k1} from any algorithm equivalent to D_{k2} ;

(2.8) for all $(k1, k2) \in K$: the pairs E_{k1} and D_{k2} are easily computed;

The encryption function E_k is a trapdoor one way function and the decryption algorithm D_k contains the necessary privileged information to allow the plaintext message to be easily recovered. Condition (2.7) states that if the keyed encryption algorithm is revealed then determining the keyed decryption algorithm (or any equivalent algorithm) from this information alone, is computationally infeasible. The flow of information in a PKC is shown in Diagram 2.1.

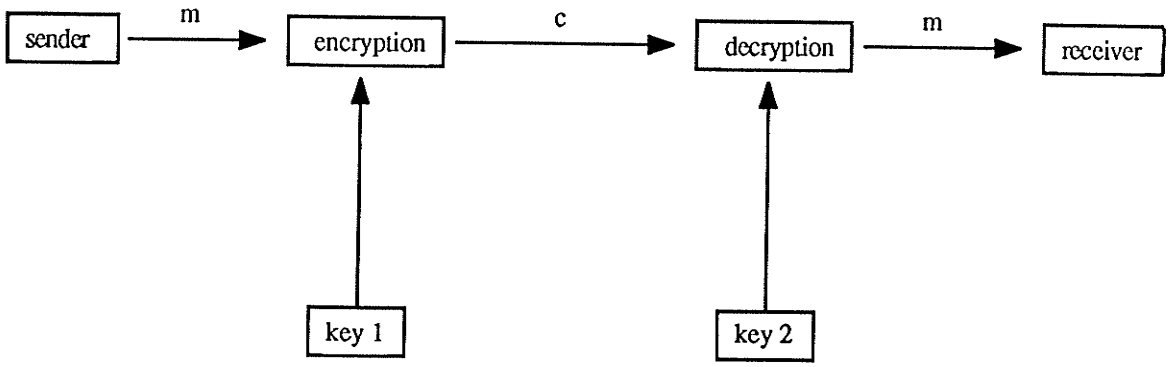


Diagram 2.1

SECTION 3 The Complexity of Algorithms

In classifying problems we make our first distinction between computable (ie. decidable) and uncomputable problems. Intractability is concerned with the next level of distinction, which is the delineation between what can be computed in a feasible amount of time and what cannot. Complexity theory provides a theoretical framework to define the concept of an intractable or difficult problem. Simply put, the more resources a given problem consumes in its solution the harder it is considered to be. The difficulty of an algorithm is gauged by measuring its resource consumption; it is grouped with other algorithms that have a similar resource usage. By establishing classes of problems that require a similar amount of work to solve, we can identify a

set of problems that are easily solved and define all problems that are *not easily solved* to be intractable.

If a problem instance is to be solved, the insatnce must be encoded into a form that can be used as input to an algorithm or program. The length of this encoding is called the size of the problem instance, which will be denoted by n. For example, consider the problem of multiplying two integers a_1 and a_2. One encoding is to write both integers as binary numbers. The size of the problem instance is then size$(a_1 * a_2) = n = \lceil log_2 a_1 \rceil + \lceil log_2 a_2 \rceil + 1$, where we have assumed that the integers are separated by a space and $\lceil x \rceil$ denotes the least integer greater than x. A complexity measure for a problem is an expression that indicates how the running time of the best algorithm for a problem grows as a function of the size of the problem. The growth rate of this expression is the (time) complexity of the problem.

A growth rate may contain several terms involving the problem size n, as for example in the cubic function $f(n) = 7n^3 + 5n^2 + 2$. This expression is an example of a running time function for an algorithm . The complexity of an algorithm is an asymptotic measure, which is defined to be the dominant term of the running time function $f(n)$, as n tends to infinity, (n^3 in the above example). If $f(n)$ is an upper bound on the running time of problem p then there is no instance of problem p that requires more than $f(n)$ elementary operations to solve. Similarly, if $f(n)$ is a lower bound on problem p, then there is no instance of p that requires less then $f(n)$ operations to be solved. The big "O" notation is used to express upper bounds. The complexity of $f(n)$ is said to be $O(g(n))$ if there exist rationals c and n_0 such that

$$|f(n)| \leq c \cdot g(n), \quad all\ n > n_0 .$$

The little "o" notation is also used to define upper bounds of functions. A function $f(n)$ is said to be $o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(x)}{g(x)} = 0 .$$

For example, x is $o(x^2)$. We have a similar notation for denoting lower bounds. The complexity of $f(n)$ is $\Omega(g(n))$ if there exist rationals c and n_0 such that

$$|f(n)| \geq c \cdot g(n), \quad all\ n > n_0 .$$

And finally, the complexity of $f(n)$ is $\Theta(g(n))$ if there exists rationals n_0, c_1, c_2 such that

$$c_1 \cdot g(n) \leq |f(n)| \leq c_2 \cdot g(n) \quad all\ n > n_0 .$$

Some typical values of $g(n)$ are $\Theta(n^2)$, $\Theta(n\ logn)$, $\Theta(2^{n!})$. If an algorithm is $\Theta(g(n))$ then the upper and lower bounds of the running time for the algorithm can be expressed as $k \cdot g(n)$, $k > 0$. If the lower bound for a problem p is known to be $\Omega(f(n))$ then algorithm AL for problem p is said to be optimal if its complexity is $\Theta(f(n))$.

If the growth rate of a problem can be expressed as a polynomial in the size of the problem n, ie. $f(n) = a_k n^k + a_{k-1} n^{k-1} \ldots + a_1 n + a_0$, then the problem has a polynomial growth rate and is said to be *solvable in polynomial time.* A growth rate $f(n)$ which is $O(c^{o(n)})$ for $c > 0$ is said to grow subexponentially, (eg. $2^{\log\log n}$). All other growth rates are said to exponential. Table 3.1 shows some growth rates.

f(n)	10	20	30
n	10	20	30
n^2	100	400	900
n^{10}	1.0×10^{10}	1.024×10^{13}	5.904×10^{14}
n^{30}	1.0×10^{30}	1.0737×10^{39}	2.059×10^{44}
$5^{\sqrt{n}}$	1.623×10	1.3362×10^{3}	6.7362×10^{3}
2^n	1024	1048576	1073741824
5^n	9765625	9.573×10^{13}	9.313×10^{20}

Table 3.1

Improving the efficiency of an algorithm means closing the gap between the current upper and lower bounds for the problem it solves. For example if a problem is known to be $\Omega(n^2)$ and the best known algorithm for the problem is $O(n^3)$ then there still maybe a possible order of magnitude in efficiency to be gained. Lower bounds are useful in that they indicate how much efficiency we may expect to gain at best from improving an algorithm. If a lower bound is not known, then reducing the upper bound is a gain in efficiency, but the magnitude of the improvement is not known. Nevertheless, in the absence of a known lower bound, an improvement from an exponential running time to a polynomial running time is considered a noteworthy gain in efficiency.

A computation where each step is uniquely determined is called deterministic. The following algorithm for performing a binary search on a sorted n-element list L is deterministic:

Example 3.1. Binary search algorithm; looking for the key X in the list L;

1. $k \leftarrow 1; m \leftarrow n$
2. **while** $k \leq m$ **do**
3. $j \leftarrow \lfloor (k+m)/2 \rfloor$
4. **if** $X = L$ **then return** j
5. **if** $X < L$ **then** $m \leftarrow j-1$
   else $k \leftarrow j+1$
6. **end_do**
7. $j \leftarrow 0$; •

The current program state (ie. current instruction and values of the variables) will uniquely determine which step of the algorithm will be executed next. For example the if statement at line 5 will choose the alternative according to the value of X. A deterministic Turing machine (DTM) is an abstract computation model for deterministic algorithms. A DTM consists of a set of internal states Q, a read/write (r/w) head, a set of state transitions δ and an input tape. The input tape is infinite and consists of tapes squares labelled ... –2, –1, 0, 1, 2, ... , and the r/w head points to one tape square at any time. A configuration of the DTM is a pair (s_i, q_j) where s_i is the current tape symbol pointed to by the r/w head and q_j is the current internal state of the DTM. The DTM is initially in the start state q_0 with the r/w head pointing to the first tape symbol, say s_1. Then (s_1, q_0) is the intial configuration. A DTM executes its program by moving through a sequence of internal states, beginning with q_0, until a designated halt state is reached. The flow of control is determined by the set of state transitions δ. The set δ consists of quintuples of the form (s_i,q_j,s_k,q_m,M), which says that if (s_i, q_j) is the current configuration, then write s_k on the current tape square, make q_m the current internal state and reposition the r/w head according to M. Let $M \in \{+1, -1\}$, where $M = +1$ indicates that the r/w head is to be moved one square to the right, and moved one square to the left if $M = -1$. One step of the computation is said to be completed after one quintuple has been executed. See [PaS] [HaU] [LaP] for examples of DTM programs.

The input to a DTM is written from left to right, beginning at tape square 1. If the input requires n tape squares to be specified, then the size of the input is n. The (time) complexity of a DTM that computes a program H, whose instruction are encoded in δ, is defined as

$$T_H(n) = \max \left\{ w : \text{there is a valid input sequence } x, \text{ size}(x) = n, \text{ such that } H \text{ requires } w \text{ steps to compute on input } x. \right\}.$$

The program H is called a polynomial time DTM program if there exists a polynomial $p(x)$ such that for all $n \in Z^+$, $T_H(n) \leq p(n)$. For a more detailed discussion of DTMs see [GaJ] [HaU].

The class **P** consists of those problems that can be solved polynomial time DTM (program). The problems of finding the largest element in a list if n elements and inverting an $n \times n$ matrix are members of the class **P**. It is a generally accepted thesis that the class **P** is equivalent to the collection of feasibly computed problems. There are several reasons to believe this claim. Recall that complexity theory deals with asymptotic quantities. Even though Table 3.1 shows the polynomial function $n30$ growing very quickly, if n were large enough the exponential functions would begin to grow more rapidly than their polynomial counterparts (ie. $2^n < 2^{100}$ for only finitely many n but for infinitely many n the converse holds). Table 3.2 shows how increased computing power will affect the number of problems that can be solved in a fixed time period, for various running time functions. Very little is gained in the cases where the running time function is exponential.

$f(n)$	present computer	100 times faster	1000 times faster
n^2	N_1	$10N_1$	$31.6N_1$
n^3	N_2	$4.64N_2$	$10N_2$
n^5	N_3	$2.5N_3$	$3.98N_4$
2^n	N_4	$N_4+6.64$	$N_4+9.97$
3^n	N_5	$N_5+4.19$	$N_5+6.29$

Table 3.2*

The class **P** is closed under certain operations. Let $f(x)$ and $g(x)$ be polynomials. Then each of the following expressions is also a polynomial: $f(g(x))$, $g(x)+f(x)$ and $f(x)g(x)$. Let AL and BL be algorithms where $AL, BL \in$ **P**. The following algorithms, $C1$, $C2$ and $C3$, are also members of **P**;

* [GaJ, p8].

C1 : running *AL* followed by *BL*; return;

C2 : *AL* uses *BL* as a subroutine; return;

C3 : run *AL* on the output of *BL*; return.

SECTION 4 The Theory of Hard Problems

Let us begin by restricting ourselves to decision problems. A decision problem D consists of the instance I_D and the property X, with the problem being to determine if instance I_D possesses the property X. Since each instance of D will either possess the property X or it will not, the domain of instances can be partitioned into yes–instances, D_y, and no–instances, D_n. As an example consider the Hamiltonian circuit problem,

INSTANCE: graph *G(E, V)* ;

QUESTION: does *G* have a Hamiltonian circuit ?

A non–deterministic TM (NDTM) has two stages, a guessing stage and a verification stage. During the guessing stage the machine produces a *certificate* which will provide a proof that the current instance is a yes–instance, if it possible to do so. By the nature of non–determinism, if such a certificate exists, then the guessing stage will find it. The verification stage deterministically checks that the certificate is in fact a proof that the current instance is a yes–instance. The class **NP** is defined as those problems whose certificates can be generated and verified by a NDTM in polynomial time. This is sometimes called a short proof of validity. For example, a NDTM for the Hamiltonian circuit problem would guess the circuit as the certificate. The verification consists of following the given circuit through the graph and ensure that the sequence of vertices is a circuit and that every vertex is visited only once, except the starting vertex. This can be done in time which is polynomial in the number of vertices, which is the size of the problem.

Example 4.1. The certificate and verification stage for the Hamiltonian circuit problem: let G be a graph with $E(G)$ as its edge set, $V(G)$ the vertex set, and $|V(G)| = n$. The certificate will be some permutation of the vertex set $v_{\pi(1)}v_{\pi(2)} \ldots v_{\pi(n)}$ which is a circuit of the graph that will be a Hamiltonian circuit (if one exists). This is produced by the guessing stage in n steps. One verification algorithm is

```
1. node_set ← {v_π(1)}; H_circuit ← true;

2. for index ← 2 to n do

3.     if (v_π(i-1),v_π(i)) ∉ E(G) or v_π(i) ∈ node_set then

4.         H_circuit ← false; return;
5.     else
           node_set ← node_set + {v_π(i)};
6. end_do

7. if (v_π(1),v_π(n)) ∉ E(G) then H_circuit ← false;

8. return H_circuit;
```

The main loop, lines 2 to 6 will execute n times which means the algorithm is linear in the problem size. •

A NDTM can also be thought of as an infinitely parallel machine. The guessing stage always finds a solution if one exists, as the guess is chosen by considering all possible solutions (computation) paths simultaneously. At each point where it is necessary to compute several

branches of the computation in parallel, the NDTM "magically" selects that branch that will lead to the desired proof. This is referred to as a non-deterministic computation.

Example 4.2. This example will show a non-deterministic algorithm for the *SUBSET SUM* problem, which is defined as

> INSTANCE: set of integer weights $(a_1, a_2, \ldots a_n)$, $a_i \in Z+$, and positive integer B.
>
> PROPERTY: does there exist a subset of the weights that exactly sums to B ?

The following algorithm will non-deterministically solve the *SUBSET SUM* problem, given a_1, a_2, ... a_n and B as input;

1. $sum \leftarrow \{ \varnothing \}$; $index \leftarrow 1$;

2. **for** $index \leftarrow 1$ **to** n **do**

3. let $sum \leftarrow sum + [a_{index}]$ or do nothing;

4. **end_for**

5. **if** $sum = B$ **then** *return yes*
   else *return no*

The non-deterministic step is in line 3. The algorithm has the choice of adding the currently indexed weight to the sum. •

The class **NP** can also be characterised as the set of non-deterministic algorithms that run in polynomial time. Recall that the class **P** is the set of deterministic algorithms that run in polynomial time. We have that $\mathbf{P} \subseteq \mathbf{NP}$ since a NDTM may ignore the result of the guessing stage and then use verification stage to simulate any other polynomial DTM program. The most controversial

question in complexity theory is whether **P** is a proper subset of **NP**, or does **P** = **NP** ? If **P** = **NP** then all decision problems in **NP** could be solved in polynomial time. This is not expected to be the case since certain problems in **NP** have resisted concerted attempts at solving them in polynomial time, but the **P** ≠ **NP** conjecture has not been proven. Diffie and Hellman [DIF] have indicated the general difficulty of the *GCP*:

> "*the cryptanalytic difficulty of a system whose encryption and decryption operations can be done in P time cannot be greater than NP*"

The designer can decrypt in polynomial time as he knows the decryption key(s). Each of these keys will be drawn from a finite domain say K, and $|K|$-fold parallelism will be sufficient for a NDTM to "guess" these keys. Since the cryptosystem has decryption and encryption algorithms that are members of **P**, the key selection can be verified in polynomial time.

A special subclass of **NP** is the **NP**-complete problems, which are the "hardest" decision problems in **NP** (see §14). Before defining this subclass, we need to introduce the notion of a polynomial transformation. A polynomial transformation from problem A to B takes any instance A and constructs an instance of B in polynomial time such that yes–instances are mapped to yes–instances and no–instances are mapped to no–instances. This is denoted as $A \propto B$, read as "A transforms to B".

Example 4.3. The Hamiltonian circuit (HC) problem is polynomially transformable to the Travelling Salesman problem (TSP). The TSP is defined as

> INSTANCE: cost matrix (weighted graph) N and positive integer C ;
> QUESTION: is there a Hamiltonian circuit of N with cost $\leq C$?

The transformation takes an instance of HC and forms an instance of TSP. Let G be an n-vertex graph be the instance of HC and define an $n \times n$ cost matrix N to be constructed as follows:

$N(i, j) = 1$ if v_i, v_j is an edge in G and $i \neq j$;

$N(i, j) = 0$ if v_i, v_j is not an edge in G and $i \neq j$;

$N(i, j) = \infty$ if $i = j$

then the polynomial transformation f is $f(G) = (N, n)$, where N is the cost matrix and n the cost in the TSP instance. If the graph G has a HC, then there will be a tour of the graph if length n, so the yes–instances of the HC problem are mapped to the yes–instances of the TSP. Conversely, if there is a yes-instance of the TSP with cost at most n, then since each edge is length 1, then the cost C must be n since each edge is included in the tour. Therefore this will also be a yes–instance of the HC problem. •

It is often convenient to consider **NP** problems as set recognition problems. For example,

$$D_y = \{ I_D \mid I_D \textit{ has property } X \} .$$

Computing a decision problem is equivalent to computing a set characteristic function. Another way to view polynomial transformations is to state that f is an algorithm which executes in polynomial time, such that if A and B are two problems characterised by A_y and B_y, then

$$x \in A_y \textit{ if and only if } f(x) \in B_y ,$$

and $A \propto B$. This is true in example 5, as graph G will have a HC if and only if $f(G)$ has a HC of cost less then n. If $A \propto B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$. Similarly if $A \notin \mathbf{P}$ then B $\notin$ **P**. A problem Π is defined to be **NP**-complete if and only if

- $\Pi \in NP$
- $\forall K \in NP,\ K \propto \Pi$

The consequence of this definition is that **P** = **NP** if and only if Π can be solved in polynomial time. Since the **NP**–complete problems have resisted all attempts to be solved in polynomial time,

this is taken as (compelling) evidence that the **NP**–complete problems are intractable, and it is conjectured that $\mathbf{P} \neq \mathbf{NP}$.

For each problem $D \in \mathbf{NP}$ there exists a NDTM that in polynomial time identifies yes–instances of D, ie. D_y. What can be said about the no–instances of D ? The class **Co–NP** is the complement of **NP**, and $D \in$ **Co–NP** if a polynomial time NDTM identifies no–instances of D. The problem of showing that an encryption function is one–to–one is apparently in **Co–NP**. There is another apparent asymmetry in that short proofs of yes–instances exist but there is no known (polynomial time) algorithm to identify no–instances. Consider the *SUBSET SUM* problem. There are 2^n possible subsets to consider. To prove that an instance is a yes–instance only one subset that satifies the property need be exhibited. To prove that an instance is a no–instance it apparently needs to be shown that each of the possible subsets will fail to have the property of summing to B. Presenting an enumeration of all the subsets to the verification stage is an exponentially long certificate, all of which cannot be checked in a polynomial amount of time. There are other problems for which this predicament holds, namely all **NP**–complete problems.

Given this apparent asymmetry, it is conjectured that $\mathbf{NP} \neq$ **Co–NP**, ie. to compute all **Co–NP** problems in polynomial amount of time we need a stronger abstract machine than a NDTM. Consider the intersection of these two classes, $\mathbf{NP} \cap$ **Co–NP**. Since $\mathbf{P} =$ **Co–P**, as it is closed under complementation, we have, $\mathbf{P} \subseteq \mathbf{NP} \cap$ **Co–NP**. We also have the following implications,

(4.1) $P = NP \quad \Rightarrow \quad NP = Co\text{–}NP$

(4.2) $NP \neq Co\text{–}NP \quad \Rightarrow \quad P \neq NP$

(4.3) $\Pi \in NP\text{–}complete,\ \Pi \in NP \cap Co\text{–}NP \quad \Rightarrow NP = Co\text{–}NP$

Assertions (4.1) and (4.2) follow from the fact that $\mathbf{P} =$ **Co–P**. The last assertion is very interesting. Polynomial transformations induce an equivalence relation on the **NP**–complete problems (see §14.1) and if one **NP**–complete problem can be solved in polynomial time then all

NP–complete problems can be solved in polynomial time. Hence, if Π lies in **NP** $\cap$ **Co–NP** then all instances of Π can be verified in polynomial time by a NDTM, and **NP = Co–NP**. The relations of the complexity classes that have been discuseed so far are shown in Diagram 4.1.

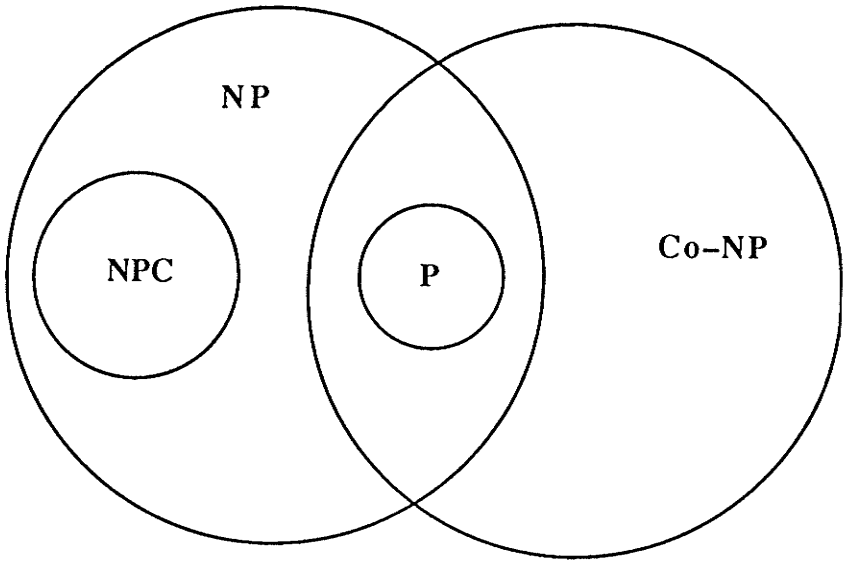


Diagram 4.1

But, from the discussion above, we do not suspect that **NP = Co–NP** and, consequently, no **NP**–complete problem is expected to lie in **NP** $\cap$ **Co–NP**. It is therefore conjectured that

(4.4) $\Pi \in NP \cap Co\text{–}NP \Rightarrow \Pi \notin NP\text{–}complete$

This conjecture will be used later to prove some properties about one way functions and the difficulty of the *GCP*.

SECTION 5 Reductions and Equivalent problems

In some ways decision problems are very limited, as we often are not only interested in the existence of a solution (does a graph have a hamiltonian circuit ?) but also in finding the solution (the said circuit). Associated with many decision problems is an anlaogous search problem. Consider the *SUBSET SUM* problem once again; the analogous search problem is

PROBLEM: find the subset of the weights that sums exactly to B (if it exists).

If we cannot prove that an apparently hard problem is **NP**–complete, then there are other methods that can be used to suggest its difficulty. An oracle TM (OTM) computes deterministically, except that at some point(s) of the computation the oracle function is called. The oracle function solves some problem in one step and leaves the answer to be used for the remainder of the computation. Oracles are used for *relativised* models of computation, meaning that oracles are used to show the relative difficulty of two problems. A problem R is *Turing reducible* to R' in polynomial time, denoted $R' \propto_T R$, if there is a polynomial time bounded OTM with an oracle for R that solves R'. The problem R is said to be **NP**–hard if in addition R' is **NP**–complete. Turing reductions are generalisations of polynomial transformations and all **NP**-complete problems are automatically **NP**-hard, but in general polynomial reductions do not imply polynomial transformations [LLS]. From example 4.3, if there were an algorithm that solved the TSP then it could be used to solve the HC problem. The **NP**–hard problems have the property that they are not solvable in polynomial time unless **P** = **NP**. That is, **P** = **NP** is a necessary but not suffcient condition for all **NP**-hard problems to be solvable in polynomial time. This means that some **NP**-hard problems may remain intractable even if all decision problems in **NP** are easily solved, (the *KTH LARGEST SUBSET* [GaJ] is such a problem). Thus proving that the *GCP* of a system is **NP**-hard seems a strong method of certifying its security.

Usually, there are trivial reductions from the decision form of a problem to a related search problem. Consider the analogous search problem for the TSP:

PROBLEM: find a Hamiltonian circuit of length $\leq C$.

Let *AL(G, N, n)* be an algorithm that finds this Hamiltonian circuit, if it exists, and returns 0 otherwise. Let TSP be the decision form of the TSP_{dec} and TSP_{sch} the related search problem. Then $TSP_{dec} \propto_T TSP_{sch}$ since

if *AL(G, N, n)* $\neq 0$ **then** TSP_{dec} *answered yes*
else TSP_{dec} *answered no;*

Since the reduction is one way, from the decision form to the search problem, if the TSP_{sch} is solved in polynomial time then so is the TSP_{dec}. Nothing is said about the complexity of TSP_{sch} though. Two problems are said to be (polynomially) *equivalent*, denoted $R \equiv_T R'$, if an efficient algorithm for one problem yields an efficient algorithm for the other problem as well. Using reductions, two problems R and R' are equivalent if and only if

$$R' \propto_T R \text{ and } R \propto_T R'$$

If $R' \propto_T R$ and $R \propto_T R'$ and one of the problems is **NP**–complete then both of these problems are solvable in polynomial time if and only if **P = NP**. The **NP**-complete problems are automatically equivalent . Consider the *SUBSET SUM* problem in its decision and search forms, SUB_{dec} and SUB_{sch} respectively;

Example 5.1. The *SUBSET SUM* decision and search problems are equivalent. Trivially, $SUB_{dec} \propto_T SUB_{sch}$. The $SUB_{sch} \propto_T SUB_{dec}$ is the interesting, and possibly surprising reduction. Let $AL(a_1, \ldots a_i, i, B)$ return yes if there is a subset of the i weights that sums to B. This algorithm is an oracle for the SUB_{dec} problem. Assume that the weights are arranged in non–decreasing order. Then the following algorithm shows that $SUB_{sch} \propto_T SUB_{dec}$;

1. **if not** $AL(a_1, \ldots a_n, n, B)$ **then return** ;
2. $subset \leftarrow \{\varnothing\}$;
3. **for** $index \leftarrow n$ **downto** 2 **do**
4. **if not** $AL(a_1, \ldots a_{index-1}, index-1, B)$ **then**
5. $B \leftarrow B - a_{index}$
   $subset \leftarrow subset + [a_{index}]$
6. **end_if**
7. **end_do**
8. **if** $B > 0$ **then** $subset \leftarrow subset + [a_1]$ •

The notion of equivalence is very strong; it indicates that *any* algorithm for a problem p can be used to solve all problems that are equivalent to p. This will prove useful to the designer who wishes to show that any attack on his system will be equivalent to solving an intractable problem. In fact, reductions must be appealed to if condition (2.6) of the definition of a public key cryptosystem is hoped to be satisfied. Normally it is not.

CHAPTER 2 Cryptocomplexity

SECTION 6 Some Hard Cryptographic Problems

In the relatively short history of intractable secrecy systems, a number of suitably formulated problems have established themselves as difficult cryptographic problems. These problems are: factoring, determining quadratic residuosity, the discrete logarithm problem, and the knapsack problem. These are defined formally as

NAME: *factorisation* ;

INSTANCE: integer n ;

PROBLEM: find the unique prime factorisation of n.

NAME: *quadratic residuosity* ;

INSTANCE: integer n, integer a such that $0 < a < n$;

QUESTION: does $x^2 \equiv a \bmod n$ have a solution ?

NAME: *discrete logarithm problem* ;

INSTANCE: prime power q, primitive element g of GF(q), $y \in$ GF(q) ;

PROBLEM: find x such that $0 \leq x \leq q-2$ and $y \equiv g^x \bmod q$.

The form of the knapsack problem that is of interest to cryptography is exactly the search form of the *SUBSET SUM* problem defined in §4. Three of these problems are search problems and one is a decision problem. In sections 13 and 14 we will discus the merits of decision and search problems in cryptography. These shall be referred to as the *basic problems* for this discussion. Each of these problems is intractable in the the sense that no algorithm is known for any of them that runs in polynomial time. The first three basic problems originate from computational number theory, and the last is a mathematical programming problem.

Each of these basic problems has been used to construct a PKC, but is also adaptable to symmetric protocols. We intend to address several issues concerning intractability in general, and what it specifically has to say about the basic problems. In particular consider the following

(6.1) how to quantify the security of a PKC (§7),

(6.2) how to select difficult instances of the basic problems (§8),

(6.3) the merits of worst case analysis (§8),

(6.4) reducibility amongst the basic problems (§14).

The last question is perhaps the most interesting. Demonstrating that several problems have equivalent difficulty strengthens the conjecture that they are in fact difficult. If one such problem can be proved hard, then, by the nature of reducibilty, all of the equivalent problems are also difficult. There is a similar corollary if one of the problems is strongly suspected of being difficult. If a designer devises a new system, then it would be advantageous to demonstrate that a cryptanalysis of the new system would yield methods of cryptanalysing other systems or problems known or thought to be difficult. This is not denying the invention of a novel system that is intuitively hard, yet bears no functional dependence to any existing difficult systems. But it appears that cryptosystems derived from the basic problems are moving towards a concept of equivalent security, in at least a general sense (see §14.3).

There are two places where reductions will be applied. Let S_1 and S_2 be cryptosystems where S_1 is based on $p_1(I)$, an instance of basic problem p_1, and similarly S_2 is based on $p_2(I)$. If problem p_1 is an intractable problem and $p_1(I)$ an intractable problem instance, can we show that solving the *GCP* of S_1 is equivalent to solving $p_1(I)$? If this were the case, it would imply that *any successful cryptanalytic attack on* S_1 would yield an algorithm to solve an intractable problem. This is the strongest sense of security that a designer can gain from a computationally secure system. By using reductions it is possible to anticipate every attack of a cryptanalyst. The designer can exactly quantify the efforts of the cryptanalyst and be assured that he must perform a minimum amount of

work to solve the cryptosystem. This is the precise notion Diffie and Hellman were stating in condition (2.7) of the defintion of a public key cryptosystem – " *each algorithm equivalent* to D_{k2} is computationally infeasible to compute from E_{k1}".

Secondly, S_1 and S_2 are of equivalent difficulty if any efficient algorithm *AL* which solves either of their *GCPs* leads to an efficient solution of the other system's *GCP*. Hence, if such an algorithm can be found for either S_1 or S_2, then both systems are insecure. Shannon called such systems *similar*. Establishing a set of cryptographically equivalent systems that are hard suggests that these systems are strong in a natural sense.

There are some hard realities to contend with in PKCs. Difficult problems have proved to be less adaptable to cryptography than designers had anticipated. Even though the basic problems carry the promise of difficulty, they are also unwieldly and awkward to handle, as the difficulty of solving each of these problems is directly proportional to their size. How does one arbitrarily choose a number that is hard to factor? Making it a few hundred digits is a good start. In 1978, an analysis of the best algorithms available for factoring suggested the results in Table 6.1, and a 50 digit number could be factored in reasonable amount of time.

digits	time
50	3.9 hours
75	104 days
100	74 years
200	3.8×10^9 years
300	4.9×10^{15} years
500	4.2×10^{25} years

Table 6.1[1]

Using intractability as a basis for security is a double–edged sword. Large problems are not only difficult to solve but messy to work with as compared to more streamlined conventional systems such as DES. There is a definite tradeoff between efficiency and security derived from intractability. Are the public key systems worth the trouble? There are probably no simple, elegant solutions to the problem of establishing a secure channel that provides authentication, but we are bound to accept the best solutions at our disposal. Is the facility of being able to hold a secure yet slow conversation without the need for users to be previously accquainted more convenient than using a fast one key system? Possibly, the ultimate configuration of a cryptosystem will be a hybrid that posesses both conventional (for rapid data exchange) and public key (for key/signature exchanges) components. Further, we wish to discuss the limitations of intractability and describe the considerations a designer must undertake in constructing a secure system.

[1][RSA].

SECTION 7 Public Key Complexity

The security of a public key cryptosystem was first conceived as a relative measure between the time required by the designer to decrypt, as compared to the time needed for an opponent to perform an illegal decryption. Call this the *security ratio* of a system. An upper bound on the security ratio of any system is provided by the size of the keyspace. The time required to determine the key by exhaustive search should be exponentially large compared to the time of selecting a key.

Ralph Merkle [MER1] began by offering a key exchange protocol that enabled two parties to select a key in linear time and left an opponent a number of "puzzles" which required quadratic time to solve for the key. He showed how a key exchange protocol could be built out of conventional components. Here the security ratio of the system is considered to be in favour of an opponent, as a problem with a quadratic complexity is an insufficient deterrent for the opponent who wishes to compromise the system. Merkle conjectured that exponential puzzles could be devised, and more to the point, that they are necessary. Pohlig and Hellman indicate that the security ratio should be at least 10^9 [PaH]. The best scenario for a security ratio would be a system that guarantees linear time decryption for the designer with respect to some system parameter (eg. message length, size of the public key), while leaving an opponent with an exponential task to cryptanalyse the system, as measured by the same parameter. For several reasons this ideal will never be attained.

Guaranteeing a linear upper bound for legal decryption is unlikely, unless a system can be based on some simple set of operations such as addition and subtraction. The Merkle-Hellman [MER2] knapsack cryptosystems (see §12) can be encrypted using addition alone, but decryption becomes more complicated. A more realistic goal for the designer is to devise a system that is "easy" for him to decrypt but "not easy" to decrypt for anyone else. Encryption should be "easy" for everyone. An "easy" task has come to be a pseudonym for a problem that can be computed in

time that is proportional to some polynomial in the size of the problem instance (ie. a member of P). The basic problems can be manipulated using modular arithmetic including, exponentiation. Since each operation is easily performed [KNU2], it is possible to achieve the desired upper bound for encryption and decryption. Of course these operations do not form the exclusive set of efficient encryption and decryption operations[2]. Guaranteeing a "not easy" task for an opponent, though, requires some craft.

Public key cryptography is concerned with intractable problems, and in fact these problems are a great concern to theorists and practicioners alike. We begin by looking at intractability in practice – tasks that seem to take a long time to compute. Any algorithm for a problem yields an upper bound, whereas to prove a lower bound it must shown that every algorithm for a problem will require at least $\Omega(g(n))$ steps to complete. But anticipating all algorithms seems difficult to conceive, and this problem is very similar to a designer attempting to anticipate all cryptanalytic attacks.

Consider establishing a resource threshold Φ, expressed numerically, which indicates the limit to the amount of resources that will be spent to solve a problem. Given this bound Φ, we can then take an arbitrary problem, examine its complexity, and then determine how large an instance of this problem need be constructed before the cost of its solution exceeds Φ.

More efficient algorithms, better coding of these algorithms, faster machine cycles, and growing paralellism are some of the many factors that are contributing to the redefinition of what a feasible computation is, although a bound can be set that is not likely to be approached in the forseeable future. Some people have already given estimates of what practical computing is. Knuth [KNU1, p45] suggested that an algorithm that required less than $10! \approx 3.6 \times 10^6$ steps was

[2]The lowest and most efficient operations are bit operations which can be performed directly in hardware. Even though the RSA system has a polynomial encryption algorithm it is only possible to achieve a 6 kilobit per second transmission whereas DES, which can be implemented by hardware operations can achieve rates of 10 megabits per second [DIFF2].

feasible. With a millisecond instruction cycle 10! steps could be computed in a day or so. Simmons [SIMM] gave 10^{12} operations as the "sound barrier" for computation, and a useful basis of comparing the asymptotic performances of algorithms. As a benchmark, consider the reception of DES in 1977. When the DES system was adopted there was some disagreement over how much security the 56-bit key could provide (led by Hellman *et al.* [HELL]). There are $2^{56} \approx 7.2 \times 10^{16}$ possible keys, which seems a large number, but expensive special purpose hardware could search the keyspace in a day. It was suggested that if the length of the key were doubled, a comfortable margin of safety could be achieved. Given this, it is asserted that it is beyond the economic and physical resources of any opponent to expend $\Phi = 2^{100} \approx 1.267 \times 10^{30}$ arithmetic operations solving a problem. This was also the lower bound of the general cryptanalytic problem as suggested by Merkle and Hellman [MER2].

Now let us return to the system proposed by Merkle where the opponent is faced with a task that has a quadratic complexity. Diagram 7.1 represents Merkle's scheme. There is a polynomial algorithm *AL* that, given an ordered pair (KEY_i, ID_i), will produce a problem p_i that requires $c \cdot N$, $c > 0$, operations to solve.

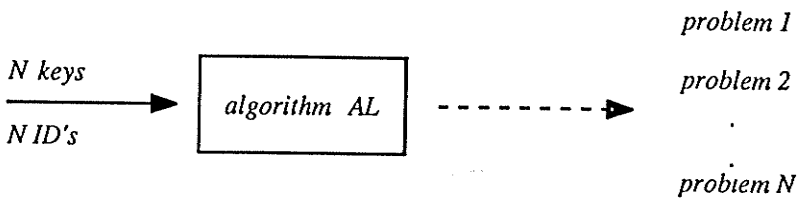


Diagram 7.1. Merkle's system.

Suppose that users Alfred and Barbara wish to communicate. Alfred generates N puzzles and sends them to Barbara. Barbara solves one puzzle of her choice, say p_k, that yields the pair (KEY_k, ID_k). Barbara then sends ID_k to Alfred who deduces that Barbara has selected key KEY_k. Since each pair (KEY_i, ID_i) is encoded in a puzzle, an opponent has to solve (on average) half of the puzzles before he finds the key that corresponds to ID_k.

Quite intentionally *AL* does not produce hard problems. The difficulty of a cryptanalysis of Merkle's system resides in the number of puzzles to solve more than the task of actually solving an individual puzzle. It is assumed that hard encryption functions exist though. How large must N be made before an opponent is faced with an infeasible task? Unfortunately, if the quadratic security ratio could not be increased, then 2^{51} puzzles would have to be transmitted, where each puzzle would require 2^{50} operations to solve. Given that are less than 3.2×10^{13} milliseconds a year, if each puzzle could be encoded and sent in a millisecond, then someone would be sending puzzles for about 100 years. Clearly a quadratic security parameter is not very useful at all. Polynomials of higher degree could be considered but it is argued that the impracticality of the system is inherent when dealing with a polynomial security ratio. The problem is that Merkle's system is inherently polynomial but the designer is required to guard against an opponent with an exponential amount of resources. Merkle is attempting to create an exponential security ratio through the concatenation of polynomial problems.

Admittedly, a quadratic is a low degree polynomial and Merkle expresses disatisfation with it. Consider a security ratio of $O(n^{30})$ which has a rapid growth rate (see Table 3.1). A system that provided easy encryption/decryption procedures and an $O(n^{30})$ security ratio would appear to be practical. But have we not disregarded our assumption that illegal decryption should be hard, by giving a polynomial bound on this task? Clearly there is an inconsistency between the theory and application – an algorithm that has a polynomial time solution but is intractable (by Table 3.1). The class **P** is the most thoroughly understood complexity class; it is only convention and convenience that permits this class to be considered as the set of "well solved" problems.

Proving that the computational lower bound of a problem is $\Omega(n^{30})$ would only be of theoretical interest because the result has the practical translation that the problem cannot be efficiently solved for most n. The problem is not "well solved" even though a polynomial time algorithm suggests a deep insight into the nature of the problem. The broad definition of **P** accepts

all problems that have a polynomial solution, but more often than not these polynomials are of a low degree, not normally more than four.

Recall that the designer is looking for a problem that will give him a "favourable" security ratio, a means of puzzling his opponent. The designer is looking for a problem with a complexity that has a rapid growth rate. We have attempted in the preceding discussion, and in the comments of §3, to convince the reader that, even though most polynomial growth rates are rapid, we do not expect to find an algorithm whose complexity is dominated by a large polynomial term in the problem size. It contradicts the interpretation of **P**. But, in accordance with the notion of intractability, there are problems that have rapid non-polynomial growth rates, or simply exponential growth rates. There are problems of great practical significance that present themselves as being difficult to solve, which can be used to provide the designer with favourable and practical security ratios. Public key cryptography is intrinsically concerned with problems that lie outside **P**, in the domain of intactable problems.

Complexity theory has identified collections of problems for which there are no known polynomial time algorithms. Even so, there is no well defined set where each member of this set has the property of being difficult and all difficult problems are in this set. The best definitions of an intractable problem may seem trivial,

(7.1) an intractable problem is a problem that is not considered to be tractable;

(7.2) hard problems are those that are not known to lie in **P**;

(7.3) a problem is considered to be hard until it is found to be easy.

The reader should perceive that these statements are not a particularly constructive notion of intractability, given that the concept of a tractable problem is only convention. Without ridding ourselves of the **P** =? **NP** sphinx, intractability is only a strong suspicion, embodied in the conjecture that $\mathbf{P} \neq \mathbf{NP}$ (ie. $\Omega(c^n)$ bounds will exist). Still, there is a need for a working definition

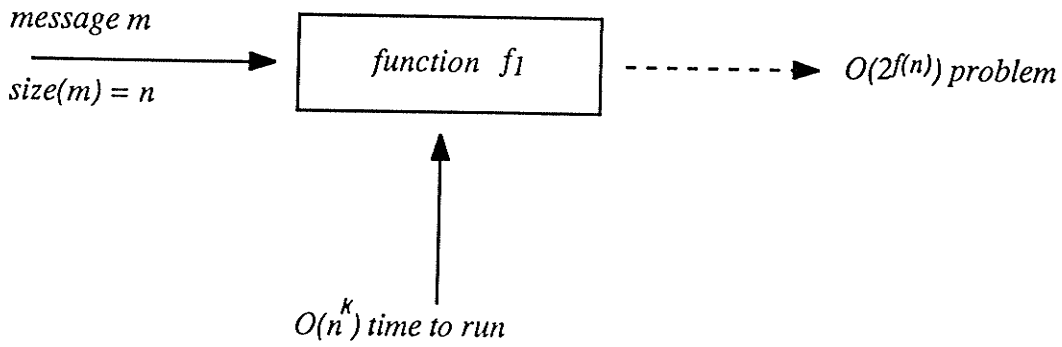
of the collection of hard problems. For convenience, define the set **IP** (intractable problems) by the following characteristic function,

$$\chi(x)\begin{cases} 1 \text{ if the best algorithm known for } x \text{ has a non–polynomial time growth rate;} \\ 0 \text{ if } x \in \mathbf{P}. \end{cases}$$

The class **EXPTIME** consists of all problems that can be solved in $O(2^{p(n)})$ time where p is a polynomial and n the size of the input. Then **IP** = **EXPTIME** – **P**. Each of the basic problems is a member of **IP**, which also contains the **NP**–hard and **NP**–complete problems.

For a relatively small problem size the task of solving an **IP** problem is computationally infeasible. Cryptographers have turned to these problems not only because they are hard, but because they are a convenient means of constructing a difficult task. Consider the TSP on k cities. The number of possible tours is $(k–1)!/2$ and when k is as small as 30, the time required to solve the problem exceeds $\Phi = 2^{100}$.

The designer is therefore searching for an algorithm or black box that encrypts a message of size n in polynomial time and produces a cryptogram, where the time required to cryptanalyse the ciphertext is (apparently) $O(2^{f(n)})$*. Diagram 7.2 represents the one way function f_1.



*Or any task that is non-polynomial. We use $(2^{f(n)})$ as a symbol of difficulty as much as an actual measure of it. There is nothing essential about choosing 2 as the base of the exponent, but it is a number encountered often in numerical computing and combinatorial problems. Every problem in NP can be solved in time $O(2^n)$.

Diagram 7.2

How does the set **IP** assist the system designer? He now has at his disposal a collection of problems that to the best of anyones' knowledge are difficult to solve. Forming a set of candidate problems is the easiest hurdle to overcome, but a more serious obstacle awaits. How can he harness the difficulty of one of these problems? The process being portrayed in Diagram 7.2 is the encryption of a message, and to make the system viable another black box is needed to decrypt (see Diagram 7.3). The input to this black box is the work of the problem facing the general public, and $O(n^k)$ will represent the relative ease with which the intended receiver can decrypt . The secret of this black box is the secret to the whole system, that being the trapdoor information that allows the receiver to reduce an (apparent) $O(2^{f(n)})$ task to a problem that has a relatively simple solution, and hence a favourable security ratio.

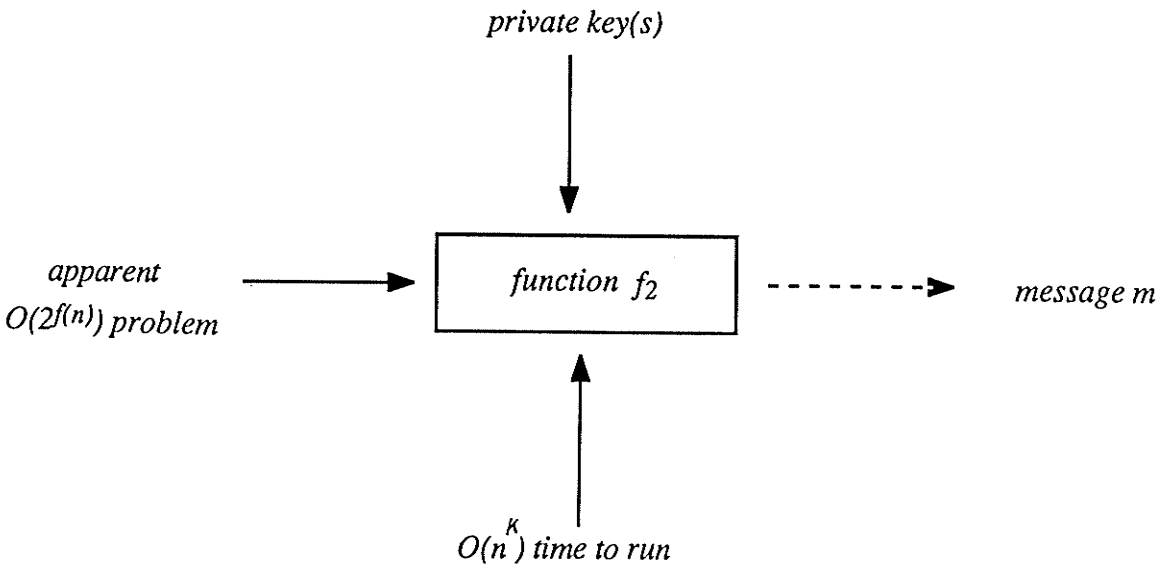


Diagram 7.3

Designers are not strictly dealing with hard problems, as will be shown, but with various methods that obscure the information that will lead to the efficient solution of problems that are generally intractable.

SECTION 8 Search Problems and Worst Case Analysis

There are certain problems for which the set of possible solutions, patterns or arrangements, can be enumerated. This enumeration, though finite, may be extremely large, which is precisely the nature of our basic problems. If we are given a 500-bit number to factor then its factors must be contained in the set $\{1, 2, ..., 2^{500}\}$. The adjacency matrix of a graph defines the set of possible Hamiltonian circuits on the graph, as it does all the possible k-cliques, if either exist. Similar statements can be made about other decision and optimisation problems. It is a necessary condition for a problem to be difficult that the number of possible solutions is large compared to the size of the problem.

For a given problem a search space can be identified that exactly contains the set of possible solutions or considerations for the problem. For example, the *SUBSET SUM* problem with k weights has all 2^k k-bit binary vectors as its search space. An algorithm for a problem acts on the search space of that problem by successively examining members of the search space in some fashion until one is accepted as the answer to the problem (include "there is no solution" as a possible solution). The efficiency of an algorithm can be seen as the strategy it employs to examine the search space of a problem, and how it prunes the search space if it is large (ie. how directly it seeks out the solution).

Let $W(p)$ be the worst case complexity of problem p. Then for any algorithm $AL(p)$ that solves problem p there is at least one instance of p that requires $W(p)$ time to solve. The worst case complexity of each basic problem is some function of the cardinality of its search space. If a problem is deemed intractable, then its worst case complexity is a non-polynomial function of the problem size. This implies that if a problem is (considered) intractable then the best methods for solving it do not differ significantly (say by a polynomial factor) from the task of exhaustively searching an exponentially large number of possible solutions. For example, the best known algorithm for the knapsack problem of n weights runs in time $O(2^{n/2})$ and space $O(2^{n/4})$ [SHC]

where the number of possible solutions is 2^n; the best known algorithm for factoring a number n is $O(L(n)^{\alpha})$ where $L(n)$ is exp(sqrt(*log n loglog n*)) and $\alpha < 3/2$ [POM1].

How is a designer to find a hard instance of an intractable problem? Little is learned from a worst case analysis, and we state that there is no known general transformation from hard problems to hard cryptosystems. A worst case analysis is the result of a game between two agents, one who has an algorithm $AL(p)$ for problem p and the other who devises the instance of p that $AL(p)$ will run on.

Game 1: (8.1) agent 1 submits his algorithm to agent 2;

(8.2) agent 2 examines $AL(p)$ and proceeds to construct an instance of p that taxes the efficiency of $AL(p)$ in the worst possible way.

It is an irrelevant consideration that the instance of p that agent 2 supplies may be totally contrived and have a negligible chance of ever occuring in reality. The worst case estimate for the problem p is taken to be the complexity of the most efficient algorithm for p as designated by agent 2, taken across all known algorithms for p. If p has an exponential worst case complexity then on each submission of an algorithm $AL(p)$, agent 2 was able to find a difficult instance of p with respect to $AL(p)$. Such problems are considered inherently difficult as each known algorithm that purports to solve them efficiently fails. The problem p is solvable in a feasible amount of time if $AL(p)$ is total with respect to p (ie. halts on every instance of p) and $AL(p) \in \mathbf{P}$. If a problem p is considered to be intractable then there is no known algorithm that is total with respect to p *and* is a member of the class **P**. However, it may be the case that a suitably chosen subset of problem instances can be solved in polynomial time.

An intractable instance of problem p is qualified by the running time of an algorithm or set of algorithms for p. This is far from a rigorous proof that a problem is intractable. It merely says that there are witnesses to the intractability of p, and these witnesses are only as useful as far as they are prepared to be trusted. If a problem is accepted as intractable, by default the conjecture that

P ≠ NP is assumed to be true. In Game 2 the problem p is a PKC and the particular instance of p is a selection from the keyspace associated with the PKC. The rules are changed slightly though. Previously, the algorithm was fixed, the problem instance could vary, and it was possible to prove the work of agent 2 was correct. Now the problem instance is fixed (ie, the cryptosystem), the algorithm (ie. the method cryptanalysis) can vary, and the work of the agent 1 can be proved (ie. that the system has been broken). The designer must make his selection of the system instance without full knowledge of what methods and information are at the disposal of the cryptanalyst. The new game is,

Game 2: (8.3) the designer submits his system to public scrutiny;

(8.4) the cryptanalyst attempts to compromise the system.

Game 2 is an informal statement of the *GCP*. In both games, if one agent can produce a proof, then the game is over with that agent giving the proof, "winning" in the sense that he has frustrated his opponents' aims. If the cryptanalyst can demonstrate that he can decrypt efficiently, then the system is useless. Similarly if the designer can prove that illegal decryption by an opponent is an intractable problem then the system is secure and the cryptanalyst cannot win (in polynomial time). The difference is that proofs of efficiency exist, but proofs of infeasible computational lower bounds, generally, do not. Hence, the designer usually can not guarantee that he will "win" the confrontation against the cryptanalyst, but it is obvious when the designer has lost.

How can a designer, in the absence of a proof, convince himself that he has selected an intractable instance of a problem he wishes to base his system on? The plain answer is that he cannot. An instance of a problem is only hard until it is solved efficiently, an uncertain state of affairs indeed. The only definite property that he has at his disposal concerning an intractable problem is that the number of possible solutions grows exponentially in the problem size. The designer is able to generate an instance of a problem that has a corresponding search space which cannot be exhaustively examined in polynomial time.

Simply guaranteeing a large search space does not imply that the cryptanalyst has to consider every possible solution to effect a decryption though. For example, the complete graph on n vertices, K_n, has n^{n-2} distinct spanning trees, yet for a weighted graph G, the *MINIMUM SPANNING TREE* problem can be solved in polynomial time [KRUS]. Creating an infeasibly large problem amounts to either having a large number of problem parameters or, large problem parameters, depending upon the nature of the problem. For problems such as factoring and determining quadratic residuosity the number of parameters is small and fixed, hence a difficult instance is induced by selecting very large numbers. For example the RSA system is based on factoring one 200 digit number. Generating a large instance of the knapsack problem requires a large number of parameters *and* also that these parameters be large. Knapsack systems not only have to guard against direct search, but also against a technique called dynamic programming, which runs in time proportional to the product of the number of the weights and the target sum. Hence a large number of large weights are necessary to provide an exponentially large search space. The Merkle-Hellman system (see §12) requires 100 weights each of which is over 200 bits in length.

Notice that by creating a large instance of a problem we are not increasing the difficulty of solving it *per se*. An algorithm for deciding the satisfiability problem with 10 clauses also works for an instance of 100 clauses. But the time required to solve these respective instances differs dramatically. This is the essence of how the notion of difficulty has been adopted in complexity theory. All of the problems in **IP** are decidable, and there are effective procedures to realise this, but they are difficult to solve in the sense that the time required to realise these effective procedures is prohibitively large.

In the next four sections we will consider four problems, how difficult they are to solve in general and how they have been adapted to cryptography, and used as potential one way functions.

CHAPTER 3 Some Potential One Way Functions

Section 9 The Problem of Factoring

The problem of factoring has puzzled great minds for centuries. As it has evaded efficient solution during this time, it establishes itself as a strong candidate for a one way function. Factoring is the basis for at least seven cryptosystems [SIMM], some providing secrecy and some for signatures, the most noteworthy being the RSA proposal. Also, for our basic number theoretic problems, instances known to involve composite moduli are much harder to compute than those involving prime moduli (such is the case for computing square roots, determining quadratic residuosity and squarefreeness). Thus, finding factors is important for the effcient solution of these problems. A system based on factoring means that one method of cryptanalysis would be to find an efficient method for factoring a system parameter. Hence, factoring is an upper bound on the security of these systems. These systems would be as secure as factoring, no more no less, if it could be shown that *any* cryptanalysis would yield an efficient algorithm to factor a system parameter. The last 10 years or so have been the most dramatic period of successful factorisations. The quadratic sieve method [POM2] when introduced increased the size of a number that could be factored in a day by 13 digits [SIMM]. At present we know of 6 factoring methods which share the same asymptotic complexity bound of

$$L(N) = \exp\left((1 + o(1)) \operatorname{sqrt}(\ln n \ln\ln n)\right)$$

where n is the number to be factored. These six methods are:

(i) the elliptic curve method [LEN],

(ii) the class group algorithm [SCLN],

(iii) the linear sieve algorithm [COS],

(iv) the quadratic sieve [POM2],

(v) residue list sieve [LEN],

(vi) continued fraction method [MaB]

The question to be asked is how large a number can we factor in a reasonable amount of time? The following 1984 estimates due to Rivest are based on the above *L(N)* estimate with no constant term in the exponent[1].

digits	elementary operations	time to factor
75	9×1012	1 day
100	2×1015	255 days
125	3×1017	103 years
150	3×1019	9755 years
200	1×1023	3.6×107 years
300	1×1029	5×1013 years

Table 9.1

It is interesting to compare this table to Table 6.1 to see the improvement in factoring in just six years. A 75 digit that number was infeasible to factor in 1978 can now be factored in a day, but the proposition that a 200 digit number is hard to factor still remains true. Simmons [SIMM] suggests the rule of thumb that doubling the computing power at your disposal, for a fixed amount of work, roughly increases the size of a number that can be factored by 3 digits. Accordingly, by increasing his computing power 45-fold Simmons could factor a 200 digit RSA modulus in a day. Reaping such an increase seems unlikely, as the current bottlenecks to improving the efficiency of factoring lie in hardware. For example, the implementation of the quadratic sieve method at Sandia

[1][RIV]. Rivest also noted that if n is a k-bit number then a crude estimate of the complexity bound to factor n is $5 \cdot 10^{9+(k/50)}$ and the difficulty grows one order of magnitude more difficult for each additional 50 bits (15 decimal) digits of RSA modulus.

Laboratories is approaching the clock speed of the Cray XM-P. It should be stressed that methods such as the quadratic sieve, are *factoring techniques*. Their success depends upon a subexponential algorithm, a taut coding of this algorithm and a hardware specific implementation. The next logical step is to introduce parallel architectures. Each of the 6 best factoring algorithms subscribes to the divide and conquer philosophy, which means that if k identical machines simultaneously act on a number, it will essentially be factored k times as fast[2]. We cite two parallel implementations. Caron and Silverman [CaS] have implemented the quadratic sieve in a network of 9 SUN 3 work stations and were able to factor an 81 digit number in a week (1260 CPU hours). It is extapolated that a 126-digit number could be factored in a year long run of 2000 SUN 3's [PST]. Pomerance *et al.* [PST] have pipelined the quadratic sieve method and describe a machine that has the factoring capabilities described in Table 9.2.

digits	time	cost to build($US)
100	2 weeks	50,000
144	1 year	10,000,000
200	1 year	1011

Table 9.2

Given these advances it is still plausible to assert that a 200 digit number cannot be computed with a feasible amount of resources. Since factoring is so important, we will give a short description of Morrison and Brillhart's continued fraction (CFRAC) method, which is one of the most efficient general methods of factoring known.

[2]See [PST]. In [CaS] for a 60 digit number, 1 processor required 6 hours to factor the number while 8 processors only required 1 hour.

9.1 The Continued Fraction Method

Using the Euclidean algorithm, rational numbers can be expressed as simple continued fractions. For example, from the Euclidean algorithm gcd(62, 23) yields the following equations

$$\begin{aligned}
62 &= 2\cdot 23 + 16 &\Rightarrow \quad 62/23 &= 2 + 1/(23/62)\\
23 &= 16\cdot 1 + 7 &\Rightarrow \quad 23/16 &= 1 + 1/(16/7)\\
16 &= 2\cdot 7 + 2 &\Rightarrow \quad 16/7 &= 2 + 1/(7/2)\\
7 &= 3\cdot 2 + 1 &\Rightarrow \quad 7/2 &= 3 + 1/2
\end{aligned}$$

and combining we have

$$\begin{aligned}
\frac{62}{23} &= 2 + \frac{1}{23/16}\\
&= 2 + \frac{1}{1+\frac{1}{16/7}}\\
&= 2 + \frac{1}{1+\frac{1}{2+7/2}}\\
&= 2 + \frac{1}{1+\frac{1}{2+\frac{1}{3+\frac{1}{2}}}}
\end{aligned}$$

which is the simple continued fraction (SCF) expansion of 62/23. Formally a finite SCF is an expression of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots + \cfrac{1}{a_n}}}}$$

denoted $[a_0;a_1, a_2, \ldots, a_n]$ where the a_i are positive integers. Thus 62/23 is represented as [2;1,2,3,2].

The basic idea in the CFRAC method, as with the other 5 main factoring algorithms, is that if N is the number to be factored, then we search for integers x and y such that

$$x^2 \equiv y^2 \mod N, \quad 0 < x, y < N, \quad x \neq y \text{ and } x + y \neq N \ . \tag{9.1}$$

If this is the case, N is a divisor of $(x^2 - y^2) = (x + y)(x - y)$, yet N divides neither $(x + y)$ or $(x - y)$. Hence $\gcd(N, x + y)$ and $\gcd(N, x - y)$ will yield proper factors of N. The CFRAC method begins by finding solutions to the congruence

$$x^2 \equiv a \mod N \tag{9.2}$$

for "small" integers a and then combines several solutions to eq.(9.2) to obtain a solution to eq.(9.1). Some number theory is required before the details of the algorithm are given.

Let k be a non–negative integer less than n, then $[a_0;a_1,a_2, \ldots, a_k]$ is the *kth* convergent of the SCF $[a_0;a_1,a_2, \ldots, a_n]$, denoted C_k. It is well known (see for example [ROS]) that if the sequences $p_0, p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$ are defined recursively as

$$p_0 = a_0 \ , \quad q_0 = 1$$

$$p_1 = a_0a_1 + 1 \ , q_1 = a_1$$

and

$$p_k = a_k p_{k-1} + p_{k-2}\,, \quad q_k = a_k q_{k-1} + q_{k-2}$$

for $k = 2, \ldots, n$, then the *kth* convergent is given by $C_k = p_k/q_k$.

Lemma 9.3. $p_k q_{k-1} - p_{k-1} q_k = (-1)^{k-1}$.

proof. by induction. Let $k = 1$ then

$$p_1 q_0 - p_0 q_1 = (a_0 a_1 + 1)\cdot 1 - a_0 a_1 = 1$$

where a_0 and a_1 are the first two integers in the SCF expansion. Assume true for $1 \le k < n$ so that

$$p_k q_{k-1} - p_{k-1} q_k = (-1)^{k-1}$$

then

$$p_{k+1} q_k - p_k q_{k+1} = (a_{k+1} p_k + p_{k-1}) q_k - p_k (a_{k+1} q_k + q_{k-1})$$

$$= p_{k-1} q_k - p_k q_{k-1} = -(-1)^{k-1} = (-1)^k \bullet$$

We are interested in the SCF expansion of the general surd

$$\frac{P_0 + \sqrt{D}}{Q_0} = \left[a_0; a_1, a_2 \ldots a_{n-1}, \frac{P_n + \sqrt{D}}{Q_n} \right] \tag{9.3}$$

by the rules

$$P_{n+1} = a_n Q_n - P_n, \; Q_{n+1} = (D - P^2_{n-1})/Q_n$$

$$a_{n+1} = \lceil (P_{n+1} + \sqrt{D})/Q_{n+1} \rceil$$

where $\lceil x \rceil$ is the greatest integer less than x. The following lemma will be useful later.

Lemma 9.2. If r, s, t, u are rationals and $r + s\sqrt{d} = t + u\sqrt{d}$ and d is a positive integer but not a perfect square, then $r = t$ and $u = s$.

proof. since $r + s\sqrt{d} = t + u\sqrt{d}$ then if $s \neq u$

$$\sqrt{d} = \frac{r-t}{u-s}$$

since $(r–t)/(u–s)$ is rational, then $\sqrt{d}$ is irrational and $s = u$ and $r = t$. •

The particular SCF expansion of interest is $\sqrt{(KN)}$, for some K, which is the following surd

$$\frac{\sqrt{KN} + 0}{1}$$

and hence can be expanded as in eq.(9.3) to

$$\sqrt{(KN)} = [a_0;a_1,a_2, \ldots , \alpha]$$

where $\alpha = (P_n + \sqrt{(KN)})/Q_n$. Then the *kth* convergent of $\sqrt{(KN)}$ is

$$\sqrt{KN} = \frac{\alpha p_{n-1} + p_{n-2}}{\alpha q_{n-1} + q_{n-2}} \tag{9.4}$$

and

$$\sqrt{KN} = \frac{(P_n + \sqrt{KN})p_{n-1} + Q_n p_{n-2}}{(P_n + \sqrt{KN})q_{n-1} + Q_n q_{n-2}} \tag{9.5}$$

and cross–multiplying

$$KNq_{n-1} + \sqrt{(KN)}(P_n q_{n-1} + Q_n q_{n-2}) = \sqrt{(KN)}p_{n-1} + P_n p_{n-1} + Q_n q_{n-2}$$

From Lemma 9.4, we have that

$$KNq_{n-1} = P_n p_{n-1} + Q_n p_{n-2} \tag{9.6}$$

$$p_{n-1} = P_n q_{n-1} + Q_n q_{n-2} \tag{9.7}$$

Multiplying eq.(9.6) by q_{n-1} and eq.(9.7) by p_{n-1} then subtracting

$$p^2_{n-1} - KNq^2_{n-1} = (p_{n-1}q_{n-2} - p_{n-2}q_{n-1})Q_n \tag{9.8}$$

By Lemma 9.3

$$p^2_{n-1} - KNq^2_{n-1} = (-1)^n Q_n \tag{9.10}$$

which implies the congruence

$$p^2_{n-1} \equiv (-1)^n Q_n \bmod N \tag{9.11}$$

If Q_n is a perfect square, then eq.(9.11) is of the same form as eq(9.1). The CFRAC algorithm generates pairs (p_i, Q_i) that are solutions to eq.(9.11) and then attempts to combine several pairs $(p_1, Q_1), (p_2, Q_2), \ldots, (p_k, Q_k)$ such that

$$\prod_{i=1}^{k} (-1)^i Q_i \tag{9.12}$$

is a perfect square. Morrison and Brillhart (MaB) call such a set of (p_i, Q_i) pairs that satisfies eq.(9.12) an *S*–set. The CFRAC algorithm has three stages:

Stage 1. Select K and form the SCF expansion of $\sqrt{(KN)} = [a_0; a_1, a_2, \ldots, a_n]$ for some $n = n_0$ and collect (p_i, Q_i) pairs ;

Stage 2. From the (p_i, Q_i) pairs collected in stage 1, find all *S*-sets. If no *S*–sets can be found then return to stage 1 ;

Stage 3. Each *S*–set *S* found in stage 2 gives rise to a congruence of the form

$$p^2 = \prod_{i \in S} p_{i-1}^2 \equiv \prod_{i \in S} (-1)^i Q_i = Q^2 \mod N \tag{9.13}$$

where $1 \le p \le N$. From eq.(9.13), let $x = \gcd(N, p-Q)$ for each S–set S found in stage 2. If $1 < x < N$ then x is a non–trivial factor of N. Otherwise return to stage 1.

We will consider the first two stages of the algorithm in detail. In stage 1, $\sqrt{(KN)}$ is being expanded and we are looking for (p, Q) pairs. The SCF expansion of $\sqrt{(KN)}$ as in eq.(9.3) proceeds using the following algorithm

(i) set $p_{-2} = 0, p_{-1} = 1, Q_{-1} = KN, g = \lceil \sqrt{(KN)} \rceil, P_0 = 0, Q_0 = 1, r_{-1} = g$;

(ii) $n = 0$

repeat

generate q_n and r_n	from eq.(9.14);
generate $p_n \bmod N$	from eq.(9.15);
generate $g + P_{n+1}$	from eq.(9.16);
generate Q_{n+1}	from eq.(9.17);

until sufficient (p, Q) pairs have been found;

$$g + P_n = q_n Q_n + r_n \quad 0 \le r_n \le Q_n \tag{9.14}$$

$$p_n \equiv q_n p_{n-1} + p_{n-2} \mod N \tag{9.15}$$

$$g + P_{n+1} = 2g - r_n \tag{9.16}$$

$$Q_{n+1} = Q_{n-1} + q_n(r_n - r_{n-1}) \tag{9.17}$$

example 9.1. Table 9.1 shows the some of the SCF expansion of $N = 5364877$ for $K = 1$.

n	$g+P_n$	Q_n	q_n	r_n	$A_{n-1} \bmod N$
1	2316	1	2316	0	2316
2	4632	1021	4	548	9265
3	4084	$3 \cdot 17 \cdot 43$	1	1891	11581
4	2741	$2^2 \cdot 3 \cdot 197$	1	397	20846
9	3446	$3 \cdot 1091$	1	173	2133239
17	4339	$(2 \cdot 3^2)^2$	13	127	5046913
19	3665	$2^2 \cdot 3 \cdot 167$	1	1661	4501169
26	4561	$2^2 \cdot 3 \cdot 11$	34	73	208344
29	4119	$2^2 \cdot 11 \cdot 23$	4	71	690257
33	3758	$(3 \cdot 11)^2$	3	491	5232305
44	4331	$2^2 \cdot 3 \cdot 29$	12	155	1216814
70	4361	$2^2 \cdot 3^2 \cdot 103$	1	653	626044
90	4258	$3^2 \cdot 10^3$	4	550	5332699
99	3751	$2^2 \cdot 23 \cdot 29$	1	1083	2286668

Table 9.1

Notice that the expansion may yield solutions to eq.(9.1) directly. In this example, Q_{17} and Q_{33} are already perfect squares. We find that gcd(5364877, 5045913–18) = 1 but gcd(5364877, 5232305–33) = 7, and hence 5364877 = 7·766411. •

Th expansion of $\sqrt{N}$, or any surd, will be periodic [ROS], and if the period is short it is possible that an insufficent number of (p, Q) pairs will be generated, and S–sets that produce factorisations may not be found. The constant K is used to increase the period of the expansion. Assume for the moment that the prime factorisation of each Q_i in a (p, Q) pair from stage 1, is known, ie.

$$Q_i = \prod_{i=1}^{k} b_i^{\alpha_i}$$

as shown in the 3rd column of example 9.1. Recall that the algorithm is looking for perfect squares. For a number to be a perfect square each of its prime factors must have an even exponent (ie. each α_i is even). Define the exponent vector $\mathbf{e_i} = (\alpha_0, \alpha_1, \ldots, \alpha_r)$ for each Q_i to be

$$\alpha_0 = \begin{cases} 1 \text{ if } i \text{ is odd} \\ 0 \text{ otherwise} \end{cases}$$

and for $1 \le j \le r$,

$$\alpha_i = \begin{cases} 1 \text{ if } b_j \mid Q_i \text{ to an odd power} \\ 0 \text{ otherwise} \end{cases}$$

Let F be a set of (p, Q) pairs where the cardinality of F is f, $F = \{ (p_i, Q_i) \mid i = 1 \ldots f \}$, and define the history vector $\mathbf{h_i} = (\beta_1, \beta_2, \ldots, \beta_f)$ as

$$\beta_m = \begin{cases} 1 \text{ if } m = i \\ 0 \text{ otherwise} \end{cases}$$

example 9.2 Let $F = \{Q_{26} = 2^2 \cdot 3 \cdot 11,\ Q_{29} = 2^2 \cdot 11 \cdot 23,\ Q_{44} = 2^2 \cdot 3 \cdot 29,\ Q_{99} = 2^2 \cdot 23 \cdot 29\ \}$. Then $|F| = f = 4$, and $r = 5$, where $b_1 = 2$, $b_2 = 3$, $b_3 = 11$, $b_4 = 23$, $b_5 = 29$. The exponent and history vectors are

$\mathbf{e_1} = (1, 0, 1, 1, 0, 0)$ $\quad \mathbf{h_1} = (1, 0, 0, 0)$

$\mathbf{e_2} = (0, 0, 0, 1, 1, 0)$ $\quad \mathbf{h_2} = (0, 1, 0, 0)$

$\mathbf{e_3} = (1, 0, 1, 0, 0, 1)$ $\quad \mathbf{h_3} = (0, 0, 1, 0)$

$\mathbf{e_4} = (0, 0, 0, 0, 1, 1)$ $\quad \mathbf{h_4} = (0, 0, 0, 1)$ •

An exponent vector can be associated with the product of two Q_i's from F, which will be the sum of the exponent vectors of the respective Q_i in the product. The sum will be computed in the $r + 1$

dimensional vector space $Z^{r+1}{}_2$ over Z_2. The addition of the corresponding history vectors will indicate which Q_i have been multiplied.

example 9.3. From example 9.2, computing in $Z^6{}_2$, multiplying Q_{26} and Q_{29} the exponent vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ are added, ie. $\mathbf{e}_1 + \mathbf{e}_2 = (1, 0, 1, 1, 0, 0) + (0, 0, 0, 1, 1, 0) = (1, 0, 1, 0, 1, 0)$. The history vector, computed in $Z^4{}_2$, is $(1, 0, 0, 0) + (0, 1, 0, 0) = (1, 1, 0, 0)$. •

An S–set S is a subset of F such that the product of each $Q_i \in S$ forms a perfect square. This would imply that the exponent vector resulting from the product if the Q_i must be the null vector (ie. all zeros), which indicates that all the exponents are even. Thus if F has an S–set, there must be a set of linearly dependent vectors (over $Z^{r+1}{}_2$) in F. MaB use an algorithm similar to Gaussian elimination to determine if a set of exponent vectors is linearly dependent over $Z^{r+1}{}_2$. The rightmost one refers to the first one encountered when examining a vector from right to left, eg. the rightmost 1 is highlighted here $(0, 1, 0, 0, 1, \mathbf{1}, 0)$. The reduction algorithm is

(i) set $j = r$;

(ii) find the pivot vector $\mathbf{e}_i$ of the smallest subscript whose rightmost 1 is in the *jth* component. If none exist, go to (iv).

(iii) (a) replace every vector $\mathbf{e}_m$, $i < m \leq f$, whose rightmost 1 is in the *jth* component, by the sum $\mathbf{e}_i + \mathbf{e}_m$, computed in $Z^{r+1}{}_2$. (b) when $\mathbf{e}_m$ is repleced by $\mathbf{e}_i + \mathbf{e}_m$, replace $\mathbf{h}_m$ by $\mathbf{h}_i + \mathbf{h}_m$, computed in $Z^f{}_2$.

(iv) $j = j - 1$; if $j \geq 0$ then goto (ii) otherwise stop.

example 9.4. Finding S–sets and a factorisation. Using the information in Table 9.1, let $F = \{ Q_{19}, Q_{26}, Q_{44}, Q_{70}, Q_{90}, Q_{99} \}$. Each Q_i can be represented by a product of the primes 2, 3, 11, 23, 29, 103 taken to suitable powers. Construct an $f \times (r + 1 + f)$ matrix, with each row being the exponent vector followed by the history vectors for each Q_i. The initial matrix is

\+ 2 3 11 23 29 103

$$\begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & \quad & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and the final matrix after running the reduction algorithm is

$$\begin{array}{c} \begin{matrix} + & 2 & 3 & 11 & 23 & 29 & 103 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & \quad & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

The last two vectors are linearly dependent over Z^6_2. Using the 6th row of the final matrix, we have that

$$(2286668 \cdot 1216814 \cdot 690257 \cdot 208344)2 \equiv (24 \cdot 32 \cdot 11 \cdot 23 \cdot 29)2 \mod 5364877$$

which is equivalently

$$(1659408)2 \equiv (17941)2 \mod 5364877$$

but yields on trivial factor of N as gcd(5364877, 1659408–17941) = 1. Using the 5th row of the final matrix though, we have that

$$(626044 \cdot 5332699)2 \equiv (2 \cdot 32 \cdot 103)2 \mod 5364877$$

which is equivalently

$$(937236)2 \equiv (1854)2 \mod 5364877$$

and gcd(5364877, 937236–1854) = 7 and that N = 5364877 = 7·766411. •

The CFRAC method requires that the complete prime factorisations of the Q_i be known, if the S–sets are to constructed. In the examples of this chapter, we have assumed that the factorisations are known. Finding these factorisations is the most time consuming part of the algorithm. Clearly if $Q_i = a_1 a_2$ has a large prime factor a_1, then there must be another $Q_j = a_1 a_3$ if Q_i (and Q_j) are to be included in an S–set. But as a_1 increases in size finding a suitable Q_j becomes time consuming and unlikely. MaB introduce the idea of a "factoring base" , where any Q_i that cannot be factored into the primes in the factoring base is dropped from consideration and is regarded as being "too large". A large portion of numbers not only have small prime factors but are entirely composed of small prime factors. A number a is said to be *smooth* with respect to a bound d if and only if

$$a = \prod_{i=1}^{k} p_i^{b_i} \text{ and } p_i < d$$

where the p_i are prime. The observation of Morrison and Brillhart indicates that smooth numbers are common. From example 9.4, the factor base consisting of the primes {2, 3, 11, 23, 29, 103} produced 2 S–sets quickly. Additionally, MaB use a bound UB, where if Q_i has a prime factor larger than this bound then it is discarded. Let FB be the number of primes in the factor base, then MaB gave Table 9.2 as a guide to factoring large numbers.

number of digits	FB	UB
≤ 20	60	3000
21–23	150	10000
24–25	200	14400
26, 28	300	22500
29, 30	400	29000
31, 32	450	36000

33, 34	500	36000
35, 36	550	36000
37, 38	600	44000
39, 40	650	53000
41, 46	700–1000	63000

Table 9.2

MaB used the CFRAC algorithm to give a factorisation of F_7.

9.2 Factoring and Security

The RSA system is based on the assumption that multiplication is a one way function when restricted to prime numbers. Given $n = pq$ it is difficult to find p and q, as the above discussion attests. The designer selects an integer e such that $(e, \phi(n)) = 1$ and then calculates d such that

$$ed \equiv 1 \mod \phi(n)$$

where $\phi(n) = n - (p + q) + 1$. The public key is the pair (e, n). The message m, $1 \leq m \leq n$, is encrypted by the rule $c \equiv m^e \mod n$. The designer decrypts by the rule $m \equiv c^d \mod n$. The message is recovered since $c^d \equiv m^{ed} \equiv m^{\phi(n)+1} \equiv m^1 m^{\phi(n)} \equiv m \mod n$. The problem of inverting the encryption function E is the task of computing roots in finite fields. To recover the message m we need to know the *eth* root of

$$c = m^e \mod n \ .$$

There is no known way of finding this root without the factors of n.

Let RSA^{-1} be a routine such that when given the triplet $<m^e \mod n, e, n>$ as input, returns the encrypted message m. How hard is this "magic box" to realise? Consider the following three problems,

(9.1) FAC_n : factoring the modulus;

(9.2) PHI_n : determining the Euler totient function of the modulus;

(9.3) KEY_d : finding the secret key d.

Using the notation of Miller [MILL], we introduce the concept of functional reducibility (a restricted form of Turing reducibility). Given two functions f and g, we say that f is polynomial time reducible to g , denoted $f \leq_p g$, if there exists a Turing machine such that on inputs n and $g(n)$, computes $f(n)$ in time $O(n^k)$. Further, if $f \leq_p g$ and $g \leq_p f$ then f and g are polynomial equivalent, denoted by $f \approx_p g$. Immediately, we have the following chain of reductions

$$RSA^{-1} \leq_p KEY_d \leq_p PHI_n \leq_p FAC_n \tag{9.18}$$

From right to left, the reductions mimic the sequence of problems the designer has solved in the construction of the system. These reductions imply that the task of compromising the security of the system can be no harder than factoring the modulus. But the order of the last three problems is arbitrary. We have

Lemma 9.4. $PHI_n \approx_p FAC_n$.

proof. The lemma follows easily from the following observation.

$$\begin{aligned} \phi(n) &= n - (p + q) + 1, \\ (p - q) &= ((p + q)^2 - 4n)^{1/2}, \\ q &= ((p + q) + (p - q))/2. \end{aligned} \tag{9.19}$$

•

Miller [MILL] has proved the general equivalence between finding the prime factorisation of a number n and computing the function $p(n)$, where $\lambda'(n) \mid p(n)$†, assuming the Extended Reimann Hypothesis (ERH)[3]. In §13 we will show that KEY_d is randomly equivalent to FAC_n and hence

†Where if $n = p_1^{h1} p_2^{h2} \ldots p_k^{hk}$, then $\lambda'(n) = \text{lcm}(p_1-1, p_2-1, \ldots, p_k-1)$ and lcm is the least common multiple. Thus Miller's result also includes an equivalence between the Euler totient and the Carmichael functions, given the ERH.

[3]For an explanation of the ERH see [KRAN], [KNU2], [WILL1].

PHI_n when a certain protocol is used. Even so, there is no known reduction from any of these problems to RSA^{-1}, though in §16 we show that solving RSA^{-1} is equivalent to factoring the modulus when n is of a special form.

SECTION 10 Quadratic Residuosity

Quadratic residuosity, like factoring, is recognised as a difficult problem, and was one of the four algorithmic problems addressed by Gauss in his *Disquisitiones Arithmeticae* (1801). Let $\gcd(a, n) = 1$; if there is a solution to the congruence

$$x^2 \equiv a \bmod n \tag{10.1}$$

then a is said to be a quadratic residue mod n. Otherwise, a is said to be a quadratic nonresidue of n. Let QR_n denote the set of quadratic residues mod n. Unlike factoring, the problem of determining quadratic residuosity is a decision problem, stated by the predicate $QR_n(x)$,

$$QR_n(a) = \begin{cases} 1 & \text{if } a \in QR_n \\ 0 & \text{if } a \notin QR_n \end{cases}$$

The set $Z_n = \{0, 1, 2, \ldots, n-1\}$ is referred to as the ring of integers modulo n . If we define the set Z^*_n to be $\{ a \in Z_n \mid \gcd(a, n) = 1 \}$, then QR_n will be a subset of Z^*_n (see example 10.1). Let $\gcd(a, n) = 1$; then the least positive integer x such that $a^x \equiv 1 \bmod n$ is called the order of a modulo n, denoted $\mathrm{ord}_a n$. If $\mathrm{ord}_a n = \phi(n)$ then a is said to be a primitive root modulo n.

Lemma 10.1. If a is a primitive root modulo n, then

$$Z^*_n = \{ a^1 \bmod n, a^2 \bmod n, \ldots, a^{\phi(n)-1} \bmod n, a^{\phi(n)} \bmod n \}.$$

proof. Since $\gcd(a, n) = 1$ then $\gcd(a^k, n) = 1$. Thus each $a^i \bmod n$, $i = 1 \,..\, \phi(n)$ is relatively prime to n. To show that they are distinct assume that

$$a^i \equiv a^j \bmod n .$$

Since $\mathrm{ord}_a n = \phi(n)$, we must have

$$i \equiv j \bmod \phi(n) .$$

This congruence implies that $i = j$ since $1 \le i, j \le \phi(n)$. •

We also have the well known (see [ROS, Theorem 8.7], [HaW] [BAK] for proofs)

Lemma 10.2. Let n be a prime and let d be a positive divisor of $n-1$. Then the number of incongruent integers of order d modulo n is equal to $\phi(d)$. •

As an immediate consequence of Lemma 10.2, we have

Corollary 10.1 Every prime n has $\phi(n-1)$ primitive roots. •

Lemma 10.3. If n is prime, the congruence $x^2 \equiv a \bmod n$ has either two solutions or none.

proof. Follows easily from the fact that if $x^2 \equiv y^2 \pmod{n}$ and n is a prime, then $x \equiv \pm y \pmod{n}$. •

Corollary 10.2. If n is an odd prime, there are $\phi(n)/2$ quadratic residues mod n.

proof. Let g be a primitive root mod n and let a be a quadratic residue of n. Then $x^2 \equiv a \bmod n$ has a solution for x and x can be written as $x \equiv g^i \bmod n$, for some i, $1 \le i \le \phi(n)$. Since QR_n is a subset of Z^*_n, every quadratic residue a can be written as $a \equiv g^k \bmod n$ for some k, $1 \le k \le \phi(n)$. Further, k must be even as $a \equiv g^i g^i \equiv g^{2i} \bmod n$, and $2i \equiv k \bmod \phi(n)$. As k can only assume $\phi(n)/2$ even values modulo $\phi(n)$, there are exactly $\phi(n)/2$ quadratic residues modulo n. •

Let g_1 and g_2 be primitive roots of Z^*_p and Z^*_q respectively, where p and q are primes. We have

Lemma 10.4. Let a and b be such that

$$a \equiv g_1 \bmod p \quad \text{and} \quad a \equiv 1 \bmod q\,,$$
$$b \equiv 1 \bmod p \qquad b \equiv g_2 \bmod q\,.$$

These integers will exist by the Chinese Remainder Theorem. Then, any $x \in Z_n^*$, where $n = pq$, can be written as $a^i b^j \bmod n$, where $1 \le i \le p-1$, $1 \le j \le q-1$.

proof. The cardinality of the set $G = \{ a^i b^j \bmod n \mid 1 \le i \le p-1, 1 \le j \le q-1 \}$ is $\phi(p)\phi(q)$, and for each $x \in G$, $\gcd(x, n) = 1$. The cardinality of Z^*_n is by definition $\phi(n) = (p-1)(q-1)$. It remains to show that the members of G are distinct. Let

$$a^i b^j \equiv a^k b^m \bmod n$$

where $n = pq$. But since $b \equiv 1 \bmod p$, we have

$$a^i \equiv a^j \bmod p \ ,$$

and since $a \equiv g_1$ and $1 \le i, k \le \phi(p)$, we get $i = k$. Similarly $j = m$. •

If n is prime, there is an efficient method of deciding $QR_n(a)$. The Legendre symbol of a with respect to n is defined as,

$$(a \mid n) = \begin{cases} 1 & \text{if } a \in QR_n \\ -1 & \text{if } a \notin QR_n \\ 0 & \text{if } n \mid a \, . \end{cases}$$

Information about the residuosity of a mod n can be gained from Euler's criterion, stated as

Lemma 10.5. For all primes $p > 2$, and all $a \in Z^*_p$, $a^{(p-1)/2} \equiv (a \mid p) \bmod p$. •

Since modular exponentiation can be computed in time no greater than $O(log^3 n)$ [KNU2], for prime n, we have $QR_n(a) \in \mathbf{P}$. A generalisation of the Legendre symbol to composite moduli is provided by the Jacobi symbol. If

$$n = \prod_{i=1}^{k} p_i^{b_i}$$

where p_i is prime, then the Jacobi symbol $(a \mid n)$ is defined as

$$(a \mid n) = (a \mid p_1)^{b_1} (a \mid p_2)^{b_2} \ldots (a \mid p_k)^{b_k} \ .$$

Even so, we need not know the factors of n to evaluate $(a \mid n)$. Let $R_0 = a$ and $R_1 = n$. We then divide R_0 by R_1, and factor out the highest power of two from the remainder;

$$R_0 = R_1 q_1 + 2^{s_1} R_2 ,$$

where s_1 is a nonnegative integer and R_2 is an odd positive integer less than R_1. Using a technique similar to that employed by the Euclidean algorithm, we successively factor out the highest power of two from the remainder to obtain

$$\begin{aligned} R_1 &= R_2 q_2 + 2^{s_2} R_3 \\ R_2 &= R_3 q_3 + 2^{s_3} R_4 \\ &\cdot \\ &\cdot \\ R_{k-3} &= R_{k-2} q_{k-2} + 2^{s_{k-2}} R_{k-1} \\ R_{k-2} &= R_{k-1} q_{n-1} + 2^{s_{k-1}} \cdot 1 \end{aligned}$$

where s_j is a nonnegative integer and R_j is an odd positive integer less than R_{j-1} for $j = 2, 3, \ldots, k-1$. We have

Lemma 10.6. Let R_j and s_j, $j = 1, 2, \ldots k-1$ be as described above. Then

$$(a \mid n) = (-1)^{s_1 \frac{R_1^2-1}{8} + \cdots + s_{k-1} \frac{R_{k-1}^2-1}{8} + \frac{R_1-1}{2} \cdot \frac{R_2-1}{2} + \cdots + \frac{R_{k-2}-1}{2} \cdot \frac{R_{k-1}-1}{2}} \quad \bullet$$

See [ROS, Theorem 9.7] for a proof. Since the division of two h–bit numbers can be performed in $O(\log^3 h)$ operations [KNU2], it follows that

Corollary 10.3. The Jacobi symbol $(a \mid n)$ can be evaluated in polynomial time. •

On first impressions, there appears to be no problem of determining $QR_n(a)$ for arbitrary n, as both the Legendre and Jacobi symbols can be efficiently evaluated. However, the Jacobi symbol is not

an indicator of quadratic residuosity. In what follows, let n be the product of two odd primes p and q.

Lemma 10.7. For integers x and y, $x \equiv y \bmod n$ if and only if $x \equiv y \bmod p$ and $x \equiv y \bmod q$.

Corollary 10.4 $a \in QR_n$ if and only if $a \in QR_p$ and $a \in QR_q$. •

Thus a is a quadratic residue mod n if and only if both the Legendre symbols $(a \mid p)$ and $(a \mid q)$ are positive. We will only be interested in those cases where $\gcd(a, n) = 1$, and from the definition of the Legendre symbol, $(a \mid n)$ can only be ± 1. If $a \in QR_n$ then $(a \mid n) = 1$. But, for integers a that are quadratic nonresidues of both p and q, the Jacobi symbol will be positive as

$$(a \mid n) = (a \mid p)(a \mid q) = (-1)(-1) = 1 .$$

Thus, $(a \mid n) = -1$ indicates quadratic nonresiduosity; but, if $(a \mid n) = 1$, a could be either a quadratic residue or nonresidue. The Jacobi symbol can be used to partition Z^*_n into 2 distinct subsets, $Z^*_n(+1)$ and $Z^*_n(-1)$, where

$$a \in Z^*_n(+1) \quad \text{if} \quad (a \mid n) = 1,$$

$$a \in Z^*_n(-1) \quad \text{if} \quad (a \mid n) = -1.$$

Set membership of $Z^*_n(-1)$ can be decided by computing the Jacobi symbol, which directly indicates nonresiduosity. Denote these nonresidues as $QNR_n{-}1$. When n is composite, the set $Z^*_n(+1)$ can itself be partitioned into two subsets, QR_n (the residues) and $QRN_n{+}1$, where $a \in QRN_n{+}1$ if $(a \mid n) = 1$ and a is a nonresidue mod n. We have

Lemma 10.8. $|QR_n| = |QNR_{n+1}| = |QNR_{n-1}|/2 = \phi(n)/4$.

proof. Let a, b, p and q be as defined in Lemma 10.4. If $x \in QR_p$ then $x \equiv a^{2i} \bmod p$ for some i, $1 \le i \le \phi(p)/2$. Similarly, if $x \in QR_q$ then $x \equiv b^{2j} \bmod q$ for some j, $1 \le j \le \phi(q)/2$. By Corollary 10.3, $x \in QR_n$ if and only if $x \in QR_p$ and $x \in QR_q$. By Lemma 10.4 every $x \in Z^*_n$, $n = pq$, can be written as $x \equiv a^i b^j \bmod n$. Hence, $x \in QR_n$ if and only if x can be written as $a^{2i}b^{2j} \bmod n$, where $1 \le 2i \le \phi(p)$, $1 \le 2j \le \phi(q)$. Since $2i$ may only assume $\phi(p)/2$ values and $2j$ may only assume $\phi(q)/2$ values, there can only be $\phi(q)\phi(p)/4$ quadratic residues. From Corollary 10.1 we have that QNR^{-1}_n is characterised by the set of indices

$$\{\, a^i b^j \bmod n \mid i, j \equiv 1 \bmod 2 \,\}.$$

There are exactly $\phi(q)\phi(p)/4$ pairs of indices i and j that satisfy $i, j \equiv 1 \bmod 2$. The set $Z^*_n(-1) = QNR^{-1}_n$ is precisely the following set union:

$$\{\, a^i b^j \bmod n \mid i \equiv 1 \bmod 2, j \equiv 0 \bmod 2 \,\} \cup \{\, a^i b^j \bmod n \mid j \equiv 1 \bmod 2, i \equiv 0 \bmod 2 \,\}.$$

Each of the sets in this union has cardinality $\phi(q)\phi(p)/4$; thus, $|QNR_{n-1}|/2 = \phi(n)/4$. •

Example 10.1. Let $n = 35 = 5 \cdot 7 = p \cdot q$. Then $|Z^*_n| = 24 = \phi(q)\phi(p) = 6 \cdot 4$. The integer 3 is a primitive root in Z^*_5 since

$$Z^*_5 = \{\, 3^0 = 1, 3^1 = 3, 3^2 \equiv 4 \bmod 5, 3^3 \equiv 2 \bmod 5 \,\}.$$

Also, 5 is a primitive root in Z^*_7 as

$$Z^*_7 = \{\, 5^0 = 1, 5^1 = 5, 5^2 \equiv 4 \bmod 7, 5^3 \equiv 6 \bmod 7, 5^4 \equiv 2 \bmod 7, 5^5 \equiv 3 \bmod 7 \,\}.$$

The values of $a = 8$ and $b = 26$ are solutions to the two sets of congruences

$$a \equiv 3 \bmod 5 \quad \text{and} \quad a \equiv 1 \bmod 5$$

$$b \equiv 1 \bmod 7 \qquad b \equiv 5 \bmod 7$$

In Table 10.1 we show that every integer in Z^*_{35} can written as $8^i26^j \bmod 35$, $1 \le i \le 4$, $1 \le j \le 6$

i	$8^i26^j \bmod 35,\ 1 \le j \le 6$
1	{33, 18, 13, 23, 3, 8}
2	{19, 4, 34, 9, 24, 29}
3	{12, 32, 27, 2, 17, 22}
4	{26, 11, 6, 16, 31, 1}

Table 10.1

Table 10.2 generates Z^*_{35} according to Lemma 10.7. Column 1 shows the QR_{35}, column 2 $QNR{+}1_{35}$ and column 3 shows $Z^*_{35}(-1) = QRN{-}1_{35}$, where $i \oplus j \equiv 1 \bmod 2$ indicates i or j are $\equiv 1 \bmod 2$ but not both,

$i, j \equiv 0 \bmod 2$	$i, j \equiv 1 \bmod 2$	$i \oplus j \equiv 1 \bmod 2$
4	33	18 23
9	13	8 19
29	3	34 24
11	27	12 32
16	17	2 22
1	31	26 6

Table 10.2

and $|QR_{35}| = |QNR{+}1_{35}| = |QNR{-}1_{35}|/2 = 6 = \phi(q)\phi(p)/4$. •

The only known way to compute the characteristic functions on the sets QR_n and $QRN_n{+}1$ is to evaluate the respective Legendre symbols for the factors of n. In other words, without knowing the

factorisation of n, it is apparently difficult to determine whether $a \in QR_n$ or $a \in QRN^{+1}$, ie. no polynomial time algorithm is known for determing set membership for either partition of $Z^*_n(+1)$. We define the quadratic residuosity problem with respect to n, QRP_n below.

INSTANCE : given $n = pq$, where p and q primes, and $a \in Z^*_n(+1)$.

QUESTION : Is $a \in QR_n$? .

There is no precise complexity measure for computing QRP_n, but since

$$QRP_n \leq_p \mathrm{FAC_n} \ ,$$

its difficulty cannot exceed that of factoring. It is generally accepted that knowing the factors of n is also a necessary, as well as sufficient condition, to determine quadratic residuosity. In §16 we show that factoring and determining quadratic residuosity are equivalent problems when n is specially chosen.

Measuring complexity in terms of elementary operations and running times is derived from the concept of Turing machines, the first model of an effectively computable function. Alternatively, circuit complexity, another computational model, could be used to measure the difficulty of QRP_n. The reference to factoring however remains.

Let p and q be primes and let the notation $|x|$ denote be the number of bits in x. Then we define

$$H_k = \{ n \mid n = pq, \ \log_2 p = \log_2 q = k \}$$

If factoring has intractable instances, then H_k will contain some of these instances. The intractability of QRP_n is formulated in the Quadratic Residuosity Assumption (QRA), given below.

QRA [GaM]. Let p_1 be a fixed polynomial. For each integer k, let C be a cicuit with $2k$-bit inputs, and one Boolean output. Let C_k be the minimum size of circuit C such that for

a fraction $1/p_1(k)$ of $h \in H_k$, $C[n, a] = QR_n(a)$, for all $a \in Z^*_n(+1)$. Then for all polynomials Q, and for all large k : $C_k > Q(k)$.

To certify the diffculty of QRP_n we may appeal to either the QRA or the problem of factoring.

One of the designer's problems is to convince himself that he can find a hard instance of a hard problem. Quadratic residuosity is blessed, in that the delineation between easily solved instances, and those instances which are not known to be easily solved, is quite obvious. We simply inspect the modulus. By Lemma 10.5, if n is prime, QRP_n can be decided in polynomial time. On the other hand, if n is composite, in particular if n is the product of 2 large primes, QRP_n is assumed hard. Also, by Corollary 10.2 and the properties of the Jacobi symbol, QRP_n is solved easily if the factorisation of n is known. Hence, we have a problem, a hard instance of the problem and a trapdoor – all the necessary tools for a cryptosystem. This system, devised by Goldwasser and Micali [GaM], encrypts bit by bit, and the cipherspace is $Z^*_n(+1)$. If the message bit is one, then a quadratic residue will be sent, otherwise, a nonresidue that has a positive Jacobi symbol will be sent.

How is the encryption realised? The public key consists of a composite n and some $y \in QNR+1_n$. The importance of y will become apparent from the following lemma.

Lemma 10.9. (i) the product of two quadratic residues is a quadratic residue; (ii) the product of a non-residue with a residue is a non-residue; (iii) The product of two quadratic nonresidues is a quadratic residue.

proof. Let $y_1, y_2 \in QR_n$. Then using the notation from Lemma 10.4, $y_1 \equiv a^{2i}b^{2j} \bmod n$ and $y_2 \equiv a^{2k}b^{2m} \bmod n$, where $1 \leq i, k \leq \phi(p)/2$, and $1 \leq j, m \leq \phi(q)/2$. Then $y_1 \cdot y_2 = a^{2(i+k)}b^{2(j+m)} \bmod n$. Arithmetic in the exponent of a can be done modulo $\phi(p)$ as $a^{\phi(p)} \equiv 1$, and since $\phi(p)$ is even, then $2(i+k) \bmod \phi(p)$ is also even. Similarly, $2(j+m) \bmod \phi(q)$ will be even, and hence $y_1 \cdot y_2 \bmod n \in QR_n$, as the indices of a and b will be even. Let $y_3 \equiv a^{2i+1}b^{2k+1} \bmod n$ be a nonresidue mod n. Then $y_2 \cdot y_3 = a^{2(i+j)+1}b^{2(j+m)+1} \bmod n$, which will be a nonresidue, as reducing the indexes of a and b will leave odd numbers. Part (iii) easily follows. •

To encrypt, the sender must be able to generate random elements of QR_n and $QRN^{+1}{}_n$. The square of any $a \in Z^*_n$ is a quadratic residue, and by Lemma 10.6 y can be used to produce the non-residues. For each bit m_i of the message m, the sender randomly generates an x and computes $a \equiv x^2 \bmod n$.

If $m_i = 1$, then send $c = a$;

else, send $c = ya \bmod n$.

The receiver who knows the factorisation of n can compute $(c \mid p)$ and $(c \mid q)$ to determine $QR_n(c)$.

Notice that each member of the plaintext alphabet can be encrypted into more than one member of the cipherspace. That is, a 1 can be encrypted to any residue, and a 0 encrypted to any element of $QRN^{+1}{}_n$. As the encryption algorithm involves randomly selecting a member from the particular subset of the cipherspace, encryption is said to be *probabilistic*. The technique of mapping plaintext characters into disjoint subsets of the cipherspace is called *homophonic substitution*.

$$\{1\} \longrightarrow \{QR_n\}$$

$$\{0\} \longrightarrow \{QRN+1_n\}$$

Those elements of the cipherspace for which the Jacobi symbol is negative are not used. Encryption is the mapping $\{0, 1\} \longrightarrow Z^*_n(+1)$. There is a finer partition on $Z^*_n(+1)$ but only the designer can realise this partition in a feasible amount of time. Even though the system is secure, a practical implementation is uneconomic. First, each bit requires a modular squaring to be encrypted which is costly as compared to conventional bit systems based on *exclusive or*-ing. Secondly, to enforce intractability, n must be large, say 200 decimal digits, which means that each bit is expanded into approximately 645 bits. Both encryption and decryption can be performed in $O(k\ log3n)$ time where k is the number of bits in the message. Even though messages can be manipulated in polynomial time, the size of the modulus makes communication unacceptably slow.

SECTION 11 The Discrete Logarithm Problem

In this section we will be considering the relation between two problems in a finite field GF(q):

$$y \equiv g^x \pmod{q},$$

and its inverse

$$x = \log_g y \text{ over GF}(q),$$

which are known as the exponential and logarithmic functions to the base g (mod q) respectively. Let g be a primitive element of the field. By Lemma 10.1 there exists an integer x, $0 \le x \le q-2$ such that for each $y \in$ GF(q), $y \equiv g^x \bmod q$. We call x the discrete logarithm of y to the base g, denoted by $x = \log_g y \pmod{q}$.The discrete logarithm problem (DLP) is stated as, given $y \in$ GF(q), $y \neq 0$, compute $x = \log_g y \pmod{q}$.

The field elements of GF(p^n) are interpreted as polynomials of degree at most $n-1$, with coefficients over GF(p). Multiplication of polynomials in this field can be reduced modulo an irreducible polynomial of degree n. Fields of the form GF(p^n) are of particular interest to cryptography, since arithmetic can be preformed much more efficiently than in GF(q).

11.1 Algorithms for the Discrete Logarithm Problem

We begin with the general remark that exponentiation can be performed easily while there is at present no efficient solution to the DLP. For this reason exponentiation is thought to be a one way function. Using the power method of repeated squaring[1], $g^x \bmod q$ can be computed in $2\cdot\lceil \log_2 q \rceil$ multiplcations using 3 words of memory, each of $\lceil \log_2 q \rceil$ bits, where g and x are $\log_2 q$ bits in length [PaH]. The remainder of this section will be concerned with general and special purpose algorithms for solving the DLP, and how these algorithms are applied to cryptography.

[1] See [KNU2].

The polynomial ring GF(2^n) is of particular interest as

- polynomials over GF(2) can be factored easily [BERK];
- addition and multiplication only require addition of coefficients with no carries;
- inverses can be easily found;

The importance of factoring polynomials will unfold as we examine the general algorithms, though the time spent factoring will be subsumed in the other stages of the algorithms.

One of the earliest published algorithms for the DLP was given by Bouniakowsky in 1870. This algorithm solved the congruence $g^x \equiv y \bmod p$ in O(p) operations [BOUN]. Since then, the complexity of computing the DLP has roughly decreased to O($p^{1/2}$). In fact, the best algorithm for solving the DLP is asymptotically equivalent to the best algorithm for factoring (cf. §6). In 1979 Leonard Adleman [ADL1] (and independently Merkle [MER3]) published a subexponential algorithm for the DLP. Subsequently, this algorithm and its variants have been referred to as the index calculus method [ODLY]. Adleman's algorithm was originally proposed for computing logarithms in GF(q), but it has been adapted for fields of the form GF(2^n) (this adaptation was first proposed in [HELL3]). The success of Adleman's algorithm relies on the factoring heuristic introduced by Morrison and Brillhart [MaB] (see §9.1), namely that a large portion of numbers not only have small prime factors but are entirely composed of small prime factors. The success of the index calculus method depends on the probability that a random polynomial over GF(q) can be expressed as the product of small irreducible polynomials.

The algorithm is in two parts: first, a precomputation stage, followed by the extraction of the actual logarithm. Let $g(x)$ be a primitive element of GF(2^n). The first stage of the algorithm consists of determining the discrete logarithm for each member of a set S with respect to $g(x)$. The set S will consist of all irreducible polynomials over GF(2) with degrees less than some integer b (the choice of which will be discussed later; but at present we note that it is not arbitrary). Table 11.1 shows the number of irreducible polynmials less than degree b, $I(n, b)$, for some b.

b	$I(n, b)$
1	2
3	2
5	6
10	99
12	335
15	2182
18	14532
20	52377

Table 11.1#

The set S will be referred to as the factor base. The logarithms of the elements of S, the set of irreducible polynomials with degree less than b, are found by solving $|S|$ linear equations in $|S|$ unknowns, where the unknowns are the logarithms (see Table 11.1 for the sizes of $|S|$). The methods of Adleman and Merkle differ in how these equations are formed and solved, and we discus Adleman's method here. Clearly, the number of unknown logarithms will depend upon b. Let s be an integer such that $1 \leq s \leq 2^{n-1}$. Form polynomials

$$h(x)^* \equiv (g(x))^s \ (mod\ f(x)), \text{ where degree } h(x)^* < n .$$

Next, check if the irreducible factors $h(x)^*$ are all have degree less than b, ie. they all are contained in the factor base S. If so, then $h(x)^*$ can be expressed as

$$h(x)^* = \prod_{p_i(x) \in S} (p_i(x))^{b_i} \quad ;$$

thus we obtain the congruence

[ODLY, p298].

$$S \equiv \sum_{p_i(x) \in S} b_i log_g p_i(x) \quad mod\ 2^n - 1 . \tag{11.0}$$

Once we obtain a few more than $|S|$ congruences of the form in eq.(11.0), it should be possible to determine the logarithms of the irreducible polynomials $p_i(x)$. Solving the system using Gaussian elimination should take $O(|S|^3)$ steps. Generating the system of congruences that when solved will yield the logarithms of all the polynomials in S will take about $|S|p(n, b)^{-1}$ steps, where $p(n, b)$ is the probability that a randomly chosen polynomial of degree n will have all its irreducible factors less than degree b. If we let $N(n, b)$ be the number of polynomials of degree n with irreducible factors less then degree b, then

$$p(n, b) = \frac{N(n, b)}{2^n} .$$

In implementing Adleman's algorithm in GF(2^{127}), Blake *et al.* [BFMV] used "systematic equations" to efficiently produce about half the necessary equations for the logarithms of the factor base. This method takes advantage of the property that squaring is a linear operation in this field. Coppersmith [COP] extended this idea to obtain a richer set of equations that would produce all the necessary equations for finding the logarithms in the precomputation stage. The systematic equation technique can be used to rapidly generate the logarithm database for GF(2^{127}).

Now, suppose that we want to find the discrete logarithm of $h(x)$, ie. we wish to solve

$$h(x) \equiv (g(x))^t \ (mod\, f(x)) \tag{11.1}$$

for $t \in Z_{2^h-1}$ where $f(x)$ is an irreducible polynomial. Choose a random integer s, where $0 \le s \le 2^n-2$, and compute

$$h(x)^* \equiv h(x)(g(x))^s \ (\text{mod}\ (f(x)) \tag{11.2}$$

The polynomial $h(x)^*$ is then factored into irreducible polynomials $p_i(x)$, and if all of these factors are in S, then we are done. Since, if

$$h(x)^* \equiv h(x)(g(x))^s \equiv \prod_{i=1}^{k} (p_i(x))^{b_i} \pmod{f(x)} \tag{11.3}$$

where $p_1, p_2, \ldots p_k \in S$ then,

$$log_{g(x)}h(x) = \sum_{i=1}^{k} b_i log_{g(x)}p_i(x) - s \ . \tag{11.4}$$

This can be evaluated by consulting the table of logarithms computed in the first stage if each $p_i(x)$ is in the factor base. The success of the algorithm, then, depends upon finding a polynomial whose factors are all contained in the factor base S. We would expect on average that $p(n, b)^{-1}$ selections of s for eq.(11.2) would need to be made before the logarithm of $h(x)$ can be determined. Odlyzko [ODLZ] estimates that

$$p(n, b)^{-1} = \left(\frac{n}{b}\right)^{(1 + o(1))(n/b)} \tag{11.5}$$

which is the running time of the second stage of the algorithm. Table 11.2 shows some of his estimates for $p(n, b)^{-1}$[ODLY].

	n, the degree of the polynomial				
b	20	60	200	300	500
1	2.00272E–05	5.29091E–17	1.25083E–58	1.47764E–88	1.53052E–148
6	4.19769E–02	2.58718E–09	1.90979E–43	2.48750E–70	3.16656E–126
11	4.24762E–01	1.80902E–04	3.98840E–24	2.33703E–41	7.79518E–80
16	7.83160E–01	1.10167E–02	7.8766E–15	7.51397E–26	6.27 531E–51

21	1.00000E+00	7.13458E–02	3.17240E–10	5.17007E–18	5.96843E–36
26	1.00000E+00	1.91288E–01	1.37605E–07	1.80862E–13	3.64471E–27
31	1.00000E+00	3.47376E–01	6.44534E–06	1.47287E–10	1.76903E–21
36	1.00000E+00	4.94689E–01	8.76481E–05	1.46051E–08	1.53163E–17

Table 11.2

When n is 10 and $b \geq 10$ then with probability 1 a polynomial of degree 10 will factor into the factor base S. Adleman [ADL1] analysed the running time for his algorithm in GF(q) to be *exp(c sqrt(log q loglog q))* .The adpation by Hellman *et al.* [HELL3] yielded a running time of *exp(c sqrt(n log n))* for GF(2^n)* .

The second stage of index calculus method can be improved. Instead of factoring the polynomial $h(x)^*$ of (11.2), Blake *et al.* [BFMV] suggest finding two ploynomials $p_1(x)$ and $p_2(x)$ such that

$$h(x)^* \equiv \frac{p_1(x)}{p_2(x)} \pmod{f(x)} \tag{11.6}$$

where degree $p_i(x) \leq n/2$ for i = 1, 2. These polynomials can be found by applying the extended Euclidean algorithm [KNU2] to the polynomials $h^*(x)$ and $f(x)$ of eq.(11.2). This algorithm produces two polynomials $p_1(x)$ and $p_2(x)$ such that $p_1(x)h^*(x) + p_2(x)f(x) = 1$. The algorithm achieves this iteratively and produces a sequence of triples of polynomials $(p_{1,i}(x), p_{2,i}(x), k_i(x))$ such that

$$p_{1,i}(x)h^*(x) + p_{2,i}(x)f(x) = k_i(x) \tag{11.7}$$

*Each of these estimates is based on certain assumptions about the randomness and independence of polynomials. An analysis of probabilistic algorithms will be given in §13.

where deg $k_1(x) >$ deg $k_2(x)$ ..., and eventually $k_i(x) = 1$. If we select that i such that $k_i(x)$ is closest to $n/2$ then eq.(11.7) is satisfied most of the time with $p_2(x) = p_{1,i}(x)$ and $p_1(x) = k_i(x)$. The advantage of this method is that it is more probable that two polynomials of degree less than $n/2$ will factor into the logarithm database than one polynomial of degree n. The expected number of polynomials $h(x)^*$ that need to be tested becomes

$$p(\lfloor n/2 \rfloor, b)^{-2} = \left(\frac{n}{2b}\right)^{(1+o(1))(n/b)} \tag{11.8}$$

which is an improvement of approximately $2^{n/b}$ from (11.5). This is not an asymptotic gain since n/b is o(1). The idea of representing a polynomial of degree $n-1$ as the quotient of 2 polynomials of degree $n/2$ can also be used to speed up the precomputation stage of the algorithm. Coppersmith was able to further improve the running time of the second part of the algorithm to

$$exp(\, cn^{1/3} (\log n)^{2/3}) .$$

The improvements of Coppersmith only apply to fields of the form GF(2^n). He advises, as does Blake *et al.* [BFMV] that GF(2^{127}) is unsuitable for use in algorithms that are described in §11.3. Using the HEP computer [BFMV], the logarithms of the factor base for GF(2^{127}) can be formed in 7 hours and extracting a logarithm then takes about a second (using the method of eq.(11.9) [BFMV]). If we use $n \approx 400$, then calculating the logarithms of the factor base in a reasonable amount of time becomes a task reserved for a supercomputer. However groups of opponents could collaborate and easily execute both stages of the index calculus algorithm. When $n \approx 700$ the first stage becomes infeasible for even a supercomputer, but a general purpose device using 10^4 semi-custom chips could still compute the first stage. Odlyzko [ODLZ] recommends that $n \geq 800$ for fast users and $n \geq$ 1800-2000 for special users.

Finally, we make a comment on the bound b, which greatly affects the optimality of performance. A tradeoff exists for the programmer. There are approximately $b^{-1}2^{b+1}$ irreducible polynomials of degree $\leq b$, each requiring n bits of storage. If b is too small, then the probability

that the random polynomial will factor into the factor base becomes negligible (finding the random polynomial dominates computation). If b is too large, then checking for smoothness dominates the computation. As b increases the time spent in the precomputation stage increases proportionally, but the chance of finding a polynomial that factors into the factor base also increases.

11.2 The Pohlig–Hellman Algorithm

Pohlig and Hellman [PaH] have devised an algorithm for computing the DLP in GF(q) that runs in time O($p1/2$) where p is the largest prime factor of $q-1$. This algorithm is most efficient when q is a Fermat prime, ie. $q = 2^m + 1$, and most inefficient when q is a Mersenne prime, ie. $q = 2^m - 1$. In 1984 there were less than 30 known Mersenne primes, which means that it will be difficult for a designer to select a field for which the Pohlig–Hellman algorithm performs at its worst.

First, consider the case when $q - 1$ is a power of 2. We wish to determine x such that $y = g^x \bmod q$, where g is a primtive root mod q. The Pohlig–Hellman method determines the logarithm one bit at a time, ie. if the logarithm $x = (b_{n-1}, b_{n-2}, \ldots, b_0)_2$, where b_{n-1} is the most significant bit, then b_0 is determined first, then b_1 and so on. Since q is prime, then $g^{q-1} \equiv 1 \bmod q$ and

$$g^{(q-1)/2} \equiv -1 \bmod q \quad .$$

The least significant bit of x is determined by raising y to the power $(q-1)/2 = 2^{n-1}$, ie.

$$y^{(q-1)/2} \equiv (g^x)^{(q-1)/2} \equiv (g^{(q-1)/2})^x \equiv (-1)^x \bmod q$$

and then observing that if $y^{(q-1)/2} \bmod q = 1$ then $b_0 = 0$, otherwise $b_0 = 1$. Subsequent bits are determined by

$$z = y \cdot g^{-b_0} \cdot g^{-2b_0} \cdot g^{-4b_2} \cdots g^{-2^i b_{i-1}} \equiv g^{x_i}$$

where

$$x_i = \sum_{j=i}^{n-1} b_j 2^j \ .$$

If we let $m = (q-1)/2^{i+1}$, then raising z to the *mth* power gives

$$z^m \equiv g^{(x_i m)} = g^{[(q-1)/2]x_i/2^i} \equiv (-1)^{x_i/2^i} \equiv (-1)^{b_i} \bmod q \ .$$

So $z^m \equiv 1 \bmod q$ if and only if $b_i = 0$, and $z^m \equiv -1 \bmod q$ if and only if $b_i = 1$.

Example 11.1. Determine $x = \log_g y \bmod q$ where $g = 14$, $y = 5$, $q = 17 = 2^4 + 1$;

b_0 : $5^{(16-1)/2} \equiv -1 \mod 17$, so $b_0 = 1$;

b_1 : $(5 \cdot 14^{-1})^{(16-1)/4} \equiv (5 \cdot 14^{-1})^4 \equiv 1 \mod 17$, so $b_1 = 0$;

b_2 : $(5 \cdot 14^{-1} \cdot 1)^2 \equiv -1 \mod 17$, so $b_2 = 1$;

b_3 : $(5 \cdot 14^{-1} \cdot 1 \cdot 14^{-4})^1 \equiv -1 \mod 17$, so $b_3 = 1$;

therefore the logarithm is $1101_2 = 13$. •

This technique can be generalised to arbitrary primes q where

$$q - 1 \equiv \prod_{i=1}^{k} p_j^{e_j}$$

Since each of the p_i are distinct we can use the Chinese Remainder Theorem to uniquely represent x as a system of linear congruences

$$x \equiv [\, r_1 \bmod p_1^{e_1} , r_2 \bmod p_2^{e_2} , \ldots , r_k \bmod p_k^{e_k} \,]$$

Each of the r_k is calculated using the following argument. Consider expanding x as

$$x \bmod p_i^{e_i} = r_i = \sum_{j=0}^{e_i-1} b_j p_i^j$$

where $0 \le b_j \le p_i - 1$. The least significant digit in base p_i is determined by

$$y^{(q-1)/p_i} \equiv g^{(q-1)x/p_i} \equiv \gamma_i^x \equiv (\gamma_i)^{b_0} \bmod q \ ,$$

where

$$\gamma_i = g^{(q-1)/p_i} \tag{11.9}$$

is a primitive $p_i th$ root of unity. The value of

$$y^{(q-1)/p_i}$$

will be a member of the sequence

$$H_i = \left\{ h_i^0 = 1,\ h_i^1 , h_i^2 , \ldots , h_i^{p_i-1} \right\},$$

where

$$h_i = g^{(p-1)/p_i} \ . \tag{11.10}$$

and by examining the sequence H_0 the value of b_0 will be found, ie.

$$h_i^{b_i} = \gamma_i$$

for some index in the sequence. Odlyzko suggests using Shanks' "baby steps, giant steps" method to determine the index of the H sequence that corresponds to b_i. The value of b_1 is determined by first forming z

$$z = y \cdot g^{-b_0} \equiv g^{x_1} \bmod q \ ,$$

where

$$x_1 = \sum_{j=1}^{e_i-1} b_j p_i^j ,$$

then raising z to the $(q-1)/p_i 2$ power. This yields

$$z^{(q-1)/p_i^2} \equiv g^{(q-1)x_i/p_i^2} \equiv \gamma_i^{x_1/p_i} \equiv \gamma_i^{b_1} \bmod q$$

and searching through the sequence H_1 reveals the value of b_1. The method is using the same technique as in example 11.1 except we are working in base p_i instead of base 2. Subsequent bits of the logarithm (mod p_i) are are found are by forming z

$$z = yg^{-b_0} \cdot g^{-b_1 p_i} \cdot g^{-b_2 p_i^2} \cdot \ldots \cdot g^{-b_j p_i^j} \equiv g^{x_i}$$

and

$$z^{(q-1)/p_i^j} \bmod q$$

gives b_j. After repeating this e_i times we have the representation of r_i, ie.

$$r_i = \sum_{i=0}^{e_i-1} b_j p_i^j$$

This procedure is repeated k times, once for each r_i .At this point, the logarithm x can be determined.

example 11.2. Let $g = 24$, $y = 29$, $q = 31$, $q-1 = 2 \cdot 3 \cdot 5$, where $p_1 = 2$, $p_2 = 3$, $p_3 = 5$.

(i) determine r_1, ie. x mod 2; then

$$h_1 = g^{(q-1)/p_1} = 24^{15} = -1 \bmod 31$$

and the first two values of the h_1 are $h_1 0 = 1$, $h_1 1 = -1$ (mod 31). Then

$$z = 29^{(31-1)/2} = 29^{15} = -1 \mod 31 \Rightarrow r_1 = 1;$$

(ii) determine r_2, ie, $x \bmod 3$; then

$$h_2 = 24^{(31-1)/3} = 24^{10} = 25 \mod 31$$

and expanding h_2 we find $h_2^0 = 1$, $h_2^1 = 25$, $h_2^2 = 5 \pmod{31}$. Then

$$z = 29^{(31-1)/3} = 29^{10} = 1 \mod 31 \Rightarrow r_2 = 0;$$

(iii) determine r_3, ie. $x \bmod 5$; then

$$h_3 = 24^{(31-1)/5} = 24^{6} = 4 \mod 31$$

and expanding h_3 we have $h_3^0 = 1$, $h_3^2 = 4$, $h_3^3 = 16$, $h_3^4 = 8 \pmod{31}$. Then

$$z = 29^{(31-1)5} = 29^{6} = 2 \mod 31 \Rightarrow r_3 = 3;$$

and lastly

$$x \equiv [\, 1 \bmod 2, 0 \bmod 3, 3 \bmod 5 \,] \equiv 3 \mod 31 \bullet$$

11.3 Cryptosystems based on the DLP

As we have no polynomial time algorithm for solving the DLP, we see that from the discussion above that exponentiation can be used as a one way function. Diffie and Hellman [DIF] gave a key distribution scheme based on exponentiation in finite fields, which according to the literature was the first public key system. A finite field GF(q) is selected and a primitive element $g \in$ GF(q) is made public. Imagine that Alfred wishes to exchange a key with Barbara. They choose random integers a and b, $2 \le a, b \le q - 2$ respectively. Alfred then transmits ga to Barbara, who transmits gb to Alfred. Both parties can obtain the key gab by raising the received message to the appropriate power. The exchanged key can be used as the private key for a conventional cipher

such as DES. This schene has been implemented by the Mitre Corporation [SCHA] using GF(2127), and Hewlett-Packard have built a special purpose chip for the same field [YP], though, as we have seen above, they offer little security. McCurley [McC] has recently devised the Composite Diffie and Hellman (CDH) exchange system where the modulus is composite. Further, McCurley shows that breaking the CDH system is at least as hard as factoring the modulus n, (this result is discussed in §16.3).

The DLP can also be used to transmit information in a public environment. We will consider two schemes here, one based on an idea due to Shamir [KON] and the other due to Elgamal [ELGA]. Note that each of the systems presented (including the key distribution scheme) could be compromised efficiently if there was an efficient algorithm for the DLP. As with the Diffie and Hellman scheme, Shamir's scheme assumes that some field GF(q) has been chosen and made public. Each message m is a nonzero element of the field. If Alfred wishes to communicate to Barbara, then the following steps are taken

- Alfred selects a random integer x, where $1 \le x \le q-1$, $(x, q-1) = 1$;
- Alfred transmits $c_1 = m^x \bmod q$;
- Barbara selects a random integer y, where $1 \le y \le q-1$, $(y, q-1) = 1$;
- Barbara transmits $c_2 = c_1^y = m^{yx} \bmod p$;
- Alfred forms $k = c_2^{x'} \bmod p$ where $xx' \equiv 1 \bmod q-1$.

Since

$$k = c_2^{x'} = m^{xyx'} = m^y \bmod q\,,$$

Barbara need only compute the y' where $yy' \equiv 1 \bmod q-1$ to recover[2] m.

[2]Notice that this system differs slightly form the defintion given by Diffie and Hellman [DIF] when they devised public key cryptography, essentially in the protocol used. The users must communicate several times, each communication called a pass. The Shamir is a k-pass system where $k = 4$. The Elgamal system is a one pass public

In the ElGamal system, a field GF(q) and a primitive root h are selected. Each user has h^a as his public key, where the secret key is the integer a. Imagine that Barabara wishes to talk to Alfred, who has h^a in his public file. If Barbara wishes to send $m \in$ GF(q), then she randomly selects x, $1 \leq x \leq q-2$, and sends the pair (h^x, mh^{ax}). Alfred can compute hax and recover the message by a division. The system has a data expansion of 2:1 as two transmissions of $log\ q$ bits are sent for each $log\ q$–bit message. Both the ElGamal and the Shamir systems encrypt probabilistically, as the encryption algorithms involve the use of random numbers. McCurley [McC] has also generalised the ElGamal system to the case where n is the product of two special primes.

How do these systems compare to RSA? Let k be the number of bits in either the field characteristic for the DLP or in the modulus n for the RSA system. The best algorithms for solving either of these problems is

$$exp\ (1 + o(1)(k\ log\ k)^{1/2})\ .$$

Thus, if the modulus and the field characteristic are about the same size, both systems afford the same level of security asymptotically. The RSA will have shorter messages while the DLP systems have smaller public files but also require a random number for each transmission. The fact that the running times are roughly equivalent comes from the fact that algorithms for both problems rely on the Morrison and Brillhart [MaB] heuristic and smooth numbers. Consider the following four problems and their best running times,

problem	*running time*
DLP in GF(q), p is prime	exp O($(k\ logk)^{1/2}$)
integer factoring	exp O($(k\ logk)^{1/2}$)
DLP in GF(2^n)	exp O($k^{1/3}log^{2/3}k$)

file cryptosystem, which are the type of systems modeled in §4. See [MRS] for an explanation of multiple pass systems.

factoring polynomials over GF(2) polynomial k

Table 11.3

where k is the problem dimension. This suggests that there may still be improvements to be made in computing the DLP in GF(2), ie. possibly a polynomial time algorithm.

Section 12 The Knapsack Problem

12.1 The Knapsack Problem

For cryptographic purposes, the knapsack problem is defined as follows: given a set of weights $B = \{b_1, b_2, \ldots, b_n\}$, where each $b_i \in Z^+$, and an integer S, does

$$\sum_{i=1}^{n} b_i \cdot x_i = S \tag{12.1}$$

have a solution for a binary vector $\mathbf{X} = x_1, x_2, \ldots x_n$? Equation (12.1) can also be written as $B \cdot \mathbf{X} = S$. This problem was shown to be **NP**–complete by Karp in 1972 [KARP]. Eq.(12.1) has 2^n possible solutions, and as n approaches 100 the task of exhaustively enumerating all the possible solutions becomes computationally infeasible. A solution to eq.(12.1) can be verified in linear time, though. For an arbitrary selection of b_i, there is no known algorithm for solving the knapsack problem short of examining the entire search space. The best algorithms for this problem require $O(2^{n/k})$ time and space where k varies between 2 and 4. Since $2^{n/k} = (2^n)^{1/k}$, if we can solve an instance of the knapsack problem involving n weights in time Φ using exhaustive search, then we can solve a knapsack problem in kn weights using an $O(2^{n/k})$ algorithm in time Φ. Shamir and Schroeppel [SCH] present an algorithm that requires $\Theta(2^{n/2})$ time and $\Theta(2^{n/4})$ space. The trick is to split B into two subsets $T = \{b_1, b_2, \ldots, b_{\lceil n/2 \rceil}\}$ and $U = \{b_{\lceil n/2 \rceil + 1}, b_{\lceil n/2 \rceil + 2}, \ldots, b_n\}$ and to work on each separately, then combine the results. This is the divide–and–conquer philosophy. We present an algorithm due to Horowitz and Sahni [HaS] that requires $O(2^{\lceil n/2 \rceil})$ space and $\Omega(2^{\lceil n/2 \rceil})$ time.

Given the set B and the target sum S, partition B into sets T and U as above. The results of the computations on the sets T and U will be stored in the sets V and W respectively. The strategy is to form all possible subset sums of T and U and then to combine these sums to determine if S is a subset sum of $B = T \cup U$. The set V will consist of ordered pairs (v_1, v_2) where v_1 is a sum

formed by some subset of the members of T, and v_2 is the set of weights used to form the sum v_1, ie.

$$v_1 = \sum_{j=1}^{t} x_i \cdot b_j \quad \text{then} \quad v_2 = \{ x_j \mid x_j = 1,\ j = 1 \,..\, t \}. \tag{12.2}$$

For example, if $T = \{1, 5, 7\}$ and $v_1 = 1 + 7 = 8$, then $v_2 = \{1, 7\}$. Similarly, the set W consists of ordered pairs (w_1, w_2). We define the "+" operator as follows: let A be a set of ordered pairs, as defined above, (a_i, A_i), where $i = 1 \,..\, j$, then $A + (b, B)$ is the set of ordered pairs $\{(a_i + b, A_i \cup B) \mid i = 1 \,..\, j\}$. We define the ordered union operator "$\cup$*" as follows: let $A = \{(a_i, A_i) \mid i = 1 \,..\, j\}$ and $B = \{(b_i, B_i) \mid i = 1 \,..\, k\}$ be two sets of ordered pairs; then $C = A \cup^* B$ is set formed by merging A and B and ordering the pairs by the first coordinate, ie.$C = \{(c_i, C_i) \mid i = 1 \,..\, j+k\}$ where $c_i \le c_{i+1}$ for $1 \le i \le j + k - 1$. The Horowitz–Sahni algorithm is given below.

1. $V \leftarrow \{(0,0)\}$;
2. **for** $i \leftarrow 1$ **to** $\lfloor n/2 \rfloor$ **do**
3. $\quad V \leftarrow V \cup^* \{ V + (b_i, b_i) \}$;
4. $W \leftarrow \{(0,0)\}$; $IC \leftarrow 1$;
5. **for** $i \leftarrow \lfloor n/2 \rfloor + 1$ **to** n **do**
6. $\quad W \leftarrow W \cup^* \{ W + (b_i, b_i) \}$;
7. pick off pairs $(v_i, v_j) \in V$, $(w_k, w_m) \in W$ such that $v_i + w_k = S$.
8. **return** (v_j, w_m). •

Example 12.1. Let $B = \{1, 4, 5, 9, 11\}$, $S = 10$, $T = \{1, 4\}$, $U = \{5, 9, 11\}$. Then using the H–S algorithm we obtain,

$V = \{ (0,0), (1, \{1\}), (4, \{2\}), (5, \{1, 4) \};$

W = { (0,0), (5, {5}), (9, {9}), (11, {11}), (14, {5, 9}), (16, {5, 11}),
(20, {9, 11}), (25, {5, 9, 11}) };

Picking off the pairs we see that $10 = 1 + 9$ and the solution is (1, 0, 0, 1, 0). •

The sets are ordered so that step 7 of the algorithm can be computed in the most efficient fashion. Let $V = \{ (v_i, V_i), 1 \le i \le |V| \}$, W = $\{ (w_j, W_j), 1 \le j \le |W| \}$ where V_i and W_j are sets. The following algorithm will determine if S is a subset sum of $B = T \cup U$;

1. $i \leftarrow 1$; $j \leftarrow |W|$;

2. **while** ($i \le |V|$ **and** $j \ge 1$) **do**

3. **if** $v_i + w_j < S$ **then**
$i \leftarrow i + 1$;
else

4. **if** $v_i + w_j > S$ **then**
$j \leftarrow j - 1$;

5. **else**
return all combinations V_i, W_j ;
$i \leftarrow i + 1$;

6. **end_while.** •

Since $|V|, |W| \le 2^{\lceil n/2 \rceil}$ then the running time of this algorithm, and the whole Horowitz–Sahni algorithm is $\Omega(2^{\lceil n/2 \rceil})$.

One might consider if a similar gain in efficiency can be achieved by splitting B into more than two subsets. If B is partitioned into k subsets $V_1, V_2, \ldots, V_k$ then all the possible subset sums of the V_i's can be computed in time $O(k2^{n/k})$. However, there does not appear to be anyway

of combining the intermediate results in time less than $O(2^{n/2})$, so nothing is gained from using more than two partitions of B.

Even though the general knapsack problem is hard to solve, there are some instances which can be solved in polynomial time. A sequence $B = \{b_1, b_2, \ldots, b_n\}$ is said to be superincreasing if

$$b_j > \sum_{i=1}^{j-1} b_i \qquad 2 \leq j \leq n \quad . \tag{12.3}$$

For example, $B = (3, 10, 15, 38)$ and $B = (2^0, 2^1, 2^2, 2^3, \ldots, 2^{n-1})$ are both superincreasing series. The associated knapsack problems defined by a superincreasing series are solved in linear time by the *SINC* algorithm;

1. algorithm *SINC (B, S)*
2. **for** $i \leftarrow n$ **downto** 1 **do**
3. **if** $b_i \leq S$ **then**
   $x_i \leftarrow 1$;
   $S \leftarrow S - b_i$;
4. **else** $x_i \leftarrow 0$
5. **return** $X \leftarrow x_1, x_2, \ldots, x_n$; •

12.2 Cryptosystems based on the Knapsack Problem.

One of the earliest PKCs was proposed by Merkle and Hellman [MER2] (M–H). This was the first use of the knapsack problem in cryptography. A set of weights B was chosen that satisfied eq.(12.3) , which was then transformed to $\mathbf{b}^* = b^*_1, b^*_2, \ldots, b^*_n$ by the rule

$$b^*_i \equiv Wb_i \bmod M, \quad 1 \leq i \leq n \tag{12.4}$$

where

$$M > \sum_{i=1}^{n} b_i \tag{12.5}$$

and gcd(W, M) = 1. After performing the modular transformation of eq.(12.4) the weights can be permuted. The vector $\mathbf{b}^*$ is the public key. An n–bit message is encrypted as $S = m \cdot \mathbf{b}^*$. To decrypt the designer calculates

$$W^{-1}S \equiv \sum_{i=1}^{n} m_i \cdot b_i^* W^{-1} \equiv \sum_{i=1}^{n} m_i \cdot b_i \mod M \ . \tag{12.6}$$

By eq.(12.5)

$$\sum_{i=1}^{n} m_i \cdot b_i \leq \sum_{i=1}^{n} b_i < M$$

and the *SINC* algorithm of §12.1 can be used to recover the message. The message is uniquely recovered since the mapping from $B \cdot \mathbf{X} \rightarrow \{0, \Sigma b_i\}$ is one–to–one[1]. The basic M–H system can be iterated by repeatedly applying transformations as in eq.(12.4). A H–iterated knapsack is formed as follows,

1. **for** $i \leftarrow 1$ **to** H **do**
2. select $M_j > \Sigma b_{j-1,i}$
3. select W_j such that $(W_j, M_j) = 1$;
4. $b_{j,i} \leftarrow W_j b_{j-1,i} \mod Mj$
5. **end_do.** •

[1]Generally the problem of determining if a knapsack system is one–to–one is Co–NP [SHA2].

The designers of the system were so confident of its security that they offered a reward for its cryptanalysis.

Consider the inverse functions of eq.(12.4),

$$W^{-1}b^*_i - k_iM = b_i \ , \tag{12.7}$$

for the integers k_i. Lemma 12.1 was independently proven by Desmedt, Vanderwalle, Govaerts [DVG] and Eier, Lagger [EL],

Lemma 12.1. There exists an $\varepsilon > 0$ such that if V/M' is a rational and $|V/M' - W^{-1}/M| \leq \varepsilon$, then the sequence $b' = Vb^*_i - k_iM'$ is superincreasing. •

Several cryptanalysts, including Shamir [SHA3], observed that if the k_i's of eq.(12.7) could be found, then a rational V/M' that satisfies Lemma 12.1 could be found also. If a PKC is constructed using the parameters suggested by M–H [MER2], then the k_i's satisfy

$$|b^*_i k_1 - b^*_1 k_i| \leq \frac{M}{2^{n-i}} \tag{12.8}$$

and Shamir showed that using eq.(12.8) when $i = 1, 2, 3, 4$, an integer programming problem could be formulated that would yield the values of the k_i's. Dividing eq.(12.7) by $b_i{}^*M$ gives

$$\frac{W^{-1}}{M} - \frac{k_i}{b^*_i} = \frac{b_i}{b^*_i M} \tag{12.9}$$

and each $k_i/b_i{}^*$ is a good approximation to W^{-1}/M, since the b_i are superincreasing and $b_i{}^* \approx M > \sum b_i$. By using the integers obtained from Shamir's integer programming problem, a small area on the unit interval can be searched which is expected to yield a rational V/M' that satisfies Lemma 12.1, and hence, can be used to decrypt ciphertexts of the form $S = m \cdot \mathbf{b}^*$. We note that the general integer programming problem is **NP**–complete, but when the number of variables is fixed ($n = 4$ in Shamir's case), the problem can be solved in polynomial time [LEN2]. Table 12.1 shows

that the sequence $\mathbf{b}^* = \{ 9, 5, 10, 7 \}$ can be transformed into a superincreasing sequence by 8 pairs of rationals V and M' (there are an infinite number of such rationals).

V	permutation	M'	SI series
125	9, 5, 10, 7	558	9, 67, 134, 317
28	9, 5, 10, 7	125	2, 15, 30 71
13	9, 5, 10, 7	58	1, 7, 14, 33
125	9, 7, 5, 10	279	9, 38, 67, 134
56	9, 7, 5, 10	125	4, 17, 30, 60
13	9, 7, 5, 10	155	5, 10, 38, 71
37	5, 10, 7, 9	61	2, 4, 15, 28
17	5, 10, 7, 9	28	1, 2, 15, 28

Table 12.1

Shamir used linear inequalities to find the k_i's of eq.(12.7), but simple diophantine approximation (SDA) can also be used to find integers that satisfy eq.(12.8). We will be interested in approximating the vector of real numbers $\alpha = (b^*_1/A, b^*_2/A, \ldots, b^*_n/A)$ by the vector of rationals $\beta = (p_1/p, p_2/p, \ldots, p_n/p)$. The quality of the approximation is given by

$$\{\{\beta\ \alpha\}\} = \max_{1 \le i \le n} \{ |pb^*_i - p_iA| \}. \tag{12.10}$$

The SDA is said to be of δ–quality if $\{\{\beta\ \alpha\}\} \le A^{1-\delta}$. We see that finding a δ–quality SDA can be formulated as an integer programming problem,

$$|pb^*_i - p_iA| < A^{1-\delta} \tag{12.11}$$

which is similar to eq.(12.8). Alternatively, the problem of finding δ–quality SDAs is equivalent to finding short vectors in a particular lattice. A lattice L is the set of points generated by the linear combination of n independent vectors $v_1, v_2, \ldots, v_n$, ie.

$$L = \left\{ v : v = \sum_{i=1}^{n} z_i v_i \text{ , where } z_i \in Z,\ 1 \le i \le n \right\}$$

and the vectors $v_1, v_2, \ldots, v_n$ are the basis for the lattice L. Consider the lattice L generated by the row vectors $v_1, v_2, \ldots, v_{n+1}$ in the matrix

$$\begin{bmatrix} -A & 0 & 0 & \cdots & 0 & 0 \\ 0 & -A & 0 & \cdots & 0 & 0 \\ & & \ddots & & & \\ 0 & 0 & 0 & \cdots & -A & 0 \\ b_1^* & b_2^* & b_3^* & \cdots & b_n^* & \varepsilon \end{bmatrix}.$$

Let $v \in L$, then

$$v = \sum_{i=1}^{n+1} h_i v_i = (h_{n+1} b_1^* - h_1 A, \ldots, h_{n+1} b_n^* - h_n A, \ \varepsilon h_{n+1})$$

and if $(h_1/h_{n+1}, h_2/h_{n+1}, \ldots, h_n/h_{n+1})$ is a δ–quality SDA of $(b^*_1/A, b^*_2/A, \ldots, b^*_n/A)$ and $\varepsilon < A^{-\delta}$ then the length, defined by the Euclidean norm, $|v| \le A^{1-\delta}\sqrt{(n+1)}$, which is considered to be a short vector in the lattice. Conversely if $|v| \le A^{1-\delta}$ then $(h_1/h_{n+1}, h_2/h_{n+1}, \ldots, h_n/h_{n+1})$ is a δ–quality SDA. Thus by finding short vectors in a lattice we can find good SDA and approximate solutions to eq.(12.9). Using the L^3 basis reduction algorithm [LLL] short vectors can be found quickly, but they are not the shortest vectors in the lattice. The reader is referred to Brickell [BR2] to see how basis reduction and short vectors can be used to break the iterated M–H knapsack.

A more general attack, that assumes nothing about the structure of the private weights, is based on the density of the knapsack $\mathbf{b}^*$ defined as,

$$\text{den}(\mathbf{b}^*) = \frac{n}{log_2 \, max \, (b_1^*, b_2^*, \ldots, b_n^*)}$$

satisfying

$$| b^*_i p_1 - b^*_1 p_i | \leq b^*_1 / n \tag{12.12}$$

and $0 < p_1 < b^*_1$. If $\text{den}(\mathbf{b}^*) < (\log_2 n)^{-1}$ then there is expected to be many good SDAs (p_2/p_1, p_3/p_1, ... , p_n/p_1) to (b^*_2/b^*_1, b^*_3/b^*_1, ... , b^*_n/b^*_1). Both Brickell [BR1] and Lagarias, Odlyzko [LaG] have devised attacks on low density knapsacks. Brickell's method will be sucessful if (n–1) independent approximations to (12.12) can be found. The Lagarias–Odlyzko algorithm constructs a lattice from the rows of the following matrix;

$$\begin{bmatrix} 1 & 0 & . & . & . & 0 & b_1^* \\ 0 & 1 & . & . & . & 0 & b_2^* \\ & & . & & & & \\ & & & . & & & \\ & & & & . & & \\ 0 & 0 & . & . & . & 0 & b_n^* \\ 0 & 0 & . & . & . & 0 & S \end{bmatrix}.$$

If $S = \sum x_i b_i$ then $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ will be a vector in the lattice. Since $\mathbf{X}$ is binary, then $|\mathbf{X}| < \sqrt{n}$, and the algorithm will work if $\mathbf{X}$ is found in the reduced basis. Lagarias and Odlyzko have shown that this is the case for almost all knapsacks of density less than n^{-1}.

12.3 The Chor–Rivest Knapsack

This system was designed by Chor and Rivest [CaR], and specifically avoids a low density attack by relying on modular multiplication for security. At the moment, it is the only knapsack system that has not been cryptanalysed. Decryption is based on the a theorem due to Bose and Chowla [BaC], which states that a field can be constructed such that the subset sum of any two subsets of h elements from the field is distinct. First, the designer picks a field GF(p^h) where

logarithms can be efficiently computed (such as GF(2^n) or a field that is well suited to the Pohlig–Hellman algorithm) and an irreducible polynomial $f(x)$ of degree at most $h - 1$ over the field. Let $g(x)$ be a primitive polynomial of the field. Pick an integer t randomly, and compute the logarithms $a_i = \log_{g(x)}(t + i)$ for $i = 0, 1, \ldots, p-1$. Permute the a_i's, ie. define $b_i = a_{\pi(i)}$, for some permutation π, and add some noise c, ie. define $c_i = b_i + c$. The public key is the vector $\mathbf{C} = (c_1, c_2, \ldots, c_{p-1})$ and h, and the private key is the parameters used to construct **C**.

To encrypt a message m, that has at most h 1's (ie. the Hamming weight cannot exceed h), a user forms the subset sum $S = m \cdot \mathbf{C}$. To decrypt, the designer computes $S' = S - hd \pmod{p^h - 1}$, then $p(x) = g(x)^{S'} \bmod f(t)$. Compute $d(t) = t^h + p(t) - r(t)$, a polynomial of degree h, where $r(t) = t^h \bmod f(t)$. The polynomial $d(t)$ factors into linear terms over GF(p) since

$$d(t) = (t + i_1)(t + i_2) \ldots (t + i_h)$$

and the ones of the message are simply the roots of $d(t)$. The inverse of the permutation is applied and the correct message is recovered.

Chapter 4 The Theory and Practice of Secure Systems

Section 13 Random Algorithms

13.1 How accurate is Worst Case analysis ?

In §4 we discussed the worst case analysis of a problem and how this is determined. This is a strict analysis which has merits and faults. The complexity of a problem is determined by the worst case running time of the best algorithm for the problem. The complexity measures the necessary work to solve the problem. Even though the algorithm may be deterministic and sequential there are certain conditional statements (such as a while loop) whose execution time will depend upon the given problem instance being solved. Consider the problem of sorting n numbers, A[i], $i = 1 .. n$ in non-decreasing order. An inversion is said to exist on the pair (A[r], A[s]) if A[r] < A[s] and $s > r$. The list A[n] is sorted when all the inversions have been removed. The bubblesort algorithm simply iterates until this condition is true, but what can be said about the number of iterations necessary to remove all inversions from a list? The list [4, 2, 8, 14] has one inversion while [14, 4, 2, 8] has 4. Clearly the number of iterations to remove all inversions will depend upon the list with which we are presented, but can any statement be made about the efficiency of an algorithm with respect to all inputs?

A worst case analysis yields an upper bound on the efficiency of an algorithm by deriving an expression for the running time based on the assumption that the algorithm takes as long as possible to execute. Let p be a problem and D_p its domain. Then for each instance $I(p) \in D_p$, let $t(I(p))$ be the actual running time of the instance. The worst case complexity is defined as

$$\max t(I(p)) \qquad \forall I(p) \in D_p \text{ and size}(I(p)) = n.$$

In the sorting example, since there are a maximum of $n(n-1)/2$ inversions on a list of n numbers, the worst case complexity for the bubble sort algorithm is $O(n^2)$. What can be deduced from a worst case analysis of problem p is true for every problem instance of p. Probably the most

pertinent aspect of worst case analysis to cryptography is that not all instances of a hard problem are necessarily hard. There need only be a single hard instance for a problem to be considered difficult, in a worst case analysis.

Example 13.1. Lempel [LEMP] presented a symmetric cryptosystem for which the *GCP* is known to be **NP**–hard, but which can be broken most of the time. Let a message m_t be encrypted in the following way,

$$c_t = m_t \oplus k_t.$$

The vector k_t is the output of the key function $K_t = F(X, R_t)$ where X is the secret key possessed by both sender and receiver, and R_t is a random vector. Each transmission consists of the pair (c_t, R_t) and F must be a one way function if X is to remain secret. No trapdoor is required. Lempel considers a family of F functions based on the knapsack problem. Let $\mathbf{A} = (a_1, a_2, \ldots, a_n)$ be a vector of positive integers known to all, and let $G(X, R_t)$ be a function whose image is a binary n-vector. Then let

$$F = A \cdot G(X, R_t)$$

or equivalently,

$$F = \sum_{i=1}^{n} a_i \cdot g_i \qquad (13.1)$$

where the bits $g_1, g_2 \ldots, g_n$ are the output of G . Assume that the function G is also public, and further that an opponent has access to some triples of the form (m_i, c_i, R_i), which is equivalent to knowing (k_i, R_i) pairs. Define a history to be h pairs of key and random bit vectors (k_1, R_1), (k_2, R_2), $\ldots (k_h, R_h)$ where h is a polynomial in m, the size of the weights of **A** (ie. $m = \log (\sum a_i + 1)$. We consider a system to be broken if a key k^* can be found such that

$$k_i^* = [A \cdot G(X, R_i)] \quad i = 1, \ldots, h,$$

$$k^* = [\, A \cdot G(X, R_t)].$$

Finding such a k^* will be called the *cracking problem* , and is specified by the (h+3)–tuple (A, R_t, h, $G(X, R_1)$, $G(X, R_2)$, ... , $G(X, R_h)$). Lempel defined the function G to be $G = X \oplus R_t$. The following lemma shows that the cracking problem is **NP**–hard for the case where $h = 1$.

Lemma 13.1. When $G = X \oplus R_t$ and $h = 1$, the cracking problem is **NP**–hard.

proof. Assume that algorithm AL solves the cracking problem when $h = 1$, ie. it finds a consistent k^*. We show how to use AL to solve any instance of the knapsack problem. Let (**A**, b) be an instance of the knapsack problem. Define an instance of the cracking problem to be **A**, $R = (1, 0, 0, \ldots , 0)$, and a history of one pair (B, (0, 0, ..., 0)) , where B is the m-bit representation of b. Then use AL to solve the cracking problem,

$$B = A\,(\,X \oplus (0, 0, \ldots , 0)\,),$$
$$k^*_1 = A\,(X \oplus (1, 0, 0, \ldots , 0)\,).$$

Now if a_1 is included in the solution of $\sum a_i x_i = B$, then $x_1 = 1$ and a_1 is not included in the solution of $\sum a_i x_i = k^*_1$, and vice versa. Then $x_1 = 0$ if and only if $k^*_1 > b$. The algorithm AL returns the value of k^*_1. Now let $\mathbf{A_1} = (a_2, a_3, ..., a_n)$ and $b_1 = b - x_1 a_1$ and repeat the above procedure (using $\mathbf{A_1}$ and b_1 and solve for x_2 and so on) $n - 1$ times, until the entire X vector is recovered. •

Despite this result, in most cases the system is easily broken when $h = n$. Given that $k_t = \mathbf{A} \cdot (\, X \oplus R_t\,)$ can be written as

$$K_t = \sum_{i=1}^{n} a_i \cdot (\, x_i \oplus r_{ti}\,) \qquad (13.2)$$

and since $u \oplus v = u + v - 2uv$ for $u, v \in \{0, 1\}$ then (13.2) becomes

$$K_t = \sum_{i=1}^{n} a_i \cdot (x_i + r_{ti} - 2x_i r_{ti}) \tag{13.3}$$

or

$$K_t - \sum_{i=1}^{n} a_i \cdot r_{ti} = \sum_{i=1}^{n} x_i \cdot (a_i - 2a_i r_{ti}). \tag{13.4}$$

Equation (13.4) is linear in the secret key X, and if the R_t are randomly generated then the probability that the system is linearly independent is at least 1/3, when N is the number of equations in the system and $N = n$. If $N = n + 1$ then the probability of solving the system doubles, and as $N - n$ increases, this probability approaches unity. •

In deciding the difficulty of a problem, complexity theory assumes that the person attempting to solve the problem has very little information, beyond the problem description, at their disposal. It is very difficult for the system designer to limit the information the cryptanalyst possesses or can deduce. The use of reductions seems a natural recourse for the designer. If a system's security can be shown to be equivalent to problem p, then no matter what information the cryptanalyst possesses or can deduces, problem p must be solved in order to break the system.

The worst case analysis is theoretical, easily calculated for deterministic algorithms and pessimistic. In 1947 George Dantzig [HaL] devised the simplex algorithm for linear programming problems. The simplex algorithm has an empirical complexity of $O(mn^3)$* but in 1972 Glee *et al.* [GlaM] proved that the algorithm had an exponential worst case complexity and hence is an inefficient algorithm. A worst case analysis binds the **NP**-complete problems together (eg. if there was an algorithm *AL* that could solve every instance of the Hamiltonian circuit problem, then *AL* could be used to solve every instance of the Travelling Salesman problem). A strict interpretation of the result that a problem is **NP**-complete is to abandon any hope of an efficient solution, since the problem is more than likely intractable. But can a solution be found some or perhaps most of

* Here m is the number of constraints and n is the number of variables.

the time? Another approach would be that, even though an optimal solution to problem p may be difficult to find, it may be possible to find a partial or approximate solution in a reasonable amount of time. An optimisation problem is one where each feasible or candidate solution has a positive rational number associated with it. An approximation algorithm quickly finds a feasible sub–optimal solution to an optimisation problem. Let AL be an approximation algorithm to $\Pi \in$ **NP**–complete. Let $AL(\Pi)$ denote the value of the approximate solution determined by AL and $OPT(\Pi)$ the optimal solution to Π. Let $R_{AL(\Pi)}$ be defined as

$$\frac{AL(\Pi)}{OPT(\Pi)} \qquad \left(\frac{OPT(\Pi)}{AL(\Pi)}\right)$$

for a minimisation (maximisation) problem. Sahni [SAHN] has shown that for any $k \geq 1$, there is a polynomial time algorithm $AL(k, knap)$ for the knapsack optimisation problem[1] and $R_{AL(k, knap)} \leq 1 + k^{-1}$. There are polynomial approximation algorithms to the "bin packing" problem that will always find a feasible solution that is no more than 22% worse than the optimal solution [JOHN]. Even though the *MAXIMUM INDEPENDENT SET* problem is **NP**–complete [GaJ], there is an algorithm for this problem that solves most instances in time $O(n^{\log n})$ [WILF]. On the other hand, the Travelling Saleman problem cannot be solved by a polynomial approximation algorithm unless **P = NP** [GaJ].

[1]This problem is defined as follows: given an integer vector $B = (b_1, b_2, \ldots, b_n)$ and an integer S, find a binary vector X that maximises XB such that $XB \leq S$.

13.2 Average Case analysis and probabilitic computations

A probabilistic or average case analysis begins by assuming that the instances for a problem are drawn according to some probability distribution. The average case complexity of an algorithm is defined as

$$\sum Pr(I(p))\cdot t(p)$$

where $Pr(I(p))$ is the probability that instance $I(p)$ occurs. This approach has been successful for yielding close to optimal solutions for several combinatorial optimisation problems [KATA]. The validity of an average case analysis is contingent on the accuracy of the probability distribution on the instance domain, which is beyond the control of the programmer. In the extreme case, the instances for which the algorithm performs most poorly will occur most frequently, and the analysis will degenerate to a worst case scenario anyway .

Keep in mind that we are searching for efficient algorithms. On the one hand, algorithms can be designed to take advantage of an expected set of inputs, while on the other hand, we can employ probabilistic arguments *in the actual computation* rather than on the input to the computation. Techniques that call upon upon random experiments to obtain qualitative information about a large systems (eg. such as the expected number of disk requests in a distributed environment) or to estimate complex quantities (eg. the area under a curve) are called Monte Carlo methods. This technique is quite old, but it did not gain acceptance as an established computing practice till the late 60's, and later still as a useful method of solving deterministic combinatorial problems. Rabin [RATM] argues that the Monte Carlo principle and injecting randomness into algorithms are distinct methods, used for different purposes to produce different results. After presenting a more precise definition of a probabilistic computation we argue that this distinction is not so clean, but there are complexity classes that capture the distinction tersely.

A probabilistic algorithm is computed by a probabilistic Turing machine (PTM) that bears all the traits of a deterministic TM (DTM) with the additional feature of having access to a random number generator. At points during the exexcution, the generator is consulted to determine which computational path will be selected next, ie. the next state is randomly selected. Contrast this with deterministic computation where the next state of the computation is uniquely determined from the current configuration, and non–determinism, where all of the successor states are computed simultaneously[2]. Probabilistic algorithms identify themselves by having at least one intruction that requires the selection of a random number. For example, Adleman's algorithm for the DLP uses random field elements to try and find a smooth number. A polynomial PTM is used to compute the index calculus algorithm, to construct the RSA system, and perform the encryption in the Golwasser–Micali quadratic residue system.

Let AL be a probabilistic algorithm for problem p. Let $I(p)$ be an instance of p and size($I(p)$) = n. At certain points during the execution of AL a random number r is required, where $1 < r < n$. Let r_i be the *ith* random number chosen and let $R = (r_1, r_2, \ldots, r_n)$ denote the sequence of random numbers used in the execution of AL. Assume that all R sequences are equally likely. Algorithm AL is said to run in expected time $f(n)$ if for every instance $I(p)$, size($I(p)$) = n, AL solves $I(p)$ in expected time less $\leq f(n)$. Expected time is defined as the average running time of AL on $I(p)$ for all possible choices of R.

An intriguing issue is whether probabilistic algorithms are more powerful than deterministic ones. Can anything be gained from "a throw of a dice"? Ultimately, the computing power of a PTM and DTM are equivalent [SANT], but by the same token, DTMs are equally as powerful as NDTMs as well. The real issue is what can be computed in polynomial time, and with this restriction, the NDTM is seemingly the most powerful machine. Are there problems that can be

[2]Strictly speaking there is only deterministic and non-deterministic (ND) computation, and a PTM is a type of NDTM, though the type of NDTM that is used to characterise the NP class is more powerful than a PTM when restricted to polynomial running time. For convenience, we will consider the DTM, PTM and NDTM as three separate computing models.

solved in polynomial expected time but not in polynomial deterministic time ? It would appear that when resticted to a polynomial running time, a PTM is more powerful than a DTM.

Example 13.2. The quadratic residue system of Goldwasser–Micali (see §10.2) cannot be constructed by a polynomial time DTM, but can be constructed by a PTM in polynomial expected time. There is no known polynomial time deterministic algorithm for finding an integer a such that $(a \mid n) = 1$ and a is a nonresidue mod n. The following probabilistic algorithm is used to generate an instance of the Goldwasser–Micali system where n is the product of two k–bit primes and y is a $2k$–bit nonresidue mod n,

step 1. flip $4k$ fair coins ;

step 2. (i) check whether the first $2k$ outcomes constitute two k–bit primes p and q. If so let $n = pq$; otherwise, go to step 1; (ii) check whether the last $2k$ outcomes constitute a nonresidue y mod n and $(a \mid n) = 1$. If so, the public key is (n, y) otherwise, goto 1.

Step 2 is the verification stage and its success depends upon the likelihood of randomly selecting primes and nonresidues mod n. In §10 we saw that 1/4 of the elements of Z^*_n will be suitable nonresidues (Lemma 10.8). A candidate nonresidue can also be verified quickly (Lemma 10.5). The Prime Number Theorem indicates the density of primes in a given range $[0 .. x]$. This theorem states that if $\pi(x)$ is the number of primes less than x, then

$$\lim_{x \to \infty} \frac{\pi(x) \ln x}{x} = 1 .$$

The Prime Number Theorem implies that if a is chosen randomly from Z_n then the probability that a is prime is

$$\frac{1}{\ln n} + O\left(\frac{1}{\ln n}\right).$$

Verifying primality can be done in subexponential time, namely $O(\log n^{c \log\log\log n})$ for some positive constant c [DIX]. The Lenstra–Cohen implemantation [CaL] of the Adleman–Pomerance–Rumely algorithm [APR] routinely provides proofs of primality for 200 digits in 10 minutes. •

Recall the game used to explain how the worst case of an algorithm is defined. It is much more difficult for an opponent to make a probabilitic algorithm run inefficiently, as the opponent cannot predict the behaviour of an algorithm which executes randomly. Selecting an instance of the problem that will trouble the algorithm is more difficult than for the deterministic case.

Probabilitic algorithms may provide one of three answers to a problem – the correct one, a wrong one and no answer at all. A probabilistic algorithm is said to lie if the conclusions of the computation are false. There are two types of probabilitsic algorithms,

- those that never lie but may fail to produce a meaningful output: Pr(*no answer*) $< \varepsilon$,
- those that always answer but may lie: Pr(*false conclusion*) $< \varepsilon$,

where $\varepsilon < 1$. The first type of algorithms are known as Las Vegas algorithms, and the second type as Monte Carlo algorithms. The index calculus algorithm is of the Las Vegas variety, as it provides an answer if a field element can be factored into irreducible polynomials, but no answer is given if this factorisation is not found. One of the first probabilistic algorithms, due to Berlekamp [BERL], was for factoring polynomials in finite fields, and ran with a probability of $< 1/2$ of not finding a factorisation. This algorithm runs in (expected) time that is a function of the degree of the polynomial and the logarithm of the field characteristic[3], and (when it finds a factorisation) is faster than any known deterministic algorithm for the same problem. As an example of a Monte Carlo algorithm, consider the problem of prime recognition. Solovay and Strassen [SaS] and Rabin [RAPT] have both devised probabilistic algorithms for this problem, that run in polynomial expected time, which indicate that a number is composite with 100% certainty (ie. the algorithm

[3]Let q be the field characteristic. Let r be the number of factors in the *nth* degree polynomial $p(x)$ to be factored. The Berlekamp algorithm runs in time $O(n^2 (\log q)^3 \log r)$ [KNU2, p429].

yields a proof) and assert that a number is prime with a probability of error at most ε. For the Solovay-Strassen algorithm $\varepsilon = 1/2$, and $\varepsilon = 1/4$ for Rabin's algorithm. These algorithms may be iterated to obtain a chance of error that is less likely that the probability of a hardware fault. Rabin quickly identified $2^{400}-593$ as a (probable) prime using an error of 2^{-60}. This number is only a probable prime as the probabilistic algorithm yields proofs of compositeness but only assertions of primality. These algorithms do not produce evidence of primality but conclude this state based on the lack of evidence found for compositeness. Rivest *et al.* [RSA] advised to use a probabilistic algorithm to select the prime factors of the modulus for the RSA cryptosystem. Subsequently Cohen and Lenstra [CaL] have devised a powerful deterministic tester that furnishes a proof of primality. We can classify these ideas using formal languages for decision problems.

The class **ZPP** is the class of languages accepted by a polynomial time PTM that never lies. If $L \in$ **ZPP** then there is a NDTM M such that $\exists \varepsilon \in (0, 1)$, and $\forall x$

$$x \in L \Rightarrow \textit{no path of } M \textit{ rejects } x,$$
$$x \notin L \Rightarrow \textit{no path accepts } x, \textit{ and}$$
$$pr(M(x) = ?) \leq \varepsilon,$$

where "?" is the no answer state. The last condition indicates that there is a fraction of computations that will not be computed in a polynomial number of steps in the average case. The class **ZPP** is said to operate with zero error probability since a false computation is never reached. Since every DTM computes with zero error probability then $\mathbf{P} \subseteq \mathbf{ZPP}$. Let the Δ class be defined by altering the qualification of **ZPP** to "$\forall x \exists \varepsilon$". Then

Lemma 13.1. $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \Delta$.

proof. $\forall \varepsilon \forall x \Rightarrow \exists \varepsilon \forall x \Rightarrow \forall x \exists \varepsilon$ •

Next we define the class **BPP** (bounded probabilistic error), which is the class of languages for which there is a polynomial time NDTM M, such that $\exists \varepsilon > 0$, $\forall x$,

$$x \in L \Rightarrow pr(M(x) = yes) \geq 1/2 + \varepsilon,$$
$$x \notin L \Rightarrow pr(M(x) = yes) \geq 1/2 + \varepsilon,$$

and $0 < \varepsilon < 1/2$. Finally we define the class **RP** which is the class of languages for which there is a polynomial time PTM M such that $\exists\varepsilon,\forall x$

$$x \in L \Rightarrow pr(M(x) = yes) \geq \varepsilon,$$
$$x \notin L \Rightarrow \textit{no path of } M \textit{ accepts } x.$$

If we replace "$\exists\varepsilon\forall x$" by "$\forall x\, \exists\varepsilon$" then we have a characterisation of the class **NP**. The qualification indicates that for each instance of a language some path of the computation will accept the instance. The set

$$COMPOSITES = \{\, p \mid p \textit{ is not a prime} \,\}$$

is a member of **RP**, and is recognised by the primality algorithms mentioned before. The class **PP** consists of those algorithms computed by PTM in polynomial expected time. Diagram 13.1 shows the relations between the classes introduced in this section. An arrow from class A to class B indicates that $A \subseteq B$.

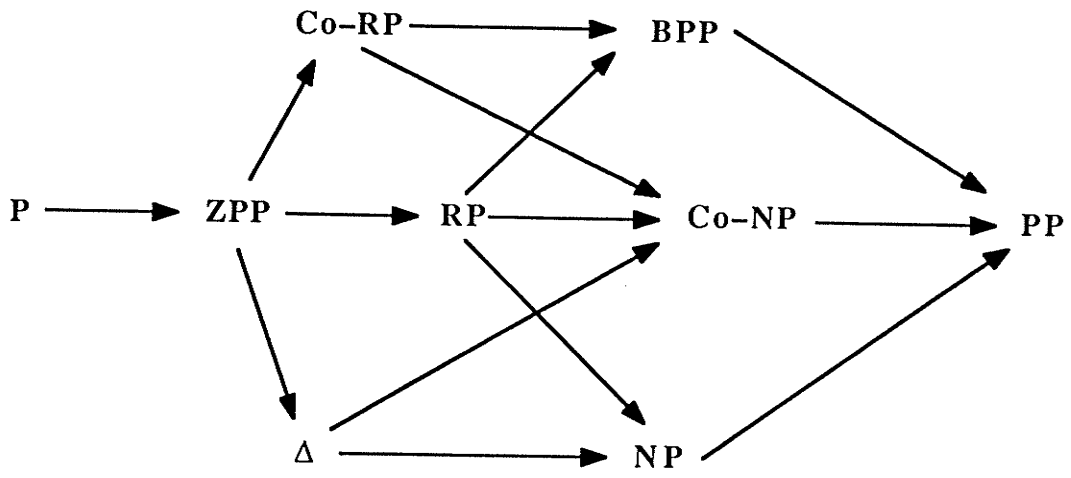


Diagram 13.1

We note that **ZPP** = **RP** ∩ **Co–RP** and Δ = **Co–NP** ∩ **NP**. The *COMPOSITES* problem is not known to lie in **P**, and along with the problem of recognising non-singular matrices whose entries are polynomial in many variables [SCHW], these algorithms are witnesses that probabilistic computation is more powerful than deterministic computation when restricted to polynomial running times. We have evidence for suspecting that the inclusion of **P** in **RP** is proper. Note that all of the probabilistic algorithms are using psuedorandom generators and hence the computation is not truly random. How pseudorandomness alters the behaviour of a polynomial time PTM is not precisely known, but the empirical achievements indicate that the algorithms adapt well to the pseudorandom decisions. Cook observes that if probabilistic algorithms always perform well using pseudorandom numbers then nothing is gained from true randomness and **RP** = **P**. It has also been suggested that problems which can be solved by a polynomial time PTM conform to our notion of tractable problems, as **P** has done, if we accept a small error. The Δ class is errorless but on occasion may return no answer. Adleman and Manders [AaM] explore the possibilty that this class conforms to the notion of an easily computed task.

13.3 Random Reductions

A cryptosystem that can be broken most of the time or even for a fraction of the time is considered an inadequate means of providing security. Such systems may be deemed intractable to compromise in the worst case, but weak in the average case. The reductions that have been presented thus far have been based on the worst case analysis, ie. the reduction $A \propto_T B$ indicates that *any* algorithm *AL* that solves problem *B* will also solve *all* instances of problem *A*. Such a reduction is deterministic and errorless. The designer will also be interested to know if algorithm *BL* that solves problem *C* may be used to decipher (say) half of his ciphertexts. We say that problem *A* probabilistically reduces to problem *B*, denoted $A \propto_E B$, if given a polynomial time algorithm *BL* for *B* then $1/\varepsilon$ of the problem instances of *A* will be solvable in polynomial time.

Two problems A and B are randomly equivalent, denoted $A \approx_r B$, if $A \propto_E B$ and $B \propto_E A$. Most of the interesting probabilistic reductions for cryptography are between number theoretic problems. The success of these reductions relies on several facts from number theory, such as the Prime Number Theorem, $\phi(n) \geq n/\ln\ln n$ [HaW, Theorem 328] and that the number of primitive elements in a Z^*_p is $\phi(\phi(p))$ for a prime p [LIPS].

Example 13.3. We will show that FAC_n and KEY_d (defined in §9) are randomly equivalent in the RSA common modulus protocol (CMP). Recall that in the CMP there is one modulus n and each user is issued with a pair e_i and d_i such that $e_i d_i \equiv 1 \mod n$. Since $KEY_d \leq_p FAC_n$ by the construction of the RSA system, we need only show the reduction from factoring to determining the secret key d_i. Let n be an RSA modulus. If we can find a integer c such that

$$c^2 \equiv 1 \mod n \tag{13.5}$$

and c is nontrivial, then n can be factored. By Fermat's theorem, if $(a, n) = 1$ then

$$a^{\phi(n)} \equiv 1 \mod n\,. \tag{13.6}$$

Since we know e_i and d_i, then $(e_i d_i - 1)$ is a multiple of $\phi(n)$ and

$$a^{e_i d_i - 1} \equiv 1 \mod n \tag{13.7}$$

let $(e_i d_i - 1) = 2^k p$. If

$$a^{2^{k-1}p} \not\equiv -1 \mod n \tag{13.8}$$

then

$$c = a^{2^{k-1}p}$$

is a non-trivial square root of 1. Delaurentis [DeL] shows that this procedure succeeds with a probability of 1/2 for a randomly chosen integer a. Hence, two selections are expected before a non–trivial square root is found. Thus $KEY_d \approx_r FAC_n$ (CMP). •

One way to show that a cryptanalysis of the RSA system is equivalent to factoring would be to give a reduction to RSA^{-1} to one of the problems in Lemma 9.4. At present, such a reduction has neither been given nor denied to exist, and exactly how difficult RSA^{-1} is to compute remains an open question, ie. it is not known whether an efficient method for computing *eth* roots leads to an efficient method for factoring. Clearly none of the obvious (known) ways to solve RSA^{-1} result in a problem that is easier to solve than factoring the modulus. Some other random reductions are discussed in §16.

SECTION 14 Decision Problems and Reductions

14.1 NP–Complete Systems

In §4 the difficulty of the *GCP* was shown to be **NP**. If the *GCP* were as hard as possible to solve, then it would be equivalent in difficulty to the hardest problems in **NP**. We have already indicated that the **NP**–complete problems are the hardest decision problems in **NP**[1]. Can cryptography gain anything from the theory of **NP**–completeness ? Can cryptosystems be based on **NP**–complete problems? Can the *GCP* be **NP**–complete?

Each **NP**–complete problem is a decision problem, dealing with the existence of some property (such as a Hamiltonian circuit). Hence each **NP**–complete problem carries with it one bit of information that is infeasible to compute, namely the question "does instance *I(p)* of problem *p*, have property *X* ?" An encryption function based on a difficult decision problem can only ensure that one bit of information is secure per transmission, and a bit–by–bit encryption scheme is not expected to be very efficient. Consider the Goldwasser–Micali (see §10) system based on the difficulty of determining quadratic residuosity. Each non–zero element of Z_n will either be a quadratic residue or a quadratic nonresidue. Only one bit is encrypted with each random selection $x \in Z_n$. In order that the modulus is large enough so that factoring it is infeasible, n will need to be approximately 650 bits. That is, each message bit is expanded to over 600 ciphertext bits through modular exponentiation. So the system is of little practical value (it is slower than the RSA system). There is no theoretical consolation either, as the quadratic residuosity problem is not known to be **NP**–complete. Certainly its difficulty cannot exceed that of factoring.

The **NP**–complete problems are related via polynomial transformations. If solving the *GCP* of a cryptosystem is to be equivalent to solving an **NP**–complete problem then there must be a

[1]There are problems not known to be solvable by a NDTM in polynomial time, and hence can be considered harder than any problem in **NP**. Hartmanis [HART, p12–13] gives some examples. We may consider the condition that a problem be **NP**–complete as a minimum requirement for a problem to be intractable.

polynomial transformation from a known **NP**–complete problem to the *GCP* of a cryptosystem. However, the *GCP* is not usually a decision problem. One decision form of the *GCP* would be given E_{kI} and c, does there exist a message m such that $c = E_{kI}(m)$? Trivially, such an m must exist if the ciphertext is a valid encryption. This is nowhere more evident than in the Merkle–Hellman knapsack system, which many people thought had the best chance of being proven intractable, since the general knapsack problem is **NP**–complete. The cryptanalytic problem cannot be equivalent to the knapsack decision problem since the encryption algorithm ensures that this problem is always answered yes. As further evidence of the difficulty of applying of decision problems to cryptography, consider the systems based on the DLP (see §11). By definition every element of a field will have a discrete logarithm to the base g if g is a primitive root of the field. There is no doubt that the logarithm of transmissions in the ElGamal system will exist. Further it is known that the moduli of the RSA and quadratic residue systems are composite and have two large prime factors.

The only information that separates the opponent from the designer of a PKC is the knowledge of the trapdoor used to construct the system. Then, the only meaningful hard problem in such a cryptosystem is to recover a plaintext message from its corresponding ciphertext without the trapdoor information. If the message space is restricted to {0, 1} then the *GCP* becomes a decision problem, namely "does $m = 1$?". But once again we are dealing with inefficient bit–by–bit systems.

14.2 Partial Solutions To The GCP

Once a problem has been classified in a complexity class, by making assumptions about the relations amongst the complexity classes (such as $\mathbf{P} \neq \mathbf{NP}$) something can be said about the difficulty of problems (eg. under the assumption $\mathbf{P} \neq \mathbf{NP}$, the **NP**–complete problems are intractable). For most cryptosystems, the complexity of their respective *GCPs* is not known, ie. it is not known what complexity class they belong to. For example, it is not known if breaking the

RSA system is as hard as factoring a number which is the product of two primes. Moreover, the complexity of the general factoring problem is itself is unknown. Similar comments apply to the DLP, with respect to the Diffie and Hellman key exchange and ElGamal systems.

If anything is to be gained from the theory of **NP**–completeness, then there must be some interpretation of the *GCP* as a decision problem. Given a decision problem, it is usually a straightforward matter to convert the problem to a related search problem. The search form is at least as hard as the decision form. Most encryption functions act on n-bit messages, and the corresponding *GCPs* are search problems (ie. given C and E_{kI}, find the message m such that $E_{kI}(m) = C$). As has been argued, decision problems are not particularly useful in the design of cryptosystems, nor do they adequately model the difficulty of determining the trapdoor information of a system.

The *GCP* is a formulation of the problem of recovering the corresponding plaintext message, in its entirety, from an intercepted ciphertext message. Since the message is to be recovered entirely, the message space must be {0, 1} if the *GCP* is to be a decision problem. Alternatively, can we gain anything from determining one bit of information about the message ? For example, we might ask if the parity of the message is even, or if the message $m \leq k$ for some positive integer k . Both of these problems are decision problems. The first question concerning the parity of the message will be discussed in the §14.4. The second question, " *is* $m \leq k$? ", is an example of Miller's projection technique, which has been used to prove some results about *GCPs* and one way functions (see §16).

Inspired by the work of Brassard *et al.* [BRNP], Even and Yacobi [EaY] showed that the *GCP* is not expected to be **NP**–hard. Let the *GCP* be defined as in §1, and ask the question

QUESTION: is the message $m > t$?

for a given value t, which is part of the problem instance. Let this decision problem be called P_{GCP}, the projection of the *GCP*. A NDTM, when given t as input, can guess the message m and easily decide P_{GCP}. Similarly, the complement of the P_{GCP} (ie. is $m \leq t$?) can be decided by a NDTM in polynomial time. Hence $P_{GCP} \in \mathbf{NP} \cap \mathbf{Co\text{–}NP}$. Further, $GCP \propto_T P_{GCP}$, by the following binary search algorithm

1. $high \leftarrow 2|M|$
2. $low \leftarrow 0$
3. **while** $low < high$ **do**
4. $\quad mid \leftarrow \lfloor (high+low)/2 \rfloor$
5. $\quad$ **if** $\langle E(m), t \rangle \in P_{GCP}$ **then** $low \leftarrow mid + 1$
   else $high \leftarrow mid$
6. **end_do**
7. **return** low

where $|M|$ is the size of the message space. Clearly, $P_{GCP} \propto_T GCP$ and hence $P_{GCP} \equiv_T GCP$. If the *GCP* were **NP**–hard, then $A \propto_T GCP$ for some **NP**–complete problem A. Then, P_{GCP} would also be **NP**–hard, and hence **NP**–complete. However, since $P_{GCP} \in \mathbf{NP} \cap \mathbf{Co\text{–}NP}$ then from §4, this would imply **NP** = **Co–NP**, which most researchers feel is very unlikely. Hence it is concluded that the *GCP* is not expected to be **NP**–hard. In §16 we will see that a similar technique can be used to demonstrate that certain one way functions are not expected to be **NP**–hard.

14.3 Bit Security

Let us assume for the moment that the *GCP* of some n–bit block encryption system is **NP**–hard to solve. Thus, given a ciphertext $c = E_k(m)$, the problem of determining m is difficult. This

statement is not contradicted if every 5th bit of m, ie. m_5, m_{10}, m_{15} ..., is easily determined. The *GCP* asks how hard it is to recover the n–bit message entirely, and this problem will be hard if at least one bit of the message is hard to recover. A worst case analysis would support this claim. The real issue is how much information can be gained from expending a given amount of computational resources.

Let E be a permutation from a 100–bit message space to a 100–bit cipherspace. Suppose that 75 bits of the message can be computed in polynomial time from the ciphertext. If the messages are words from a natural language, then the redundancy of the language could probably be used to infer the remaining 25 bits of the message, even though these bits cannot be computed in polynomial time. The side information has allowed the cryptanalyst to obtain the message without solving any hard problem(s) at all. If the message was numerical, say a PIN of a bank account, then the opponent will still gain information, namely a few digits of the PIN number. It may be that valuable information can be gained if even half or a quarter of the message bits are easily computed from a ciphertext. In some cases, as we shall see, the ability to recover one bit is enough to recover the whole message.

The bit security of a system is concerned with what partial information about the messages can be computed from the ciphertexts. Intuitively, we would expect that the problem of recovering all the bits of a message is about as hard as the joint problems of recovering each bit individually. Contrary to popular sayings though, a worst case analysis maintains that a chain is as strong as its strongest link, and if one bit of the message is difficult to recover, then the whole message is considered difficult to recover. One way, and possibly the only way, to avoid the extraction of partial information is to resort to a bit–by–bit encryption scheme. With this message space, no partial information can be computed without necessarily computing the bit that was sent as well. Later in the section we will show that there are certain bits in the RSA system, for example the parity bit of the message, such that, if these bits could be efficiently computed, then the entire message could be recovered in polynomial time.

An analysis of hard bits produced by an encryption function has applications to psuedorandom number generators (PNG). There are various algorithms for generating psuedorandom sequences of numbers that pass a set of statistical tests [YAO] which attest to their randomness. But to be useful to cryptography such sequences must be secure as well. By secure, we mean that given a sequence $\mathbf{x} = x_1, x_2, x_3, \ldots, x_k$, it is computationally infeasible to predict x_{k+1} (or any terms beyond x_k). A subsequence is unpredictable if it cannot be inferred to the left or right in polynomial time, ie. an observer can do no better in polynomial time than guess the terms of the sequence. A cryptographically secure pseudorandom bit generator (CSPBG) produces unpredictable pseudorandom bits $b_1, b_2, b_3, \ldots$ from seeds $s_1, s_2, s_3, \ldots$; the seeds should be hard to recover from the output of the generator. Most CSPBGs are based on some difficult encryption transformation, since a one way function is needed to hide the seeds. Shamir [SHA] has used the RSA encryption function (modular exponentiation) to build a PNG that is secure, where the first k seeds are the first k primes, ie. 2, 3, 5, 7, 11 ... and so on. The PNG produces numbers p_i by the rule $p_i = s_i^e \bmod n$, where e and n are the normal RSA parameters (see §9). There is no apparent way to determine the s_i from the p_i (for this choice of s_i), though this is not the case if the first k seeds are the first k integers, ie. 1, 2, 3, 4, 5, and so on. It still may be possible to determine partial information about the seeds from the output of Shamir's generator. In §16 we will see that a powerful CSPBG can be constructed using the quadratic residue problem, and no partial information about the sequence can be obtained without solving an intractable problem. Blum and Micali [BM] have given a similar result for a CSPBG based on the DLP.

On the one hand, we would like to know if only 1 bit is to be extracted from $f(x)$, the output of a one way function, which bit of the output gives the least information about x (or gives the hardest problem to recover x). On the other hand, we would also like to know what bits of x can be easily computed if all bits of $f(x)$ are given. Consider the Merkle–Hellman system and let $c = x\mathbf{b}^*$ where $\mathbf{b}^* = b^*_1, b^*_2, \ldots, b^*_n$ are the public weights. Let $O_1(c, \mathbf{b}^*)$ be an oracle that returns the *jth* bit of x, ie. if the oracle returns 1 then b_j is included in the sum c. We see that by permuting

each weight to the *jth* position in $\mathbf{b}^*$, n calls to oracle $O_1(\cdot,\cdot)$ are needed to recover x, and hence the whole message.

Thus, for the M–H knapsack system, the problem of recovering any bit of the plaintext message is as hard as the problem of recovering all the bits of the message. The results for the RSA system are far less general. In particular, determining the message m given m^e mod n is equivalent to finding oracles for the following bits [GMT];

- computing the parity of m (extracting the parity bit);
- computing whether $m \leq n/2$ (extracting the location bit);
- when $m = b_n \ldots b_i \ldots b_1$, where $b_i = \ldots = b_1 = 1$ and $b_{i+1} = 0$, computing any one of the least i significant bits of the message.

The following notation will facilitate our discussion. Let $B(a)$ be a function that returns the binary representation of the integer a ; let $M(t)$ be the inverse function of B for a binary string t. If t and u are binary strings such that $|u| \leq |t|$ then let $t(-)u$ be the binary sequence v where v is the last $|u|$ bits of $B(M(t) - M(u))$. The concatenation operator is "*". Consider the following oracle for extracting the parity bit, defined for all messages m,

$$O_2(m^e \bmod n) = 0 \text{ if } m \text{ is even, and 1 otherwise.}$$

Then it is possible to decrypt m^e mod n by using n calls to $O_2(\cdot)$, by means of the following algorithm due to Goldwasser *et al.* [GMT],

1. input m^e mod n.
2. *compute* $H \in Z^*_n$ *such that* $2^s H \bmod n = 1$;

3. $r[1] \leftarrow m^e \bmod n$;
4. $ANS[1] \leftarrow O_2(r[1])$;
5. **for** $i \leftarrow 1$ **to** n **do**
6. **if** $ANS[i-1] = 0$ **then** $r[i] \leftarrow r[i-1]H \bmod n$
7. **else** $r[i] \leftarrow (N-r[i-1])H \bmod n$;
8. $ANS[i] \leftarrow O_2(r[i])$;
9. **end_for**
10. $t^{(n)} \leftarrow ANS[n]$;
11. **for** $i \leftarrow n$ **downto** 2 **do**
12. **if** $ANS[i-1] = 0$ **then** $t^{(i-1)} \leftarrow t^{(i)} * 0$
13. **else** $t^{(i-1)} \leftarrow B(n)\,(-)\,(t^{(i)} * 0)$;
14. **end_for**
15. $m \leftarrow M(t^{(1)})$;
16. **return** m ; •

Example 14.1. Recovering a message using the above algorithm where $n = 37 \cdot 31 = 1127$, $e = 7$ and $H = 690$. The input is $m^e \bmod n = 81$. Table 14.1 shows the behaviour of the algorithm using O_2.

i	$r[i]$	$ANS[i]$	$t^{(i)}$
1	81	1	00001101011
1	313	0	1000001000
3	334	0	100000100
4	1060	0	10000010
5	761	1	1000001
6	236	1	011101
7	34	1	01111
8	627	0	0110
9	211	1	011
10	79	0	00
11	601	0	0

Table 14. 1

The output of the algorithm is $M^{(t)} = m = 107$. •

Consider an oracle for the location bit

$$O_3(m^e \bmod n) = 1 \text{ if } m < n/2 \text{ and zero otherwise}$$

which can be used to simulate O_2 and vice versa. For any $x \in Z^*_n$, $2x \bmod n$ is even if and only if $x < n/2$. Thus for any $x \in Z^*_n$, $O_2((2x)^e \bmod n) = 0$ if and only if $O_3(x) = 0$. Similarly, if $H = 2^{-e} \bmod n$ then for any $x \in Z^*_n$, $O_2(x) = 0$ if and only if $O_3(Hx) = 0$. It is interesting to consider the

question, "is every bit of a message m as hard to compute as inverting the encryption function, i.e. $E(m)$?" Goldwasser *et al.* show that having an oracle for determining any of the $(i+1)th$ least significant bits is equivalent to being able to decrypt. The algorithm is similar to the one using $O_2(\cdot)$ and involves n calls to an oracle for the least significant bit.

Even though it may seem unlikely someone may gain access to any of these oracles, the response to a message can give an opponent information that he could not have otherwise determined. For example, from time to time the users would expect transmission errors. Garbage will be received and messages will have to be resent. Any user can check if a string y is an image under E_k by simply sending $E_k(y)$ to user i and waiting to see how i responds. If a retransmission is requested then y is not an image under the encryption function of user i. It is also reasonable to assume that a "yes" is encrypted differently to a "no", and suppose that the parity of the messages for these responses are different, or perhaps that one of the messages is less than $n/2$ and the other is greater than $n/2$. Then either O_2 or O_3 could be implemented in the PKC: all that need be done is for users to ask questions that require yes and no answers. For this to be fruitful at least n such questions must be posed. See [GMT] for further examples of the weaknesses in the public key model. They then propose a private key system where each user exchanges a key with each user he wishes to communicate to. The actual exchange is a probabilistic game where the chance of an impersonator winning is small.

Consider the special case of the RSA system where the public exponent is 2, ie. $e = 2$, and encryption is performed by squaring the message and reducing mod n. This variation of the RSA system is due to Rabin [RAB] and breaking this system is equivalent to factoring the modulus (see §16). Goldwasser *et al.* [GMT] show that oracles for the following bits lead to an efficient algorithm for the decryption of messages in the Rabin scheme,

- deciding whether parity(x) = parity(y) where $x, y \leq n/2$ and $x^2 \equiv y^2 \bmod n$;
- deciding parity(y) such that $y \leq n/2$, $(y \mid n) = 1$ and $y^2 \equiv z \bmod n$.

Recently, Vazirani and Vazirani [VaV] have shown that *c loglog n* bits may be extracted from the X^2 mod n CSPBG, which produces random numbers according to the rule $X_{i+1} = X^2_i$ mod n, ie. the knowledge of *c loglog n* bits of output does not allow an observer to predict the sequence in either direction in polynomial time. Similarly, Peralta [PER] has also shown that *c loglog n* bits can be securely extracted from the discrete logarithm PNG, defined by the rule

$$X_i = g^{X_{i-1}} \ mod\ n$$

where g is a primitve root mod n.

14.4 Reductions Amongst The Basic Problems

We have seen that the *GCP* is not **NP**–complete because it is not generally a decision problem. Further some *GCP*s cannot be **NP**–hard unless **NP = Co–NP**, which is very unlikely. Perhaps there is another intractable complexity class, other than those considered, that the basic problems belong to.

Recall that problems that lie in **NP – P** are the most likely candidates for intractable problems. One the one hand, there are the **NP**–complete problems which are the hardest in **NP** while on the other hand, the easiest problems reside in **P**. The problems that lie between these two poles are those of "intermediate" hardness. Until recently there were three problems which many researchers thought might lie in **NPI = NP – P – NP**–complete; *COMPOSITES, LINEAR PROGRAMMING*, and *GRAPH ISOMORPHISM*. Additionally, the first two problems also lie in **NP ∩ Co–NP**. The complement of *LINEAR PROGRAMMING* is its dual [HaL] and both problems are solved in the same manner. The certificate for *COMPOSITES* is two non–trivial factors, which is verified by a multiplication, and hence is **NP**. The complement of this problem is *PRIMES* which was shown to have a "succinct" certificate by Pratt [PRAT] in 1975. Since **P = Co–P** is a subset of **NP ∩ Co–NP** and no complete problems are expected to lie in **NP ∩ Co–NP** there is far less confidence in the belief that polynomial time algorithms will not be found for

problems that lie in this intersection. In fact, in 1979, *LINEAR PROGRAMMING* was proven to be in **P** by Khachian [KHAC], and there already exists a polynomial time algorithm for *PRIMES* assuming the ERH [MILL]. Some believe that if a problem lies in **NP** $\cap$ **Co–NP** then this suggests that the problem is solvable in polynomial time.

Consider the basic problems of §2 again. The problem of *COMPOSITES* is Turing reducible to that of factoring, but this does not imply that factoring is **NP**–hard since *COMPOSITES* is not **NP**–complete. Similarly we know of no reductions from the DLP, computing square roots or determinig quadratic residuosity to any **NP**–complete problems. Albeit, both the knapsack decision and search problems are **NP**–hard, and solving either in polynomial time would imply that **P** = **NP**. This problem seems to be a strong choice as the basis for a cryptosystem.

The basic number thoeretic problems do not seem to gain any evidence for their intractability from reductions or polynomial time transformations. The failure of reductions to classify certain problems in **NP** is not surprising and was predicted a few years before the first public key cryptosystem by Ladner [LADN]. Let K_1 be a decision problem ("does $I(K_1)$ have property p_1?"), for which there is no known polynomial algorithm. Let K_2 be a decision problem ("does $I(K_2)$ have property p_2?") where $K_2 \in$ **P**. Ladner's result states that if K_3 is the decision problem that asks the question "does $I(K_3)$ have the properties p_1 and p_2" then, $K_3 \propto K_1$ but K_1 will not transform to K_3. Specifically, let $K_1 \in$ **NP**–complete, and assuming **P** $\neq$ **NP**, then K_3 is neither **NP**–complete nor a member of **P**, ie. **NPI** will not be empty. For example, let K_1 be the *SUBGRAPH ISOMORPHISM* problem, which is **NP**–complete. Ladner's result indicates that there exist a class of graphs such that *SUBGRAPH ISOMORPHISM*, when restricted to these graphs is neither **NP**–complete nor a member of **P**. The *GRAPH ISOMORPHISM* problem, which is a special case of *SUBGRAPH ISOMORPHISM*, is neither **NP**–complete nor a member of **P** and is suspected of being in **NPI**.

Further evidence that the number theoretic problems may not be **NP**–complete or **NP**–hard is that there exists a number of reductions amongst these problems, creating their own heirachy of difficulty. Diagram 14.1 shows the reductions between various number theory is problems. Those that have not been defined previously are:

NAME: *squarefreeness* ;

INSTANCE: integer n ;

QUESTION: does there exist a prime p such that $p^2 \mid n$?

NAME: *order* ;

INSTANCE: integer n and $a \in Z^*_n$;

PROBLEM: find $\mathrm{ord}_n a$.

NAME: *index* ;

INSTANCE: integers b, i , prime p, and integer a that generates GF(p^i)

PROBLEM: find the least integer x such that $a^x \equiv b \bmod p^i$?

NAME: *generator* ;

INSTANCE: integers a, i and prime p ;

QUESTION: does a generate the cyclic group of p^i ?

The *SQUARE_ROOTS(a, n)* problem is the search form of the quadratic residue problem. Similarly *FIND_GENERATOR* is the search form of the generator problem. In the Diagram 14.1 a line from problem A to B indicates $A \propto_T B$. Reductions labelled with E indicate that they are members of **ZPP**, ie. the reduction may give no answer but never a wrong answer. Those reductions labelled with a one belong to **RP**, ie. may give an incorrect answer. Diagram 14.1 has

three dividing lines. Each of the problems in the top section is at least as hard as factoring when computed by a PTM. The problems in the

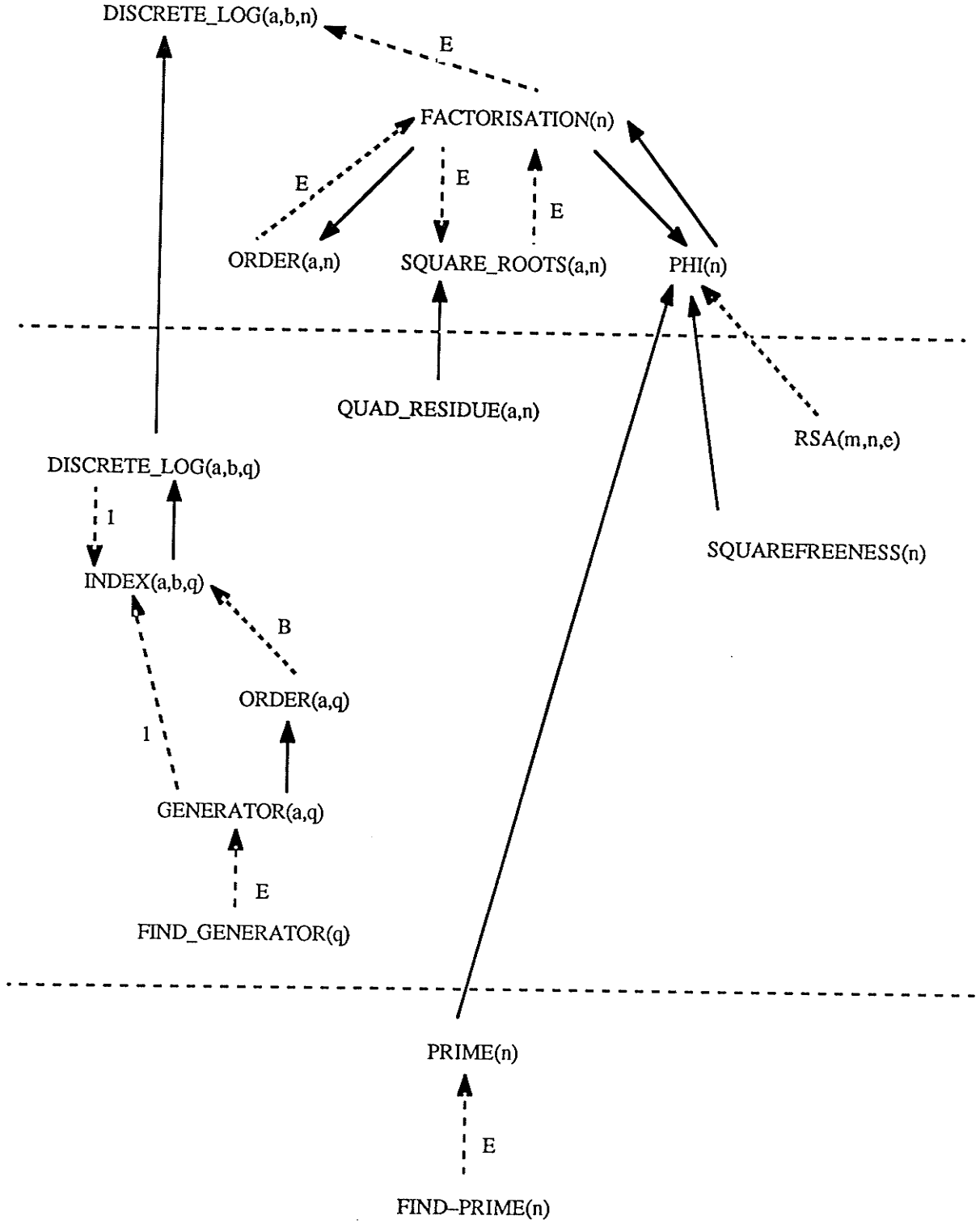


Diagram 14.1

bottom section belong to **RP**. Nothing much is really known about those in the middle section, where the *GCPs* for at least 3 cryptosytems lie[2]. In Diagram 14.1, q is an integer where $q = p^i$ and p is a prime.

The deterministic reductions are straightforward (eg. $\phi(n) \propto_T \mathrm{FAC_n}$, $QRP_n(a) \propto_T$ *SQUARE_ROOTS(a, n)*). The probabilistic reductions are interesting, and we will present the reductions that are relevant to our discussion (Woll [WOLL] gives most of the reductions). Using the ideas of Long [LONG], Finn [FINN] and Bach [BACH], a general algorithm can be constructed that will serve as a basis for the following reductions

$$\mathrm{FAC_n} \propto_T order(a, n),\ \phi(n),\ \mathrm{DLP}(a, b, n)$$

The algorithm [WOLL] is

1. If n is a perfect power, return one of its integral roots;

2. If oracles for *order* or $\phi(n)$ are available then check for primality; return n if it is prime;

3. Choose $a \in Z_n \backslash \{0\}$ randomly and uniformly;

4. Compute $k = \gcd(a, n)$. If $k > 1$ then k is a non–trivial factor of n, return k ;

5. Compute an odd number u such that

   - $u\, 2^t = \mathrm{order}(a, n)$ for some t ;
   - $u\, 2^t = \phi(n)$, for some integer t ;
   - $u\, 2^t = x$, where $a^x \equiv 1$;

[2]Blum and Micali have devised a system based on the index problem [BaM].

6. Compute $odd = a^u \bmod n$;

7. Repeatedly square *odd* until arriving at the first square root of 1.

8. If $r = \pm 1 \pmod{n}$ then goto step 2. Otherwise return gcd(r–1, n) as a non–trivial factor of n. •

The choice of selection in step 5 will depend upon which oracle is available, ie. $\phi(n)$, discrete log or the order oracle. There is no known reduction from the DLP to FAC, nor from the *index* problem to FAC.

Adleman and Manders [AaM], motivated by Ladner's result, use a new type of reduction to find problems that are not **NP**–complete yet are not in **P** either. A γ reduction is similar to a polynomial transformation except that the transformation is non–deterministic. Gamma reductions can be used to show that even though Π is not reducible to every set in **NP** (as the **NP**–complete sets are), if Π is γ complete then $\Pi \notin$ **P** if and only if **NP** $\neq$ **Co–NP**. This is a weaker notion of intractability since

$$NP \neq Co\text{–}NP \Rightarrow P \neq NP$$

but the reverse inplication is not known to be true. Even though a γ reduction is weaker than polynomial time reductions, it is hoped that this weakness will make γ reductions more general, and able to relate more hard problems than other types of reductions.

We formally define a γ reduction here. Let $R_M\langle x, y\rangle$ be a relation from an input string x for a NDTM M to the set of possible output strings y , ie.

$$R_M = \left\{ \begin{array}{l} \langle x, y\rangle : \text{there is a guess } z \text{ such that on the input} \\ \text{sequence } x \text{ and guess } z, M \text{ produces output } y. \end{array} \right\}.$$

where the output of the NDTM is the contents of the tape at the end of the computation. Then problem A is γ–reducible to problem B , denoted $A \propto_\gamma B$, if and only if there is a NDTM M running in polynomial time such that

(14.1) $\forall x, \exists y, \langle x, y \rangle \in R_M$;

(14.2) $\forall x, \forall y\ [\ \langle x, y \rangle \in R_M \Rightarrow [\ x \in A$ if and only if $y \in B\ ]]$.

These two conditions state that M will always halt on a valid input string x, and x is a yes–instance of problem A if and only if the output of M is yes–instance of B. We stress that this is a non-deterministic reduction and that $A \propto B \Rightarrow A \propto_\gamma B$. A γ–complete problem is an **NP** problem such that every problem is **NP** is reducible to it. Then

Lemma 14.2. If A is γ–complete then $A \in NP \cap Co\text{–}NP$ *if and only if* $NP = Co\text{–}NP$ •

Adleman *et al.* show that the linear divisibilty problem is γ–complete by showing that a multisack version of the knapsack problem (which is **NP**–complete) reduces to linear divisibilty. It is a tedious non–deterministic reduction. Gamma reductions are useful for characterising "hard" problems other than **NP**–complete ones, but still these methods only apply to decision problems, and possibly only have application to quadratic residuosity.

Section 15 One Way Functions

In §14 we examined the difficulty of the *GCP*. Here we wish to discuss the components of a PKC more closely, namely the various algorithms that are combined to make a system viable. Creating a PKC is essentially the problem of forming the black boxes of diagrams 7.2 and 7.3, where $f_1 = f_2^{-1}$, and together they realise a trapdoor one way function. The greatest ingenuity involved in designing a system lies in selecting the trapdoor. A worst case analysis does not guarantee that every instance of a problem is diffcult to solve; and, in fact, this is exactly what PKCs are based on: the existence of difficult and easy instances of the same problem. For the one problem, the designer wishes to have an easy algorithm for himself and a hard instance for the general public.

Obviously the encryption function E must be one way, but whether or not it is a trapdoor will depend upon another function. The information that allows E to be a trapdoor one way function is contained in a one way function itself. An acknowledged theoretical weakness of public key cryptosystems is their susceptibility to an opponent with a large (practically infinite) amount of resources[1]. Although this is a theoretical flaw, it is not a weakness in practice. Are we really dealing with infeasible problems at all?

We will argue that a PKC is made up of 3 main algorithms that make it viable: an encryption algorithm, a decryption algorithm and an *obscuring algorithm*. The obscuring algorithm is the transformation that hides the private information which allows the receiver to easily decrypt each ciphertext. The standard defintion of a PKC (see §2) indirectly refers to the obscuring algorithm in condition (2.7),

"for almost all $k \in K$, it is computationally infeasible to derive each equivalent algorithm to D_k from E_k".

[1]Brassard maintains; Theorem 1 [BRASS1]. Public key cryptosystems do not exist.

Notice there is no explicit reference to messages here. A cryptosystem, whether it be symmetric or asymmetric, must guard against two threats: (i) the problem of compromising the key (which is hidden in the obscuring algorithm), and (ii) the problem of compromising the encryption function. These have been called the cryptographic and cryptanalytic problems. Showing that the encryption function E is intractable to invert implies that recovering the keying information (ie. inverting the obscuring algorithm) is also difficult to achieve. For example, in the RSA system, modular exponentiation is a difficult function to invert, given the assumption that the modulus cannot be factored easily. The security of the RSA system is based on this assumption.

Each public key cryptosystem has a public and private key, where the private key contains enough information to reduce a problem instance involving the public key to an easily solved task. The designer begins with an instance of a problem that he can easily solve, or information that leads to a simple solution of a problem. Denote this problem or information as S_{pi}. Knowledge of S_{pi} leads directly to the private key of the system. The designer applies a transformation g to S_{pi}, to produce $g(S_{pi})$, which forms part or all of the public key. Let $S_h = g(S_{pi})$, then Diagram 15.1 shows the effect of g;

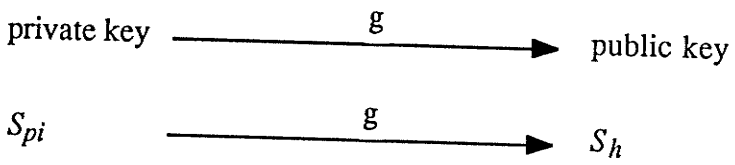


Diagram 15.1

We say that the pair (g, S_{pi}) forms the secret of the system. Clearly g should be a one way function, as an opponent knows that when he is given S_h he has a direct route to S_{pi} via g^{-1}. It is not necessary that g be a trapdoor one way function because the designer is aware of the arguments that were used by g, ie. that g applies to S_{pi} gives S_h. Let E be the encryption function, which is separate from g. The function E should be one way, and for anyone who knows S_{pi}, E has a trapdoor. Let $Easy_p$ denote the set of problem instances of p that can be solved effciently, and

$Hard_p$ those instances that are not known to have an efficient solution. Let M be the message space and $Hard_E$ the cipherspace. Then encryption and decryptions functions E and D are defined as,

$$E : \text{Hard}_\text{p} \times M \longrightarrow \text{Hard}_\text{E} \tag{15.1}$$

$$E(g(S_{pi})), m) = C, \tag{15.2}$$

$$D : \text{Hard}_\text{E} \times \text{Easy}_\text{p} \longrightarrow M \tag{15.3}$$

$$D(C, g^{-1}(S_h)) = m. \tag{15.4}$$

Both E and g are one way functions because they map into "hard" ranges. E is a trapdoor one way function as shown by the existence of D, but we must have the knowledge of g–1. A public key cryptosystem is defined as the 5-tuple,

$$\text{PKC} = <S_{pi}, g, E, D, M> .$$

Using the notation from the previous section, the RSA system is defined as

$$\text{RSA} = < (p,q), \text{g}(p, q) = p \cdot q = n, c = m^e \bmod n, m = c^d \bmod n, 0 < m < n - 1 >^* .$$

In this case the large primes p and q form the instance S_{pi}. The function g is then the binary operation of multiplication ,

$$g(p, q) = \{ n = p \cdot q : \log_2 p = \log_2 q = k, \log_2 n = 2k \}$$
$$g : P_k \times P_k \longrightarrow N_{2k}$$

where P_k is the set of k-bit primes and N_{2k} the set of $2k$-bit integers. The encryption function is a permutation of the $2k$–bit integers, as is the decryption function, which uses the factors of n.

---

*Though not a difficulty issue, the RSA encryption function is a *one way permutation* [DIF], which allows signatures. Shannon in *Secrecy Systems* called such systems *endomorphic*.

$$E : N_{2k} \longrightarrow N_{2k}$$

$$E(m) = (m^e \bmod n)$$

$$D : N_{2k} \times P_k \times P_k \longrightarrow N_{2k}$$

$$D(E(m), p, q) = m$$

For the RSA system both Hard_p and Easy_p are the set of $2k$–bit integers, which seems odd at first since one feels that the domain should be either hard or easy. The point to take is that computing roots in the domain is hard *if* the factors of the modulus (ie. $g^{-1}(S_i)$) are not known.

For the quadratic residue cryptosystem, as with those based on factoring, the factors of n constitute S_{pi}, g is the binary operation of multiplication, and S_h is the product of p and q. The two one way system functions are

$$g(p, q) = \{ n = p \cdot q : \log_2 p = \log_2 q = k, \log_2 n = 2k \}$$
$$E(m_i) : m_i \longrightarrow Z^*_n(+1)$$

The ElGamal system is described by the 5-tuple

$$< a, ha, (h^x, m g^{ax}), g^{ax}, 0 \leq m \leq q - 1 > .$$

The integer a is the information that the designer alone possesses, and hence a is the instance of S_{pi}. It is obscured by exponentiation mod q. Inverting g is an instance of the DLP.

Even though E and g are distinct functions they could be based on the same one way function, such as factoring. Each public key cryptosystem is derived from some hard problem, such as those which we have identified as the basic problems. Consider these definitions:

Definition 15.1. A function f is f–secure if the task of computing the inverse of f is equivalent to solving an intractable problem S, ie. $f\text{-}1 \equiv_T S \in$ **IP**.

Hence, if f is f-secure then f is a one way function. We have,

Definition 15.2. A public key cryptosystem $<S_{pi}, g, E, D, M>$ is E–secure if the encryption function E is E-secure.

An implication of Definition 15.2 is that an E–secure PKC is also g–secure. If an E–secure PKC is compromised, then the efficient illegal decryption will produce an efficient algorithm for solving an intractable problem. We also define a weaker notion of security.

Definition 15.3. A public key cryptosystem $<S_{pi}, g, E, D, M>$ is E–difficult if E^{-1} is reducible in polynomial time to a problem S that is intractable, (ie. $E^{-1} \propto_T S$ and $S \in$ **IP**).

Compromising a system that is E–difficult simply indicates that either g or E was a poor selection as a one way function. This was the case with modular transformations and the M–H knapsack cryptosystems. When a public key cryptosystem is said to be based on a problem p, this normally means that the system is E–difficult, with problem p being involved in the reductions, (eg. RSA based on factoring). Table 15.1 classifies some systems according to the above definitions.

Cryptosystem	Security
RSA	E–difficult
Rabin[2]	E–secure
M–H	not g–secure
Diffie–Hellman DLP	E–difficult
DLP El Gamal	E–difficult
Quad. Res.	E–secure
Chor–Rivest	E–difficult

[2]This system is described in §16.

DES ?

Table 15.1

In Definition 15.1 we say that f is one way if f^{-1} is equivalent to some intractable problem. It would be a stronger assertion to say that f^{-1} is **NP**–hard, **NP**–complete or in **Co–NP**, ie. classify its complexity. Brassard [BRNP] has shown that we cannot expect to be more concise in our defintion of a one way function than we were in Definition 15.1.

Consider the DLP for some finite field. Note that we have no natural decision problem for the DLP since we know that every member of the field will have a discrete logarithm with respect to a generator (by definition). Let $DLP(q, g, y)$ be an instance of the DLP, where we wish to determine x such that $y = g^x$ *in* GF(q) . Brassard [BRNP] defines the projection P_{log} as

$$P_{log} = \{ < q, g, y, t > / DLP(q, g, y) > t \}$$

Using Pratt's certificate [PRAT] and Fermat's theorem [ROS] q and g can be verified as a prime and a primitive root respectively by a NDTM in polynomial time. The NDTM can also produce the certificate for x and thus $P_{log} \in$ **NP**. Similarly the complement of P_{log} (ie. DLP(q, g, y) $\leq$ t ?) can be checked by a NDTM and hence $P_{log} \in$ **NP** $\cap$ **Co–NP**. Finally, DLP(q, g, y) $\propto_T$ P_{log} by the following algorithm,

1. $i \leftarrow 0$
2. $j \leftarrow q - 1$
3. **while** $i < j$ **do**
4. $k \leftarrow \lfloor (i+j)/2 \rfloor$
5. **if** $<q, g, y, k> \in P_{log}$ **then** $i \leftarrow k + 1$
   else $j \leftarrow k$

6. **end_do**
7. *return i*

This method of binary search is a common technique of producing a Turing reduction, as the search spaces for some problems are well defined. Clearly, $P_{log} \propto_T$ DLP. If the DLP were **NP**–hard, ie. Turing reducible to some **NP**–complete problem Π, then P_{log} would also be Turing reducible and hence **NP**–complete. However since $P_{log} \in$ **NP** $\cap$ **Co–NP**, then from the remarks in section 4, this would imply that **NP** = **Co–NP**, which is very unlikely. Thus it is concluded that the DLP is not expected to be **NP**–hard. By changing the oracle so that for n it determines if $y > t$, where y is a prime factor of n, it can be shown that the problem of factoring the RSA modulus is also unlikely to be **NP**–complete.

CHAPTER 5 Specific Problems

A reduction, such as a Turing reduction or a polynomial transformation, is used to indicate the relative difficulty of one general problem with respect to another. It is a powerful concept, and appears to be the only way to answer all cryptographic threats simultaneously. Reductions are a means of distributing the risk of a cryptosystem, since if the system is broken, other problems can also be solved efficiently. The most theoretically satisfying system would be one whose security is given not in terms of an equivalence to several problems, but in terms of complexity class relations. For example, if the *GCP* of a system were shown to be **NP**–complete, then breaking the system efficiently would be equivalent to deciding **P** =? **NP**. Settling **P** =? **NP** will affect all (decision) problems in **NP**. Also, when we deal with problems that will affect class relations we are necessarily dealing with the most general form of the problem and not a form that has been contrived for cryptography. For example, we see that there is no known reduction from the general *SUBSET SUM* problem to the M–H knapsack system, which is based on a superincreasing sequence, though both have similar descriptions. Since there does not appear to be any transformation from the general form to the specific M–H form of the knapsack problem, it is not expected that these problems are of equivalent difficulty (as was shown in §12).

Since there has been little success with showing the equivalence between general and specific forms of one problem, or even between two problems, designers have settled for reductions between specific forms of different problems. Recall from §7 that we cannot identify a hard instance of any problem, but by a worst case analysis, it is expected that at least one hard instance exists for each intractable problem. If a problem instance is solved in polynomial time it is agreed that this is an easy problem instance. But there is a risk with selecting an instance of that problem and then purporting that it is a hard instance of the general problem. Merkle and Hellman seemed convinced that a set of weights based on a modular transformation from a superincreasing series constituted a difficult instance of the *SUBSET PROBLEM*. Since no other problem reduced to the *GCP* of the M–H knapsack, its cryptanalysis was an isolated result, in that it did not alter the

complexity of any general problems. Let the universe of knapsackproblems **UK** consist of all instances of the knapsack problem. We can partition **UK** into instances that are easily solved and those that are not known to be so. From §12 we see that knapsack problems where the weights are derived from a superincreasing series via a modular transformation are easily solved. Also, the iterated M–H knapsack, low density knapsacks and various other knapsack–like systems [BRIC2] have been compromised. By increasing the number of partitions of easily solved knapsack problems in **UK**, the region where a difficult instance can lie is narrowing. Possibly this will be the greatest contribution of cryptography to complexity theory – polynomial time algorithms for specific problem instances.

The reader might have the impression that deciding the relationship between **P** and **NP** is the major obstacle to demonstrating that cryptosystems are computationally secure. For a moment let us assume that someone gave a proof that $\mathbf{P} \neq \mathbf{NP}$. That is, not every decision problem in **NP** could be solved in polynomial time. How would this result effect the security of the cryptosystems that have been devised to date? There would be no effect at all. $\mathbf{P} \neq \mathbf{NP}$ would not automatically prove the security of any cryptosystem as no *GCP* is known to be **NP**–hard, or **NP**–complete. Plainly, no system is that general. On the other hand, what can we expect to gain of the *GCP* if certain systems were solvable efficiently ? Generally a cryptosystem is said to based on a hard problem p, which indicates that the difficulty of compromising the system cannot exceed the complexity of p. But the situation is even worse. A system based on problem p normally means that the system is based on a very specific form of problem p. Asserting, as is often done, that the RSA system is based on factoring sounds quite reasonable. If the modulus $n = pq$ is factored then the system is broken. For sake of argument, assume that the problem of recovering a message m from $m^e \bmod n$ is equivalent to determining the factors of n. Let $AL(m^e \bmod n)$ run in polynomial time and return m. How does AL affect the general problem of determing the complete prime factorisation of a arbitrary number x ? Can $AL(\cdot)$ be used to factor x ? It is not obvious that it could, and we would draw a similar conclusion that the demise of the M–H brought with itself,

namely, the specific case of factoring a number that has exactly two prime factors is easily solved. But, since algorithm *AL* is not known to exist, no one can demonstrate that the specific instance of $n = pq$ is easier to solve than the general case, and the designers of RSA would hope that this is the case. An efficient algorithm *BL* that broke the Goldwasser–Micali system would yield an algorithm to decide quadratic residuosity in a ring where the modulus is the product of two distinct primes, but again, how would the existence *BL* alter the general problem form ?

Perhaps we are crossing our bridges before we arrive at them. It is possible, and quite probable, that a method of factoring $n = pq$ will provide techniques that will prove useful in solving the general factoring problem. By travelling to the moon we appeciate the immensity of travelling further distances. At present we cannot see how an algorithm for factoring $n = pq$ can be used to factor arbitrary integers, but its significance may not be realised until it has actually been found.

For what it may be worth, there are several cryptosystems that have been shown to be equivalent in difficulty to specific instances of intractable problems. The assumption is that the specific problem instance is probably as hard as any other general instance of the problem, though it is not known if an algorithm for the specific problem will alter the complexity of the general problem. However, it stands to reason from the discussion of hard and complete problems in §14, that if *I(p)* is one of the hardest instances of problem *p*, and *I(p)* can be easily solved, then the majority of problem instances should be easily solved. In special cases, solving

$$x^k \equiv a \mod n \tag{16.1}$$

is equivalent in difficulty to factoring *n*. Rabin [RAB] has proved that solving eq.(16.1) for $k = 2$ is randomly equivalent to factoring the modulus which is the product of two primes. Williams [WIL2] [WIL3] further shows that for $k = 3$, and more generally for all even *k*, solving eq.(16.1) is equivalent to factoring the modulus when the factors of *n* are of a special form. We present here

variations of the RSA and discrete logarithm systems that are equivalent in difficulty to specific instances of factoring. Also, we present a CSPBG that is as secure as factoring.

16.1 More on Quadratic Residuosity

In integer arithmetic, if a is a perfect square and x is a square root of a, then so is $-x$. In Z_n we have the following

Theorem 16.1. Let n be the product of two distinct primes p and q. Any $a \in QR_n$ has four distinct square roots modulo n.

proof. Let $x = x_0$ be a solution to eq.(16.1) when $k = 2$. Then let $x_0 \equiv x_1 \bmod p$ and $x_0 \equiv x_2 \bmod q$. By Corollary 10.2, $a \pmod p \in QR_p$ and $a \pmod q \in QR_q$. The congruence $x^2 \equiv a \bmod p$ will have two incongruent solutions, namely x_1 and $p - x_1$. Similarly, x_2 and $q - x_2$ are incongruent solutions to $x^2 \equiv a \bmod q$. Thus, solving solving the following four sets of congruences will yield the four distinct roots modulo n,

$$\begin{aligned} x &\equiv x_1 \bmod p & \qquad x &\equiv p - x_1 \bmod p \\ x &\equiv x_2 \bmod q & \qquad x &\equiv x_2 \bmod q \end{aligned}$$

$$\begin{aligned} x &\equiv x_1 \bmod p & \qquad x &\equiv p - x_1 \bmod p \\ x &\equiv q - x_2 \bmod q & \qquad x &\equiv q - x_2 \bmod q \end{aligned}$$

By the Chinese Remainder Theorem, the four distinct roots modulo n will exist. •

Example 16.1. Let $n = 77 = p \cdot q = 7 \cdot 11$ and consider finding the 4 distinct roots of $x^2 \equiv 15 \bmod 77$, given it is known that $13^2 \equiv 15 \bmod 77$. Following Theorem 16.1, $x_0 = 13$ and

$$\begin{aligned} 13 &\equiv 6 \bmod 7 \quad (x_0 \equiv x_1 \bmod p) \\ 13 &\equiv 2 \bmod 11 \quad (x_0 \equiv x_2 \bmod q) \end{aligned}$$

We see that $x^2 \equiv 15 \equiv 1 \bmod 7$ has two incongruent solutions,

$$x \equiv 6 \bmod 7 \quad (x \equiv x_1 \bmod p)$$
$$x \equiv 1 \bmod 7 \quad (x \equiv p - x_1 \bmod p)$$

as $6^2 \equiv 36 \equiv 1 \bmod 7$. Similarly, we see that $x^2 \equiv 15 \equiv 4 \bmod 11$ has two incongruent solutions,

$$x \equiv 2 \bmod 11 \quad (x \equiv x_2 \bmod q),$$
$$x \equiv 9 \bmod 11 \quad (x \equiv q - x_2 \bmod q),$$

as $9^2 \equiv 81 \equiv 2^2 \equiv 4 \bmod 11$. Thus, to find the four incongruent solutions mod 77 we solve the following linear congruences

$x_1 \equiv 6 \bmod 7$	$x_2 \equiv 1 \bmod 7$
$x_1 \equiv 2 \bmod 11$	$x_2 \equiv 9 \bmod 11$
$\Rightarrow x_1 = 13$	$\Rightarrow x_2 = 64$
$x_3 \equiv 6 \bmod 7$	$x_4 \equiv 1 \bmod 7$
$x_3 \equiv 9 \bmod 11$	$x_4 \equiv 2 \bmod 11$
$\Rightarrow x_3 = 20$	$\Rightarrow x_4 = 57$.

Notice that $x_1 \equiv -x_2 \bmod 77$ and $x_3 \equiv -x_4 \bmod 77$. Thus, ± 13 and ± 20 are the four distinct square roots of 15 mod 77. •

For the remainder of the section, assume that $p \equiv q \equiv 3 \pmod 4$. The modulus from example 16.1 satisfies this condition. When $n = pq$ is of this form, we have that

Theorem 16. 2. $(-x \mid n) = (x \mid n)$.

proof. Since $(p-1)/2$ is odd, $(-1 \mid p) = -1$ by Euler's criterion (Lemma 10.4). Similarly $(-1 \mid q) = -1$. Thus, $(-1 \mid n) = +1$. It follows that $(-x \mid n) = (-1 \mid n)(x \mid n) = (x \mid n)$. •

Example 16.2. From the previous example, $(13 \mid 77) = (-13 \mid 77) = (64 \mid 77)$. To show this, first calculate $(13 \mid 7)$ and $(64 \mid 7)$ using Lemma 10.4:

$$13^{(7-1)/2} \equiv 2197 \equiv -1 \bmod 7, \text{ and } (13\,|\,7) = -1,$$
$$64^3 \equiv 2^6 \cdot 2^6 \cdot 2^6 \equiv 1 \bmod 7, \text{ and } (64\,|\,7) = 1\ .$$

Similarly, we find that $(13\,|\,11) = -1$ and $(64\,|\,11) = 1$. Then $(13\,|\,77) = (13\,|\,7)(13\,|\,11) = 1$ and $(64\,|\,77) = (64\,|\,7)(64\,|\,11) = -1$. •

Theorem 16.3. If $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$, then $(x\,|\,n) = -(y\,|\,n)$.

proof. Let x and y be as stated. Then $n \mid (x^2 - y^2) = (x + y)(x - y)$, and since $x \not\equiv \pm y \bmod n$, then p and q cannot both divide $(x + y)$ or $(x - y)$. Let $x \equiv -y \bmod p$ and $x \equiv y \bmod q$. It follows that $(x\,|\,q) = (y\,|\,q)$, and $(x\,|\,p) = -(y\,|\,p)$ since $-(y\,|\,p) = (-1\,|\,p)(y\,|\,p) = (-1\,|\,p)(x\,|\,p)$. The theorem follows. •

The next theorem is the most important one of the section. It states that every quadratic residue has a square root that is itself a quadratic residue.

Theorem 16.4. If $a \in QR_n$ then the mapping $a \longrightarrow a^2 \bmod n$ is a bijection.

proof. Let a be as stated. Then by Theorem 16.1 it will have four square roots modulo n, say $\pm x$ and $\pm y$. By Theorem 16.3, $(x\,|\,n) = -(y\,|\,n)$. Let $(x\,|\,n) = 1$ and $(y\,|\,n) = -1$. By Theorem 16.2, it follows that $y, -y \notin QR_n$. As $(-1\,|\,p) = -1$, then $(x\,|\,p) = -(x\,|\,p)$ and similarly, $(x\,|\,q) = -(x\,|\,q)$. As we have assumed arbitrarily that $(x\,|\,n) = (-x\,|\,n) = 1$, then we have either $(x\,|\,p) = (x\,|\,q)$ or $(-x\,|\,p) = (-x\,|\,q)$. Thus, either x or $-x$ will be a quadratic residue of n, but not both. •

Example 16.3 From example 15.1, ± 13 and ± 20 are the four solutions to $x^2 \equiv 15 \bmod 77$. From Example 16.2, $(13\,|\,77) = (-13\,|\,77)$. Also, $(20\,|\,77) = (-20\,|\,77) = (57\,|\,77)$ since,

$$20^3 \equiv 6 \bmod 7 \Rightarrow (20\,|\,77) = -1,\ 57^3 \equiv 1 \bmod 7 \Rightarrow (57\,|\,7) = 1,$$
$$20^5 \equiv 1 \bmod 11 \Rightarrow (20\,|\,11) = 1,\ 57^5 \equiv 10 \bmod 11 \Rightarrow (57\,|\,11) = -1\ .$$

Simialrly,

$$(64\,|\,7) = (64\,|\,11) = 1,\ (13\,|\,7) = (13\,|\,11) = -1,$$
$$(57\,|\,7) = -(57\,|\,11) = 1,\ (20\,|\,7) = -(20\,|\,11) = -1\ .$$

Hence, $64 \in QR_{77}$, $13 \in Z^*_{77}(+1)$, $57, 20 \in Z^*_{77}(-1)$. Clearly, $64 = 8^2$ is the only root which is a quadratic residue. •

Consider the sequence $\mathbf{x} = x_0, x_1, x_2, \ldots, x_k$ generated by the following rule;

$$x_{i+1} \equiv (x_i)^2 \bmod n\ , \tag{16.2}$$

where $x_i \in QR_n$, $i = 0, 1, 2, \ldots, k$. By Theorem 16.4, given the factorisation of n and any x_i, $0 \leq i \leq k$, all the terms in the sequence that precede x_i, ie. x_j for $0 \leq j \leq i - 1$ can be uniquely determined in polynomial time. Given n and an arbitrary x_i, the sequence can always be generated in the forward direction by applying eq.(16.2). But, by appealing to Corollary 10. 2, given an $x_i \in \mathbf{x}$, efficiently generating $x_{i-1}, x_{i-2}, \ldots, x_0$ is not known to be possible without the factors of n.

In the Goldwasser–Micali system the problem was given $x_i \in Z^*_n(+1)$, determine if $x_i \in QR_n$. Now, consider the problem where we know that $x_i \in QR_n$, and we wish to find x_{i-1} such that $(x_{i-1})^2 \equiv x_i \bmod n$. We are essentially switching between decision and search forms of the quadratic residue problem.

Blum, Blum and Shub [BBS] have devised a bit–by–bit encryption scheme that relies on Theorem 16.4 to decrypt. Equation 16.2 is used to generate random bits; let *RNG* be a psuedo random number generator, such that when given a seed $x_0 \in QR_n$, generates the sequence of bits $\mathbf{b} = b_1, b_2, \ldots, b_i$ by

$$x_{i+1} \equiv (x_i)^2 \bmod n \text{ and output } b_i = \text{parity}(x_{i+1})\ . \tag{16.3}$$

The next theorem shows that an algorithm for determining b_{i-1} given b_i can be used to decide $QR_n(a)$.

Theorem 16.5. Let *AL* be an algorithm that, given $b_i = \text{parity}(x_i) = \text{parity}((x_{i-1})^2 \bmod n)$ as input, returns $b_{i-1} = \text{parity}(x_{i-1})$ where $x_{i-1} \in QR_n$. If *AL* runs in polynomial time then the predicate $QR_n(a)$ can be decided in polynomial time as well.

Proof. Imagine that we wish to determine $QR_n(x_0)$ for an arbitrary x_0. Let $x_1 = (x_0)^2 \bmod n$ and set $b_1 = \text{parity}(x_1)$. Assume that $\pm x$ are the solutions to $x^2 \equiv x_1 \bmod n$, and $\pm x \in Z^*_n(+1)$. Since n is odd, then $\text{parity}(x) \neq \text{parity}(-x)$ and only one of x and $-x$ is a quadratic residue. Thus $x_0 \in QR_n$ if and only if $AL(\text{parity}(x_1)) = \text{parity}(x_0)$. •

Corollary 16.2. Let $\mathbf{b} = b_1, b_2, \ldots, b_k$ be generated as in eq.(16.3), then given any b_i, $1 \leq i \leq k$, generating the sequence in the backwards direction is at least as hard as deciding $QR_n(a)$ where $n = pq$ and $p \equiv q \equiv 3 \bmod 4$. •

The $X^2 \bmod N$ generator defined by eq.(16.3) is CSPBG, and produces a sequence that cannot be predicted in polynomial time. Blum, Blum and Shub (BBS) have used the output of this generator to approximate Shannon's one time pad scheme [KAHN]. In the one time pad each message bit m_i is encrypted by the rule

$$c_i = m_i \oplus k_i \tag{16.4}$$

where each k_i is a truly random bit, and $\oplus$ is the *exclusive–or* operator. To decrypt, the key is added to the ciphertexts again. This system is unconditionally secure since if each bit k_i is random then an opponent has no means of unambiguously determing if m_i was 1 or 0. He can do no better than guess.

In the BBS system, the encryption function is the same as eq.(16.4) except that the k_i are produced by the $X^2 \bmod N$ generator. The BBS systems can also be used as a PKC. Imagine that Alfred wishes to communicate to Barbara the message $m = m_1, m_2, \ldots, m_s$. Barbara has as her public key $n = pq$. Alfred randomly selects $x \in Z^*_n$ and sets $x_0 = x^2 \bmod n$. Alfred then forms the sequence $\mathbf{b} = b_1, b_2, \ldots, b_s$ by invoking the $X^2 \bmod N$ generator n times, and then exclusive–

oring the sequence **b** with the message m, ie. $c = \mathbf{b} \oplus m$. Alfred sends Barbara (c, x_{s+1}). The secret key of the system is the factorisation of n, and with this Barbara can generate the sequence **b** from x_{s+1} and recover the message.

The next theorem, due to Rabin, shows that any algorithm that could be used to generate the sequence produced by eq.(16.2) backwards could be used to factor the modulus n. This theorem requires a procedure that will compute the square roots of a residue modulo a prime. Let $a \in QR_n$. Since $p \equiv q \equiv 3 \bmod 4$ and $n = pq$, then $a^{(p+1)/4}$ is a square root of $a \bmod p$ since $a^{(p+1)/4}a^{(p+1)/4} \equiv aa^{(p-1)/2} \equiv a \bmod p$. Similarly, $a^{(q+1)/4}$ is a solution to $x^2 \equiv a \bmod q$.

Theorem 16.6. The sequence $\mathbf{x} = x_0, x_1, \ldots, x_k$ of eq.(16.2) can be generated in the backwards direction if and only if modulus of can be factored.

proof. Let *AL(a, n)* be an algorithm that returns z where $z^2 \equiv a \bmod n$ and $z \in QR_n$. Choose a random y such that $(y \mid n) = -1$ and let $x \equiv y^2 \bmod n$. If $z = AL(x, n)$ then by Theorem 16.3, gcd($y + z, n$) will be one of the prime factors of n. Conversely, let $BL(n)$ be an algorithm thats returns the factors of $n = pq$. Given $x \in QR_n$, the following procedure will find a square root of x that is also a quadratic residue. Compute $u \in QR_p$ and $v \in QR_q$ such that $x \equiv u^2 \bmod p$ and $x \equiv v^2 \bmod q$. Compute a and b such that $ap + bq = 1$ and let $c = bq$ and $d = ap$. Since $c \equiv 1 \bmod p$ and $d \equiv 1 \bmod p$ then $w = (cu + dv)^2 \equiv x \bmod n$. Further $c \equiv 1 \bmod p$, $c \equiv 0 \bmod q$, $d \equiv 0 \bmod p$, and $d \equiv 1 \bmod q$. Since, $(w \mid p) = (u \mid p) = 1$ and $(w \mid q) = (v \mid q) = 1$, then $w \in QR_n$. •

16.2 RSA and Factoring

Section 9.2 showed an equivalence between several problems that are naturally considered when discussing the RSA system. The security of the RSA system remains undecided, or the complexity of RSA^{-1} is unknown. It is known that the difficulty of RSA^{-1} cannot exceed the difficulty of $\mathrm{FAC_n}$, $\mathrm{KEY_d}$ or $\mathrm{PHI_n}$ but can we find a lower bound (ie. an Ω–bound) or equivalent

problems ? Finding lower bounds for general problems has remained fruitless as evidenced by the **P** =? **NP** question remaining unanswered. Some results have been wrought by placing restrictions on the problem or model of computation, which permit certain assumptions to assist the analyst. Galil [GAL] has shown that a subclass of algorithms for the satisfiabilty problem are intractable, while Chvatal [CHV] has shown a similar result for the independent set problem. Borodin [BORO] introduced the concept of a structured model, where the abstract computing machine has its operations restricted (say to purely arithmetic operations, comparisons of list elements) and has deduced some lower bounds.

For the RSA system, an intractability result has been proved by fixing the public key (exponent) to 2, ie. to encrypt a message we square it and reduce it modulo n. We delayed the discussion of this system until quadratic residuosity had been introduced. By insisting that $e = 2$ we are restricting ourselves to a subclass of RSA encryption functions. Recall that a problem is considered intractable if at least one instance of the that problem is intractable. Thus if RSA^{-1} with $e = 2$ is intractable, then RSA^{-1} in general is intractable. We show that when $e = 2$, RSA^{-1} is equivalent to factoring the modulus. This will allow us to show that some instances RSA are E–difficult.

When $e = 2$, RSA^{-1} is equivalent to the problem of computing square roots: $\text{SQRT}_{n,m}$ is defined as determing a random solution to the congruence,

$$m^2 \equiv c \bmod n \quad . \tag{16.5}$$

If n is an odd prime, there are probabilistic polynomial time algorithms due to Shanks, and others (see [SHK]) that when given n and $c \in QR_n$ will compute a square root of c mod n. This can now be used to show that $\text{SQRT}_{n,m} \approx_r \text{FAC}_n$. The theorem is due to Rabin [RAB] (see [KRAN]).

Theorem 16.7. (Rabin's Factorisation Theorem). Let n be the product of two primes. Then the following assertions are equivalent:

(i) there is a polynomial time algorithm such that for all $c \in QR_n$, $AL(n,c)$ yields a random solution to the congruence $m^2 \equiv c \bmod n$;

(ii) there is a polynomial time algorithm for factoring n;

proof. (i) $\rightarrow$ (ii) Given c, apply Shanks' or some other algorithm to determine roots x and y of the congruences $m^2 \equiv c \bmod p$ and $m^2 \equiv c \bmod q$. Since $\gcd(p, q) = 1$, use an application of the Euclidean algorithm to compute integers k and h such that $kp + hq = 1$. Then $hqx + kpy$ is a solution of $m^2 \equiv c \bmod p$ and $m^2 \equiv c \bmod q$ and hence is a solution to $m^2 \equiv c \bmod n$.

(ii) $\rightarrow$ (i) Randomly choose m such that $(m, n) = 1$ and let $m^2 \equiv c \bmod n$. If $u = AL(n, c)$, then both m and u are solutions to the congruence $m^2 \equiv c \bmod n$. If $u \notin \{m, n-m\}$, ie. the roots are distinct mod n, then $\gcd(n, u+m)$ yields a prime factor of n. If the roots are not distinct then repeat the procedure. Since $u \in \{m, n-m\}$ with a probability of at least 1/2, the procedure is expected to yield a factor of n after 2 trials. •

Williams [WIL2] proves a similar result, that does not rely on any probabilistic arguments, by restricting the form that the modulus can take, namely $n = pq$ where $p \equiv 3 \bmod 8$ and $q \equiv 7 \bmod 8$. There is no evidence to suggest that factoring a number of this form is any easier than the general factorisation problem. One of the problems with Rabin's system is that decryption can be ambiguous. There are four distinct solutions to eq.(16.5) if n is composite and if the plaintext contains any redundancy, then the message should be obvious. However, if numbers were sent, then the ambiguity cannot be resolved simply. Williams' system uses the properties of the Jacobi symbol to uniquely isolate the decrypted message. Later, Williams [WIL4] was able to devise an RSA–like encryption function that is as difficult to break as factoring a modulus which is the product of arbitrary primes.

16.3 The Discrete Logarithm Problem and Factoring.

McCurley [McC] has recently devised variations of the Diffie–Hellman and ElGamal systems (§11.3) that are at least as secure as factoring a specific integer. The protocol of these systems remains unchanged from the original systems. The essential difference is that the modulus is now composite.

Consider choosing numbers r and s such that

(16.6) $2r + 1$ has a large prime factor, $4r + 1$ is prime and $8r + 3$ is prime;

(16.7) s has a large prime factor, $4s–1$ is prime, and $8s - 1$ is prime.

Let $p = 8r + 3$, $q = 8s - 1$ and $n = pq$. Let algorithm AL return g^{ab} mod n when given g^a and g^b, ie. $AL(g, n, g^a, g^b) = g^{ab} \bmod n$. If p and q are chosen as suggested by McCurley, he shows that the ability to compute $AL(g, n, x, y)$ for the fixed base $g = 16$ implies the ability to factor n. McCurley's main theorem is as follows.

Theorem 16.8. [McC] Let $n = pq$ be as specified by (16.6) and (16.7). Let $0 < \delta < 1$. Let $<g>_n$ be the subgroup of Z^*_n generated by g. Let BL be an algorithm using at most $R(n)$ bit operations that will output $AL(16, n, x, y)$ for at least $\delta(\text{ord}_n\ 16)^2$ of the input pairs $(x, y) \in <16>_n \times <16>_n$. Then there exists an algorithm CL that will output p and q with probability at least 1/2, using at most $O(\delta^{-1}(R(n) + G(n)))$ bit operations, where $G(n) = O(\log^2 n)$ represents the time to calculate the gcd's. •

The reader is referred to the paper for the actual proof; we simply mention here that algorithm CL attempts to generate a pair (j, k) of incongruent integers mod n, such that $j^2 \equiv k^2 \bmod n$. In the Composite Diffie–Hellman system, a common modulus n is chosen according to (16.6) and (16.7). If Alice and Bob wish to exchange a key, then Alice randomly selects a, and sends 16^a mod n. Bob randomly selects b, and sends 16^b mod n. The common key is then 16^{ab} mod n. McCurley has elegantly combined the problems of factoring and determing discrete logarithms.

Even if an opponent can determine the factors of n, he must still deal with breaking the original Diffie and Hellman scheme for the two prime factors p and q to recover the message. Using the base $g = 16$ and selecting n appropriately, the ElGamal system can also be modified to be as secure as factoring the modulus.

Section 17 Conclusion

At the outset of this paper we began by establishing the connection between cryptography and hard transformations. We have proceeded to examine several problems, each of which is considered to be intractable and appear to be a useful means of furnishing secrecy. What has been learned from this examination ?

First, the security of a PKC, or even the strength of its individual components (such as the encryption function), are not expected to be provably difficult. That is, no known *GCP* is expected to lie in a complexity class that contains intractable problems exclusively (such as **NP**–complete or **NP**–hard). But still there are cryptosystems that cannot, as yet, be broken in polynomial time. There are two conjectures one can make here: either polynomial algorithms for these systems will be found eventually; or a worst case analysis is an inappropriate method of appraising the difficulty of a cryptographic problem. There is nothing certain to be said about which intractable problems have polynomial algorithms, but it is clear that a worst case analysis is a poor model of cryptographic difficulty.

There is a central contradiction that seemingly defeats the use of complexity theory. Designers have welcomed intractable problems, because there are no known general polynomial algorithms to solve these problems (which characterises their intractability). The designer takes an easy instance of the problem, and then a hard instance, and combines both instances into an hybrid problem which becomes the public key. Decryption of ciphertexts is easy for the designer but hard for the general public. But by a worst case analysis the encryption function *is not difficult to invert* as there *does exist* an algorithm that will invert the encryption function in polynomial time, namely the algorithm that the designer uses. Bluntly, if other users cannot decrypt in polynomial time it merely means that they are using an inefficient suboptimal algorithm, *not that the problem cannot be solved in polynomial time.* This brings us to our second conclusion.

The notion of difficulty in cryptography is knowledge–based rather than complexity or algorithm–based. The real issue is not whether the problem can be solved in polynomial time but rather, if the knowledge about a problem be restricted so that it cannot be solved in polynomial deterministic or expected time. This is the purpose of the obscuring function g in a PKC. The modular mapping in the M–H system was unsuccessful, but multiplication in the RSA system seems to be successful in restricting the knowledge about the encryption function. The obscuring function implicitly reveals the knowledge that permits rapid decryption of ciphertexts. The craft of the designer lies in how the knowledge that is encoded by g can be used to pose a hard problem for those who cannot decode the information that $g(S_i)$ conceals. The knapsack systems have been discredited by the cryptanalysis of all but one system based on this problem. The attraction of the knapsack problem is that it adapts so elegantly to the encryption of a binary message ($n - 1$ additions and maybe a modular redcution), but there has been little research on how $S = \sum x\mathbf{b}^*$ may be used as an obscuring function. How can we use the knowledge encoded in the integer S to construct a public key ? No one has discredited this possible use of the knapsack problem.

There is a new branch of analysis called information centered theory [TaW], which deals with the solution of problems according to the quality of the knowledge available. The algorithm based approach to a problem establishes an algorithm for the task, which is then analysed. Another algorithm (usually an improvement to the first) is then proposed, and analysed and so on. The process of achieving an optimal solution is often iterative (if an optimal algorithm is known to exist, that is). The information centered approach states that given a particular amount of a particular type of knowledge, we can solve a problem with a particular degree of certainty. This theory is not primarily concerned with the time taken to solve a problem, but how well it can be solved with the knowledge available.

Consider the problem of interpolating $F(x)$ and say that for the knowledge $N(t) = [\, F(t_1), F(t_2), \ldots, F(t_n) \,]$, $F(t_i) = f(t_i)$, $i = 1, 2, \ldots n$. Then we say that F and f are indistinguishable under the knowledge $N(t)$. What we wish to establish in cryptography is a notion of knowledge

equivalence. For the case of the RSA system, given an encryption function $E(m) = m^e \bmod n$ and the problem of determining m, can it be shown that this problem is indistinguishable from the problem of taking roots in a ring given the knowledge $E(m)$, e, n ?

In a sense, cryptography and the information centered approach are considering the same problem. Cryptographic problems have exact solutions, but they can only be realised with a large amount of computing resources. By restricting the knowledge available, the designer wishes to make the *GCP* computationally difficult, even though there is an easy algorithm for solving it. The error associated with the information approach arises from an ill–defined problem description, ie. an the exact solution cannot be obtained with the knowledge given. The problem for the cryptographer is determining just what knowledge $N(t)$ the opponent has gathered, and whether this knowledge will be enough to break his cryptosystem.

A designer may further wish to construct a set $N^*(t)$ such that any knowledge included in $N^*(t)$ allows rapid decryption, and then attempt to analyse how difficult it is for an opponent to acquire any of the knowledge in $N^*(t)$. For the RSA system we see that $N^*(t)$–*RSA* = [d such that $ed \equiv 1 \bmod n, p$ or q where $n = pq$, O_2 or O_3 from §14.3, an algorithm for taking roots in rings, ...] but we do not know all the members of $N^*(t)$–*RSA*. At best, the designer can only guard against the possible attacks that he is aware of. Possibly $N^*(t)$–*RSA* is infinite or very large. Alternatively, we see that for the Rabin system $N^*(t)$–*RAB* = [{factoring a number that is the product of two primes}] has one representative element, since any knowledge that can be used to decrypt can be used to factor the modulus. Hence if we are to clearly define $N^*(t)$ for a system then we must use reductions from complexity theory to establish equivalent knowledge (cf. equivalent problems). Knowledge is just a form of oracle in this sense.

The reader may wonder where a study of approximating solutions, such as the information centered approach is, applies to cryptography. The first example is of course the cryptanalysis of the M–H knapsack system where the modular transformation pair (W, M) can be approximated by

(V, M') to form a superincreasing series. More generally an approximate solution is any partial solution which gives some knowledge about the actual solution to a problem, such as the parity of the message, or the first k bits of an n–bit message. Exact solutions can be obtained in exponential time but the cryptanalyst is searching for approximate solutions that can found in polynomial time. We note that many **NP**–complete problems can be approximately solved [GaJ].

So we conclude that certain facets of complexity theory and the information based approach are useful to cryptography. The designer wishes to select a specific easy instance of a generally intractable problem so that it appears to be an arbitrary instance of the intractable problem. The only real certainty about an intractable problem is that the number of possible solutions grows exponentially in the size of the problem.

Are reductions the only useful tool from complexity theory for the designer? Recall that in the first chapter we presented a short discussion on difficult problems, first in terms of non–linear functions and then with intractable problems.The appeal of complexity theory and the analysis of algorithms is that they provide a measure of difficulty, ie. a complexity analysis indicates how the work involved to solve a problem, $f(x)$, grows as a function of the problem size n. When a problem is posed, it is possible to translate the work $f(n)$ into a measure of actual time (ie. days, weeks etc.). We have seen that the best known algorithms for factoring a 200 digit number will take roughly 10^7 years to complete. The basic distinction between running times is polynomial and exponential. Even though some problems, such as factoring, the DLP, quadratic residuosity and computing roots, cannot be classified in a complexity class, secure cryptosystems can still be built using these problems. It is a question of security ratios and establishing resource bounds such as the one used in §7, and attempting to leave the cryptanalyst with a task that requires more resources than he can have at his disposal. In §7 we argued that the security ratio of a cryptosystem should be exponential, but it is not necessary. The security ratios for the RSA, ElGamal and Goldwasser–Micali system are subexponential, but still non–polynomial. But once a security ratio for a system

$s(n)$ is defined, the block length n can always be adjusted so that the work required to break the system is huge (recall the bound from §3). We see that the security ratio of the RSA system is

$$n^{c\sqrt{\log n \, \operatorname{loglog} n}}$$

which provides a comfortable margin of security when $n = 200$. We expect that progress in algorithms and hardware will decrease the time required to factor numbers, to solve the DLP and other problems. The Georgia factoring project is building a machine that will be able to factor a 100 digit number in less than 3 weeks[1]. But the designer can always raise the stakes. Say the running time for factoring was reduced so the *loglog n* term was removed from the exponent, then to keep the security ratio time favourable (ie. 10^7) the block size must be increased. But on the other hand, we are decreasing the speed of encryption and decryption as the block size increases. Speed is already the main problem with the RSA system. We must either look for functions that grow very rapidly or accept that transmission speeds will decrease in the future, as a response to better algorithms for intractable problems.

An average case anlalysis calculates the expected difficulty of a problem and this analysis also seems inadequate for cryptography (cf. example 13.1). Shamir [SHA] has introduced the notion of the percentile complexity of a problem p, denoted as $C(n, \varepsilon)$, which states that $1/\varepsilon$ of all instances of p can be solved in $C(n)$ steps. Even if most problem instances are hard, an algorithm that solves a small fraction of instances deterministically can be used to solve all instances probabilistically. For example, let AL be an algorithm that solves $m^e \bmod n$ for one percent of the possible messages m. Let $x = m^e \bmod n$ and choose a random sequence of messages $m_1, m_2, m_3 \ldots m_n$. Compute $x_i = (xm_i^e) = (xm_i)^e \bmod n$. By repeatedly using the m_i we will eventually arrive at a xm_i for which AL works; the message is then recovered by a division. Rabin [RAB2] has defined another type of complexity called universal intractability which states that a problem is

[1]"The Georgia Factoring Project", an address delivered by Carl Pomerance at the 17th Conference on Numerical Mathematics and Computation, University of Manitoba , Winnipeg, Canada, October 3, 1987.

difficult to solve with respect to algorithm *AL* if the number of possible solutions *AL* can check in polynomial time is negligible as compared to the possible number of key settings and problem instances. The ideal notion of intractability is one without exception.

Lastly we turn to two issues that have been largely neglected by this paper, namely cryptographic protocols and digital signatures. No matter how strong a cryptosystem is, a poor protocol may render it weak. We have seen that the common modulus protocol for the RSA system leads directly to the the factors of the modulus n. Also, imagine an opponent has intercepted a message $m = m_1, m_2, \ldots, m_n$ intended for user A, which is encrypted using the Goldwasser–Micali system, so that for each m_i either $m_i \in QR_n$ or $m_i \in QRN^{+1}{}_n$. The opponent then encrypts a meaningful message m^* to $c = c_1, c_2, \ldots, c_n$, and randomly selects an m_i and froms $b = b_1, b_2, \ldots, b_n$ as follows,

1. **for** $i \leftarrow 1$ **to** n **do**
2. **if** $c_i \in QR_n$ **then** $b_i \leftarrow c_i m_i \bmod n$
3. **else** $b_i \leftarrow c_i$;
4. **end_for.**

By sending $= b_1, b_2, \ldots, b_n$ to user A and waiting for a response the opponent can determine the residuosity of m_i. If $m_i \in QR_n$ then $c_i m_i \in QR_n$ by Lemma 10.6 and the original meaning of m^* will be accepted. If m_i is not a quadratic residue then the message b is not expected to be meaningful and A will ask for a retransmission.

In *New Directions in Cryptography*, Diffie and Hellman explain how two users, A and B, can communicate and provide signatures. Since then, these issues have become increasingly complicated. The first proposals for PKCs were the simplest and cleanest algorithms. A system is of little use if it authentication cannot be provided and forgeries prevented. The original solution

was for A to send $E_B(D_A(m))$ to authenticate the message, which B recovered by using A's public key. This solution assumes that E_A is a one way permutation, which is generally not true. In most cryptosystems only bit strings that are valid encryptions can be decrypted, and hence a signature system must be superimposed on the secrecy system. There have been several proposed solutions to this problem, including [RAB] [RAB2] [OSS] [GMY]. Each system is based on some problem that cannot be solved rapidly.

References

[AaM] Adleman, L., Manders, K., Reducibility, Randomness and Intractability, procs. 9th ACM STOC, 151-163 (1977).

[APR] Aldeman, L., Pomerance, C., Rumely, R., On distinguishing prime numbers from composite numbers, Annals of Math. (2) 117, 173–206 (1983).

[ADL1] Adleman, L., A subexponential algorithm for the discrete logarithm problem with applications to cryptography, procs. 20th IEEE FOCS, 55-60 (1979).

[AMM] Adleman, L., Manders, K., Miller, G., On taking roots in finite fields, procs. 18th IEEE FOCS, 175-178, (1977).

[BACH], see [WOLL].

[BAK] Baker, A., A concise introduction to the theory numbers, Cambridge University Press (1984).

[BERL] Berlekamp, R. E., Factoring polynomials over large fields, Math. Comp., 24, 713-735 (1970).

[BFMV] Blake, I. F., Fuji-Hara, R., Mullin, R. C., Vanstone, S. A., Computing logarithms in fields of characteristic 2, SIAM J. Alg. Disc. Methods, 5, 276-285 (1984).

[BM] Blum, M., Micali, S., On generating cryptographically secure sequences of psuedorandom bits, procs. 23rd IEEE FOCS, 112–117 (1982).

[BaM] Blum, M., Micali, S., How to generate cryptographically strong sequences of pseudo random bits, procs. 23rd IEEE FOCS, 112–117 (1982).

[BBS] Blum. L., Blum, M., Shub, M., Comparison of two simple psuedorandom number generators, Advances in Cryptology, procs. Crypto' 82, 61–78 (1982).

[BORO] Borodin, A., Structured vs. General models in computational complexity, In "Logic and Algorithmic" Monographic #30 de L'Enseignement Mathematipue Universite de Geneve (1982).

[BOUN] Bouniakowsky, V., Bull. Ac. Sc. St. Petersburg, 14, 356-375 (1870).

[BRASS1] Brassard, G., Relativized Crtpyography, procs. 23rd IEEE FOCS, 383-391 (1979).

[BR1] Brickell, E. F., Solving low density knapsack systems, Advances in Cryptology, procs. Crypto' 83, Plenum, 25–37 (1983).

[BR2] _____, The cryptanalysis of knapsack systems, preprint.

[BRNP] Brassard, G., A note on the complexity of cryptography, IEEE Transactions on Information Theory, vol. 25, no.2, (1979).

[CaS] Caron, T. R., Silverman, R. D., Parallel implmentation of the quadratic sieve, Papers and Abstracts, Crypto' 86, article 4, (1986).

[CaR] Chor, B., Rivest, R., A Knapsack Type Public key Cryptosystem Based on Arithmetic in Finite Fields, Advances in Cryptology, procs. Crypto' 84, 54–65 (1985).

[CHV] Chvatal, V., Determining the stability number of a graph, SIAM J. of Comp., no.6, 643-662 (1977).

[CaL] Cohen, H., Lenstra H. W. jr., Primality testing and Jacobi sums, report 82-18, Uni. of. Amsterdam, Dept. Math (1982).

[CaL] Cohen, H., Lenstra, A. K., Implementation of a new primality test, Math. of Computation, vol. 48, no. 177, Jan., 103–121 (1987).

[COP] Coppersmith, D., Evaluating logarithms in $GF(2^n)$, procs. 16th ACM STOC, 201-207 (1984).

[COS] Coppersmith, D., Odlyzko, A. M., Schroeppel, R., Discrete Logarithms in GF(p), Algorithmica 1, 1-15 (1986).

[COOK] Cook, S. A., The complexity of theorem proving procedures, procs. 3rd annual ACM STOC, p151-158 (1971).

[COOT] _____, An overview of computational complexity, Turing Award lecture, CACM, vol. 26, no.6, 401-408 (June 1983).

[DeL] Delaurentis, J. M., A further weakness of the common modulus protocol for the RSA cryptoalgorithm, Cryptologia, vol. 8, no. 3, 253-259 (July 1984).

[DVG] Desmedt, Y. G., Vanderwalle, J., P., Govaerts, R., J., A critical analysis of the security of knapsack public key algorithms, IEEE Transactions on Information Theory, vol. IT–30, no. 4, July, 601–611 (1984).

[DIF] Diffie, W., Hellman M. E., New Directions in Cryptography, IEEE Transactions on Information Theory IT-22, 622-654 (1976).

[HELL1] Diffie, W., Hellman, M. E., Exhaustive Cryptanalysis of the Data Encryption Standard, Computers, vol. 10, no.6, 74–84 (1977).

[DIX] Dixon, J. D., Factorisation and primality tests, AMM, vol. 91, no. 6 June–July, 333–352 (1984).

[EL] Eier, R., Lagger, H., Trapdoors in Knapsack Cryptosystems, Lecture Notes in Computer Science, vol. 149, Springer–Verlag, New York, 316–322 (1983).

[ELGA] ElGamal, T., A public key cryptosystem based on discrete logarithms, IEEE Transactions on Information Theory, vol. 31, no. 4, July (1985).

[EKW] Evans, A. jr., Kantrowitz, W., Weiss, E., A User Authentication Scheme not requiring security in the computer, CACM, vol. 17, 437–442, Aug. (1974).

[EaY] Even, S., Yacobi, Y., Cryptocomplexity and NP–completeness, 7th ICALP lecture notes in Com. Sci., vol. 85, 195–207, Springer Verlag Berlin, New York (1980).

[FNS] Reistel, H., Notz. W. A., Smith, J. L., Some cryptographic techniques for machine to machine communications, procs. IEEE vol. 63, 1545–1554, Nov. (1975).

[FINN] Finn, J., Finding square roots modulo a prime, prime testing and factoring, Tech. Report 295, Princeton University, Dept. Elect. Eng. and Com. Sci., Feb. (1981).

[GAL] Galil, Z., On the resolution of clauses with bound size, SIAM J. of Comp. no.6, 444-459 (1977).

[GIR] Girsdansky, M. B., Cryptology, the computer and data privacy, Computer Automation, vol. 21, no. 4, 12–19, April (1972).

[GlaM] Glee, V. Minty G., How good is the simplex algorithm, in Inequalities 3, O. Shisha ed., NY Academic press, 159-175 (1972).

[GMT] Goldwasser, S., Micali, S., Tong, P., Why and How to establish a private code on a punlic network, procs. 23rd IEEE FOCS, 134–144 (1982).

[GMY] Goldwasser, S., Micali, S., Yao, A., Strong Signature Systems, procs. 15th ACM STOC, 413–439 (1983).

[GaM] Goldwasser, S., Micali, S., Probabilistic Encryption, JCSS 28, 270-299 (1984).

[HaW] Hardy, G. H., Wright, E. M., An introduction to the theory of numbers, 5th edition, Oxford University Press (1979).

[HRU] Harrison, M. A., Ruzzo, W. L., Ullman, J. D., Protection in Operating Systems, CACM, vol. 19, no.8, 461–471 (1976).

[HART] Hartmanis, J., Feasible Computations and Provable Complexity Properties, CBMS–NSF Regional Conference Series in Applies Math., SIAM (1978).

[HELL] Hellman, M. E., Merkle, R., Schroeppel, R., Washington, L., Diffie, W., Pohlig, S. and Scheitzer, P., Results of an initial attempt to cryptanalyse the NBS Data Encryption Standard, Information Systems Lab., Dept. of Electrical Engineering, Stanford University.

[HELL3] Hellman, M. E., Reyneri, J. M., Fast computation of discrete logarithms in GF(p), Advances in Cryptology, proc. Crypto' 82, 3-13 (1982).

[HaL] Hillier, F. S, Lieberman, G. J., Operations Research, San Francisco Holden Day (1974).

[HaU] Hopcroft, J. E., Ullman, J. D., Introduction to Automata Theory, Languages and Computation, Addison Wesley (1979).

[HaS] Horowitz, E., Sahni, S., Computing partitions with applications to the knapsack problem, JACM, vol. 21, no. 2, April, 277–292 (1974).

[JOHN] Johnson, D. S., Near Optimal Bin Packing algorithms, Doctoral Thesis, Dept. Math., MIT, Cambridge, MA, 1973.

[KAHN] Kahn, D., The Codebreakers, Macmillan, NY, 1967.

[KARP] Karp, R. M., Reducibility among combinatorial problems, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher eds., Plenum, NY (1972).

[KATA] Karp, R. M., Combinatorics, Complexity, and Randomness, Turing Award Lecture, CACM, vol. 29, #2, 98-109 (1986).

[KEY] Keyes, P. W., Physical limits in digital electronics, proceedings of IEEE, vol. 3, 740-767 (1975).

[KHAC] Khachuan, L. G., A polynomial algorithm in linear programming, Dolk. Akad SSSR, 224 (1979).

[KNU1] Knuth, D. E., Art of Computer Programming: Fundamental algorithms, vol. 1, Addison Wesley (1973).

[KNU2] _____, Art of Computer Programming: Seminumerical algorithms, vol. 2, Addison Wesley (1973).

[KON] Konheim, A., Cryptography: a primer, Wiley and sons (1981).

[KRAN] Kranakis, E., Primality and Cryptography, Wiley and Sons (1986).

[KRUS] Kruskal, J. B., On the shortest spanning subtree of a graph and the traveling salesman problem, procs. Amer. Math. Soc., 7, 48–50 (1956).

[LADN] Ladner, R. E., On the structure of polynomial time reducibility, JACM 22, **1** , 155–171 (1975).

[LLS] Ladner, R. E., Lynch, N. A., Selman, A. L., A comparison of polynomial time reducibilities, Theo. Com. Sci., **1**, 103–123 (1975).

[LaG] Lagarias, J., Odlyzko, A. M., Solving low density subset sum problems, procs. 24th IEEE FOCS, 1–10 (1983).

[LEMP] Lempel, A., Cryptology in transition: a survey, Computing Surveys 11, 285–304, Dec. (1979).

[LEN] Lenstra, H. W. jr., Factoring and Elliptic curves, to appear.

[LLL] Lenstra, A. K., Lenstra, H., W., Lovasz, L., Factoring polynomials with rational coefficients, Mathematische Annalen 261, 515–534 (1982).

[LEN2] Lenstra, H. W., Integer Programming with a fixed number of variables, Math. of Operations Research, vol. 8, no. 4, Nov., 538–548 (1983).

[LaP] Lewis, H. R., Papidimitrou, C. H., Elements of the Theory of Computation, Prentice Hall (1981).

[LIPS] Lipson, J. D., Elements of Algebra and Algebraic Computing, Addison Wesley Reading, MA., (1981).

[LIU] Liu, C. L., Elements of Discrete Mathematics, 2nd edition, McGraw Hill (1985).

[LONG] Long, D., Random equivalences of factoring and computing orders, Tech. Report 284, Princeton University, Dept. Elect. Eng. and Com. Sci., April (1981).

[McC] McCurley, K. S., A Key Distribution Scheme Equivalent to Factoring, Advances in Cryptology, procs. Crypto' 87, preprint.

[MEL] Melhorn, K., Data Structures and Algorithms 2: Graph Algorithms and NP–completeness, Springer–Verlag, Berlin (1984).

[MER1] Merkle, R., Secure Communication over insecure channels, CACM, vol. 21, no. 4, 294-299 (April 1978).

[MER2] Merkle, R., Hellman, M. E., Hiding Information and Signatures in Trapdoor Knapsacks, IEEE Transactions on Information Theory IT-24 (1978).

[MER3] Merkle, R., Secrecy, Authentication and Public Key Systems, PhD dissertation, Dept. of Elec. Eng., Stanford Uni. June, 1979.

[MILL] Miller, G., L., Reimann's Hypothesis and Tests for Primality, JCSS, 300-317 (1976).

[MRS] Micali, S., Rackoff, C., Sloan, R., The notion of security for probabilistic cryptosystems, Advances in Cryptology, procs. Crypto' 86, p31–1 to 31–12 (1986).

[MaB] Morrison, M. A., Brillhart, J., A method of factoring and the factorisation of F_7, Math. Comp. 29, 183-205 (1975).

[ODLY] Odlyzko, A. M., Discrete logarithms in finite fields and their cryptographic significance, Advances in Cryptology, proc. Eurocrypt, 225-314 (1984).

[OaS] Okamoto, T., Shivaishi, A., A fast signature scheme based on quadratic inequalities, procs. 1985 Symposium on Secirity and Privacy, April, Oakland, CA. (1985).

[OSS] Ong, H., Schnorr, C., P., An efficient signature scheme based on quadartic equations, procs. 16th ACM STOC, 208–216 (1984).

[PER] Peralto, R., Simultaneous security of bits in the discrete logarithm problem, Advances in Cryptology, procs. Eurocrypt, 62–72 (1985).

[PaS] Pohl, I., Shaw, A., The Nature of Computation: an introduction to Computer Science, Computer Science Press (1981).

[PaH] Pohlig, S. C., Hellman, M. E., An improved algorithm for computing logarithms over GF(p) amd its cryptographic significance, IEEE Transactions on Information Theory, IT-24, no.1, 106-110 (Jan. 1978).

[POM2] Pomerance, C., Analysis and computation of some factoring algorithms, in Number Theory and Computers, ed. by H. W. Lenstra jr., R. Tijdeman, Math. Centrum Tracts, no. 154, part1, Amsterdam , 189-139, (1982).

[POM1] Pomerance, C., Wagstaff Jr., S., Implementation of the continued fraction factoring algorithm, Congressus Numerantium, 37, 99-118 (1983).

[PST] Pomerance, C., Smith, J., Tuler, R., A pipeline architecture for factoring large integers with the quadratic sieve method, Papers and Abstracts, Crypto' 86, article 3, (1986).

[POST] Post, E. L., Recursively enumerable sets of positive integers and their decision problems, Bull. AMS, 50, 284-316 (1944).

[PRAT] Pratt, V., Every prime has a succinct certificate, SIAM J. Comp., 4, 214-220 (1975).

[PUR] Purdy, G. B., A high security log–in procedure, CACM, vol. 17, 442–445, Aug. (1974).

[RATM] Rabin, M. O., Complexity of Computations, Turing Award Lecture, CACM, vol. 20, no.9, 625-633 (Sept. 1977).

[RAB2] _____, Digitalised Signatures, in Foundations of Secure Computing, eds. R. Demillo, A. Jones, R. Lipton, Academis Press, 155–166 (1978).

[RAB] _____, Digitalised Signatures and Public Key functions as intractable as factoring, MIT lab. for Comp. Sci., TR 22, Jan. (1979).

[RAPT] _____, Probabilistic algorithms for testing primality, J. Number Theory, 12, 128-138 (1980).

[RSA] Rivest, R. L., Shamir, A., Adleman, L., A method for obtaining digital signatures and public key cryptosystems, CACM, vol. 21, no. 2, 120-126 (Feb. 1978).

[ROS] Rosen, K., Elementary Number Theory and its applications, Addison Wesley (1984).

[SAHN] Sahni, S., Approximation algorithms for the 0/1 knapsack problem, JACM 22, 115–124, 1975.

[SANT] Santos, E. S., Probabilistic Turing machines and computability, procs. AMS 22, 704-710 (1969).

[SCHA] Schanning, B. P., Data encryption with public key distribution, EASCON Conf. Rec., Wash. DC, 653-660 (Oct. 1979).

[SCHW] Schwartz, J. T., Probabilistic algorithms for verification of polynomial identities, JACM, vol. 27, 4, 701-717 (Oct. 1980).

[SCLN] Schnorr, C. P., Lenstra, H. W. jr., A Monte Carlo factoring algorithm with linear storage, Math. Comp. 43, 289-311 (1984).

[SCH] Schroeppel, R., Shamir, A., A $T.S^2 = \Theta(2^n)$ time space tradeoff for certain NP-complete problems, procs. 20th IEEE FOCS, 328-336 (1979).

[SHA2] Shamir, A., On the cryptocomplexity of knapsack system, procs. 11th ACM STOC, 118–128 (1979).

[SHA] _____, On the generation of cryptographically strong sequences of pseudorandom numbers, procs. 8th ICALP, 544–550 (1981).

[SHA3] _____, A polynomial time algorithm for breaking the basic Merkle–Hellman cryptosystem, IEEE Transactions on Information Theory, vol. 30, no. 5, Sept., 699–704 (1984).

[SHK] Shanks, D., Five Number Theoretic Algorithms, Congressus Numeratium VII, procs. 2nd Manitoba Con. on Numerical Maths., Oct. 5–7, 51–70 (1972).

[SHAN1] Shannon, C. E., Communication Theory of Secrecy Systems, Bell Sys. J., 28, Oct., 656-715 (1949).

[SIMM] Simmons, G. S., Status Report on factoring (at Sandia National Lab.), Advances in Cryptology, procs. of Eurocrypt, 183-215 (1984).

[SaS] Solovay, R., Strassen V., A fast monte carlo test for primality, SAIM J. Comp., 6, 84-85 (1977).

[TaW] Traub, J., F., Wozniakowski, Information and Complexity, Advances on Computation, vol.23, 35–92 (1980).

[VaV] Vazirani, U., Vazirani, V., Efficient and Secure pseudorandom number generation, procs. 25th IEEE FOCS, 458–463 (1984).

[WAGN] Wagner, N. R., Searching for Public Key Cryptosystems, IEEE Symposium on Security and Privacy, 91-98 (1984).

[WILF] Wilf, H. S., Algorithms and Complexity, Prentice Hall, 211–215 (1986).

[WILL1] Williams, H., C., Primality Testing on a computer, Ars Combinatoria, 126-185 (1978).

[WIL2] _____, A modification to the RSA public key encryption procedure, IEEE Transactions on Information Theory, vol. IT-26, no.6, 726-729 (1980).

[WIL3] _____, An M^3 Public–Key Encryption Scheme, Advances in Cryptology, procs. Crypto' 85, 358–368 (1985).

[WIL4] _____, Some Public–Key Crypto–Functions as Intractable as Factorisation, Cryptologia, vol. 9, no. 3, 223-237 (1985).

[WOLL] Woll, H., Reductions among Number Theoretic Problems, Info. and Comp. 72, 167–179 (1987).

[YAO] Yao, A., Theory and applications of Trapdoor Functions, FOCS, 80–91 (1982).

[YP] Yiu, K., Peterson K., A single chip VLSI implementation of the discrete exponential public key distribution system, proc. GLOBCOM-82 IEEE, 173-179 (1982).

[ZACH] Zachos, S., Robustness of probabilistic computational classes under definitional perturbation, Info. and Control, 54, 143-154 (1982).