...han Demirel University
Engineering Faculty

Arslanov M.Z.

A course in Information Theory

Almaty – 2009

Süleyman Demirel University
Engineering Faculty

Arslanov M.Z.

A course in Information Theory

Almaty – 2009

ББК 32.81
А 80

Рекомендовано к печати Ученым Советом Университета имени Сулеймана Демиреля.

Arslanov M.Z...
А 80 A course in information theory. – Алматы, 2009 - 152с.

ISBN 9965-792-31-3

ББК 32.81

А 1404000000
00(05)-09

ISBN №

Contents

Part I

Shannon Information Theory

CHAPTER 1

Entropy, Relative Entropy and Mutual Information

This chapter introduces most of the basic definitions required for the subsequent development of the theory. It is irresistible to play with their relationships and interpretations, taking faith in their later utility. After defining entropy and mutual information, we establish chain rules, the non-negativity of mutual information, the data processing inequality. The concept of information is too broad to be captured completely by a single definition. However, for any probability distribution, we define a quantity called the entropy, which has many properties that agree with the intuitive notion of what a measure of information should be. This notion is extended to define mutual information, which is a measure of the amount of information one random variable contains about another. Entropy then becomes the self-information of a random variable. Mutual information is a special case of a more general quantity called relative entropy, which is a measure of the distance between two probability distributions. All these quantities are closely related and share a number of simple properties. We derive some of these properties in this chapter. In later chapters, we show how these quantities arise as natural answers to a number of questions in communication, statistics, complexity. That will be the ultimate test of the value of these definitions.

ENTROPY. We will first introduce the concept of entropy, which is a measure of uncertainty of a random variable. Let X be a discrete random variable with alphabet $\mathcal{X}$ and probability mass function $p(x) = Pr\{X = x\}, x \in \mathcal{X}$. We denote the probability mass function by $p(x)$ rather than $p_X(x)$ for convenience. Thus, $p(x)$ and $p(y)$ refer to two different random variables, and are in fact different probability mass functions, $p_X(x)$ and $p_Y(y)$ respectively.

DEFINITION 1.1. The entropy $H(X)$ of a discrete random variable X is defined by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{1}$$

We also write $H(p)$ for the above quantity. The log is to the base 2 and entropy is expressed in bits. For example, the entropy of a fair coin toss is 1 bit. We will use the convention that $0 \log 0 = 0$, which is easily justified by continuity since $x \log x \to 0$ as $x \to 0$. Thus adding terms of zero probability does not change the entropy. If the base of the logarithm is b, we will denote the entropy as $H_b(X)$. If the base of the logarithm is e, then the entropy is measured in nuts. Unless otherwise specified, we will take all logarithms to base 2, and hence all the entropies will be measured in bits. Note that entropy is a functional of the distribution of X. It does not depend on the actual values taken by the random variable X, but only on the probabilities. We shall denote expectation by E. Thus if $X \sim p(x)$, then the expected value of the random variable $g(X)$ is written

$$E_p g(X) = \sum_{x \in \mathcal{X}} g(x)p(x) \tag{2}$$

or more simply as $E_g(X)$ when the probability mass function is understood from the context. We shall take a peculiar interest in the eerily self-referential expectation of $g(X)$ under $p(x)$ when $g(X) = \log \frac{1}{p(X)}$.

Remark: The entropy of X can also be interpreted as the expected value of $\log \frac{1}{p(X)}$, where X is drawn according to probability-mass function $p(x)$. Thus

$$H(X) = E_p \log \frac{1}{p(X)} \tag{3}$$

This definition of entropy is related to the definition of entropy in thermodynamics; some of the connections will be explored later. It is possible to derive the definition of entropy axiomatically by defining certain properties that the entropy of a random variable must satisfy. This approach is illustrated in a problem at the end of the chapter. We will not use the axiomatic approach to justify the definition of entropy; instead, we will show that it arises as the answer to a number of natural questions such as "What is the average length of the shortest description of the random variable. ?" First, we derive some immediate consequences of the definition.

LEMMA 1.1. $H(X) \geq 0$.

PROOF. $0 \leq p(x) \leq 1$ implies $\log \frac{1}{p(x)} \geq 0$. □

LEMMA 1.2. $H_b(X) = (\log_b a) H_a(X)$.

PROOF. $\log_b p = \log_b a \log_a p$. □

The second property of entropy enables us to change the base of the logarithm in the definition. Entropy can be changed from one base to another by multiplying by the appropriate factor.

EXAMPLE 1.1. Let

$$X = \begin{cases} 1 & \text{with probability } p, \\ 0 & \text{with probability } 1-p. \end{cases} \tag{4}$$

Then

$$H(X) = -p \log p - (1-p) \log(1-p) \stackrel{\text{def}}{=} H(p) \tag{5}$$

In particular, $H(X) = 1$ bit when $p = 1/2$. The graph of the function $H(P)$ is shown in Figure 1. The figure illustrates some of the basic properties of entropy. It is a concave function of the distribution and equals 0 when $p = 0$ or 1. This makes sense, because when $p = 0$ or 1, the variable is not random and there is no uncertainty. Similarly, the uncertainty is maximum when $p = \frac{1}{2}$, which also corresponds to the maximum value of the entropy.

EXAMPLE 1.2. Let

$$X = \begin{cases} a & \text{with probability } 1/2, \\ b & \text{with probability } 1/4, \\ c & \text{with probability } 1/8, \\ d & \text{with probability } 1/8. \end{cases} \tag{6}$$

The entropy of X is

$$H(X) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{8} - \frac{1}{8}\log\frac{1}{8} = \frac{7}{4}\text{bits} \tag{7}$$

Suppose we wish to determine the value of X with the minimum number of binary questions. An efficient first question is "Is $X = a$?" This splits the probability in half. If the answer to the first question is no, then the second question can be "Is $X = b$?" The third question can be "Is $X = c$?" The resulting expected number of binary questions required is 1.75. This turns out to be the minimum expected number of binary questions required to determine the value of X. Later we show that the minimum expected number of binary questions required to determine X lies between $H(X)$ and $H(X) + 1$.

JOINT ENTROPY AND CONDITIONAL ENTROPY.

We have defined the entropy of a single random variable in the previous section. We now extend the definition to a pair of random variables. There is nothing really new in this definition because (X, Y) can be considered to be a single vector-valued random variable.

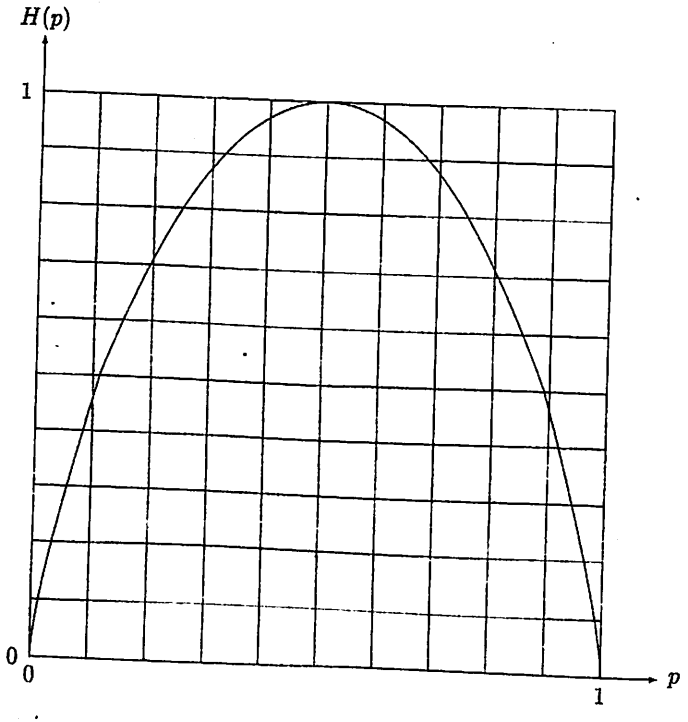

FIGURE 1. $H(p)$ versus p.

DEFINITION 1.2. The joint entropy $H(X,Y)$ of a pair of discrete random variables (X,Y) with a joint distribution $p(x,y)$ is defined as

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y), \tag{8}$$

which can also be expressed as

$$H(X,Y) = -E\log p(X,Y). \tag{9}$$

We also define the conditional entropy of a random variable given another as the expected value of the entropies of the conditional distributions, averaged over the conditioning random variable.

DEFINITION 1.3. If $(X,Y) \sim p(x,y)$, then the conditional entropy $H(Y|X)$ is defined as

$$H(Y|X) = \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \tag{10}$$

$$= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \tag{11}$$

$$= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \tag{12}$$

$$= E_{p(x,y)}\log p(Y|X). \tag{13}$$

The naturalness of the definition of joint entropy and conditional entropy is exhibited by the fact that the entropy of a pair of random variables is the entropy of one plus the conditional entropy of the other. This is proved in the following theorem.

THEOREM 1.3. *(Chain rule):*

$$H(X,Y) = H(X) + H(Y|X). \tag{14}$$

PROOF.

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) \tag{15}$$

$$= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x)p(y|x) \tag{16}$$

$$= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \tag{17}$$

$$= -\sum_{x\in\mathcal{X}} p(x)\log p(x)p(y|x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \tag{18}$$

$$= H(X) + H(Y|X). \tag{19}$$

Equivalently, we can write

$$\log p(X,Y) = \log p(X) + \log p(Y|X) \tag{20}$$

and take the expectation of both sides of the equation to obtain the theorem. □

COROLLARY 1.4. *H(X, Y—Z) = H(X—Z) + H(Y—X, Z).*

PROOF. The proof follows along the same lines as the theorem. □

EXAMPLE 1.3. Let (X, Y) have the following joint distribution:

$X\backslash Y$	1	2	3	4
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
4	$\frac{1}{4}$	0	0	0

The marginal distribution of X is $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ and the marginal distribution of Y is $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$, and hence $H(X) = 7/4$ bits and $H(Y) = 2$ bits. Also,

$$H(X|Y) = \sum_{i=1}^{4} p(Y=i)H(X|Y=i) =$$

$$= \frac{1}{4}H(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}) + \frac{1}{4}H(\frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{8})$$

$$+\frac{1}{4}H(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}) + \frac{1}{4}H(1,0,0,0) =$$

$$= \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times \frac{7}{4} + \frac{1}{4} \times 2 + \frac{1}{4} \times 0 = 11/8 \text{ bits.}$$

Similarly $H(Y|X) = 13/8$ bits and $H(X,Y) = 27/8$ bits.

Remark: Note that $H(Y|X) \neq H(X|Y)$. However, $H(X) - H(X|Y) = H(Y) - H(Y|X)$, a property that we shall exploit later.

RELATIONSHIP BETWEEN ENTROPY AND MUTUAL INFORMATION. We can rewrite the definition of mutual information $I(X,Y)$ as

$$I(X,Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} =$$

$$\sum_{x,y} p(x,y) \log \frac{p(x|y)}{p(x)} =$$

$$-\sum_{x,y} p(x,y) \log p(x) + \sum_{x,y} p(x,y) \log p(x|y) =$$

$$-\sum_{x,y} p(x,y) \log p(x) - (-\sum_{x,y} p(x,y) \log p(x|y)) =$$

$$H(X) - H(X|Y).$$

Thus the mutual information $I(X,Y)$ is the reduction in the uncertainty of X due to the knowledge of Y. By symmetry, it also follows that

$$I(X,Y) = H(Y) - H(Y|X).$$

Thus X says as much about Y as Y says about X. Since $H(X,Y) = H(X) + H(Y|X)$, we have

$$I(X,Y) = H(X) + H(Y) - H(X,Y).$$

Finally, we note that

$$I(X,X) = H(X) - H(X|X) = H(X).$$

Thus the mutual information of a random variable with itself is the entropy of the random variable. This is the reason that entropy is sometimes referred to as *self-information*. Collecting these results, we have the following theorem.

THEOREM 1.5. *(Mutual information and entropy):*

$$I(X,Y) = H(X) - H(X|Y),$$

$$I(X,Y) = H(Y) - H(Y|X),$$

$$I(X,Y) = H(X) + H(Y) - H(X,Y),$$

$$I(X,Y) = I(Y,X),$$

$$I(X,X) = H(X).$$

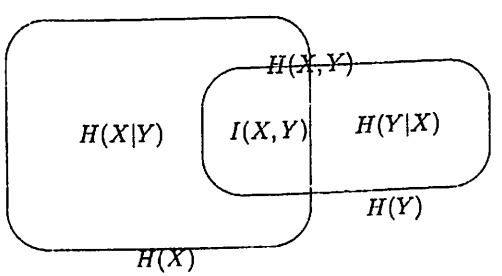

FIGURE 2. Relationship between entropy and mutual information.

The relationship between $H(X), H(Y), H(X,Y), H(X|Y), H(Y|X)$ and $I(X,Y)$ is expressed in a Venn diagram (Figure 2). Notice that the mutual information $I(X,Y)$ corresponds to the intersection of the information in X with the information in Y.

EXAMPLE 1.4. For the joint distribution of example 1.3, it is easy to calculate the mutual information $I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = 0.375$ bits.

Properties of information measures. Remember next fact of mathematical analysis:

$$\ln x \le x - 1,\ \ln x = x - 1 \Leftrightarrow x = 1.$$

THEOREM 1.6. *(Non-negativity of mutual information): For any two random variables X, Y,*

$$I(X,Y) \ge 0$$

with equality if and only if X and Y are independent.

PROOF.

$$-I(X,Y) = H(X|Y) - H(X) =$$

$$= \sum_{XY} p(x,y) \log p(x|y) + \sum_{X} p(x) \log p(x) =$$

$$= \sum_{XY} p(x,y) \log \frac{p(x)}{p(x|y)} \le$$

$$\le \log e \sum p(x,y) \left[\frac{p(x)}{p(x|y)} - 1 \right] =$$

$$= \log e \left[\sum p(x)p(y) - \sum p(x,y) \right] = 0.$$

□

THEOREM 1.7. *$H(X) \le \log |\mathcal{X}|$, where $|\mathcal{X}|$ denotes the number of elements in the range of X, with equality if and only if X has a uniform distribution over $\mathcal{X}$.*

PROOF.

$$H(X - \log |\mathcal{X}|) =$$

$$= -\sum_{x \in \mathcal{X}} p(x) \log p(x) - \log |\mathcal{X}| \sum_{\mathcal{X}} p(x) =$$

$$= \log e \sum_{x \in \mathcal{X}} p(x) \ln \frac{1}{p(x)|\mathcal{X}|} \le$$

$$\le \log e \left[\sum \frac{1}{|\mathcal{X}|} - \sum p(x) \right] = 0.$$

□

THEOREM 1.8. *(Conditioning reduces entropy):*

$$H(X|Y) \le H(X)$$

with equality if and only if X and Y are independent.

PROOF. $0 \le I(X,Y) = H(X) - H(X|Y)$. □

Intuitively, the theorem says that knowing another random variable Y can only reduce the uncertainty in X. Note that this is true only on the average. Specifically, $H(X|Y=y)$ may be greater than or less than or equal to $H(X)$, but on the average $H(X|Y) = \sum_y p(y) H(X|Y=y) \le H(X)$. For example, in a court case, specific new evidence might increase uncertainty, but on the average evidence decreases uncertainty.

EXAMPLE 1.5. Let (X,Y) have the following joint distribution:

$X \backslash Y$	1	2
1	0	$\frac{3}{4}$
2	$\frac{1}{8}$	$\frac{1}{8}$

Then $H(X) = H(\frac{1}{8}, \frac{7}{8}) = 0.544$ bits, $H(X|Y=1) = 0$ bits and $H(X|Y=2) = 1$ bit. We calculate $H(X|Y) = \frac{3}{4} H(X|Y=1) + \frac{1}{4} H(X|Y=2) = 0.25$ bits. Thus the uncertainty in X is increased if $Y=2$ is observed and decreased if $Y=1$ is observed, but uncertainty decreases on the average.

THEOREM 1.9. *(Independence bound on entropy): Let $X_1, X_2, ..., X_n$, be drawn according to $p(x_1, x_2, ..., x_n)$. Then*

$$H(X_1, X_2, ..., X_n) \le \sum_{i=1}^{n} H(X_i)$$

with equality if and only if the X_i are independent.

PROOF. By the chain rule for entropies,

$$H(X_1, X_2, ..., X_n) \le \sum_{i=1}^{n} H(X_i | X_{i+1}, ..., X_n) \le$$

$$\le \sum_{i=1}^{n} H(X_i)$$

where the inequality follows directly from the previous theorem. We have equality if and only if X_i is independent of $X_{i-1}, ..., X_1$ for all i, i.e., if and only if the X_i's are independent. □

PROBLEMS FOR CHAPTER

PROBLEM 1.1. *Coin flips. A fair coin is flipped until the first head occurs. Let X denote the number of flips required.*

(a) Find the entropy $H(X)$ in bits. The following expressions may be useful:

$$\sum_{n=1}^{\infty} r^n = \frac{r}{1-r}, \sum_{n=1}^{\infty} nr^n = \frac{r}{(1-r)^2}$$

(b) A random variable X is drawn according to this distribution. Find an "efficient" sequence of yes-no questions of the form, "Is X contained in the set S?" Compare $H(X)$ to the expected number of questions required to determine X.

PROBLEM 1.2. *Entropy of functions. Let X be a random variable taking on a finite number of values. What is the (general) inequality relationship of $H(X)$ and $H(Y)$ if*

(a) $Y = 2^X$?

(b) $Y = \cos X$?

PROBLEM 1.3. *Minimum entropy. What is the minimum value of $Np_{,,} \cdot \cdot \cdot , p, \dot{\iota} = H(p)$ as p ranges over the set of n-dimensional probability vectors? Find all p's which achieve this minimum.*

PROBLEM 1.4. *Zero conditional entropy. Show that if $H(Y|X) = 0$, then Y is a function of X, i.e., for all x with $p(x) > 0$, there is only one possible value of y with $p(x,y) > 0$.*

PROBLEM 1.5. *Infinite entropy. This problem shows that the entropy of a discrete random variable can be infinite. Let $A = \sum_{n=2}^{\infty} \frac{1}{n \log^2 n}$. (It is easy to show that A is finite.) Show that the integer-valued random variable X defined by $Pr(X = n) = \frac{1}{An \log^2 n}$ for $n = 2,3,\ldots$ has $H(X) = \infty$.*

PROBLEM 1.6. *Average entropy. Let $H(p) = -p \log_2 p - (1 - p) \log_2(l - p)$ be the binary entropy function.*

(a) Evaluate $H(l/4)$ using the fact that $\log_2 3 = 1.585$. Hint: Consider an experiment with four equally likely outcomes, one of which is more interesting than the others.

(b) Calculate the average entropy $H(p)$ when the probability p is chosen uniformly in the range $0 \leq p \leq 1$.

(c) (Optional) Calculate the average entropy $H(p_1, p_2, p_3)$ where (p_1, p_2, p_3) is a uniformly distributed probability vector. Generalize to dimension n.

PROBLEM 1.7. *Coin weighing. Suppose one has n coins, among which there may or may not be one counterfeit coin. If there is a counterfeit coin, it may be either heavier or lighter than the other coins. The coins are to be weighed by a balance.*

(a) Find an upper bound on the number of coins n so that k weighings will find the counterfeit coin (if any) and correctly declare it to be heavier or lighter.

(b) (Difficult) What is the coin weighing strategy for $k = 3$ weighings and 12 coins?

PROBLEM 1.8. *Drawing with and without replacement. An urn contains r red, w white, and b black balls. Which has higher entropy, drawing $k \geq 2$ balls from the urn with replacement or without replacement? Set it up and show why. (There is both a hard way and a relatively simple way to do this.)*

PROBLEM 1.9. *A metric. A function $\rho(x,y)$ is a metric if for all x, y,*

- $\rho(x,y) \geq 0$
- $\rho(x,y) = \rho(y,x)$
- $\rho(x,y) = 0$ *if and only if* $x = y$
- $\rho(x,y) + \rho(y,z) \geq \rho(x,z)$

(a) Show that $\rho(X,Y) = H(X|Y) + H(Y|X)$ satisfies the first, second and fourth properties above. If we say that $X = Y$ if there is a one-to-one function mapping X to Y, then the third property is also satisfied, and $\rho(X,Y)$ is a metric.

(b) Verify that $\rho(X,Y)$ can also be expressed as

$$\begin{aligned}\rho(X,Y) &= H(X) + H(Y) - 2H(X,Y) \\ &= H(X,Y) - I(X,Y) \\ &= 2H(X,Y) - H(X) - H(Y)\end{aligned}$$

PROBLEM 1.10. *Example of joint entropy. Let $p(x,y)$ be given by*

$X\backslash Y$	0	1
0	$\frac{1}{3}$	$\frac{1}{3}$
1	0	$\frac{1}{3}$

Find

(a) $H(X), H(Y)$.

(b) $H(X|Y), H(Y|X)$.

(c) $H(X,Y)$.

(d) $H(Y) - H(Y|X)$.

(e) $I(X,Y)$.

(f) Draw a Venn diagram for the quantities in (a) through (e).

PROBLEM 1.11. *Entropy of a sum. Let X and Y be random variables that take on values $x_1, x_2, ..., x_r$ and $y_1, y_2, ..., y_s$ respectively. Let $Z = X + Y$.*

(a) Show that $H(Z|X) = H(Y|X)$. Argue that if X, Y are independent, then $H(Y) \leq H(Z)$ and $H(X) \leq H(Z)$. Thus the addition of independent random variables adds uncertainty.

(b) Give an example (of necessarily dependent random variables) in which $H(X) > H(Z)$ and $H(Y) > H(Z)$.

(c) Under what conditions does $H(Z) = H(X) + H(Y)$?

PROBLEM 1.12. *Entropy of a disjoint mixture. Let X_1 and X_2 be discrete random variables drawn according to probability mass functions $p_1(\cdot)$ and $p_2(\cdot)$ over the respective alphabets $\mathcal{X}_1 = \{1, 2, ..., m\}$ and $\mathcal{X}_2 = \{1, 2, ..., n\}$. Let*

$$X = \begin{cases} X_1 & \text{with probability } \alpha \\ X_2 & \text{with probability } 1 - \alpha. \end{cases}$$

(a) Find $H(X)$ in terms of $H(X_1)$ and $H(X_2)$ and α.

(b) Maximize over α to show that $2^{H(X)} \leq 2^{H(X_1)} + 2^{H(X_2)}$ and interpret using the notion that $2^{H(X)}$ is the effective alphabet size.

PROBLEM 1.13. *Maximum entropy. Find the probability mass function $p(x)$ that maximizes the entropy $H(X)$ of a non-negative integer-valued random variable X subject to the constraint*

$$EX = \sum_{n=0}^{\infty} np(n) = A$$

for a fixed value $A > 0$. Evaluate this maximum $H(X)$.

CHAPTER 2

Data Compression

We now put content in the definition of entropy by establishing the fundamental limit for the compression of information. Data compression can be achieved by assigning short descriptions to the most frequent outcomes of the data source and necessarily longer descriptions to the less frequent outcomes. For example, in Morse code, the most frequent symbol is represented by a single dot. In this chapter we find the shortest average description length of a random variable. We first define the notion of an instantaneous code and then prove the important Kraft inequality, which asserts that the exponentiated codeword length assignments must look like a probability mass function. Simple calculus then shows that the expected description length must be greater than or equal to the entropy, the first main result. Then Shannon's simple construction shows that the expected description length can achieve this bound asymptotically for repeated descriptions. This establishes the entropy as a natural measure of efficient description length. The famous Huffman coding procedure for finding minimum expected description length assignments is provided. Finally, we show that Huffman codes are competitively optimal and that it requires roughly H fair coin flips to generate a sample of a random variable having entropy H. Thus the entropy is the data compression limit as well as the number of bits needed in random number generation. And codes achieving H turn out to be optimal from many points of view.

EXAMPLES OF CODES.

DEFINITION 2.1. A source code C for a random variable X is a mapping from $\mathcal{X}$, the range of X, to $\mathcal{D}^*$, the set of finite length strings of symbols from a D-ary alphabet. Let $C(x)$ denote the codeword corresponding to x and let $l(x)$ denote the length of $C(x)$.

For example, $C(\text{Red}) = 00$, $C(\text{ Blue}) = 11$ is a source code for $\mathcal{X} = \{\text{Red, Blue}\}$ with alphabet $\mathcal{D} = \{0, 1\}$.

DEFINITION 2.2. The expected length $L(C)$ of a source code $C(x)$ for a random variable X with probability mass function $p(x)$ is given

by

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x),$$

where $l(x)$ is the length of the codeword associated with x.

Without loss of generality, we can assume that the D-ary alphabet is $\mathcal{D} = \{0, 1, ..., D-1\}$.

Some examples of codes follow.

EXAMPLE 2.1. Let X be a random variable with the following distribution and codeword assignment:

$\Pr(X = 1)$=1/2, codeword $C(1) = 0$

$\Pr(X = 2)$=1/4, codeword $C(2) = 10$

$\Pr(X = 3)$=1/8, codeword $C(3) = 110$

$\Pr(X = 4)$=1/8, codeword $C(4) = 111$.

The entropy $H(X)$ of X is 1.75 bits, and the expected length $L(C) = El(X)$ of this code is also 1.75 bits. Here we have a code that has the same average length as the entropy. We note that any sequence of bits can be uniquely decoded into a sequence of symbols of X. For example, the bit string 0110111100110 is decoded as 134213.

EXAMPLE 2.2. Consider another simple example of a code for a random variable:

$\Pr(X = 1)$=1/3, codeword $C(l) = 0$

$\Pr(X = 2) = 1/3$, codeword $C(2) = 10$

$\Pr(X = 3) = 1/3$, codeword $C(3) = 11$.

Just as in the previous case, the code is uniquely decodable. However, in this case the entropy is $\log 3 = 1.58$ bits, while the average length of the encoding is 1.66 bits. Here $El(X) > H(X)$.

EXAMPLE 2.3. (Morse code): The Morse code is a reasonably efficient code for the English alphabet using an alphabet of four symbols: a dot, a dash, a letter space and a word space. Short sequences represent frequent letters (e.g., a single dot represents E) and long sequences represent infrequent letters (e.g., Q is represented by "dash, dash, dot, dash"). This is not the optimal representation for the alphabet in four symbols - in fact, many possible codewords are not utilized because the codewords for letters do not contain spaces except for a letter space at the end of every codeword and no space can follow another space. It is an interesting problem to calculate the number of sequences that can be constructed under these constraints. The problem was solved by Shannon in his original 1948 paper. The problem is also related to coding for magnetic recording, where long strings of O's are prohibited.

We now define increasingly more stringent conditions on codes. Let x^n denote $(x_1, x_2, ..., x_n)$.

DEFINITION 2.3. A code is said to be *non-singular* if every element of the range of X maps into a different string in $\mathcal{D}^*$, i.e.,

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j)$$

Non-singularity suffices for an unambiguous description of a single value of X. But we usually wish to send a sequence of values of X. In such cases, we can ensure decodability by adding a special symbol (a "comma") between any two codewords. But this is an inefficient use of the special symbol; we can do better by developing the idea of self-punctuating or instantaneous codes. Motivated by the necessity to send sequences of symbols X, we define the extension of a code as follows:

DEFINITION 2.4. The extension C^* of a code C is the mapping from finite length strings of $\mathcal{X}$ to finite length strings of $\mathcal{D}$, defined by

$$C(x_1 x_2 ... x_n) = C(x_1)C(x_2)...C(x_n),$$

where $C(x_1)C(x_2)...C(x_n)$ indicates concatenation of the corresponding codewords.

EXAMPLE 2.4. If $C(x_1) = 00$ and $C(x_2) = 11$, then $C(x_1 x_2) = 0011$.

DEFINITION 2.5. A code is called uniquely decodable if its extension is non-singular.

In other words, any encoded string in a uniquely decodable code has only one possible source string producing it. However, one may have to look at the entire string to determine even the first symbol in the corresponding source string.

DEFINITION 2.6. A code is called a prefix code or an instantaneous code if no codeword is a prefix of any other codeword.

An instantaneous code can be decoded without reference to the future codewords since the end of a codeword is immediately recognizable. Hence, for an instantaneous code, the symbol x_i can be decoded as soon as we come to the end of the codeword corresponding to it. We need not wait to see the codewords that come later. An instantaneous code is a "self-punctuating" code; we can look down the sequence of code symbols and add the commas to separate the codewords without looking at later symbols. For example, the binary string 01011111010 produced by the code of Example 2.1 is parsed as 0, 10, 111, 110, 10. To

X	Singular	Non-singular, but not Uniquely decodable	Uniquely decodable but not instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

TABLE 1. Classes of codes

illustrate the differences between the various kinds of codes, consider the following examples of codeword assignments $C(x)$ to $x \in \mathcal{X}$ in Table 1. For the non-singular code, the code string 010 has three possible source sequences: 2 or 14 or 31, and hence the code is not uniquely decodable. The uniquely decodable code is not prefix free and is hence not instantaneous. To see that it is uniquely decodable, take any code string and start from the beginning. If the first two bits are 00 or 10, they can be decoded immediately. If the first two bits are 11, then we must look at the following bits. If the next bit is a 1, then the first source symbol is a 3. If the length of the string of 0's immediately following the 11 is odd, then the first codeword must be 110 and the first source symbol must be 4; if the length of the string of 0's is even, then the first source symbol is a 3.

By repeating this argument, we can see that this code is uniquely decodable. Sardinas and Patterson have devised a finite test for unique decodability, which involves forming sets of possible sufkes to the codewords and systematically eliminating them. The fact that the last code in Table 1 is instantaneous is obvious since no codeword is a prefix of any other. We wish to construct instantaneous codes of minimum expected length to describe a given source. It is clear that we cannot assign short codewords to all source symbols and still be prefix free. The set of codeword lengths possible for instantaneous codes is limited by the following inequality:

THEOREM 2.1. *(Kraft inequality): For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths $l_1, l_2, ..., l_m$ must satisfy the inequality*

$$\sum_i D^{-l_i} \leq 1.$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

PROOF. Consider a D-ary tree in which each node has D children. Let the branches of the tree represent the symbols of the codeword. For example, the D branches arising from the root node represent the D possible values of the first symbol of the codeword. Then each codeword is represented by a leaf on the tree. The path from the root traces out the symbols of the codeword. A binary example of such a tree is shown in Figure 1

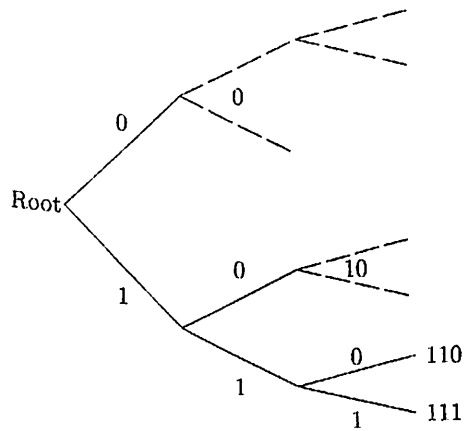

FIGURE 1. Code tree for the Kraft inequality.

The prefix condition on the codewords implies that no codeword is an ancestor of any other codeword on the tree. Hence, each codeword eliminates its descendants as possible codewords. Let $l_{\max}$ be the length of the longest codeword of the set of codewords. Consider all nodes of the tree at level $l_{\max}$. Some of them are codewords, some are descendants of codewords, and some are neither. A codeword at level l_i has $D^{l_{\max}-l_i}$ descendants at level $l_{\max}$. Each of these descendant sets must be disjoint. Also, the total number of nodes in these sets must be less than or equal to $D^{l_{\max}}$. Hence, summing over all the codewords, we have

$$\sum D^{l_{\max}-l_i} \leq D^{l_{\max}}$$

or

$$\sum_i D^{-l_i} \leq 1,$$

which is the Kraft inequality. Conversely, given any set of codeword lengths $l_1, l_2, ..., l_m$, which satisfy the Kraft inequality, we can always construct a tree like the one in Figure 1. Label the first node (lexicographically) of depth l_1, as codeword 1, and remove its descendants from the tree. Then label the first remaining node of depth l_2, as codeword 2, etc. Proceeding this way, we construct a prefix code with the specified $l_1, l_2, ..., l_m$. □

OPTIMAL CODES. In the previous section, we proved that any codeword set that satisfies the prefix condition has to satisfy the Kraft inequality and that the Kraft inequality is a sufficient condition for the existence of a codeword set with the specified set of codeword lengths. We now consider the problem of finding the prefix code with the minimum expected length. From the results of the previous section, this is equivalent to finding the set of lengths $l_1, l_2, ..., l_m$, satisfying the Kraft inequality and whose expected length $L = \sum p_i l_i$ is less than the expected length of any other prefix code. This is a standard optimization problem:

$$\text{Minimize } L = \sum p_i l_i$$

over all integers $l_1, l_2, ..., l_m$, satisfying

$$\sum_i D^{-l_i} \leq 1.$$

A simple analysis by calculus suggests the form of the minimizing l_i^*. We neglect the integer constraint on l_i and assume equality in the constraint. Hence, we can write the constrained minimization using Lagrange multipliers as the minimization of

$$J = \sum p_i l_i + \lambda \left(\sum_i D^{-l_i} \right).$$

Differentiating with respect to l_i, we obtain

$$\frac{\partial J}{\partial l_i} = p_i - \lambda D^{-l_i} \ln D.$$

Setting the derivative to 0, we obtain

$$D^{-l_i} = \frac{p_i}{\lambda \ln D}.$$

Substituting this in the constraint to find λ, we find $\lambda = 1/\ln D$ and hence

$$p_i = D^{-l_i},$$

yielding optimal code lengths

$$l_i^* = -\log_D p_i. \tag{21}$$

This non-integer choice of codeword lengths yields expected codeword length

$$L^* = \sum p_i l_i^* = -\sum p_i \log_D p_i = H_D(X).$$

But since the l_i must be integers, we will not always be able to set the codeword lengths as in (21). Instead, we should choose a set of codeword lengths l_i "close" to the optimal set. Rather than demonstrate by calculus that $l_i^* = -\log_D p_i$ is a global minimum, we will verify optimality directly in the proof of the following theorem.

THEOREM 2.2. *The expected length L of any instantaneous D-ary code for a random variable X is greater than or equal to the entropy $H_D(X)$, i.e.*

$$L \geq H_D(X)$$

with equality if and only if $D^{-l_i} = p_i$.

PROOF. We can write the difference between the expected length and the entropy as

$$\begin{aligned} -L + H_D(X) &= -\sum p_i l_i + \sum p_i \log_D \frac{1}{p_i} \\ &= \sum p_i \log_D D^{-l_i} - \sum p_i \log_D p_i = \\ &= \sum p_i \log \frac{2^{-l_i}}{p_i} \leq \sum p_i \left[\frac{2^{-l_i}}{p_i} - 1 \right] = \\ &= \sum 2^{-l_i} - 1 \leq 0. \end{aligned}$$

□

DEFINITION 2.7. A probability distribution is called D-adic with respect to D if each of the probabilities is equal to D^{-n} for some n.

Thus we have equality in the theorem if and only if the distribution of X is D-adic. The preceding proof also indicates a procedure for finding an optimal code: find the D-adic distribution that is closest to the distribution of X. This distribution provides the set of codeword lengths. Construct the code by choosing the first available node as in the proof of the Kraft inequality. We then have an optimal code for X. However, this procedure is not easy, since the search for the closest

D-adic distribution is not obvious. In the next section, we give a good suboptimal procedure (Shannon-Fano coding). We describe a simple procedure (Huffman coding) for actually finding the optimal code.

BOUNDS ON THE OPTIMAL CODELENGTH. We now demonstrate a code that achieves an expected description length L within 1 bit of the lower bound, that is,

$$H(X) \le L \le H(X) + 1.$$

Recall the setup of the last section: we wish to minimize $L = \sum p_i l_i$ subject to the constraint that $l_1, l_2, ..., l_m$ are integers and $\sum D^{-l_i} \le 1$. We proved that the optimal codeword lengths can be found by finding the D-adic probability distribution closest to the distribution of X. The choice of word lengths $l_i = -\log p_i$ yields $L = H$. Since $\log_D \frac{1}{p_i}$ may not equal an integer, we round it up to give integer word length assignments,

$$l_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil$$

These lengths satisfy the Kraft inequality since

$$\sum D^{-\lceil \log_D \frac{1}{p_i} \rceil} \le \sum D^{-\log_D \frac{1}{p_i}} = \sum p_i = 1.$$

This choice of codeword lengths satisfies

$$\log_D \frac{1}{p_i} \le l_i < \log_D \frac{1}{p_i} + 1.$$

Multiplying by p_i and summing over i, we obtain

$$H_D(X) \le L < H_D(X) + 1. \tag{22}$$

Since the optimal code can only be better than this code, we have the following theorem:

THEOREM 2.3. *Let $l_1^*, l_2^*, ..., l_m^*$ be the optimal codeword lengths for a source distribution p and a D-ary alphabet, and let L^* be the associated expected length of the optimal code ($L^* = \sum p_i l_i^*$). Then*

$$H_D(X) \le L^* < H_D(X) + 1.$$

PROOF. Let $l_i = \lceil -\log_D p_i \rceil$. Then l_i satisfies the Kraft inequality and from (22) we have

$$H_D(X) \le L = \sum p_i l_i < H_D(x) + 1.$$

But since L^*, the expected length of the optimal code, is less than $L = \sum p_i l_i$, and since $L^* \ge H_D$ we have the theorem. □

In the preceding theorem, there is an overhead which is at most 1 bit, due to the fact that $\log_D \frac{1}{p_i}$ is not always an integer. We can reduce the overhead per symbol by spreading it out over many symbols. With this in mind, let us consider a system in which we send a sequence of n symbols from X. The symbols are assumed to be drawn i.i.d. according to $p(x)$. We can consider these n symbols to be a supersymbol from the alphabet $\mathcal{X}^n$. Define L_n, to be the expected codeword length per input symbol, i.e., if $l(x_1, x_2, ..., x_n)$ is the length of the codeword associated with $(x_1, x_2, ..., x_n)$, then

$$L_n = \frac{1}{n} \sum p(x_1, x_2, ..., x_n) l(x_1, x_2, ..., x_n) = \frac{1}{n} El(X_1, X_2, ..., X_n).$$

We can now apply the bounds derived above to the code:

$$H(X_1, X_2, ..., X_n) \le El(X_1, X_2, ..., X_n) < H(X_1, X_2, ..., X_n) + 1.$$

Since $X_1, X_2, ..., X_n$, are i.i.d. (independent identical distributions),

$$H(X_1, X_2, ..., X_n) = \sum H(X_i) = nH(X).$$

Dividing by n, we obtain

$$H(X) \le L_n < H(X) + \frac{1}{n}.$$

Hence by using large block lengths we can achieve an expected codelength per symbol arbitrarily close to the entropy. Thus we have the following theorem:

THEOREM 2.4. *(Shannon source coding theorem) The minimum expected codeword length per symbol satisfies*

$$\frac{H(X_1, X_2, ..., X_n)}{n} \le L_n^* < \frac{H(X_1, X_2, ..., X_n)}{n} + \frac{1}{n}.$$

This theorem provides another justification for the definition of entropy rate – it is the expected number of bits per symbol required to describe the process.

HUFFMAN CODES. An optimal (shortest expected length)prefix code for a given distribution can be constructed by a simple algorithm discovered by Huffman. We will prove that any other code for the same alphabet cannot have a lower expected length than the code constructed by the algorithm. We prove by induction that the binary Huffman code is optimal. It is important to remember that there are many optimal codes: inverting all the bits or exchanging two codewords of the same length will give another optimal code. The Huffman procedure constructs one such optimal code. To prove the optimality of Huffman codes, we first prove some properties of a particular optimal

code. Without loss of generality, we will assume that the probability masses are ordered, so that $p_1 \geq p_2 \cdots \geq p_m$. Recall that a code is optimal if $\sum p_i l_i$ is minimal.

LEMMA 2.5. *For any distribution, there exists an optimal instantaneous code (with minimum expected length) that satisfies the following properties:*

1. *If $p_j > p_k$, then $l_j \leq l_k$.*
2. *The two longest codewords have the same length.*
3. *The two longest codewords differ only in the last bit and correspond to the two least likely symbols.*

PROOF. The proof amounts to swapping, trimming and rearranging. Consider an optimal code C_m:

If $p_j > p_k$, then $l_j \leq l_k$. Here we swap codewords. Consider C'_m with the codewords j and k of C_m, interchanged. Then

$$\begin{aligned} L(C'_m) - L(C_m) &= \sum p_i l'_i - \sum p_i l_i \\ &= p_j l_k + p_k l_j - p_j l_j - p_k l_k \\ &= (p_j - p_k)(l_k - l_j) \end{aligned}$$

But $p_j - p_k > 0$, and since C_m, is optimal, $L(C'_m) - L(C_m) \geq 0$. Hence we must have $l_k - l_j \geq 0$. Thus C_m, itself satisfies property 1.

The two longest codewords are of the same Length. Here we trim the codewords. If the two longest codewords are not of the same length, then one can delete the last bit of the longer one, preserving the prefix property and achieving lower expected codeword length. Hence the two longest codewords must have the same length. By property 1, the longest codewords must belong to the least probable source symbols.

The two longest codewords differ only in the last bit and correspond to the two least likely symbols. Not all optimal codes satisfy this property, but by rearranging, we can find a code that does. If there is a maximal length codeword without a sibling, then we can delete the last bit of the codeword and still satisfy the prefix property. This reduces the average codeword length and contradicts the optimality of the code. Hence every maximal length codeword in any optimal code has a sibling. Now we can exchange the longest length codewords so the two lowest probability source symbols are associated with two siblings on the tree. This does not change the expected length $\sum p_i l_i$. Thus the codewords for the two lowest probability source symbols have maximal length and agree in all but the last bit.

Summarizing, we have shown that if $p_1 \geq p_2 \cdots \geq p_m$, then there exists an optimal code with $l_1 \leq l_2 \cdots \leq l_{m-1} \leq l_m$ and codewords $C(X_{m-1})$ and $C(X_m)$ that differ only in the last bit. □

Thus we have shown that there exists an optimal code satisfying the properties of the lemma. We can now restrict our search to codes that satisfy these properties.

For a code C_m, satisfying the properties of the lemma, we now define a "merged" code C_{m-1} for $m-1$ symbols as follows: take the common prefix of the two longest codewords (corresponding to the two least likely symbols), and allot it to a symbol with probability $p_{m-1} + p_m$. All the other codewords remain the same.

The expected length of the code C_m, is

$$\begin{aligned} L(C_m) &= \sum p_i l_i \\ &= \sum_{i=1}^{m-2} p_i l'_i + p_{m-1}(l'_{m-1} + 1) + p_m(l'_{m-1} + 1) \\ &= \sum_{i=1}^{m-1} p_i l'_i + p_{m-1} + p_m \\ &= L(C_{m-1}) + p_{m-1} + p_m. \end{aligned}$$

Thus the expected length of the code C_m differs from the expected length of C_{m-1} by a fixed amount independent of C_{m-1}. Thus minimizing the expected length $L(C_m)$ is equivalent to minimizing $L(C_{m-1})$. Thus we have reduced the problem to one with $m-1$ symbols and probability masses $(p_1, p_2, ..., p_{m-2}, p_{m-1} + p_m)$. We again look for a code which satisfies the properties of Lemma 2.5 for these $m-1$ symbols and then reduce the problem to finding the optimal code for $m-2$ symbols with the appropriate probability masses obtained by merging the two lowest probabilities on the previous merged list. Proceeding this way, we finally reduce the problem to two symbols, for which the solution is obvious, i.e., allot 0 for one of the symbols and 1 for the other. Since we have maintained optimality at every stage in the reduction, the code constructed for m symbols is optimal. Thus we have proved the following theorem for binary alphabets.

THEOREM 2.6. *Huffman coding is optimal, i.e., if C^* is the Huffman code and C' is any other code, then $L(C^*) \leq L(C')$.*

Although we have proved the theorem for a binary alphabet, the proof can be extended to establishing optimality of the Huffman coding algorithm for a D-ary alphabet as well. Incidentally, we should remark that Huffman coding is a "greedy" algorithm in that it coalesces the two least likely symbols at each stage. The above proof shows that this local optimality ensures a global optimality of the final code.

ARITHMETIC CODING. From the discussion of the previous sections, it is apparent that using a codeword length of $\log \frac{1}{p(x)}$ for the codeword corresponding to x is nearly optimal in that it has an expected length within 1 bit of the entropy. The optimal codes are Huffman codes, and these can be constructed by the procedure described in previous section. For small source alphabets, though, we have efficient coding only if we use long blocks of source symbols. For example, if the source is binary, and we code each symbol separately, we must use 1 bit per symbol irrespective of the entropy of the source. If we use long blocks, we can achieve an expected length per symbol close to the entropy rate of the source. It is therefore desirable to have an efficient coding procedure that works for long blocks of source symbols. Huffman coding is not ideal for this situation, since it is a bottom-up procedure that requires the calculation of the probabilities of all source sequences of a particular block length and the construction of the corresponding complete code tree. We are then limited to using that block length. A better scheme is one which can be easily extended to longer block lengths without having to redo all the calculations. Arithmetic coding achieves this goal. The essential idea of arithmetic coding is to efficiently calculate the probability mass function $p(x^n)$ and the cumulative distribution function $F(x^n)$ for the source sequence x^n. We can use a number in the interval $(F(x^n) - p(x^n), F(x^n)]$ as the code for x^n. For example, expressing $F(x^n)$ to an accuracy of $\lceil -\log p(x^n) \rceil$ will give us a code for the source. It follows that the codeword corresponding to any sequence lies within the step in the cumulative distribution function corresponding to that sequence. So the codewords for different sequences of length n are different. However, the procedure does not guarantee that the set of codewords is prefix-free. We can construct a prefix-free set by using $\bar{F}(x)$ rounded off to $\lceil -\log p(x^n) \rceil + 1$ bits. In the algorithm described below, we will keep track of both $F(x^n)$ and $p(x^n)$ in the course of the algorithm, so we can calculate $\bar{F}(x)$ easily at any stage. We now describe a simplified version of the arithmetic coding algorithm to illustrate some of the important ideas. We assume that we have a fixed block length n that is known to both the encoder and the decoder. With a small loss of generality, we will assume that the source alphabet is binary. We assume that we have a simple procedure to calculate $p(x_l, x_2, ..., x_n)$ for any string $x_l, x_2, ..., x_n$. We will use the natural lexicographic order on strings, so that a string x is greater than a string y if $x_i = 1, y_i = 0$ for the first i such that $x_i \neq y_i$. Equivalently, $x > y$ if $\sum_i x_i 2^{-i} > \sum_i y_i 2^{-i}$, i.e., if the corresponding binary numbers satisfy $0.x > 0.y$. We can arrange the strings as the leaves

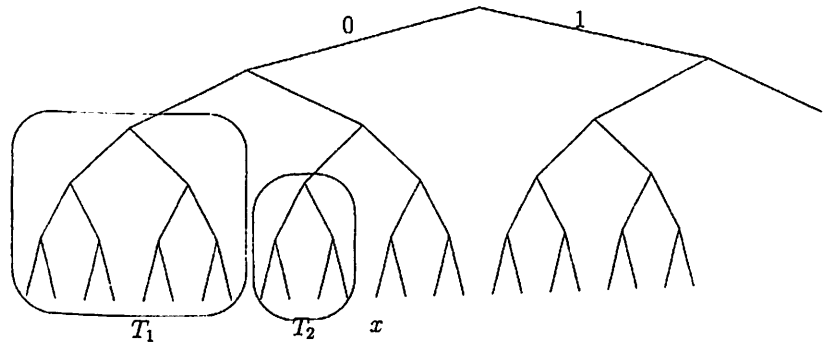

FIGURE 2. Tree of strings for arithmetic coding

of a tree of depth n, where each level of the tree corresponds to one bit. Such a tree is illustrated in Figure 2. In this figure, the ordering $x > y$ corresponds to the fact that x is to the right of y on the same level of the tree. It appears that we need to find $p(y^n)$ for all $y^n \leq x^n$ and use that to calculate $F(x^n)$. Looking at the tree, we might suspect that we need to calculate the probabilities of all the leaves to the left of x^n to find $F(x^n)$. The sum of these probabilities is the sum of the probabilities of all the subtrees to the left of x^n. Let $T_{x_1x_2...x_{k-1}0}$ be a subtree starting with $x_1x_2...x_{k-1}0$. The probability of this subtree is

$$p(T_{x_1x_2...x_{k-1}0}) = \sum_{y_{k+1}...y_n} p(x_1x_2...x_{k-1}0y_{k+1}...y_n)$$
$$= p(x_1x_2...x_{k-1}0)$$

and hence can be calculated easily.

Therefore we can rewrite $F(x^n)$ as

$$F(x^n) = \sum_{y^n \leq x^n} p(y^n)$$
$$= \sum_{T:T \text{ is to the left of } x^n} p^T$$
$$= \sum_{k:x_k=0} p(x_1x_2...x_{k-1}0).$$

Thus we can calculate $F(x^n)$ quickly from $p(x^n)$.

EXAMPLE 2.5. If $X_1, X_2, ..., X_n$ are Bernoulli(θ) in Figure 2, then

$$F(01110) = p(T_1) + p(T_2) + p(T_3) = p(00) + p(010) + p(0110)$$

$$= (1-\theta)^2 + \theta(1-\theta)^2 + \theta^2(1-\theta)^2.$$

Note that these terms can be calculated recursively. For example,

$$\theta^3(1-\theta)^3 = (\theta^2(1-\theta)^2)\theta(1-\theta).$$

To encode the next bit of the source sequence, we need only calculate $p(x^i x_{i+1})$ and update $F(x^i x_{i+1})$ using the above scheme. Encoding can therefore be done sequentially, by looking at the bits as they come in. To decode the sequence, we use the same procedure to calculate the cumulative distribution function and check whether it exceeds the value corresponding to the codeword. We then use the tree in Figure 2 as a decision tree. At the top node, we check to see if the received codeword $F(x^n)$ is greater than $p(0)$. If it is, then the subtree starting with 0 is to the left of x^n and hence $x_1 = 1$. Continuing this process down the tree, we can decode the bits in sequence. Thus we can compress and decompress a source sequence in a sequential manner. The above procedure depends on a model for which we can easily compute $p(x^n)$. Two examples of such models are i.i.d. sources, where

$$p(x^n) = \prod_{i=1}^{n} p(x_i)$$

and Markov sources, where

$$p(x^n) = p(x_1)\prod_{i=2}^{n} p(x_i|x_{i-1}).$$

In both cases, we can easily calculate $p(x^n x_{n+1})$ from $p(x^n)$. Note that it is not essential that the probabilities used in the encoding be equal to the true distribution of the source. In some cases, such as in image compression, it is difficult to describe a "true" distribution for the source. Even then, it is possible to apply the above arithmetic coding procedure. The procedure will be efficient only if the model distribution is close to the empirical distribution of the source. A more sophisticated use of arithmetic coding is to change the model dynamically to adapt to the source. Adaptive models work well for large classes of sources. The adaptive version of arithmetic coding is a simple example of a universal code, that is, a code that is designed to work with an arbitrary source distribution. Another example is the Lempel-Ziv code. The foregoing discussion of arithmetic coding has avoided discussion of the difficult implementation issues of computational accuracy, buffer sizes, etc. An introduction to some of these issues can be found in the tutorial introduction to arithmetic coding.

Arithmetic coding else. Again we assume we have information on the relative frequencies of letters. As all calculations are base 2 we probably wish to replace the actual frequencies by fractions with a power of 2 in the denominator. Let us use frequencies

$$p_a = \frac{1}{2} = 0.1, p_b = p_c = \frac{3}{16} = 0.0011, p_d = \frac{1}{8} = 0.001.$$

The idea is the following: Start from the unit interval [0, 1). Each letter is assigned a subinterval, closed to the left and open to the right, whose length is proportional to the frequency of the letter and such that those subintervals partition the unit interval. In our case those subintervals are

interval	length l	representative
$A(a) = [0, 0.1)$	0.1	0
$A(b) = [0.1, 0.1011)$	0.0011	1
$A(c) = [0.1011, 0.111)$	0.0011	11
$A(d) = [0.111, 1)$	0.001	111

Recall that we use the duadic system. For example 0.1 stands for 1/2. We subdivided the unit interval [0, 1) into four subintervals whose lengths equal the probabilities of the letters. In the right column we noted a representative for each of those subintervals. For example, representative 11 in the $A(c)$ row really stands for $0.11 \in A(c)$. Here the bitstring describing the representative has been chosen as short as possible. Next we assign intervals to pairs of letters using the same principle. The intervals $A(aa), A(ab), A(ac), A(ad)$ partition $A(a)$ and the lengths of the subintervals correspond to the additional letters: the lengths satisfy $l(A(ai)) = l(A(a)) \times p_i$. In practice it is probably easiest to compute those lengths first. The left endpoint of each subinterval corresponding to pairs equals the right endpoint of its predecessor. The right endpoint is obtained by adding the length:

interval	length l	representative
$A(aa) = [0, 0.01)$	0.01	0
$A(ab) = [0.01, 0.01011)$	0.00011	01
$A(ac) = [0.01011, 0.0111)$	0.00011	011
$A(ad) = [0.0111, 0.1)$	0.0001	0111
$A(ba) = [0.1, 0.10011)$	0.00011	1
$A(bb) = [0.10011, 0.10100001)$	0.00001001	101
$A(bc) = [0.10100001, 0.1010101)$	0.00001001	10101
$A(bd) = [0.1010101, 0.1011)$	0.0000011	101011
$A(ca) = [0.1011, 0.11001)$	0.00011	11
$A(cb) = [0.11001, 0.11010001)$	0.00001001	1101
$A(cc) = [0.11010001, 0.1101101)$	0.00001001	11011
$A(cd) = [0.1101101, 0.111)$	0.0000011	110111
$A(da) = [0.111, 0.1111)$	0.0001	111
$A(db) = [0.1111, 0.1111011)$	0.0000011	1111
$A(dc) = [0.1111011, 0.111111)$	0.0000011	11111
$A(dd) = [0.111111, 0.1)$	0.000001	111111

It is clear how this continues: intervals $A(aaa), A(aab), A(aac), A(aad)$ partition $A(aa)$ and the relative lengths correspond to the letter frequencies. In other words, if we magnify the interval $A(aa)$ such that it becomes the unit interval, then the subdivision defined by $A(aaa)$,-$A(aab), A(aac), A(aad)$ corresponds exactly to the original subdivision of the unit interval. This is the general principle. Whenever we pass to the next subdivision of an interval we generate a clone of the unit interval and its original subdivision, on a smaller scale. In each interval we choose a short representative. Here is how compression (encoding) works: imagine we wish to compress a long document (signal $x = (x_1, ..., x_N)$) consisting of N letters. The compressed document has two parts:the number N and the representative of interval $A(x)$. For example, the message bc is encoded as (2, 10101). The decompresser looks up which of the intervals corresponding to pairs has 10101 as its representative. As this is interval $A(bc)$ the decompression yields the correct result bc. Observe that the most probable signal aa of length 2 corresponds to the longest interval $A(aa)$ and to the shortest representative 0.

Lempel-Ziv compression. We know that Huffman compression is optimal if only short range effects are taken into account. If we believe that the entropy of the English language is essentially determined by sequences of 4 letters, then the best thing to do is to use Huffman compression based on signals of length 4. However, we saw that this is not justified. Shannon's experiments and arguments revealed that the

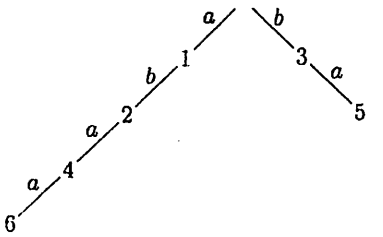

FIGURE 3. Our tree

entropy per letter goes down a lot of we take long range effects into account. In practical terms this means that it is not futile to search for methods which tap this source of redundancy. Huffman coding would theoretically still be best but it is of course hopeless in reality. We cannot manage the statistics of all sequences of 100 letters, say. There are 27^{100} such sequences after all. Two successful attempts to use long-range effects for data compression have been found by Lempel and Ziv. We start with an easy example illustrating the principle of LZ78: let the text be

aabbababaabaa

Encoding proceeds letter by letter, building up a tree where the vertices are numbered 0, 1, 2, . . . and the edges (each edge connects two vertices) are indexed by letters. Starting point is always the root of the tree, which is numbered 0. The first text letter a leads to an edge starting at the root with endpoint 1. The second letter a of the text illustrates the principle: starting from the root it is checked what is the longest section of text, starting from our second letter, which can be read following the edges. In our the a is found and that is all. The part of the compressed text we obtain so far is:

$$(0, a)(1, b).$$

The decoder (the uncompress algorithm) reads this as: at first take the empty string, corresponding to index 0 (the root), follow it by a. Then read the labels of edges, starting from the root, until vertex 1 is reached, followed by b. The gives aab so far and leads to a new edge of the tree, starting from vertex 1, ending at the new vertex 2, and labelled b. So far our memory, the tree consists of a branch labelled

a from 0 to 1 and a branch labelled b connecting 1 and 2. Now the fourth letter b needs to be encoded. As there is no edge labelled b originating at the root yet we now construct one. Its endpoint is 3 and the next encoding pair is $(0, b)$. The next step is the first success of compression: we want to encode *aba...* and we find a path, from the root to vertex 2, with labels *ab*. This means we encode the segment *aba* as as $(2, a)$ and construct a new vertex 4, joined to 2 by an edge labelled a. The principle should be clear by now: the next section is *ba*, encoded as $(3, a)$, which leads to vertex 5 in the usual manner. Finally we encode the last segment *abaa* as $(4, a)$. By now the tree looks as in Figure 3. The compressed text is $(0, a)(1, b)(0, b)(2, a)(3, a)(4, a)$ and it should be clear how decompressing works. The original text has 13 letters, the compression has 6 letters and 6 numbers. This is of course only a toy example. It seems that LZ78 works better than Huffman on ordinary English text. Haykin mentions a compression rate of about 55% for LZ78, whereas Huffman apparently compresses by about 43%. There are many practical issues which we do not want to discuss. One concern is with the size of the memory. The tree grows with the text. What happens when it gets too large? We may flush it and start from the beginning, or we may freeze at a certain point. LZ77 is a variant which needs much less memory. We illustrate the principle with an example. The text is

She sells sea shells by the seashore

The idea is the following: instead of building up a large memory we use only the section of the text in a window of fixed size. Assume we have reached the point where

ells by the seashore

needs to be codes and the window extends all the way to the beginning of our text. At this point the encoder will search for the longest section in the window (to the left) which agrees with the text we want to encode. In the example we find a section *ells* of length 5 (the space also counts) 11 positions to the left. The next entry of the compressed text is then the pair (11, 5). When decompressing this is interpreted as: go 11 to the left and read the segment of length 5 starting there.

Problems

PROBLEM 2.1. *Consider a source that produces the 4 letters a, b, c, d. In this and the following problems we use the following example string produced by the source:*

aadda — abacb — abcaa—aacdb — abacc — aaabb — aaccc — adaba

Compute $H(0.5, 0.2, 0.2, 0.1)$.

PROBLEM 2.2. *Find the Huffman code for probabilities 0.5, 0.2, 0.2, 0.1 and the average word length.*

PROBLEM 2.3. *Use the Huffman code from the preceding problem to compress the example string.*

PROBLEM 2.4. *Determine the relative frequencies of pairs in the example string, find the corresponding Huffman code based on those 16 probabilities and use this to compress the example string.*

PROBLEM 2.5. *Compress the example string using adaptive Huffman coding (based on letters).*

PROBLEM 2.6. *Consider a source producing the letters a, b, c, d, e with relative frequencies 0.3, 0.2, 0.2, 0.2, 0.1. Find $H(0.3, 0.2, 0.2, 0.2, 0.1)$, the Huffman code and the average wordlength.*

PROBLEM 2.7. *Encode the source text*
a b c b a c e a d d
produced by the source above, using adaptive Huffman encoding.

PROBLEMS FOR CHAPTER

PROBLEM 2.8. *Slackness in the Kraft inequality. An instantaneous code has word lengths $l_1, I_2, ..., l_m$, which satisfy the strict inequality*

$$\sum_{i=1}^{m} D^{-l_i} < 1.$$

The code alphabet is $\mathcal{D} = \{0, 1, 2, ..., D - 1\}$. Show that there exist arbitrarily long sequences of code symbols in $\mathcal{D}^*$ which cannot be decoded into sequences of codewords.

PROBLEM 2.9. *Huffman coding. Consider the random variable*

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ 0.49 & 0.26 & 0.12 & 0.04 & 0.04 & 0.03 & 0.02 \end{pmatrix}$$

(a) Find a binary Huffman code for X.
(b) Find the expected codelength for this encoding.
(c) Find a ternary Huffman code for X.

PROBLEM 2.10. *More Huffman codes. Find the binary Huffman code for the source with probabilities $(1/3, 1/5, 1/5, 2/15, 2/15)$. Argue that this code is also optimal for the source with probabilities $(1/5, 1/5, 1/5, 1/5, 1/5)$.*

PROBLEM 2.11. *Bad codes. Which of these codes cannot be Huffman codes for any probability assignment?*
(a) $\{0, 10, 11\}$.
(b) $\{00, 01, 10, 110\}$.
(c) $\{01, 10\}$.

PROBLEM 2.12. *Huffman code. Find the (a) binary and (b) ternary Huffman codes for the random variable X with probabilities*

$$p = \left(\frac{1}{21}, \frac{2}{21}, \frac{3}{21}, \frac{4}{21}, \frac{5}{21}, \frac{6}{21}\right)$$

(c) Calculate $L = \sum p_i l_i$ *in each case.*

PROBLEM 2.13. *Classes of codes. Consider the code* $\{0, 01\}$
(a) Is it instantaneous?
(b) Is it uniquely decodable?
(c) Is it nonsingular?

CHAPTER 3

Channel Capacity

What do we mean when we say that A communicates with B? We mean that the physical acts of A have induced a desired physical state in B. This transfer of information is a physical process and therefore is subject to the uncontrollable ambient noise and imperfections of the physical signalling process itself. The communication is successful if the receiver B and the transmitter A agree on what was sent. In this chapter we find the maximum number of distinguishable signals for n uses of a communication channel. This number grows exponentially with n, and the exponent is known as the channel capacity. The channel capacity theorem is the central and most famous success of information theory. The mathematical analog of a physical signalling system is shown in Fig. 1. Source symbols from some finite alphabet are mapped into some sequence of channel symbols, which then produces the output sequence of the channel. The output sequence is random but has a distribution that depends on the input sequence. From the output sequence, we attempt to recover the transmitted message. Each of the possible input sequences induces a probability distribution on the output sequences. Since two different input sequences may give rise to the same output sequence, the inputs are confusable. In the next few sections, we will show that we can choose a "non-confusable" subset of input sequences so that with high probability, there is only one highly likely input that could have caused the particular output. We can then reconstruct the input sequences at the output with negligible probability of error. By mapping the source into the appropriate "widely spaced" input sequences to the channel, we can transmit a message with very low probability of error and
reconstruct the source message at the output. The maximum rate at which this can be done is called the capacity of the channel.

DEFINITION 3.1. We define a discrete channel to be a system consisting of an input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$ and a probability transition matrix $p(y|x)$ that expresses the probability of observing the output symbol y given that we send the symbol x. The channel is

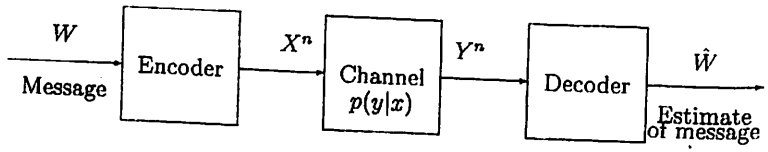

FIGURE 1. A communication system.

said to be *memoryless* if the probability distribution of the output depends only on the input at that time and is conditionally independent of previous channel inputs or outputs.

DEFINITION 3.2. We define the "information" channel capacity of a discrete memoryless channel as

$$C = \max_{p(x)} I(X,Y), \tag{23}$$

where the maximum is taken over all possible input distributions $p(x)$.

We shall soon give an operational definition of channel capacity as the highest rate in bits per channel use at which information can be sent with arbitrarily low probability of error. Shannon's second theorem establishes that the "information" channel capacity is equal to the "operational" channel capacity. Thus we drop the word "information" in most discussions of channel capacity. There is a duality between the problems of data compression and data transmission. During compression, we remove all the redundancy in the data to form the most compressed version possible, whereas during data transmission, we add redundancy in a controlled fashion to combat errors in the channel. In the last section of this chapter, we show that a general communication system can be broken into two parts and that the problems of data compression and data transmission can be considered separately.

CHAPTER 4

Examples of channel capacity

Noiseless Binary Channel. Suppose we have a channel whose the binary input is reproduced exactly at the output. This channel is illustrated in Figure 1. In this case, any transmitted bit is received without error. Hence, 1 error-free bit can be transmitted per use of the channel, and the capacity is 1 bit. We can also calculate the information capacity

$$C = \max I(X,Y) = 1 \text{ bit},$$

which is achieved by using $p(x) = (\frac{1}{2}, \frac{1}{2})$.

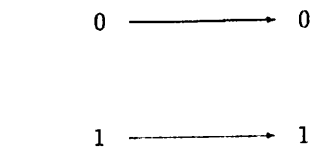

FIGURE 1. Noiseless binary channel.

Noisy Channel with Nonoverlapping Outputs. This channel has two possible outputs corresponding to each of the two inputs, as illustrated in Figure 2. The channel appears to be noisy, but really is not. Even though the output of the channel is a random consequence of the input, the input can be determined from the output, and hence every transmitted bit can be recovered without error. The capacity of this channel is also 1 bit per transmission. We can also calculate the information capacity

$$C = \max I(X,Y) = 1 \text{ bit},$$

which is achieved by using $p(x) = (\frac{1}{2}, \frac{1}{2})$.

Binary Symmetric Channel. Consider the binary symmetric channel (BSC), which is shown in Figure 3. This is a binary channel in which the input symbols are complemented with probability p. This is the simplest model of a channel with errors; yet it captures most of the complexity of the general problem. When an error occurs,

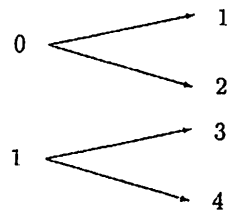

FIGURE 2. Noisy Channel with Nonoverlapping Outputs.

a 0 is received as a 1 and vice versa. The received bits do not reveal where the errors have occurred. In a sense, all the received bits are unreliable. Later we show that we can still use such a communication channel to send information at a non-zero rate with an arbitrarily small probability of error.

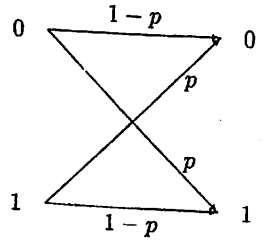

FIGURE 3. Binary Symmetric Channel.

We bound the mutual information by

$$\begin{aligned} I(X,Y) &= H(Y) - H(Y|X) \\ &= H(Y) - \sum p(x) H(Y|X=x) \\ &= H(Y) - \sum p(x) H(p) \\ &= H(Y) - H(p) \\ &\le 1 - H(p), \end{aligned}$$

where the last inequality follows because Y is a binary random variable. Equality is achieved when the input distribution is uniform. Hence the

information capacity of a binary symmetric channel with parameter p is

$$C = 1 - H(p) \text{ bits.}$$

SYMMETRIC CHANNELS. The capacity of the binary symmetric channel is $C = 1 - H(p)$ bits per transmission.

Now consider the channel with transmission matrix:

$$p(y|x) = \begin{pmatrix} 0.3 & 0.2 & 0.5 \\ 0.5 & 0.3 & 0.2 \\ 0.2 & 0.5 & 0.3 \end{pmatrix} \tag{24}$$

Here the entry in the xth row and the yth column denotes the conditional probability $p(y|x)$ that y is received when x is sent. In this channel, all the rows of the probability transition matrix are permutations of each other and so are the columns. Such a channel is said to be symmetric. Another example of a symmetric channel is one of the form

$$Y = X + Z(\mod c)$$

where Z has some distribution on the integers $\{0, 1, 2, \cdots, c-1\}$, and X has the same alphabet as Z, and Z is independent of X. In both these cases, we can easily find an explicit expression for the capacity of the channel. Letting r be a row of the transition matrix, we have

$$\begin{aligned} I(X,Y) &= H(Y) - H(Y|X) \\ &= H(Y) - H(r) \\ &\le \log|\mathcal{Y}| - H(r) \end{aligned}$$

with equality if the output distribution is uniform. But $p(x) = \frac{1}{|\mathcal{X}|}$ achieves a uniform distribution on Y, as seen from

$$p(y) = \sum_{x \in \mathcal{X}} p(y|x)p(x) = \frac{1}{|\mathcal{X}|} \sum p(y|x) = c\frac{1}{|\mathcal{X}|} = \frac{1}{|\mathcal{Y}|}$$

where c is the sum of the entries in one column of the probability transition matrix.

Thus the channel in (24) has capacity

$$C = \max_{p(x)} I(X,Y) = \log 3 - H(0.5, 0.3, 0.2),$$

and C is achieved by a uniform distribution on the input. The transition matrix of the symmetric channel defined above is doubly stochastic. In the computation of the capacity, we used the facts that the rows were permutations of one another and that all the column sums were equal. Considering these properties, we can define a generalization of the concept of a symmetric channel as follows:

DEFINITION 4.1. A channel is said to be *symmetric* if the rows of the channel transition matrix $p(y|x)$ are permutations of each other, and the columns are permutations of each other. A channel is said to be *weakly symmetric* if every row of the transition matrix $p(\cdot|x)$ is a permutation of every other row, and all the column sums C, $\sum_x p(y|x)$ are equal.

For example, the channel with transition matrix

$$p(y|x) = \begin{pmatrix} \frac{1}{3} & \frac{1}{6} & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{6} \end{pmatrix}$$

is weakly symmetric but not symmetric. The above derivation for symmetric channels carries over to weakly symmetric channels as well. We have the following theorem for weakly symmetric channels:

THEOREM 4.1. *For a weakly symmetric channel,*

$$C = \log |\mathcal{Y}| - H(\text{ row of transition matrix}),$$

and this is achieved by a uniform distribution on the input alphabet.

PROPERTIES OF CHANNEL CAPACITY. 1. $C \geq 0$, since $I(X,Y) \geq 0$.

2. $C \leq \log |\mathcal{X}|$ since $C = \max I(X,Y) \leq \max H(X) = \log |\mathcal{X}|$
3. $C \leq \log |\mathcal{Y}|$ for the same reason.
4. $I(X;Y)$ is a continuous function of $p(x)$.
5. $I(X,Y)$ is a concave function of $p(x)$.

Since $I(X,Y)$ is a concave function over a closed convex set, a local maximum is a global maximum. From properties (2) and (3), the maximum is finite, and we are justified in using the term maximum, rather than supremum in the definition of capacity. The maximum can then be found by standard nonlinear optimization techniques like gradient search. Some of the methods that can be used include the following:

1. Constrained maximization using calculus and the Kuhn-Tucker conditions.
2. The Frank-Wolfe gradient search algorithm.
3. An iterative algorithm developed by Arimoto and Blahut.

In general, there is no closed form solution for the capacity. But for many simple channels it is possible to calculate the capacity using properties like symmetry. Some of the examples considered earlier are of this form.

THE CHANNEL CODING THEOREM. We shall now give the complete statement of Shannon's second theorem:

THEOREM 4.2. *(The channel coding theorem): All rates below capacity C are achievable. Specifically, for every rate $R < C$, there exists a sequence of $(2^{nR}, n)$ codes with maximum probability of error $\lambda^{(n)} \to 0$. Conversely, any sequence of $(2^{nR}, n)$ codes with $\lambda^{(n)} \to 0$ must have $R \leq C$.*

Random coding is the method of proof for the above theorem, not the method of signalling. Codes are selected at random in the proof merely to symmetrize the mathematics and to show the existence of a good deterministic code. Although the theorem shows that there exist good codes with exponentially small probability of error for long block lengths, it does not provide a way of constructing the best codes. If we used the scheme suggested by the proof and generate a code at random with the appropriate distribution, the code constructed is likely to be good for long block lengths. However, without some structure in the code, it is very difficult to decode (the simple scheme of table lookup requires an exponentially large table). Hence the theorem does not provide a practical coding scheme. Ever since Shannon's original paper on information theory, researchers have tried to develop structured codes that are easy to encode and decode. So far, they have developed many codes with interesting and useful structures, but the asymptotic rates of these codes are not yet near capacity.

PROBLEMS FOR CHAPTER

1. Preprocessing the output. One is given a communication channel with transition probabilities $p(y|x)$ and channel capacity $C = \max_{p(x)} I(X;Y)$. A helpful statistician preprocesses the output by forming $\tilde{Y} = g(Y)$. He claims that this will strictly improve the capacity.

(a) Show that he is wrong.

(b) Under what conditions does he not strictly decrease the capacity?

2. Maximum likelihood decoding. A source produces independent, equally probable symbols from an alphabet (a_1, a_2) at a rate of one symbol every 3 seconds. These symbols are transmitted over a binary symmetric channel which is used once each second by encoding the source symbol a_1 as 000 and the source symbol a_2 as 111. If in the corresponding 3 second interval of the channel output, any of the sequences 000,001,010,100 is received, a_1 is decoded; otherwise, a_2 is decoded. Let $\epsilon < \frac{1}{2}$ be the channel crossover probability.

(a) For each possible received 3-bit sequence in the interval corresponding to a given source letter, find the probability that a_1, came out of the source given that received sequence.

(b) Using part (a), show that the above decoding rule minimizes the probability of an incorrect decision.

(c) Find the probability of an incorrect decision (using part (a) is not the easy way here).

(d) If the source is slowed down to produce one letter every $2n+1$ seconds, a_1, being encoded by $2n+1$ O's and a_2, being encoded by $2n+1$ 1's. What decision rule minimizes the probability of error at the decoder? Find the probability of error as $n \to \infty$.

3. An additive noise channel. Find the channel capacity of the following discrete memoryless channel: $Y = X + Z$, where $Pr(Z = 0) = Pr(Z = a) = \frac{1}{2}$. The alphabet for x is $\mathcal{X} = \{0, 1\}$. Assume that Z is independent of X. Observe that the channel capacity depends on the value of a.

5. Channel capacity. Consider the discrete memoryless channel $Y = X + Z(\mod 11)$, where

$$Z = \begin{pmatrix} 1 & 2 & 3 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

and $X \in \{O, 1, ..., 10\}$. Assume that Z is independent of X.

(a) Find the capacity.

(b) What is the maximizing $p^*(x)$?

9. The Z channel. The Z channel has binary input and output alphabets and transition probabilities $p(y|x)$ given by the following matrix:

$$Q = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \quad x, y \in \{0, 1\}$$

Find the capacity of the Z channel and the maximizing input probability distribution.

Part II

Coding theory

CHAPTER 5

Error-correcting codes.

This lecture in concerned with error-correcting codes. A message is encoded, and sent through a noisy channel, emerging as a garbled messaged, which then has to be decoded and the original message recovered. Examples: computers (R.W.Hamming: first error-detection, then error-correction), telephone lines, satellite links, CDs etc. A repetition code: message YES or NO, encoded as 11111 or 00000 respectively. If 00000 is sent, maybe it emerges as 01100 at the other end. This is then decoded as 00000, as this is the 'nearest' possibility.

DEFINITION 5.1. A code is a set of codewords each corresponding to a message.

E.g. the above code is {00000, 11111}. It is a binary (i.e. 0s and 1s) block (i.e. all words same length) code of length 5. The alphabet here is 0, 1, but could be any set F_q of q elements. Then we have a q-ary code - the case $q = 3$ is called ternary. Usually the alphabet is a field, so that we can do arithmetic $(+, -, \times, \div)$ with the usual rules. A code of length n is a subset of the Cartesian product F_q^n of n copies of F_q.

EXAMPLE 5.1. Four messages N, W, S, E can be encoded as 00, 01, 10, 11 respectively. Now add a check bit to make the number of 1s even, so that we have N : 000, W : 011, S : 101, E : 110. Now a single error can be detected (WHY?), but not corrected (WHY NOT?).

EXAMPLE 5.2. Now repeat the message: N : 00000 W : 01101 S : 10110 E : 11011. Now a single error can be corrected. WHY? HOW?

DEFINITION 5.2. The Hamming distance between two words ('vectors') in F_q^n is the number of places ('coordinates') where they differ.

EXAMPLE 5.3. $d(0111, 1110) = 2$.

PROPOSITION 5.1. *(The Hamming distance is a metric.)*
1. $d(x, y) \geq 0$, *with equality iff* $x = y$
2. $d(x, y) = d(y, x)$ *for all* x, y
3. *(the triangle inequality)* $d(x, z) \geq d(x, y) + d(y, z)$

PROOF. The triangle inequality follows from the fact that we can change x to z by first changing x to y and then changing y to z. □

This enables us to formalize nearest neighbour decoding-decode as the nearest codeword (in the Hamming distance) to the received vector. (Examples as above.)

Reasonable assumptions.

1. Each symbol is equally likely to be in error (with probability $p < \frac{1}{2}$)

2. Each error symbol is equally likely (with probability $p/(q-1)$) This defines a q-ary symmetric channel.

EXAMPLE 5.4. C = {000, 111}. Suppose 000 is sent. It is decoded wrongly if and only if it is received as 111, 110, 101, or 011. Now Prob(111) $= p^3$ and Prob(110) $= p^2(1-p)$ etc, so the word error probability is $p^3 + 3p^2(1-p) = 3p^2 - 2p^3$.

CHAPTER 6

Code parameters.

The crucial parameter for a code C is the minimal distance (between distinct codewords) $d(C) = \min\{d(x,y)|x,y \in C, x \neq y\}$. If this is large, we can make many errors before we lose track of which codeword we came from:

THEOREM 6.1. *1. If $d(C) \geq s+1$, then C can detect s errors.*
2. If $d(C) \geq 2t+1$, then C can correct t errors.

PROOF. If w is a codeword, and x is obtained by making $\leq t$ errors in w, so that $d(w,x) \leq t$, suppose w_0 is another codeword with $d(w_0,x) \leq t$. Then $d(w,w_0) \leq d(w,x) + d(x,w_0) \leq 2t$ which is a contradiction. □

Now $d(C) \geq s+1$ iff $s < d(C)$, and $d(C) \geq 2t+1$ iff $t < \frac{1}{2}d(C)$, so this can be rewritten:

COROLLARY 6.2. *If $d(C) = d$, then C can either detect d-1 errors, or correct $\lfloor \frac{d-1}{2} \rfloor$ errors, but not both! (WHY NOT?)*

Code parameters.
Minimal distance $d = d(C)$
Length n
Alphabet size q
Number of codewords $M = |C|$
Then C is called a $q-(n,M,d)$-code (or an (n,M,d)-code if q is understood. Different parameter values are required for different jobs:
Increase d for better error-correcting
Increase n or q for higher cost/time used
Increase M to send a greater variety of messages
Thus increasing d or M is a good thing, and increasing n or q is a bad thing.

EXAMPLE 6.1. Around 1970, the Mariner expeditions sent pictures back from Mars. The messages sent were subject to a lot of noise, so a binary (32, 64, 16) code was used. This can correct up to 7 errors in each word of length 32. But only has 64 codewords (out of a possible 2^{32}).

The Main Coding Theory Problem is to optimise the parameters q, n, M, d (i.e. maximise d and M, and minimise n and q). Usually we fix q, n and d by considering the hardware and the channel, and maximise M. Denote the maximum possible value of M for given q, n and d by $A_q(n, d)$.

EXAMPLE 6.2. $A_q(n, 1) = q^n$

PROOF. $d(C) = 1$ is no restriction, so C can contain all words $x_1 \dots x_n, x_i \in F_q$. □

EXAMPLE 6.3. $A_q(n, n) = q$

PROOF. The q-ary repetition code has parameters $q - (n, M, d)$ with $M = q$ and $d = n$, so $A_q(n, n) \geq q$. Conversely, if $M > q$ then two distinct codewords agree in the first place (by the pigeonhole principle) so the distance between them is at most $n - 1$, contradicting $d(C) = n$. □

Equivalence of codes.

The following changes make no 'essential' difference to a code:

1. Re-ordering the n places (in all codewords at once)
2. Relabelling the symbols in any one place

Two codes which can be transformed into each other by a combination of these operations are called equivalent.

EXAMPLE 6.4. By moving the second, third and fourth coordinates to the third, fourth and second places respectively, we see that the code {00000, 01101, 10110, 11011} is equivalent to the code {00000, 00111, 11010, 11101}. By swapping the symbols 0 and 1 in the first, third and fourth places, we see that both these codes are equivalent to the code {10110, 10001, 01100, 01011}.

LEMMA 6.3. *Any code C is equivalent to a code containing the zero word 00 . . . 0.*

PROOF. Choose $x_1, x_2 \dots x_n \in C$, and in the i-th place relabel x_i as 0 (and 0 as x_i). □

EXAMPLE 6.5. $A_2(5, 3) = 4$.

PROOF. We try to construct a binary $(5, M, 3)$-code with $M > 4$. Wlog $00000 \in C$. We cannot have two words each with four or five 1s (because they differ in at most two places). Therefore we must have at least two words each with three 1s. Wlog these are 11100 and 00111. Then the only word at distance at least 3 from all of these is 11011. □

We have actually proved more: there is (up to equivalence) a unique binary $(5, 4, 3)$-code.

CHAPTER 7

Binary codes.

Let $F_2 = \{0, 1\}$, so F_2^n is the set of words (or vectors) of length n, $F_2^n = \{x_1 x_2 \dots x_n | x_i = 0 \text{ or } 1\}$. Given $x = x_1 x_2 \dots x_n$ and $y = y_1 y_2 \dots y_n$ define

$$x + y = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

$$x \cup y = (x_1 y_1, x_2 y_2, \dots, x_n y_n)$$

where arithmetic is done modulo 2.

DEFINITION 7.1. The weight $w(x)$ of $x = x_1 x_2 \dots x_n$ is the number of 1s in the list $x_1, x_2, \dots, x_n$ of its coordinates.

LEMMA 7.1. $d(x, y) = w(x + y)$

PROOF. The places where x and y differ are exactly the places where $x + y$ has a 1. □

LEMMA 7.2. *For binary codes $d(x, y) = w(x) + w(y) - 2w(x \cap y)$*

PROOF. LHS counts the places where $x_i = 1$ and $y_i = 0$, or vice versa. Now $w(x) + w(y)$ counts all of these, and also counts the places where $x_i = y_i = 1$, twice. □

Construction. 'Overall parity check' for binary codes.

If $w(x)$ is even, extend $x = x_1 x_2 \dots x_n$ to $\hat{x} = x_1 x_2 \dots x_n 0$ of length $n + 1$; if $w(x)$ is odd, extend to $\hat{x} = x_1 x_2 \dots x_n 1$. Note that all weights in the new code are even. Hence by the above lemma, all distances in the new code are even.

THEOREM 7.3. *If d is odd and C is a $2 - (n, M, d)$-code, then this construction gives a $2 - (n + 1, M, d + 1)$-code.*

PROOF. The minimum distance of the new code is an even number between d and $d + 1$. □

The converse also holds: if D is a $2 - (n + 1, M, d + 1)$-code, with d odd, choose two words in D at distance $d + 1$ apart, and choose a coordinate where they differ. Remove this coordinate from all words in D, to get a $2 - (n, M, d)$-code.

COROLLARY 7.4. *$A_2(n,d) = A_2(n+1,d+1)$ if d is odd.*

EXAMPLE 7.1. The $2-(5,4,3)-$*code* $\{00000, 01101, 10110, 11011\}$ gives rise to the $2-(6,4,4)$-code $\{00000, 011011, 101101, 110110\}$. Also, since $A_2(5,3) = 4$ we have $A_2(6,4) = 4$.

Sphere-packing. If C is a t-error-correcting code, then the set of points at distance $\leq t$ from one codeword cannot intersect the set of points at distance $\leq t$ from any other codeword. (WHY NOT?) In ordinary geometry, the set of points at distance $\leq t$ from a point c is called the sphere of radius t and centre c. We use the same name here.

DEFINITION 7.2. $S_t(c) = \{v \in F_q^n | d(c,v) \leq t\}$.

How many points are in $S_t(c)$? Well, to get a point at distance i from c we have to change i of its coordinates: there are $\binom{n}{i}$ ways to choose the i coordinates to change, and for each one, we change the given coordinate to any one of the other $q-1$ elements of the alphabet. Thus there are $\binom{n}{i}(q-1)^i$ words at distance i from c. So the total number of vectors in $S_t(c)$ is

$$\sum_{i=0}^{t} \binom{n}{i} (q-1)^i.$$

Therefore, if C has M codewords, and the spheres of radius t centred on codewords do not overlap, then there are

$$M \sum_{i=0}^{t} \binom{n}{i} (q-1)^i$$

vectors in the union of all these spheres. BUT there are only q^n vectors altogether, so

THEOREM 7.5. $M \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n$

Restating this, since $d(C) \geq 2t+1$, we have

THEOREM 7.6. *If there is a $q-(n,M,2t+1)$-code, then*

$$M \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i} (q-1)^i}.$$

This important inequality is called the sphere-packing bound. For binary codes it simplifies to

$$M \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}}.$$

This bound is not always very good. Roughly speaking, if $t < \frac{1}{4}n$ it is reasonable. If t is large, then in some sense the spheres are big and don't fit into the space very well.

EXAMPLE 7.2. Binary codes with minimum distance 3 (i.e. $t = 1$) give

$$M \leq \frac{2^n}{1+n}.$$

If $n = 3$, this gives $M \leq 2$, and $\{000, 111\}$ meets the bound.

If n = 4, then $M \leq 3$, but there is no $2-(4,3,3)$-code.

If n = 5, then $M \leq 5$, but, as we have seen, there is no $2-(5,5,3)$-code.

If n = 6, then $M \leq 9$. In fact, no such code exists: $A_2(6,3) = 8$.

If n = 7, then $M \leq 16$. We shall construct a $2-(7,16,3)$-code later.

CHAPTER 8

Perfect codes.

If a $q-(n, M, d)$ code C satisfies

$$M = \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i} (q-1)^i}$$

then C is called perfect. Perfect codes are very rare.

EXAMPLE 8.1. Binary repetition codes of odd length $n = 2t+1$ are perfect.

$$C = \{00\ldots0, 11\ldots1\}$$
$$S_t(00\ldots0) = \{v \in F_2^n | w(v) \leq t\}$$
$$St(11\ldots1) = \{v \in F_2^n | w(v) \geq t+1\}$$

Thus F_2^n is the disjoint union of the two spheres.

EXAMPLE 8.2. A code with only one word is perfect-but cannot send any messages!

EXAMPLE 8.3. A code consisting of all words ($C = F_q^n$) is perfect-but cannot detect any errors!

In order to construct a nontrivial perfect code we first define a balanced block design (or (b, v, r, k, λ)-design): a set S of v points (or varieties) with a collection of b blocks, such that each block is a set of k points, each point is in exactly r blocks, and every pair of points is in exactly λ blocks.

EXAMPLE 8.4. $S = \{0, 1, 2, 3, 4, 5, 6\}$ with blocks {0, 1, 3}, {1, 2, 4}, {2, 3, 5}, {3, 4, 6}, {4, 5, 0}, {5, 6, 1}, {6, 0, 2}. (i.e. $\{n, n+1, n+3\}$ reading modulo 7.) We have $v = 7, b = 7, k = 3$ and check that $r = 3, \lambda = 1$. So this is a (7, 7, 3, 3, 1)-design. This is also called the Fano plane or projective plane of order 2. [In general, a projective plane of order n is a $(n^2+n+1, n^2+n+1, n+1, n+1, 1)$-design.]

From this example we construct a perfect code (actually a small Hamming code). Label 7 coordinates $0, 1, \ldots, 6$ and take

$$a_1 = 1101000$$

$$a_2 = 0110100$$
$$\vdots$$
$$a_7 = 1010001$$

corresponding to the 7 blocks, and their complements

$$b_1 = 0010111$$
$$\vdots$$
$$b_7 = 0101110$$
$$0 = 0000000$$
$$1 = 1111111$$

So $M = 16, n = 7$ and if we can show that $d = 3$ (i.e. $t = 1$), then

$$\frac{2^n}{\sum_{i=0}^{t}\binom{n}{i}} = \frac{2^7}{1+7} = 16 = M$$

so this code is perfect.

To prove $d = 3$. Clearly $d(0,x) = w(x) \geq 3$ for all $x \in C$, and $d(1,x) \geq 3$, and $d(a_i, b_i) = 7$. For the other cases use

$$d(x,y) = w(x) + w(y) - 2w(x \cap y)$$

Now $\lambda = 1$ implies $w(a_i \cap a_j) = 1$, so

$$d(a_i, a_j) = 3 + 3 - 2 = 4$$

so $w(a_i \cap bj) = 2$ (complement inside a_i), and

$$d(a_i, b_j) = 3 + 4 - 2 \cdot 2 = 3$$

so $w(b_i, b_j) = 2$(complement inside b_j), and

$$d(b_i, b_j) = 4 + 4 - 2 \cdot 2 = 4$$

More about balanced block designs.

1. Count the number of pairs consisting of a block and a point in the block, in two ways.

(a) There are b blocks, each with k points, so the number of these pairs is bk.

(b) There are v points, each contained in r blocks, so the number of these pairs is vr. Therefore $bk = vr$.

2. Count the number of configurations consisting of a block and a pair of points in the block, in two ways.

(a) There are b blocks, each containing k points, so containing $k(k-1)/2$ pairs of points. Thus there are $bk(k-1)/2$ of these configurations.

(b) There are v points, so $v(v-1)/2$ pairs of points. Each pair of points is contained in λ blocks, so there are $v(v-1)\lambda/2$ of these configurations. Therefore $bk(k-1) = v(v-1)\lambda$, and substituting in $bk = vr$ we obtain $vr(k-1) = v(v-1)\lambda$, and therefore $r(k-1) = (v-1)\lambda$.

COROLLARY 8.1. *It is sufficient to specify v, k and λ.*

PROOF. For we can calculate $r = (v-1)\lambda/(k-1)$ and $b = vr/k = v(v-1)\lambda/k(k-1)$. □

Notation. Since the parameters b and r can be calculated from v, k, and λ, a (b, v, r, k, λ)- design is often called a (v, k, λ)-design.

CHAPTER 9

Special types of designs.

If $b = v$ (and therefore $r = k$) the design is called symmetric. Any symmetric $(n2 + n + 1, n + 1, 1)$-design is called a projective plane of order n. For example the design constructed above is a symmetric (7, 3, 1)-design, so is a projective plane of order 2.

Generalisation. If every set of t points is in exactly λ blocks, we have a $2 - (v, k, \lambda)$- design. This generalises the case $t = 2$ which we have looked at above.

Finite fields.

EXAMPLE 9.1. Integers modulo p, for p prime. E.g. p = 2

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

E.g. $p = 3$

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

But for $p = 4$, Z_p is not a field:

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

because 2 does not have an inverse. There is a field with four elements, defined by

+	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

×	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

This is called the Galois field of order 4, denoted $GF(4)$.

In fact

THEOREM 9.1. *Every finite field has prime-power order p^n (p prime, $n > 0$).*

THEOREM 9.2. *For every such prime-power p^n, there is a unique field of order p^n, called the Galois field, denoted $GF(p^n)$.*

EXAMPLE 9.2. $GF(11) = Z_{11}$, the integers modulo 11.

a	0	1	2	3	4	5	6	7	8	9	X
a^{-1}	-	1	6	4	3	9	2	8	7	5	X

To construct inverses for such small fields, use trial and error. For large fields, use Euclid's algorithm: If $a \in GF(p)$, with $0 < a < p$, then p and a are coprime, so Euclid's algorithm finds integers x and y such that

$$1 = \gcd(a, p) = ax + py$$

so $ax \equiv 1(\mod p)$, so x is the inverse of a in $GF(p)$.

a	0	1	2	3	4	5	6	7	8	9	X
a^2	0	1	4	9	5	3	3	5	9	4	1

We can solve some quadratic equations using square roots in $GF(11)$:

$$\sqrt{1} = +1 = 1 \text{ or } X$$

$$\sqrt{3} = 5 \text{ or } 6$$

$$\sqrt{2} \text{ does not exist in } GF(11)$$

The ISBN code.

International Standard Book Number is a length 10 code over $GF(11)$, with minimum distance 2. Each book has a number $a_1a_2a_3...a_9$ assigned to it, and then a check-digit a_X is calculated by

$$a_X \equiv a_1 + 2a_2 + 3a_3 + \cdots + 9a_9 (\mod 11)$$

Therefore

$$a_1 + 2a_2 + 3a_3 + \cdots + 9a_9 + Xa_X \equiv 0(\mod 11)$$

E.g. ISBN 0-387-96617-X can be checked:

$$0+2\cdot 3+3\cdot 8+4\cdot 7+5\cdot 9+6\cdot 6+7\cdot 6+8\cdot 1+9\cdot 7+X\cdot X \equiv$$

$$0+6+2+6+1+3+9+8+8+1 \equiv 0(\mod 11).$$

THEOREM 9.3. *The ISBN code detects single errors*

PROOF. If $a_1a_2...a_X$ is the correct ISBN, and a_i is changed to k, then we calculate the checkformula:

$$a_1+2a_2+\cdots+(i-1)a_i-1+i(a_i-a_i+k)+\cdots+Xa_X \equiv i(-a_i+k)(\mod 11).$$

Now $i \not\equiv 0(\mod 11)$ and $k - a_i \not\equiv 0(\mod 11)$, so $i(k - a_i) \not\equiv 0(\mod 11)$ because $GF(11)$ is a field. Therefore the error has been detected. □

Notice that the error cannot be corrected: we know what $i(k - a_i)$ comes to, but we cannot solve for the two unknowns, i and a_i.

THEOREM 9.4. *The ISBN code detects transposition errors (i.e. errors when two digits are transposed, or interchanged).*

PROOF. If a_j and a_k are swapped in the ISBN $a_1a_2...a_X$, we calculate the check formula as $a_1 + 2a_2 + \cdots + k(a_k - a_k + a_j) + \cdots + j(a_j - a_j + a_k) + \cdots + Xa_X \equiv (k - j)(a_j - a_k)(\mod 11)$. Now $j \not\equiv k(\mod 11)$ and $a_j \not\equiv a_k(\mod 11)$, so $k - j \not\equiv 0$ and $a_j - a_k \not\equiv 0$, so $(k - j)(a_j - a_k) \not\equiv 0(\mod 11)$, and the error has been detected. □

The ISBN code can also be used to reinstate an illegible digit. HOW?

CHAPTER 10

Linear codes.

Vector spaces over finite fields obey the same rules as vectors spaces over R or C.

$$V(n,q) = GF(q)^n = \{(x_1, ..., x_n) | x \in GF(q)\}$$

with operations of addition

$$(x_1, \cdots, x_n) + (y_1, \cdots, y_n) = (x_1 + y_1, \cdots, x_n + y_n)$$

and scalar multiplication

$$\lambda \cdot (x_1, \cdots, x_n) = (\lambda x_1, \cdots, \lambda x_n)$$

All the vector space axioms that you already know are satisfied. But $V(n,q)$ is finite, so we can ask how many vectors are in $V(n,q)$? Well, there are q choices for x_1, and q choices for $x_2, \cdots$, making q^n possibilities in all. Other vector space concepts we need: a subspace of $V(n,q)$ is any non-empty subset closed under addition and scalar multiplication. Vectors $v_1, v_2, \cdots, v_k$ are linearly independent if $\lambda_1 v_1 + + \lambda_k v_k = 0$ implies that all $\lambda_i = 0$. The set

$$\{\lambda_1 v_1 + + \lambda_k v_k | \lambda_i \in GF(q)\}$$

of all linear combinations of $v_1, \cdots, v_k$ is the subspace spanned by $v_1, \cdots, v_k$. If $v_1, \cdots, v_k$ span a subspace C, and $v1, \cdots, v_k$ is a linearly independent set of vectors, then it is a basis for C, and k is the dimension of C. Since

$$C = \{\lambda_1 v_1 + \cdots + \lambda_k v_k\}$$

and there are q choices for each λ_i, we have q^k vectors in C, and they are distinct because $v_1, \cdots, v_k$ is a linearly independent set.

EXAMPLE 10.1. $V(3,2) = \{000, 001, 010, 011, 100, 101, 110, 111\}$. The subset {000, 001, 110, 111} is a subspace of dimension 2. Ignoring the zero vector, we have 7 vectors and 7 subspaces of 3 vectors each. Labelling the 7 vectors modulo 7 in the order 111, 110, 011, 001, 101, 010, 100 shows that the 2-dimensional subspaces correspond to the blocks $\{0,1,3\}, \cdots, \{6,0,2\}$ of the Fano plane. Thus we have an alternative description of the projective plane of order 2.

Remark: it is a fact that this is the only projective plane of order 2. Can you prove this? For larger fields, we can make a projective plane of order q from $V(3,q)$, by taking the 1-dimensional subspaces as points and the 2-dimensional subspaces as blocks (sometimes called lines).

Linear codes.

A linear code is any subspace (not subset!) of $V(n,q)$. If $C \leq V(n,q)$ has dimension k, we call C a linear [n, k, d]-code, where $d = d(C)$ is the minimal distance of C.

NOTE: square brackets! Since C has q^k vectors in it, it is a (n, q^k, d)-code. So every linear [n, k, d]-code over F_q is a (n, q^k, d)-code, but not conversely. As before d is the minimal distance

$$d = d(C) = min\{d(x,y)|x,y \in C, x \neq y\}$$

but the linearity makes it much easier to calculate d. Define the weight $w(x)$ of a vector $x = (x_1, \cdots, x_n)$ to be the number of non-zero coordinates x_i. We know that

$$d(x,y) = w(x-y),$$

since the coordinates $x_i \neq y_i$ iff $x_i - y_i \neq 0$, and we deduce

THEOREM 10.1. *If C is a linear code, then $d(C) = w(C)$, where*

$$w(C) = \min\{w(x)|x \in C, x \neq 0\}$$

is the minimum weight of C.

PROOF. There are two words $x, y \in C$ with

$$d(C) = d(x,y) = d(x-y,0) = w(x-y) \geq w(C).$$

Conversely, there is a word $z \in C$ with

$$w(C) = w(z) = d(z,0) \geq d(C).$$

□

Advantages of linear codes.

1. Can be defined by a basis of k words, rather than a list of all q^k words.

2. Easier to calculate $d(C)$: just check $|C|$ weights instead of $|C|(|C|-1)/2$ distances.

3. Easier to encode and decode, using linear algebra (see later).

Disadvantages of linear codes.

1. Can only be defined over alphabets of prime-power order. But we can overcome this by using a subset of the code (as in the ISBN code).

2. May only get weaker codes. For example there is a binary (8, 20, 3)-code, but if linear, only (8, 16, 3). But the extra efficiency of these non-linear codes is relatively small, and usually outweighed by the extra complications.

CHAPTER 11

A generator matrix for a linear code.

DEFINITION 11.1. A generator matrix for a linear code C is any matrix whose rows form a basis of C.

EXAMPLE 11.1. $C = \{000, 011, 101, 110\}$ has a generator matrix

$$G_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

as well as

$$G_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \cdots$$

It makes sense to choose a generator matrix which is in echelon form. E.g.

$$G_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

If we end up with a generator matrix of the form $(I_k|A)$ we say this is in standard form. BUT not all linear codes have a generator matrix in standard form. WHY NOT? To get a generator matrix in standard form, it may be necessary to permute some columns, which will change the code to an equivalent code.

EXAMPLE 11.2.

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

by swapping columns 2 and 4, but cannot be put into standard form by row operations alone.

Equivalence of linear codes.

Two linear codes are equivalent if one can be obtained from the other by a combination of

1. permuting the coordinates in all codewords simultaneously (i.e. permuting the columns of a generator matrix).

2. multiplying any particular coordinates by scalars (i.e. multiplying columns of a generator matrix by scalars)

Note that this definition of equivalence for linear codes is more restrictive than that for arbitrary codes.

EXAMPLE 11.3. Over $GF(5)$

$$G1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 4 & 1 \end{pmatrix} \to \begin{pmatrix} 2 & 1 & 3 & 0 \\ 4 & 0 & 1 & 1 \end{pmatrix} \to \begin{pmatrix} 4 & 3 & 1 & 0 \\ 3 & 0 & 2 & 2 \end{pmatrix}$$

are generator matrices for equivalent codes.

We always work with a linear code C by specifying a generator matrix G. Suppose C has dimension k, and rows of G are the vectors $r_1, \cdots, r_k$. Then C contains the q^k vectors

$$\lambda_1 r_1 + \lambda_2 r_2 + \cdots + \lambda_k r_k$$

corresponding to message vectors $(\lambda_1, \lambda_2, \cdots, \lambda_k)$. The encoded vector

$$\sum_{i=1}^{k} \lambda_i r_i = (\lambda_1 \cdots \lambda_k) \begin{pmatrix} r_1 \\ \vdots \\ r_k \end{pmatrix} = \lambda G$$

where λ is the message vector and G is the generator matrix. If G is in standard form $(I_k|A)$ then the encoded vector is $(\lambda_1, \cdots, \lambda_k, \cdots)$.

EXAMPLE 11.4. Over $GF(2)$

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Message $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ is encoded as

$$(\lambda_1\lambda_2\lambda_3\lambda_4) \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$= (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_1 + \lambda_2 + \lambda_3, \lambda_2 + \lambda_3 + \lambda_4, \lambda_1 + \lambda_2 + \lambda_4)$$

which consists of the message followed by three check bits. Similarly, if a message $(x_1, x_2, \cdots, x_7)$ is sent, we know it should satisfy the equations

$$x_1 + x_2 + x_3 = x_5$$
$$x_2 + x_3 + x_4 = x_6$$
$$x_1 + x_2 + x_4 = x_7$$

so these equations can be used to check the received vector.

Decoding.

EXAMPLE 11.5.

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

over $GF(2)$, so that $C = \{0000, 1011, 0101, 1110\}$. An error in the first place gives one of the vectors 1000, 0011, 1101, 0110, (i.e. an element of the coset $1000 + C$. An error in the second place gives 0100, 1111, 0001 or 1010. An error in the third place gives 0010, 1001, 0111 or 1100. These four cosets (including the zero coset C itself) account for all the 16 vectors. An error in the fourth place gives 0001, 1010, 0100 or 1111, a coset we have already seen.

CHAPTER 12

Slepian standard array.

Required background on cosets.

DEFINITION 12.1. If $C \leq V(n,q)$, the coset $a+C = \{a+x | x \in C\}$.

LEMMA 12.1. *If $b \in a+C$ then $b+C = a+C$*

PROOF. If $b \in a+C$, then $b = a+x$ for some $x \in C$. Every element of $b+C$ is of the form $b+y$, for some $y \in C$, so $b+y = a+x+y \in a+C$ since $x+y \in C$. Hence $b+C \subseteq a+C$. Similarly, every element of $a+C$ is of the form $a+z$ with $z \in C$, so $a+z = b-x+z \in b+C$, since $z-x \in C$. Hence $a+C \subseteq b+C$, and therefore $a+C = b+C$. □

THEOREM 12.2. *If $C \leq V(n,q)$ has dimension k, then*

1. *every vector is in some coset of C*
2. *every coset has q^k vectors, and so there are q^{n-k} cosets*
3. *any two cosets are disjoint*

PROOF. 1. $v \in v+C$

2. There are q^k values of $x \in C$ giving rise to q^k elements $a+x \in a+C$. These are distinct because if $a+x = a+y$ then $x = y$.

3. If $x \in (a+C) \cap (b+C)$ then $a+C = x+C = b+C$. □

EXAMPLE 12.1. (Continued from above)

	Coset leaders:			
Code C:	0000	1011	0101	1110
Coset 1000 + C:	1000	0011	1101	0110
Coset 0100 + C:	0100	1111	0001	1010
Coset 0010 + C:	0010	1001	0111	1100

Given any received vector v, we find the coset leader e of the coset $v+C$ (so $e+C = v+C$), and decode as $v-e \in C$. In this example, an error in the last coordinate results in the received vector being decoded wrongly.

Such an array is called a Slepian standard array, after D. Slepian, and can be constructed as follows:

1. List the codewords across the top, starting with $000\cdots0$.
2. Pick a vector not already in the table, of minimum weight, and put it in the first column. Then add it to each codeword in turn.

3. Repeat Step 2 until the table is full. Note that each entry $a+x$ is the sum of the coset leader a and the codeword x at the top of the column. To use a Slepian standard array for decoding, simply look up the received vector in the table, and decode as the codeword at the top of the column. The errors which are corrected are precisely the coset leaders. If a coset contains more than one vector of minimum weight, then there is no obvious choice of coset leader. Often one would use the array to detect such errors, and not try to correct them. This compromise scheme is called incomplete decoding. The probability of decoding a received vector correctly is just the probability that the actual error is one of the coset leaders. Now the probability that the error is a particular vector of weight i is $p^i(1-p)^{n-i}$, assuming a binary symmetric channel. So if there are α_i coset leaders of weight i, then the total probability is

$$P_{corr}(C) = \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}.$$

EXAMPLE 12.2. (continued from above) Here $\alpha_0 = 1, \alpha_1 = 3, \alpha_2 = 0$ etc so

$$P_{corr}(C) = (1-p)^4 + 3p(1-p)^3 = (1+2p)(1-p)^3.$$

CHAPTER 13

Dual codes.

A code of length n and dimension k can send k message digits in each block of n digits sent. Thus the message occupies a proportion k/n of what is sent. This fraction k/n is called the rate of the code. Shannon's theorem is a theoretical limit to the rate at which messages can be sent:

THEOREM 13.1. *For a binary symmetric channel, and any* $R < 1 + p\log_2 p + (1-p)\log_2(1-p)$, *and any* $\varepsilon > 0$, *there is a code* C *of rate* $> R$ *such that* $P_{err}(C) < \varepsilon$, *where* $P_{err}(C) = 1 - P_{corr}(C)$ *is the word error probability of* C.

The expression $1+p\log_2 p+(1-p)\log_2(1-p)$ is called the capacity of the channel. It varies from 1 when $p = 0$ (i.e. no errors, so no error-detection required) to 0 when $p = 1/2$ (i.e. 0s and 1s transmitted at random, so no message can get through).

The dual code. If $u = u_1 \cdots u_n \in V(n,q)$ and $v = v_1 \cdots v_n \in V(n,q)$ define the inner product of u and v as

$$u.v = \sum_{i=1}^{n} u_i v_i.$$

Clearly $u.v = v.u$ and $(\lambda u + v).w = \lambda u.w + v.w$. If $u.v = 0$ we say u and v are orthogonal. If $C \leq V(n,q)$, define

$$C^{\perp} = \{v \in V(n,q) | v.u = 0 \text{ forall } u \in C\}.$$

EXAMPLE 13.1. If $C = \{000, 111\}$ over $GF(2)$, then

$$C^{\perp} = \{v_1v_2v_3 | v_1 + v_2 + v_3 = 0\} = \{000, 011, 101, 110\}.$$

EXAMPLE 13.2. If $C = \{0000, 1100, 0011, 1111\}$ then $C^{\perp} = C$.

LEMMA 13.2. *If* v *is orthogonal to every vector in a basis for* C, *then* v *is orthogonal to every vector in* C.

PROOF. If $u \in C$ and $r_1, \cdots, r_k$ is a basis for C, then $u = \sum_{i=1}^{k} \lambda_i r_i$ for some $\lambda_i \in GF(q)$, so

$$v.u = v.(\sum_{i=1}^{k} \lambda_i r_i) = \sum_{i=1}^{k} \lambda_i v.r_i = 0$$

□

To check if $v \in C^\perp$ therefore, it suffices to check that v is orthogonal to the rows of a generator matrix G for C, i.e. $v \in C^\perp \Leftrightarrow vG^T = 0$.

LEMMA 13.3. *$C^\perp$ is a linear code.*

PROOF. If $v, w \in C^\perp$ and $u \in C$, then $(\lambda v + w).u = \lambda v.u + w.u = 0$ so $\lambda v + w \in C$. □

THEOREM 13.4. *If $C \leq V(n,q)$ has dimension k, then $C^\perp$ has dimension $n-k$.*

PROOF. $C^\perp$ is the solution set of k independent homogeneous linear equations $((x_1, \cdots, x_n).r_i = 0$ for $1 \leq i \leq k)$ in n unknowns $(x_1, \cdots, x_n)$. □

COROLLARY 13.5. $(C^\perp)^\perp = C$

PROOF. Every vector in C is orthogonal to every vector in $C^\perp$, so $C \subseteq (C^\perp)^\perp$. But $(C^\perp)^\perp$ has dimension $n-(n-k) = k = \dim C$. □

EXAMPLE 13.3. If $C = \{000, 011, 101, 110\}$ then $C^\perp = \{000, 111\}$.

DEFINITION 13.1. A generator matrix for $C^\perp$ is called a parity-check matrix for C, and usually denoted H.

By definition, each row r_i of H satisfies $r_i G^T = 0$, so $HG^T = 0$ (the zero $k \times (n-k)$ matrix), or $GH^T = 0$ (the zero $(n-k) \times k$ matrix). Indeed, $v \in C \Leftrightarrow vH^T = 0$. Each individual row of H (or column of H^T) may be thought of as a parity-check.

EXAMPLE 13.4. If

$$H = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

then every $v = v_1 v_2 v_3 v_4 \in C$ must satisfy two paritychecks

$$v_1 + v_2 = 0 \text{ and } v_3 + v_4 = 0.$$

CHAPTER 14

Hamming codes.

If C has a generator matrix G in standard form, then there is a simple way to write down a parity-check matrix for C:

If $G = (I_k | A)$ then there is a parity-check matrix $H = (-A^T | I_{n-k})$, i.e.

$$G = \left(\begin{array}{cccc|ccc} 1 & 0 & \cdots & 0 & a_{11} & \cdots & a_{1,n-k} \\ 0 & 1 & \cdots & 0 & a_{21} & \cdots & a_{2,n-k} \\ & & \ddots & & & \ddots & \\ 0 & 0 & \cdots & 1 & a_{k1} & \cdots & a_{k,n-k} \end{array}\right)$$

PROOF. ith row of G is $(0, \cdots, 0, 1, 0, \cdots, 0, a_{i1}, a_{i2}, \cdots, a_{i,n-k})$ with the 1 in the ith coordinate; and jth row of H is

$$(-a_{1j}, -a_{2j}, \cdots, -a_{kj}, 0, \cdots, 0, 1, 0, \cdots, 0)$$

with the 1 in the $(j+k)$th coordinate. Therefore the inner product of these two vectors is $-a_{ij} + a_{ij} = 0$. So the rows of H are in $C^\perp$. Moreover, they span an $(n-k)$-space, so they span the whole of $C^\perp$. □

EXAMPLE 14.1. If

$$G = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

over $GF(3)$ (in standard form), then

$$H = \begin{pmatrix} 2 & 2 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{pmatrix}$$

in standard form.

EXAMPLE 14.2. If

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

over $GF(2)$ (in standard form), then

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

in standard form.

Syndrome decoding. If C is a code with a parity-check matrix H, then we know

$$x \in C \Leftrightarrow xH^T = 0$$

(or $Hx^T = 0$). We can use this to determine which coset of C a given vector is in:

If $b \in a + C$ then $b = a + x$ for some $x \in C$, so

$$bH^T = aH^T + xH^T = aH^T.$$

Conversely, if $bH^T = aH^T$ then $(b - a)H^T = 0$ so $b - a \in C$, and $b \in a + C$. So

$$b \in a + C \Leftrightarrow bH^T = aH^T.$$

I.e. $b + C = a + C \Leftrightarrow bH^T = aH^T$. This means that there is a one-to-one correspondence between the cosets $a + C$ and the syndromes aH^T.

DEFINITION 14.1. $S(a) = aH^T$ is called the syndrome of a.

To use this, we first calculate the syndromes of all the coset leaders, and make a table:

EXAMPLE 14.3. (Continued from above)

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

over GF(2), and

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix},$$

or

$$H = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Coset leaders a	Syndromes aH^T
0000	00
1000	11
0100	01
0010	10

This is the syndrome look-up table.

Now given a received vector y we calculate $S(y) = yH^T$. We look this up in the table to find which of the aH^T it is. Then $yH^T = aH^T$ so $y \in a + C$, where a is the leader of the coset $a + C$, so y is decoded as $y - a \in C$.

EXAMPLE 14.4. (Continued from above) If $y = 1101$ is received, then

$$yH^T = (1101)\begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (11).$$

Look this up in the syndrome look-up table, to find $a = 1000$ and decode as $y - a = 0101$.

If $d(C) = 2t + 1$ or $2t + 2$, so that C is a t-error-correcting code, we only need to calculate the syndromes of the errors we are going to correct. Usually this means all vectors of weight $\leq t$. Then we use the table for incomplete decoding: If yH^T is in the syndrome table, corresponding to error vector a, then decode as $y - a$; otherwise, yH^T must be the syndrome of some other error vector (i.e. of weight $> t$), so we have detected $> t$ errors.

EXAMPLE 14.5.

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

over GF(2). So

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix},$$

and we calculate the syndromes of the vectors of weight 1:

e	eH^T
10000	110
01000	101
00100	100
00010	010
00001	001

Notice that the syndromes eH^T are just the rows of H^T. WHY? Now the five syndromes eH^T are all different, so the five vectors e are all in different cosets. Hence all single errors can be corrected. E.g. $y = 11111 \Rightarrow yH^T = 100$ so decode as $y - 00100 = 11011$. E.g. $y = 00111 \Rightarrow yH^T = 111$ which is not in the table, so we have detected ≥ 2 errors.

The ISBN code revisited.
C has parity-check matrix

$$H = (123456789X)$$

over $GF(11)$, since $a_1a_2\cdots a_X \in C \Leftrightarrow 1a_1 + 2a_2 + \ + Xa_X = 0$. The syndrome eH^T of an error vector $e = (0,\cdots,0,k,0,\cdots,0)$, with k in the jth coordinate, is j_k, which is non-zero, so such errors can be detected. But not corrected, for even if we know the value of j_k, we do not know the values of j and k individually.

EXAMPLE 14.6. Suppose C over $GF(11)$ has parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & X \end{pmatrix}.$$

I claim that C can correct single errors and detect transposition errors.

PROOF. Calculate the syndromes of the single errors

$$e = (0,\cdots,0,k,0,\cdots,0),$$

with k in the jth coordinate. We have $eH^T = (k, jk)$, with $k \neq 0$ and $jk \neq 0$. So if $yH^T = (A,B)$ with $A \neq 0 \neq B$ we can solve for $k = A$ and $jk = B \Rightarrow j = BA^{-1}$. A transposition error is $e = (0,\cdots,0,k,0,\cdots,0,-k,0,\cdots,0)$, with k in the ith place and $-k$ in the jth place. Hence $eH^T = (0, k(i-j)) \neq (0,0)$ so these errors are detected. □

Hamming codes. If C has length n and dimension k then (if the generator matrix is in standard form) each codeword consists of k message digits $x_1\cdots x_k$ and $n-k$ ('redundant') check digits $x_{k+1}\cdots x_n$. The number $r = n-k$ is called the redundancy of C. Note that r is the number of rows in a parity-check matrix for C. A code can be defined by specifying a parity-check matrix (PCM for short), rather than a generator matrix. (E.g. the definition of the ISBN code above.) The binary Hamming code of redundancy r is defined by a PCM whose columns are all non-zero vectors of length r (over $GF(2)$). The total number of column vectors of length r is 2^r, so the number of non-zero vectors is $2^r - 1$. Therefore the Hamming code has length $n = 2^r - 1$ and dimension $k = n - r = 2^r - 1 - r$.

EXAMPLE 14.7. If $r = 2$ we have

$$H = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix},$$

so $C = \{000, 111\}$, the binary repetition code of length 3. We have $n = 2^2 - 1 = 3$ and $k = n - r = 3 - 2 = 1$. If $r = 3$ we have

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix},$$

and $n = 2^3 - 1 = 7, k = n - r = 7 - 3 = 4$. DO YOU RECOGNISE THIS CODE?

Warning. The order of the columns is not specified in this definition, so 'the' Hamming code is not well-defined. (It is only defined up to equivalence.) Usually we order the columns as though they were binary numbers, e.g.

$$\begin{matrix} 1 \\ 1 \\ 0 \end{matrix} \rightarrow 110 \rightarrow 6$$

is the 6th column of H. Let us put H into standard form

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

and deduce a generator matrix in standard form

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

EXERCISE 14.1. Prove that this code is equivalent to the Hamming code.

So this code is a perfect 1-error-correcting code (i.e. with minimal distance 3). In fact this is true for all the binary Hamming codes.

PROOF. It is enough to show that C has no words of weight 1 or 2, and then to check equality in the sphere-packing bound. If $v \in C$ then $vH^T = 0$. But $(00100)H^T$ is the jth row of H^T, which is non-zero by definition, so no vector of weight 1 is in the code. Similarly, if v has weight 2, then vH^T is the sum of two rows of H^T. But if this were 0, then these two rows of H^T would be equal, contradicting the construction of H. So C has no words of weight 1 or 2, i.e. $d(C) \geq 3$. Now $M = 2n \quad r, q = 2, d = 3$, so $t = 1$ and therefore

$$\frac{q^n}{\sum_{i=0}^{t}\binom{n}{i}(q-1)^i} = \frac{2^n}{1+n} = \frac{2^n}{1+(2^r-1)} = 2^{n-r} = M$$

so C is perfect. □

Decoding with a Hamming code. Note that the rows of H^T are just the binary numbers from 1 to $2^r - 1$, in order. So if $e = (0, \cdots, 0, 1, 0, \cdots, 0)$, with the 1 in the jth coordinate, then $S(e) = eH^T$ is the jth row of H^T, which is just the binary representation of the integer j. Thus we can read off j immediately, and hence correct the error.

ALGORITHM 14.1. Calculate $S(y)$ for the received vector y. If $S(y) = 0$, decode as y. Otherwise, $S(y)$ represents a binary number j, with $1 \leq j \leq n = 2^r - 1$, and we correct y by changing the jth bit (from 0 to 1 or vice versa).

CHAPTER 15

The extended binary Hamming code.

The extended binary Hamming code is obtained by adding an overall parity check. Hence its parameters are

$$M = 2^{m-r}, \text{ where } m = 2^r - 1$$
$$n = m + 1 = 2^r$$
$$d = 4 \text{ (since it is even)}$$

Therefore it can correct all single errors and detect all double errors.

LEMMA 15.1. *Adding an overall parity check to a binary linear [n, k]-code C gives a binary linear [n + 1, k]-code $\hat{C}$.*

PROOF. For any $x_1x_2\cdots x_nx_{n+1}$ and $y_1y_2\cdots y_ny_{n+1} \in \hat{C}$ we have $x_1x_2\cdots x_n$ and $y_1y_2\cdots y_n \in C$, so $(x_1 + y_1, x_2 + y_2, \cdots, x_n + y_n) \in C$. Moreover

$$x_{n+1} = \sum_{i=1}^{n} x_i \text{ and } y_{n+1} = \sum_{i=1}^{n} y_i$$

so

$$x_{n+1} + y_{n+1} = \sum_{i=1}^{n} (x_i + y_i)$$

and therefore $(x_1 + y_1, x_2 + y_2, \cdots, x_n + y_n, x_{n+1} + y_{n+1}) \in \hat{C}$. □

In particular, the extended binary Hamming code is a linear $[2^r, 2^r - 1 - r, 4]$-code. The words in $\hat{C}$ satisfy all the parity check equations for C, in $x_1, \cdots, x_n$, together with the extra parity check equation

$$x_1 + x_2 + \ + x_n + x_{n+1} = 0.$$

Therefore if C has PCM H, then $\hat{C}$ has a PCM

$$\begin{pmatrix} H & \mathbf{0} \\ 1 & 1 \end{pmatrix}$$

where $\mathbf{0}$ is the zero column vector of height r, and $\mathbf{1}$ is the row vector $(1, 1, \cdots, 1)$ of length n. (Check that this has the right size! NOTE that this PCM is not in standard form.)

Decoding with the extended binary Hamming code. Calculate the syndrome $S(y) = (s_1, \cdots, s_r, s_{r+1})$. If $s_{r+1} = 1$, assume a

single error, and $s_1 s_2 \cdots s_r$ is the binary representation of the error position (with $00\cdots0$ representing the $n+1 = 2^r$). If $s_{r+1} = 0$, there must be an even number of errors, either 0 (in the case $s_1 \cdots s_r = 0 \cdots 0$) or 2 (otherwise).

EXAMPLE 15.1. The [8, 4] extended Hamming code has a PCM

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

If $y = 11011010$ then $S(y) = 1011$, so there is a single error in the 5th coordinate, and we decode as 11010010. If $y = 10101111$ then $S(y) = 1100$ so we detect a double error.

Before we define the q-ary Hamming codes, we prove an important theorem relating the minimum distance of a code to the columns of a PCM.

LEMMA 15.2. *Let C be a code with a PCM H. Then C contains a word of weight d iff some d columns of H are linearly dependent.*

PROOF. Denote the columns of H by $H_1, \cdots, H_n$, so that

$$H^T = \begin{pmatrix} H_1^T \\ \cdots \\ H_n^T \end{pmatrix}.$$

So

$$x_1 \cdots x_n \in C \Leftrightarrow (x_1 \cdots x_n) \begin{pmatrix} H_1^T \\ \cdots \\ H_n^T \end{pmatrix} = 0,$$

i.e. iff $x_1 H_1 + \cdots + x_n H_n = 0$. If $x_1 \cdots x_n$ has weight d then this is a linear dependence among d columns of H (and conversely). □

THEOREM 15.3. *Let C be a linear code with a parity check matrix H. Then $d(C)$ is the minimum number of columns of H which form a linearly dependent set. So $d(C) = d$ iff both*

1. *every set of $d-1$ columns of H is linearly independent, and*
2. *some set of d columns of H is linearly dependent.*

We can use this theorem to construct H one column at a time, to make sure the code has the required minimal distance. For example, to ensure that $d(C) \geq 3$ it is necessary and sufficient to ensure that no two columns of H are linearly dependent, i.e. no two columns of H are scalar multiples of each other. Over $GF(q)$, every non-zero column has

$q-1$ non-zero scalar multiples, so we can only choose one out of each such set of $q-1$ possibilities. E.g. in $V(2,3)$ the $3^2 - 1 = 8$ non-zero vectors come in 4 pairs:

$$\left\{\begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}\right\}, \left\{\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}2\\0\end{pmatrix}\right\}, \left\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix}\right\}, \left\{\begin{pmatrix}2\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix}\right\}.$$

Choosing one from each pair we get a PCM for a Hamming code Ham(2, 3):

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}.$$

In general, Ham(r, q) has a PCM with r rows. Each column therefore is in $V(r,q)$, which has q^r vectors. Therefore there are q^{r-1} non-zero vectors, each in a set of $q-1$ scalar multiples. So we can choose $(q^r-1)/(q-1)$ columns to make H, and get length $n = (q^r-1)/(q-1)$. By construction, $d \geq 3$, and it is easy to see that $d = 3$. HOW? Thus Ham(r, q) is a q-ary linear $[(q^r-1)/(q-1), (q^r-1)/(q-1)-r, 3]$ code.

THEOREM 15.4. *Ham(r, q) is a perfect code.*

PROOF.

$$\frac{q^n}{\sum_{i=0}^{t}\binom{n}{i}(q-1)^i} = \frac{q^n}{1+n(q-1)}$$

$$= \frac{q^n}{1+\frac{q^r-1}{q-1}(q-1)} = \frac{q^n}{1+q^r-1} = \frac{q^n}{q^r} = q^{n-r} = M.$$

□

CHAPTER 16

Decoding with a q-ary Hamming code.

EXAMPLE 16.1. Since every non-zero vector can be multiplied by a (unique) scalar to make the first non-zero coordinate a 1, we can write down a PCM for each Hamming code by just listing all the column vectors whose first non-zero coordinate is 1: E.g. Ham(2, 5) has a PCM

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & 3 & 4 \end{pmatrix}.$$

E.g. Ham(3, 3) has a PCM

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{pmatrix}.$$

Decoding with a q-ary Hamming code. We know that Ham(r, q) is a perfect 1-errorcorrecting code, so the coset leaders are $\lambda e_i = (0, \cdots, 0, \lambda, 0, \cdots, 0)$, and $S(\lambda e_i) = \lambda H_i^T$, where H_i is the ith column of the PCM H. If we have chosen H such that each column has first non-zero coordinate 1, then λ is the first non-zero coordinate of $S(\lambda e_i)$. To decode, we simply find λ and i such that $S(y) = \lambda H_i^T$, and decode as $y - \lambda e_i$.

Shortening a code. A code may be shortened by deleting a column from the parity-check matrix. This converts a code of length n and redundancy r to one of length $n-1$ and redundancy r. So the dimension has been reduced from $n-r$ to $n-r-1$. In general the minimal distance will stay the same. WHY?

... though it may increase. HOW?

The minimal distance of C is the minimum number of columns of H such that there is a linear dependence among that many columns. If we delete a column, we delete all linear dependences which involve that column. Hence that minimum number cannot decrease. But if all the linear dependences of just d columns actually involve the column we have deleted, then that minimum number can (and will) increase. So shortening an $[n, k, d]$ code gives an $[n-1, k-1, d']$ code, where $d' \geq d$.

EXAMPLE 16.2. Ham(2, 11) has PCM

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & X \end{pmatrix}.$$

Deleting the first two columns give a PCM

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & X \end{pmatrix}$$

which defines a code we have seen before. The theorem implies that this code has minimum distance at least 3. In fact, it is exactly 3.

Warning. You can only continue deleting columns of H while the rows of H remain linearly independent. Otherwise, the redundancy of the code decreases. How does shortening a code affect a generator matrix? Suppose H is in standard form, $H = (B|I_r)$, say, and $G = (I_{n-r}| - B^T)$. If we delete the ith column of B, we delete the ith row of $-B^T$. We get

$$\begin{pmatrix} 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & -b_{11} & \cdots & -b_{1r} \\ & \ddots & & & \vdots & & \vdots & & & \vdots \\ 0 & \cdots & 1 & 0 & 0 & \cdots & 0 & -b_{i-1,1} & \cdots & -b_{i-1,r} \\ 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & -b_{i+1,1} & \cdots & -b_{i+1,r} \\ & & \vdots & & & \ddots & \vdots & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 1 & -b_{n-r,1} & \cdots & -b_{n-r,r} \end{pmatrix}$$

This makes the ith column of G zero, and we then delete this zero column. This means we take all codewords with 0 in the ith coordinate, then then delete this coordinate from all the codewords which remain.

CHAPTER 17

Cyclic Codes

Among the first codes used practically were the cyclic codes which were generated using shift registers. It was quickly noticed by Prange that the class of cyclic codes has a rich algebraic structure, the first indication that algebra would be a valuable tool in code design. The linear code C of length n is a cyclic code if it is invariant under a cyclic cyclic code shift:

$$c = (c_0, c_1, c_2 \cdots, c_{n-2}, c_{n-1}) \in C$$

if and only if

$$c = (c_{n-1}, c_0, c_1 \cdots, c_{n-3}, c_{n-2}) \in C.$$

As C is invariant under this single right cyclic shift, by iteration it is invariant under any number of right cyclic shifts. As a single left cyclic shift is the same as $n-1$ right cyclic shifts, C is also invariant under a single left cyclic shift and hence all left cyclic shifts. Therefore the linear code C is cyclic precisely when it is invariant under all cyclic shifts. There are some obvious examples of cyclic codes. The 0-code is certainly cyclic as is $F^n = V(n, 2)$. Less trivially, repetition codes are cyclic. The binary parity check code is also cyclic, and this goes over to the sum-0 codes over any field. Notice that this shift invariance criterion does not depend at all upon the code being linear. It is possible to define nonlinear cyclic codes, but that is rarely done. The history of cyclic codes as shift register codes and the mathematical structure theory of cyclic codes both suggest the study of cyclic invariance in the context of linear codes.

0.1. Basics. It is convenient to think of cyclic codes as consisting of polynomials as well as codewords. With every word

$$a = (a_0, a_1, \cdots, a_i, \cdots, a_{n-2}, a_{n-1}) \in F^n$$

we associate the polynomial of degree less than n

$$a(x) = a_0 + a_1 x + \cdots + a_i x^i + \cdots + a_{n-1} x^{n-1} \in F[x]_n.$$

(We see here why in this chapter we index coordinates from 0 to $n-1$.) If c is a codeword of the code C, then we call $c(x)$ the associated

code polynomial. With this convention, the shifted codeword $\tilde{c}$ has associated code polynomial

$$\tilde{c}(x) = c_{n-1} + c_0 x + c_1 x^2 + \cdots + c_i x^{i+1} + \cdots + c_{n-2} x^{n-1}.$$

Thus $\tilde{c}(x)$ is almost equal to the product polynomial $xc(x)$. More precisely,

$$\tilde{c}(x) = xc(x) - c_{n-1}(x^n - 1).$$

Therefore $\tilde{c}(x)$ also has degree less than n and is equal to the remainder when $xc(x)$ is divided by $x^n - 1$. In particular

$$\tilde{c}(x) = xc(x)(\ \cdot \bmod x^n - 1).$$

That is, $\tilde{c}(x)$ and $xc(x)$ are equal in the ring of polynomials $F[x](\bmod x^n - 1)$, where arithmetic is done modulo the polynomial $x^n - 1$. If $c(x)$ is the code polynomial associated with some codeword c of C, then we will allow ourselves to abuse notation by writing

$$c(x) \in C.$$

Indeed, if $f(x)$ is any polynomial of $F[x]$ whose remainder, upon division by $x^n - 1$, belongs to C then we may write

$$f(x) \in C(\mod x^n - 1).$$

With these notational conventions in mind, we see that our definition of the cyclic code C has the pleasing polynomial form

$$c(x) \in C(\mod x^n - 1) \text{ if and only if } xc(x) \in C(\mod x^n - 1).$$

Since additional shifts do not take us out of the cyclic code C, we have

$$x^i c(x) \in C(\mod x^n - 1),$$

for all i. By linearity, for any $a_i \in F$,

$$a_i x_i c(x) \in C(\mod x^n - 1)$$

and indeed

$$\sum_{i=0}^{d} a_i x_i c(x) \in C(\mod x^n - 1),$$

That is, for every polynomial

$$a(x) = \sum_{i=0}^{d} a_i x_i \in F[x],$$

the product $a(x)c(x)$ (or more properly $a(x)c(x)(\bmod x^n - 1)$) still belongs to C. As linear C is certainly closed under polynomial addition, this says that the polynomial rendering of a cyclic code is precisely an ideal of the ring $F[x](\bmod x^n - 1)$. This correspondence, first noted by Prange, opened the way for the application of algebra to cyclic codes.

THEOREM 17.1. *Let $C \neq \{0\}$ be a cyclic code of length n over F.*
(1) Let $g(x)$ be a monic code polynomial of minimal degree in C. Then $g(x)$ is uniquely determined in C, and

$$C = \{q(x)g(x) | q(x) \in F[x]_{n-r}\},$$

where $r = deg(g(x))$. In particular, C has dimension $n - r$.
(2) The polynomial $g(x)$ divides $x^n - 1$ in $F[x]$.

PROOF. As $C \neq \{0\}$, it contains non-zero code polynomials, each of which has a unique monic scalar multiple. Thus there is a monic polynomial $g(x)$ in C of minimal degree. Let this degree be r, unique even if $g(x)$ is not. By the remarks preceding the theorem, the set of polynomials

$$C_0 = \{q(x)g(x) | q(x) \in F[x]_{n-r}\},$$

is certainly contained in C, since it is composed of those multiples of the code polynomial $g(x)$ with the additional property of having degree less than n. Under addition and scalar multiplication C_0 is an F-vector space of dimension $n - r$. The polynomial $g(x)$ is the unique monic polynomial of degree r in C_0. To prove (1), we must show that every code polynomial $c(x)$ is an $F[x]$- multiple of $g(x)$ and so is in the set C_0. By the Division Algorithm we have

$$c(x) = q(x)g(x) + r(x),$$

for some $q(x), r(x) \in F[x]$ with $\deg(r(x)) < r = \deg(g(x))$. Therefore

$$r(x) = c(x) - q(x)g(x).$$

By definition $c(x) \in C$ and $q(x)g(x)$ is in C_0 (as $c(x)$ has degree less than n). Thus by linearity, the righthand side of this equation is in C, hence the remainder term $r(x)$ is in C. If $r(x)$ was non-zero, then it would have a monic scalar multiple belonging to C and of smaller degree. But this would contradict the original choice of $g(x)$. Therefore $r(x) = 0$ and $c(x) = q(x)g(x)$, as desired.

Next let

$$x^n - 1 = h(x)g(x) + s(x),$$

for some $s(x)$ of degree less than $\deg(g(x))$. Then, as before,

$$s(x) = (-h(x))g(x)(\mod x^n - 1)$$

belongs to C. Again, if $s(x)$ is not zero, then it has a monic scalar multiple belonging to C and of smaller degree than that of $g(x)$, a contradiction. Thus $s(x) = 0$ and $g(x)h(x) = x^n - 1$, as in (2). □

The polynomial $g(x)$ is called the generator polynomial for the code C.

The polynomial $h(x) \in F[x]$ determined by

$$g(x)h(x) = x^n - 1$$

is the check polynomial of C.

Under some circumstances it is convenient to consider $x^n - 1$ to be the generator polynomial of the cyclic code 0 of length n. Then by the theorem, there is a one-to-one correspondence between cyclic codes of length n and monic divisors of $x^n - 1$ in $F[x]$.

EXAMPLE 17.1. Consider length 7 binary cyclic codes. We have the factorization into irreducible polynomials

$$x^7 - 1 = (x-1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Since we are looking at binary codes, all the minus signs can be replaced by plus signs:

$$x^7 + 1 = (x+1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

As there are 3 irreducible factors, there are $2^3 = 8$ cyclic codes (including 0 and F_2^7). The 8 generator polynomials are:

(i) 1=1
(ii) $x + 1 = x + 1$
(iii) $x^3 + x + 1 = x^3 + x + 1$
(iv) $x^3 + x^2 + 1 = x^3 + x^2 + 1$
(v) $(x+1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1$
(vi) $(x+1)(x^3 + x^2 + 1) = x^4 + x^2 + x + 1$
(vii) $(x^3 + x + 1)(x^3 + x^2 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$
(viii) $(x+1)(x^3 + x + 1)(x^3 + x^2 + 1) = x^7 + 1$

Here in (i) the polynomial 1 generates all F_2^7. In (ii) we find the parity check code and in (vii) the repetition code. As mentioned before, in (viii) we view the 0-code as being generated by $x^7 + 1$. The polynomials of (iii) and (iv) have degree 3 and so generate [7, 4] codes, which we shall later see are Hamming codes. The [7, 3] codes of (v) and (vi) are the duals of the Hamming codes.

PROBLEM 17.1. *How many cyclic codes of length 8 over F^3 are there? Give a generator polynomial for each such code.*

PROBLEM 17.2. *Prove that there is no cyclic code that is (equivalent to) an [8, 4] extended binary Hamming code.*

PROBLEM 17.3. *Let cyclic code C have generator polynomial $g(x)$. Prove that C is contained in the sum-0 code if and only if $g(1) = 0$.*

PROBLEM 17.4. *Let C be a cyclic code. Let C_- be the code resulting from shortening C at a single position, and let C^- be the code resulting from puncturing C at a single position.*
(a) Give all C for which C_- is cyclic.
(b) Give all C for which C^- is cyclic.

The check polynomial earns its name by the following

PROPOSITION 17.2. *If C is the cyclic code of length n with check polynomial $h(x)$, then*

$$C = \{c(x) \in F[x]_n | c(x)h(x) = 0(\mod x^n - 1)\}.$$

PROOF. The containment in one direction is easy. Indeed if $c(x) \in C$, then by Theorem 17.1 there is a $q(x)$ with $c(x) = q(x)g(x)$. But then

$$c(x)h(x) = q(x)g(x)h(x) = q(x)(x^n - 1) = 0(\mod x^n - 1).$$

Now consider an arbitrary polynomial $c(x) \in F[x]_n$ with

$$c(x)h(x) = p(x)(x^n - 1), \text{ say }.$$

Then

$$c(x)h(x) = p(x)(x^n - 1) = p(x)g(x)h(x),$$

hence

$$(c(x) - p(x)g(x))h(x) = 0.$$

As $g(x)h(x) = x^n - 1$, we do not have $h(x) = 0$. Therefore

$$c(x) - p(x)g(x) = 0 \text{ and } c(x) = p(x)g(x),$$

as desired. □

If we are in possession of a generator polynomial

$$g(x) = \sum_{j=0}^{r} g_j x^j$$

for the cyclic code C, then we can easily construct a generator matrix for C. Consider

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & \cdots & \cdots & g_{r-1} & g_r \end{pmatrix}$$

The matrix G has n columns and $k = n - r$ rows; so the first row, row g_0, finishes with a string of 1's of length $k - 1$. Each successive row

is the cyclic shift of the previous row: $g_i = \tilde{g}_{i-1}$, for $i = 1, \cdots, k-1$. As $g(x)h(x) = x^n - 1$, we have

$$g_0 h_0 = g(0)h(0) = 0^n - 1 \neq 0.$$

In particular $g_0 \neq 0$ (and $h_0 \neq 0$). Therefore G is in echelon form (although likely not reduced). In particular the $k = dim(C)$ rows of G are linearly independent. Clearly the rows of G belong to C, so G is indeed a generator matrix for C, sometimes called the cyclic generator matrix of C. For instance, if C is a [7, 4] binary cyclic code with generator polynomial $1 + x + x^3$, then the cyclic generator matrix is

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Given the cyclic generator matrix G, cyclic encoding is the process of encoding the message k-tuple $m = (m_0, \cdots, m_{k-1})$ into the codeword $c = mG$. At the polynomial level, this corresponds to encoding the message polynomial

$$m(x) = \sum_{i=0}^{k-1} m_i x^i$$

into the code polynomial $c(x) = m(x)g(x)$. Since the cyclic generator G is in echelon form, the first k coordinate positions form an information set. Therefore cyclic C has a standard generator matrix, although the cyclic generator matrix is almost never standard (or even systematic).

PROBLEM 17.5. *(a) Describe all situations in which the cyclic generator matrix for a cyclic code is the standard generator matrix.*

(b) Describe all situations in which the cyclic generator matrix for a cyclic code is systematic.

We next present for cyclic C a linear encoding method corresponding to the standard generator matrix. Namely

$$m = (m_0, \cdots, m_{k-1}) \mapsto c = (m_0, \cdots, m_{k-1}, -s_0, -s_1, \cdots, -s_{r-1}),$$

where $s(x) = \sum_{j=0}^{r-1} s_j x^j$ is the remainder upon dividing $x^r m(x)$ by $g(x)$. That is,

$$x^r m(x) = q(x)g(x) + s(x),$$

with $\deg(s(x)) < \deg(g(x)) = r$. To see that this is the correct standard encoding, first note that

$$x^r m(x) - s(x) = q(x)g(x) = b(x) \in C$$

with corresponding codeword

$$b = (-s_0, -s_1, \cdots, -s_{r-1}, m_0, \cdots, m_{k-1}).$$

As this is a codeword of cyclic C, every cyclic shift of it is also a codeword. In particular the c given above is found after k right shifts. Thus c is a codeword of C. Since C is systematic on the first k positions, this codeword is the only one with $\mathbf{m}$ on those positions and so is the result of standard encoding. To construct the standard generator matrix itself, we encode the k different k-tuple messages $(0, 0, \cdots, 0, 1, 0, \cdots, 0)$ of weight 1 corresponding to message polynomials x_i, for $0 \leq i \leq k-1$. These are the rows of the standard generator matrix. When we try this for the [7, 4] binary cyclic code with generator $x^3 + x + 1$ (so $r = 7 - 4 = 3$), we find, for instance,

$$x^3 x^2 = (x^2 + 1)(x^3 + x + 1) + (x^2 + x + 1)$$

so that the third row of the standard generator matrix, corresponding to message polynomial x^2, is

$$(m_0, m_1, m_2, m_3, -s_0, -s_1, -s_2) = (0, 0, 1, 0, 1, 1, 1).$$

Proceeding in this way, we find that the standard generator matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The code C is a Hamming code (this can also be checked easily by hand). This process of systematic encoding for cyclic codes is important in practice, since a machine can be transmitting the information symbols from m during the time it is calculating the check symbols s_j.

PROBLEM 17.6. *(a) Find the cyclic and standard generator matrices for the [7, 4] binary cyclic code D with generator polynomial $x^3 + x^2 + 1$.*

(b) Find the cyclic and standard generator matrices for the [15, 11] binary cyclic code E with generator polynomial $x^4 + x + 1$.

(c) Prove that D and E are Hamming codes.

A code equivalent to a cyclic code need not be cyclic itself. For instance, there are 70 distinct binary [7, 4] Hamming codes; but, as we saw in the example above, only two of them are cyclic. One permutation does take cyclic codes to cyclic codes. The *reverse code* $C^{[-1]}$ of a cyclic code C, gotten by reversing each codeword, is still cyclic. We have

$$(c_0, c_1, \cdots, c_i, \cdots, c_{n-1}) \in C \text{ iff } (c_{n-1}, \cdots, c_{n-1-i}, \cdots, c_1, c_0) \in C^{[-1]}.$$

In polynomial notation, this becomes

$$c(x) \in C \text{ iff } x^{n-1}c(x^{-1}) \in C^{[-1]}.$$

For the polynomial $p(x)$ of degree d, we let its reciprocal polynomial be given by reciprocal polynomial

$$p^{[-1]}(x) = \sum_{i=0}^{d} p_{d-i}x^i = x^d p(x^{-1}).$$

The roots of the reciprocal polynomial are the reciprocals of the nonzero roots of the original polynomial.

LEMMA 17.3. *If $g(x)$ generates cyclic C, then $g_0^{-1}g^{[-1]}(x)$ generates $C^{[-1]}$, the reverse code of C.*

PROOF. Starting from the cyclic generator matrix for C, we reverse all the rows and then write them from bottom to top. The result is

$$\begin{pmatrix} g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 & 0 & 0 & \cdots & 0 \\ 0 & g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & g_r & g_{r-1} & \cdots & \cdots & \cdots & \cdots & g_1 & g_0 \end{pmatrix}.$$

The rows of this matrix certainly belong to $C^{[-1]}$. As before, they are linearly independent since $g_0 \neq 0$. Therefore we have a generator matrix for $C^{[-1]}$. Its first row visibly corresponds to a non-zero code polynomial of degree less than r, which is seen to be $g^{[-1]}(x)$. By Theorem 17.1 the monic scalar multiple $g_0^{-1}g^{[-1]}(x)$ is the generator polynomial. (In fact, we have a scalar multiple of the cyclic generator matrix for $C^{[-1]}$.) □

It is easy to see that the dual of a cyclic code C is again a cyclic code. Proposition 17.2 suggests that the dual is associated with the check polynomial of C. Let the cyclic code C of length n have generator polynomial $g(x)$ of degree r and check polynomial $h(x)$ of degree $k = n - r = \dim C$. As $h(x)$ is a divisor of $x^n - 1$, it is the generator polynomial for a cyclic code D of length n and dimension $n - k = n - (n - r) = r$. We have

$$C = \{q(x)g(x) | q(x) \in F[x]_k\}$$

and

$$D = \{p(x)h(x) | p(x) \in F[x]_r\}.$$

Let $c(x) = q(x)g(x) \in C$, so that $\deg(q(x)) \leq k-1$; and let $d(x) = p(x)h(x) \in D$, so that $\deg(p(x)) < r - 1$. Consider

$$\begin{aligned} c(x)d(x) &= q(x)g(x)p(x)h(x) \\ &= q(x)p(x)(x^n - 1) \\ &= s(x)(x^n - 1) \\ &= s(x)x^n - s(x), \end{aligned}$$

where $s(x) = q(x)p(x)$ with

$$\deg(s(x)) \leq (k-1) + (r-1) = r + k - 2 = n - 2 < n - 1.$$

Therefore the coefficient of x^{n-1} in $c(x)d(x)$ is 0. If $c(x) = \sum_{i=0}^{n-1} c_i x^i$ and $d(x) = \sum_{j=0}^{n-1} d_j x^j$, then in general the coefficient of x^m in $c(x)d(x)$ is $\sum_{i+j=m} c_i d_j$. In particular, the two determinations of the coefficient of x^{n-1} in $c(x)d(x)$ give

$$\begin{aligned} 0 &= \sum_{i+j=n-1} c_i d_j \\ &= \sum_{i=0}^{n-1} c_i d_{n-i} \\ &= c_0 d_{n-1} + c_1 d_{n-2} + \cdots + c_i d_{n-i-1} + \cdots + c_{n-1} d_0 \\ &= \mathbf{c} \cdot \mathbf{d}^*, \end{aligned}$$

where

$$\mathbf{c} = (c_0, c_1, \cdots, c_i, \cdots, c_{n-1}) \text{ and } \mathbf{d}^* = (d_{n-1}, d_{n-2}, \cdots, d_{n-i}, \cdots, d_0).$$

That is, each codeword c of C has dot product 0 with the reverse of each codeword d of D. Therefore $C^\perp$ contains $D^{[-1]}$. Also

$$\dim(C^\perp) = n - \dim(C) = n - k = r = n - \deg(h^{[-1]}(x)) = \dim(D^{[-1]}),$$

so from Lemma 17.3 we conclude

THEOREM 17.4. *If C is the cyclic code of length n with check polynomial $h(x)$, then $C^\perp$ is cyclic with generator polynomial $h_0^{-1}h^{[-1]}(x)$.*

Part III

Basics cryptography

0.2. Introduction. Cryptography has a long and fascinating history. The predominant practitioners of the art were those associated with the military, the diplomatic service and government in general. Cryptography was used as a tool to protect national secrets and strategies. The proliferation of computers and communications systems in the 1960s brought with it a demand from the private sector for means to protect information in digital form and to provide security services. Beginning with the work of Feistel at IBM in the early 1970s and culminating in 1977 with the adoption as a U.S. Federal Information Processing Standard for encrypting unclassified information, DES, the Data Encryption Standard, is the most well-known cryptographic mechanism in history. It remains the standard means for securing electronic commerce for many financial institutions around the world. The most striking development in the history of cryptography came in 1976 when Diffie and Hellman published *New Directions in Cryptography*. This paper introduced the revolutionary concept of public-key cryptography and also provided a new and ingenious method for key exchange, the security of which is based on the intractability of the discrete logarithm problem. Although the authors had no practical realization of a public-key encryption scheme at the time, the idea was clear and it generated extensive interest and activity in the cryptographic community. In 1978 Rivest, Shamir, and Adleman discovered the first practical public-key encryption and signature scheme, now referred to as RSA. The RSA scheme is based on another hard mathematical problem, the intractability of factoring large integers. This application of a hard mathematical problem to cryptography revitalized efforts to find more efficient methods to factor. The 1980s saw major advances in this area but none which rendered the RSA system insecure. Another class of powerful and practical public-key schemes was found by ElGamal in 1985. These are also based on the discrete logarithm problem. One of the most significant contributions provided by public-key cryptography is the digital signature. In 1991 the first international standard for digital signatures (ISO/IEC 9796) was adopted. It is based on the RSA public-key scheme. In 1994 the U.S. Government adopted the Digital Signature Standard, a mechanism based on the ElGamal public key scheme. The search for new public-key schemes, improvements to existing cryptographic mechanisms, and proofs of security continues at a rapid pace. Various standards and infrastructures involving cryptography are being put in place. Security products are being developed to address the security needs of an information intensive society. The purpose of this chapter is to give an up-to-date treatise of the principles, techniques, and algorithms of interest in cryptographic practice. Emphasis has been placed on those aspects which are most practical and applied.

0.3. Information security and cryptography. The concept of information will be taken to be an understood quantity. To introduce cryptography, an understanding of issues related to information security in general is necessary. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met. Some of these objectives are listed in Table 1. Over the centuries, an elaborate set of protocols and mechanisms has been created to deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security cannot solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result. For example, privacy of letters is provided by sealed envelopes delivered by an accepted mail service. The physical security of the envelope is, for practical necessity, limited and so laws are enacted which make it a criminal

offense to open mail for which one is not authorized. It is sometimes the case that security is achieved not through the information itself but through the physical document recording it. For example, paper currency requires special inks and material to prevent counterfeiting. Conceptually, the way information is recorded has not changed dramatically over time. Whereas information was typically stored and transmitted on paper, much of it now resides on magnetic media and is transmitted via telecommunications systems, some wireless. What has changed dramatically is the ability to copy and alter information. One can make thousands of identical copies of a piece of information stored electronically and each is indistinguishable from the original. With information on paper, this is much more difficult. What is needed then for a society where information is mostly stored and transmitted in electronic form is a means to ensure information security which is independent of the physical medium recording or conveying it and such that the objectives of information security rely solely on digital information itself. One of the fundamental tools used in information security is the signature. It is a building block for many other services such as non-repudiation, data origin authentication, identification, and witnessing, to mention a few. Having learned the basics in writing, an individual is taught how to produce a handwritten signature for the purpose of identification. At contract age the signature evolves to take

privacy or confidentiality	keeping information secret from all but those who are authorized to see it
data integrity	ensuring information has not been altered by unauthorized or unknown means.
entity authentication or identification	corroboration of the identity of an entity (e.g., a person, a computer terminal, a credit card, etc.)
message authentication	corroborating the source of information; also known as data origin authentication.
signature	a means to bind information to an entity.
authorization	conveyance, to another entity, of official sanction to do or be something.
validation	a means to provide timeliness of authorization to use or manipulate information or resources.
access control	restricting access to resources to privileged entities.
certification	endorsement of information by a trusted entity.
timestamping	recording the time of creation or existence of information.
witnessing	verifying the creation or existence of information by an entity other than the creator.
receipt	acknowledgement that information has been received.
confirmation	acknowledgement that services have been provided.
ownership	a means to provide an entity with the legal right to use or transfer a resource to others.
anonymity	concealing the identity of an entity involved in some process.
non-repudiation	preventing the denial of previous commitments or actions.
revocation	retraction of certification or authorization.

TABLE 1. Some information security objectives

on a very integral part of the person's identity. This signature is intended to be unique to the individual and serve as a means to identify, authorize, and validate. With electronic information the concept of a signature needs to be redressed; it cannot simply be something unique to the signer and independent of the information signed. Electronic

replication of it is so simple that appending a signature to a document not signed by the originator of the signature is almost a triviality. Analogues of the "paper protocols" currently in use are required. Hopefully these new electronic based protocols are at least as good as those they replace. There is a unique opportunity for society to introduce new and more efficient ways of ensuring information security. Much can be learned from the evolution of the paper based system, mimicking those aspects which have served us well and removing the inefficiencies. Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography.

DEFINITION 17.1. Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication.

Cryptography is not the only means of providing information security, but rather one set of techniques.

Cryptographic goals

Of all the information security objectives listed in Table 1, the following four form a framework upon which the others will be derived:

(1) privacy or confidentiality ;
(2) data integrity ;
(3) authentication ;
(4) non-repudiation.

1. *Confidentiality* is a service used to keep the content of information from all but those authorized to have it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to providing confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

2. *Data integrity* is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such things as insertion, deletion, and substitution.

3. *Authentication* is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity

authentication and data origin authentication. Data origin authentication implicitly provides data integrity (for if a message is modified, the source has changed).

4. *Non-repudiation* is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

Figure 1 provides a schematic listing of the primitives considered and how they relate. Many of these will be briefly introduced in this chapter. These primitives should

be evaluated with respect to various criteria such as:

1. *level of security.* This is usually difficult to quantify. Often it is given in terms of the number of operations required (using the best methods currently known) to defeat the intended objective. Typically the level of security is defined by an upper bound on the amount of work necessary to defeat the objective. This is sometimes called the work factor.

2. *functionality.* Primitives will need to be combined to meet various information security objectives. Which primitives are most effective for a given objective will be determined by the basic properties of the primitives.

3. *methods of operation.* Primitives, when applied in various ways and with various inputs, will typically exhibit different characteristics; thus, one primitive could provide very different functionality depending on its mode of operation or usage.

4. *performance.* This refers to the efficiency of a primitive in a particular mode of operation. (For example, an encryption algorithm may be rated by the number of bits per second which it can encrypt.)

5. *ease of implementation.* This refers to the difficulty of realizing the primitive in a practical instantiation. This might include the complexity of implementing the primitive in either a software or hardware environment. The relative importance of various criteria is very much dependent on the application and resources available. For example, in an environment where computing power is limited one may have to trade off a very high level of security for better performance of the system as a whole. Cryptography, over the ages, has been an art

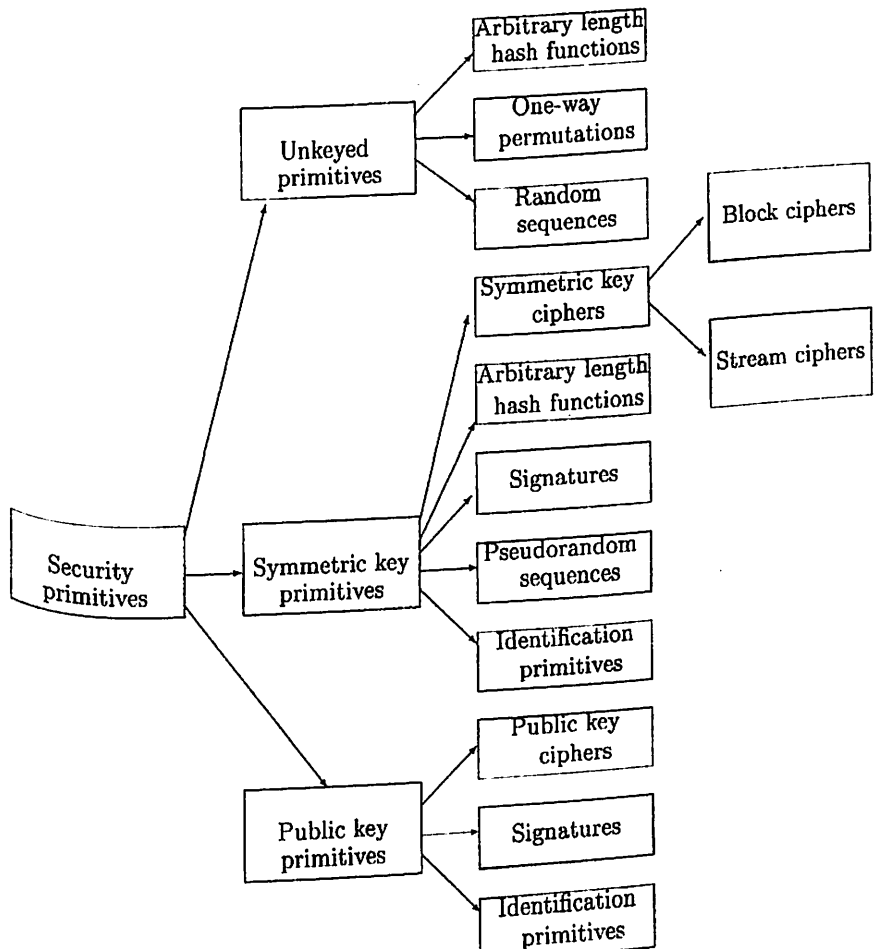

FIGURE 1. A taxonomy of cryptographic primitives.

practised by many who have devised ad hoc techniques to meet some of the information security requirements. The last twenty years have been a period of transition as the discipline moved from an art to a science. There are now several international scientific conferences devoted exclusively to cryptography and also an international scientific

organization, the International Association for Cryptologic Research (IACR), aimed at fostering research in the area.

0.4. Background on functions. While this book is not a treatise on abstract mathematics, a familiarity with basic mathematical concepts will prove to be useful. One concept which is absolutely fundamental to cryptography is that of a function in the mathematical sense. A function is alternately referred to as a mapping or a transformation.

0.4.1. *Functions (1-1, one-way, trapdoor one-way).* A set consists of distinct objects which are called elements of the set. For example, a set X might consist of the elements a, b, c, and this is denoted $X = \{a; b; c\}$.

DEFINITION 17.2. A function is defined by two sets X and Y and a rule f which assigns to each element in X precisely one element in Y. The set X is called the domain of the function and Y the codomain. If x is an element of X (usually written $x \in X$) the image of x is the element in Y which the rule f associates with x; the image y of x is denoted by $y = f(x)$. Standard notation for a function f from set X to set Y is $f : X \to Y$. If $y \in Y$, then a preimage of y is an element $x \in X$ for which $f(x) = y$. The set of all elements in Y which have at least one preimage is called the image of f, denoted $\mathrm{Im}(f)$.

EXAMPLE 17.2. (function) Consider the sets $X = \{a, b, c\}, Y = \{1, 2, 3, 4\}$, and the rule f from X to Y defined as $f(a) = 2, f(b) = 4, f(c) = 1$. Figure 2 shows a schematic of the sets X, Y and the function f.. The preimage of the element 2 is a. The image of f is $\{1, 2, 4\}$.

Thinking of a function in terms of the schematic (sometimes called a functional diagram) given in Figure 2, each element in the domain X has precisely one arrowed line originating from it. Each element in the codomain Y can have any number of arrowed lines incident to it (including zero lines).

Often only the domain X and the rule f are given and the codomain is assumed to be the image of f. This point is illustrated with two examples.

EXAMPLE 17.3. (function) Take $X = \{1, 2, 3, ..., 10\}$and let f be the rule that for each $x \in X, f(x) = r_x$, where r_x is the remainder when x^2 is divided by 11. Explicitly then

$$f(1) = 1, f(2) = 4, f(3) = 9, f(4) = 5, f(5) = 3$$
$$f(6) = 3, f(7) = 5, f(8) = 9, f(9) = 4, f(10) = 1.$$

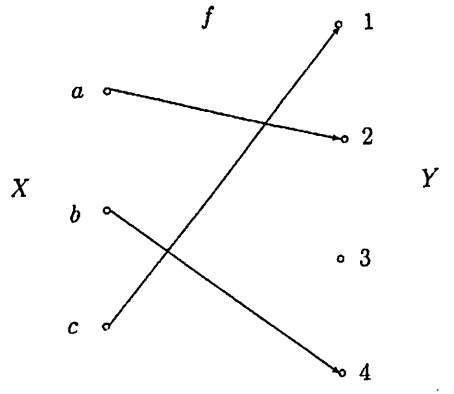

FIGURE 2. A function f from a set X of three elements to a set Y of four elements.

The image of f is the set $Y = \{1, 3, 4, 5, 9\}$.

EXAMPLE 17.4. (function) Take $X = \{1, 2, 3, ..., 10^{50}\}$ and let f be the rule $f(x) = r_x$, where r_x is the remainder when x^2 is divided by $10^{50} + 1$ for all $x \in X$. Here it is not feasible to write down f explicitly as in Example 17.3, but nonetheless the function is completely specified by the domain and the mathematical description of the rule f.

DEFINITION 17.3. A function (or transformation) is 1 - 1 (one-to-one) if each element in the codomain Y is the image of at most one element in the domain X.

DEFINITION 17.4. A function (or transformation) is onto if each element in the codomain Y is the image of at least one element in the domain. Equivalently, a function $f : X \to Y$ is onto if $\mathrm{Im}(f) = Y$.

DEFINITION 17.5. If a function $f : X \to Y$ is 1-1 and $\mathrm{Im}(f) = Y$, then f is called a bijection.

Fact If $f : X \to Y$ is 1 - 1 then $f : X \to \mathrm{Im}(f)$ is a bijection. In particular, if $f : X \to Y$ is 1 - 1, and X and Y are finite sets of the same size, then f is a bijection. In terms of the schematic representation, if f is a bijection, then each element in Y has exactly one arrowed line incident with it. The functions described in Examples 17.2 and 17.3 are not bijections. In Example 17.2 the element 3 is not the image of any element in the domain. In Example 17.3 each element in the codomain has two preimages.

DEFINITION 17.6. If f is a bijection from X to Y then it is a simple matter to define a bijection g from Y to X as follows: for each $y \in Y$ define $g(y) = x$ where $x \in X$ and $f(x) = y$. This function g obtained from f is called the inverse function of f and is denoted by $g = f^{-1}$.

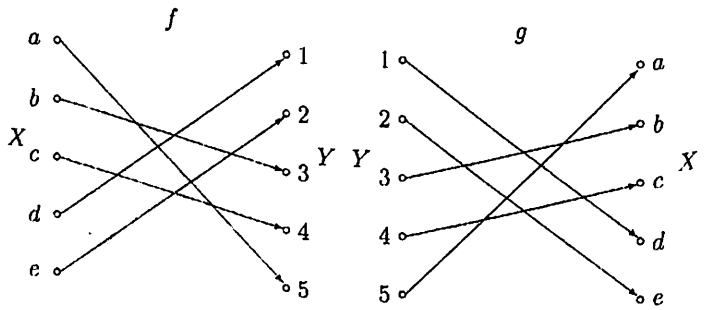

FIGURE 3. A bijection f and its inverse $g = f^{-1}$.

EXAMPLE 17.5. (inverse function) Let $X = \{a, b, c, d, e\}$, and $Y = \{1; 2; 3; 4; 5\}$, and consider the rule f given by the arrowed edges in Figure 3. f is a bijection and its inverse g is formed simply by reversing the arrows on the edges. The domain of g is Y and the codomain is X.

Note that if f is a bijection, then so is f^{-1}. In cryptography bijections are used as the tool for encrypting messages and the inverse transformations are used to decrypt. This will be made clearer when some basic terminology is introduced. Notice that if the transformations were not bijections then it would not be possible to always decrypt to a unique message.

One-way functions

There are certain types of functions which play significant roles in cryptography. At the expense of rigor, an intuitive definition of a one-way function is given.

DEFINITION 17.7. A function f from a set X to a set Y is called a one-way function if $f(x)$ is "easy" to compute for all $x \in X$ but for "essentially all" elements $y \in \mathrm{Im}(f)$ it is "computationally infeasible" to find any $x \in X$ such that $f(x) = y$.

Note (clarification of terms in Definition 17.7)

(i) A rigorous definition of the terms "easy" and "computationally infeasible" is necessary but would detract from the simple idea that is being conveyed. For the purpose of this chapter, the intuitive meaning will suffice.

(ii) The phrase "for essentially all elements in Y " refers to the fact that there are a few values $y \in Y$ for which it is easy to find an $x \in X$ such that $y = f(x)$. For example, one may compute $y = f(x)$ for a small number of x values and then for these, the inverse is known by table look-up. An alternate way to describe this property of a one-way function is the following: for a random $y \in \mathrm{Im}(f)$ it is computationally infeasible to find any $x \in X$ such that $f(x) = y$. The concept of a one-way function is illustrated through the following examples.

EXAMPLE 17.6. (one-way function) Take $X = \{1, 2, 3, ..., 16\}$ and define $f(x) = r_x$ for all $x \in X$ where r_x is the remainder when 3^x is divided by 17. Explicitly,

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(x)$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Given a number between 1 and 16, it is relatively easy to find the image of it under f. However, given a number such as 7, without having the table in front of you, it is harder to find x given that $f(x) = 7$. Of course, if the number you are given is 3 then it is clear that $x = 1$ is what you need; but for most of the elements in the codomain it is not that easy.

One must keep in mind that this is an example which uses very small numbers; the important point here is that there is a difference in the amount of work to compute $f(x)$ and the amount of work to find x given $f(x)$. Even for very large numbers, $f(x)$ can be computed efficiently using the repeated square-and-multiply algorithm, whereas the process of finding x from $f(x)$ is much harder.

EXAMPLE 17.7. (one-way function) A prime number is a positive integer greater than 1 whose only positive integer divisors are 1 and

itself. Select primes $p = 48611$, $q = 53993$, form $n = pq = 2624653723$, and let $X = \{1, 2, 3, ..., n-1\}$. Define a function f on X by $f(x) = r_x$ for each $x \in X$, where r_x is the remainder when x^3 is divided by n. For instance, $f(2489991) = 1981394214$ since $2489991^3 = 5881949859 \cdot n + 1981394214$. Computing $f(x)$ is a relatively simple thing to do, but to reverse the procedure is much more difficult; that is, given a remainder to find the value x which was originally cubed (raised to the third power). This procedure is referred to as the computation of a modular cube root with modulus n. If the factors of n are unknown and large, this is a difficult problem; however, if the factors p and q of n are known then there is an efficient algorithm for computing modular cube roots.

Example 17.7 leads one to consider another type of function which will prove to be fundamental in later developments.

Trapdoor one-way functions

DEFINITION 17.8. A trapdoor one-way function is a one-way function $f : X \to Y$ with the additional property that given some extra information (called the trapdoor information) it becomes feasible to find for any given $y \in \text{Im}(f)$, an $x \in X$ such that $f(x) = y$.

Example 17.7 illustrates the concept of a trapdoor one-way function. With the additional information of the factors of $n = 2624653723$ (namely, $p = 48611$ and $q = 53993$, each of which is five decimal digits long) it becomes much easier to invert the function. The factors of 2624653723 are large enough that finding them by hand computation would be difficult. Of course, any reasonable computer program could find the factors relatively quickly. If, on the other hand, one selects p and q to be very large distinct prime numbers (each having about 100 decimal digits) then, by today's standards, it is a difficult problem, even with the most powerful computers, to deduce p and q simply from n. This is the wellknown integer factorization problem and a source of many trapdoor one-way functions. It remains to be rigorously established whether there actually are any (true) one-way functions. That is to say, no one has yet definitively proved the existence of such functions under reasonable (and rigorous) definitions of "easy" and "computationally infeasible". Since the existence of one-way functions is still unknown, the existence of trapdoor one-way functions is also unknown. However, there are a number of good candidates for one-way and trapdoor one-way functions. Many of these are discussed in this book, with emphasis given to those which are practical. One-way and trapdoor one-way functions are the basis for public-key cryptography. The importance of these concepts will become clearer when their application

to cryptographic techniques is considered. It will be worthwhile to keep the abstract concepts of this section in mind as concrete methods are presented.

Permutations

Permutations are functions which are often used in various cryptographic constructs.

DEFINITION 17.9. Let S be a finite set of elements. A permutation p on S is a bijection from S to itself (i.e., $p : S \to S$).

EXAMPLE 17.8. (permutation) Let $S = \{1, 2, 3, 4, 5\}$. A permutation $p : S \to S$ is defined as follows:

$$p(1) = 3; p(2) = 5; p(3) = 4; p(4) = 2; p(5) = 1.$$

A permutation can be described in various ways. It can be displayed as above or as an array:

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix} \tag{25}$$

where the top row in the array is the domain and the bottom row is the image under the mapping p. Of course, other representations are possible.

Since permutations are bijections, they have inverses. If a permutation is written as an array (see 25), its inverse is easily found by interchanging the rows in the array and reordering the elements in the new top row if desired (the bottom row would have to be reordered correspondingly). The inverse of p in Example 17.8 is

$$p^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 1 & 3 & 2 \end{pmatrix}$$

EXAMPLE 17.9. (permutation) Let X be the set of integers $\{0, 1, 2, ..., pq-1\}$ where p and q are distinct large primes (for example, p and q are each about 100 decimal digits long), and suppose that neither $p-1$ nor $q-1$ is divisible by 3. Then the function $p(x) = r_x$, where r_x is the remainder when x^3 is divided by pq, can be shown to be a permutation. Determining the inverse permutation is computationally infeasible by today's standards unless p and q are known (cf. Example 17.7).

Involutions

Another type of function which is an involution. Involutions have the property that they are their own inverses.

DEFINITION 17.10. Let S be a finite set and let f be a bijection from S to S (i.e., $f : S \to S$). The function f is called an involution

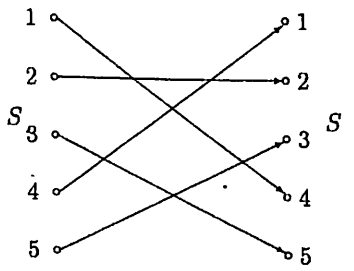

FIGURE 4. An involution on a set S of 5 elements.

if $f = f^{-1}$. An equivalent way of stating this is $f(f(x)) = x$ for all $x \in S$.

EXAMPLE 17.10. (involution) Figure 4 is an example of an involution. In the diagram of an involution, note that if j is the image of i then i is the image of j.

Basic terminology and concepts

The scientific study of any discipline must be built upon rigorous definitions arising from fundamental concepts. What follows is a list of terms and basic concepts used throughout this book. Where appropriate, rigor has been sacrificed for the sake of clarity.

Encryption domains and codomains

- $\mathcal{A}$ denotes a finite set called the alphabet of definition. For example, $\mathcal{A} = \{0, 1\}$, the binary alphabet, is a frequently used alphabet of definition. Note that any alphabet can be encoded in terms of the binary alphabet. For example, since there are 32 binary strings of length five, each letter of the English alphabet can be assigned a unique binary string of length five.
- $\mathcal{M}$ denotes a set called the message space. $\mathcal{M}$ consists of strings of symbols from an alphabet of definition. An element of $\mathcal{M}$ is called a plaintext message or simply a plaintext. For example,$\mathcal{M}$ may consist of binary strings, English text, computer code, etc.

- $\mathcal{C}$ denotes a set called the ciphertext space. $\mathcal{C}$ consists of strings of symbols from an alphabet of definition, which may differ from the alphabet of definition for $\mathcal{M}$. An element of $\mathcal{C}$ is called a ciphertext.

Encryption and decryption transformations

- $\mathcal{K}$ denotes a set called the key space. An element of $\mathcal{K}$ is called a key.
- Each element $e \in \mathcal{K}$ uniquely determines a bijection from $\mathcal{M}$ to $\mathcal{C}$, denoted by E_e. E_e is called an encryption function or an encryption transformation. Note that E_e must be a bijection if the process is to be reversed and a unique plain text message recovered for each distinct cipher text.
- For each $d \in K$, D_d denotes a bijection from $\mathcal{C}to\mathcal{M}$ (i.e., $D_d : \mathcal{C} \to \mathcal{M}$). D_d is called a decryption function or decryption transformation.
- The process of applying the transformation E_e to a message $m \in \mathcal{M}$ is usually referred to as encrypting m or the encryption of m.
- The process of applying the transformation D_d to a cipher text c is usually referred to as decrypting c or the decryption of c.
- An encryption scheme consists of a set $\{E_e : e \in \mathcal{K}\}$ of encryption transformations and a corresponding set $\{D_d : d \in \mathcal{K}\}$ of decryption transformations with the property that for each $e \in \mathcal{K}$ there is a unique key $d \in \mathcal{K}$ such that $D_d = E_e^{-1}$, that is, $D_d(E_e(m)) = m$ for all $m \in M$. An encryption scheme is sometimes referred to as a *cipher*.
- The keys e and d in the preceding definition are referred to as a key pair and sometimes denoted by (e, d). Note that e and d could be the same.
- To construct an encryption scheme requires one to select a message space $\mathcal{M}$, a cipher text space $\mathcal{C}$, a key space $\mathcal{K}$, a set of encryption transformations $\{E_e : e \in \mathcal{K}\}$, and a corresponding set of decryption transformations $\{D_d : d \in \mathcal{K}\}$.

Achieving confidentiality

An encryption scheme may be used as follows for the purpose of achieving confidentiality. Two parties Alice and Bob first secretly choose or secretly exchange a key pair (e, d). At a subsequent point in time, if Alice wishes to send a message $m \in \mathcal{M}$ to Bob, she computes $c = E_e(m)$ and transmits this to Bob. Upon receiving c, Bob computes $D_d(c) = m$ and hence recovers the original message m. The question arises as to why keys are necessary. (Why not just choose one

encryption function and its corresponding decryption function?) Having transformations which are very similar but characterized by keys means that if some particular encryption/decryption transformation is revealed then one does not have to redesign the entire scheme but simply change the key. It is sound cryptographic practice to change the key (encryption/decryption transformation) frequently. As a physical analogue, consider an ordinary resettable combination lock. The structure of the lock is available to anyonewho wishes to purchase one but the combination is chosen and set by the owner. If the owner suspects that the combination has been revealed he can easily reset it without replacing the physical mechanism.

EXAMPLE 17.11. (encryption scheme) Let $\mathcal{M} = \{m_1, m_2, m_3\}$ and $\mathcal{C} = \{c_1, c_2, c_3\}$. There are precisely $3! = 6$ bijections from $\mathcal{M}$ to $\mathcal{C}$. The key space $\mathcal{K} = \{1, 2, 3, 4, 5, 6\}$ has six elements in it, each specifying one of the transformations. Figure 5 illustrates the six encryption functions which are denoted by $E_i, 1 \leq i \leq 6$. Alice and Bob agree on a transformation, say E1.

To encrypt the message m_1, Alice computes $E_1(m_1) = c_3$ and sends c_3 to Bob. Bob decrypts c_3 by reversing the arrows on the diagram for E_1 and observing that c_3 points to m_1.

When $\mathcal{M}$ is a small set, the functional diagram is a simple visual means to describe the mapping. In cryptography, the set $\mathcal{M}$ is typically of astronomical proportions and, as such, the visual description is infeasible. What is required, in these cases, is some other simple means to describe the encryption and decryption transformations, such as mathematical algorithms.

Figure 6 provides a simple model of a two-party communication using encryption.

Communication participants

Referring to Figure 6, the following terminology is defined.

- An entity or party is someone or something which sends, receives, or manipulates information. Alice and Bob are entities in Example 17.11. An entity may be a person, a computer terminal, etc.
- Asender is an entity in a two-party communication which is the legitimate transmitter of information. In Figure6, the sender is Alice.
- A receiver is an entity in a two-party communication which is the intended recipient of information. In Figure 6, the receiver is Bob.

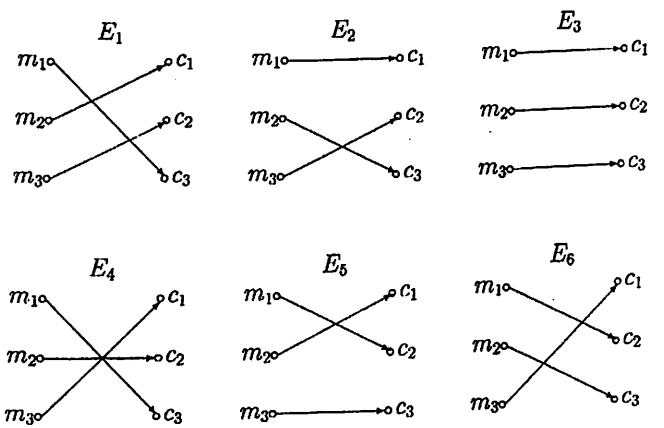

FIGURE 5. Schematic of a simple encryption scheme.

- An adversary is an entity in a two-party communication which is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver. Various other names are synonymous with adversary such as enemy, attacker, opponent, tapper, eaves dropper, intruder, and interloper. An adversary will often attempt to play the role of either the legitimate sender or the legitimate receiver.

Channels

- A channel is a means of conveying information from one entity to another.
- A physically secure channel or secure channel is one which is not physically accessible to the adversary.

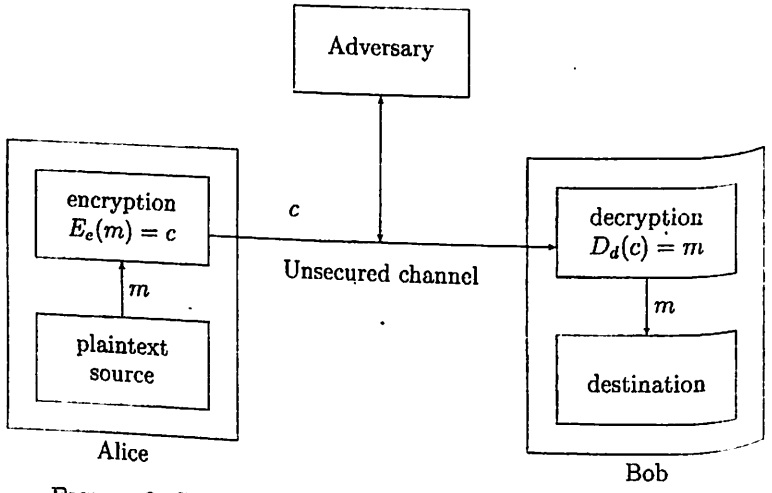

FIGURE 6. Schematic of a two-party communication using encryption.

- An unsecured channel is one from which parties other than those for which the information is intended can reorder, delete, insert, or read.
- A secured channel is one from which an adversary does not have the ability to reorder, delete, insert, or read.

One should note the subtle difference between a physically secure channel and a secured channel - a secured channel may be secured by physical or cryptographic techniques, the latter being the topic of this chapter. Certain channels are assumed to be physically secure. These include trusted couriers, personal contact between communicating parties, and a dedicated communication link, to name a few.

Security

A fundamental premise in cryptography is that the sets $\mathcal{M}, \mathcal{C}, \mathcal{K}, \{E_e : e \in \mathcal{K}\}, \{D_d : d \in \mathcal{K}\}$ are public knowledge. When two parties wish to communicate securely using an encryption scheme, the only thing that they keep secret is the particular key pair (e, d) which they are using, and which they must select. One can gain additional security by keeping the class of encryption and decryption transformations secret but one should not base the security of the entire scheme on this approach. History has shown that maintaining the secrecy of the transformations is very difficult indeed.

DEFINITION 17.11. An encryption scheme is said to be breakable if a third party, without prior knowledge of the key pair (e, d), can systematically recover plain text from corresponding cipher text within some appropriate time frame.

An appropriate time frame will be a function of the useful lifespan of the data being protected. For example, an instruction to buy a certain stock may only need to be kept secret for a few minutes whereas state secrets may need to remain confidential indefinitely. An encryption scheme can be broken by trying all possible keys to see which one the communicating parties are using (assuming that the class of encryption functions is public knowledge). This is called an exhaustive search of the key space. It follows then that the number of keys (i.e., the size of the key space) should be large enough to make this approach computationally infeasible. It is the objective of a designer of an encryption scheme that this be the best approach to break the system. Frequently cited in the literature are Kerckhoffs' desiderata, a set of requirements for cipher systems. They are given here essentially as Kerckhoffs originally stated them:

1. the system should be, if not theoretically unbreakable, unbreakable in practice;
2. compromise of the system details should not inconvenience the correspondents;
3. the key should be rememberable without notes and easily changed;
4. the cryptogram should be transmissible by telegraph;
5. the encryption apparatus should be portable and operable by a single person; and
6. the system should be easy, requiring neither the knowledge of a long list of rules nor mental strain.

This list of requirementswas articulated in 1883 and, for the most part, remains useful today. Point 2 allows that the class of encryption transformations being used be publicly known and that the security of the system should reside only in the key chosen.

Information security in general

So far the terminology has been restricted to encryption and decryption with the goal of privacy in mind. Information security is much broader, encompassing such things as authentication and data integrity. A few more general definitions, pertinent to discussions later in the book, are given next.

- An information security service is a method to provide some specific aspect of security. For example, integrity of transmitted data is a security objective, and a method to ensure this aspect is an information security service.
- Breaking an information security service (which often involves more than simply encryption) implies defeating the objective of the intended service.
- A passive adversary is an adversarywho is capable only of reading information from an unsecured channel.
- An active adversary is an adversary who may also transmit, alter, or delete information on an unsecured channel.

Cryptology

- Cryptanalysis is the study of mathematical techniques for attempting to defeat cryptographic techniques, and, more generally, information security services.
- A cryptanalyst is someone who engages in cryptanalysis.
- Cryptology is the study of cryptography and cryptanalysis.
- A cryptosystem is a general term referring to a set of cryptographic primitives used to provide information security services. Most often the term is used in conjunction with primitives providing confidentiality, i.e., encryption.

Cryptographic techniques are typically divided into two generic types: *symmetric-key* and *public-key*. Encryption methods of these types will be discussed separately. Other definitions and terminology will be introduced as required.

0.5. Symmetric-key encryption.

0.5.1. *Overview of block ciphers and stream ciphers.*

DEFINITION 17.12. Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively, where $\mathcal{K}$ is the key space. The encryption scheme is said to be *symmetric-key* if for each associated encryption/decryption key pair (e, d), it is computationally "easy" to determine d knowing only e, and to determine e from d.

Since $e = d$ in most practical symmetric-key encryption schemes, the term *symmetric key* becomes appropriate. Other terms used in the literature are single-key, one-key, private key, and conventional encryption. Example 17.12 illustrates the idea of symmetric-key encryption.

EXAMPLE 17.12. (symmetric-key encryption) Let $\mathcal{A} = \{A, B, C, ..., X, Y, Z\}$ be the English alphabet. Let $\mathcal{M}$ and $\mathcal{C}$ be the set of all strings of length

five over $\mathcal{A}$. The key e is chosen to be a permutation on $\mathcal{A}$. To encrypt, an English message is broken up into groups each having five letters (with appropriate padding if the length of the message is not a multiple of five) and a permutation e is applied to each letter one at a time. To decrypt, the inverse permutation $d = e^{-1}$ is applied to each letter of the ciphertext. For instance, suppose that the key e is chosen to be the permutation which maps each letter to the one which is three positions to its right, as shown below

$$e = \begin{pmatrix} \text{ABCDEFGH I J K LMNOPQRS T UVWXYZ} \\ \text{DEFG HI J K LMNOPQ RS TUVWXY ZABC} \end{pmatrix}$$

A message

$m =$ THISC IPHER ISCER TAINL YNOTS ECURE

is encrypted to

$c = E_e(m) =$ WKLVF LSKHU LVFHU WDLQO BQRWV HFXUH.

A two-party communication using symmetric-key encryption can be described by the block diagram of Figure 7, which is Figure 6 with the addition of the secure

(both con-fidential and authentic) channel. One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely. This problem is referred to as the key distribution problem. It is assumed that all parties know the set of encryption/decryption transformations (i.e., they all know the encryption scheme). As has been emphasized several times the only information which should be required to be kept secret is the key d. However, in symmetric-key encryption, this means that the key e must also be kept secret, as d can be deduced from e. In Figure 7 the encryption key e is transported from one entity to the other with the understanding that both can construct the decryption key d. There are two classes of symmetric-key encryption schemes which are commonly distinguished: block ciphers and stream ciphers.

DEFINITION 17.13. A block cipher is an encryption scheme which breaks up the plain text messages to be transmitted into strings (called blocks) of a fixed length t over an alphabet $\mathcal{A}$, and encrypts one block at a time.

Most well-known symmetric-key encryption techniques are block ciphers. Two important classes of block ciphers are substitution ciphers and transposition ciphers . Product ciphers combine these.

Substitution ciphers and transposition ciphers

Substitution ciphers are block ciphers which replace symbols (or groups of symbols) by other symbols or groups of symbols.

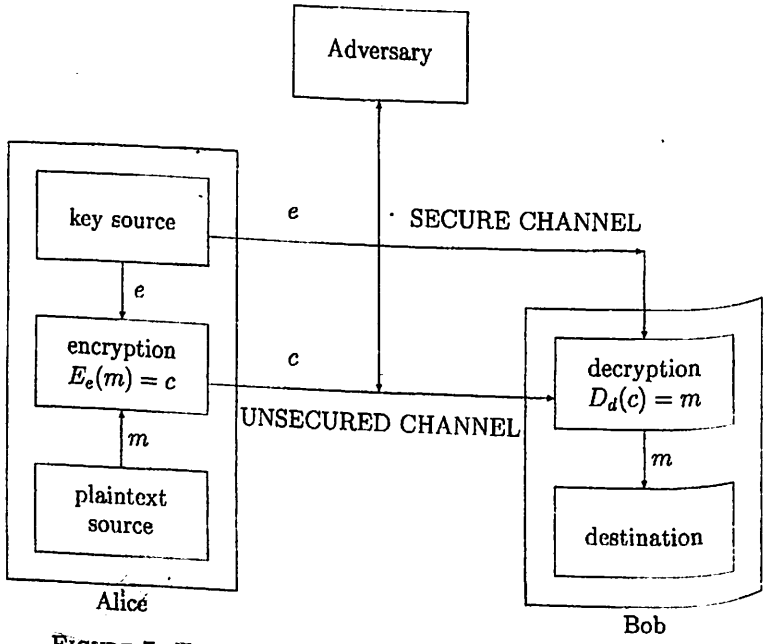

FIGURE 7. Two-party communication using encryption, with a secure channel for key exchange. The decryption key d can be efficiently computed from the encryption key e.

Simple substitution ciphers

DEFINITION 17.14. Let $\mathcal{A}$ be an alphabet of q symbols and $\mathcal{M}$ be the set of all strings of length t over $\mathcal{A}$. Let $\mathcal{K}$ be the set of all permutations on the set $\mathcal{A}$. Define for each $e \in \mathcal{K}$ an encryption transformation E_e as:

$$E_e(m) = (e(m_1)e(m_2)\cdots e(m_t)) = (c_1c_2\cdots c_t) = c,$$

where $m = (m_1m_2\cdots m_t) \in \mathcal{M}$. In other words, for each symbol in a t-tuple, replace (substitute) it by another symbol from $\mathcal{A}$ according to some fixed permutation e. To decrypt $c = (c_1c_2\cdots c_t)$ compute the

inverse permutation $d = e^{-1}$ and

$$D_d(c) = (d(c_1)d(c2)\cdots d(c_t)) = (m_1m_2\cdots m_t) = m.$$

E_e is called a *simple substitution cipher* or a *mono-alphabetic substitution cipher*.

The number of distinct substitution ciphers is $q!$ and is independent of the block size in the cipher. Example 17.12 is an example of a simple substitution cipher of block length five. Simple substitution ciphers over small block sizes provide inadequate security even when the key space is extremely large. If the alphabet is the English alphabet as in Example 17.12, then the size of the key space is $26! \approx 4 \times 10^{26}$, yet the key being used can be determined quite easily by examining a modest amount of cipher text. This follows from the simple observation that the distribution of letter frequencies is preserved in the cipher text. For example, the letter E occurs more frequently than the other letters in ordinary English text. Hence the letter occurring most frequently in a sequence of cipher text blocks is most likely to correspond to the letter E in the plaintext. By observing a modest quantity of cipher text blocks, a cryptanalyst can determine the key.

Homophonic substitution ciphers

DEFINITION 17.15. To each symbol $a \in \mathcal{A}$, associate a set $H(a)$ of strings of t symbols, with the restriction that the sets $H(a), a \in \mathcal{A}$, be pairwise disjoint. A homophonic substitution cipher replaces each symbol a in a plain text message block with a randomly chosen string from $H(a)$. To decrypt a string c of t symbols, one must determine an $a \in \mathcal{A}$ such that $c \in H(a)$. The key for the cipher consists of the sets $H(a)$.

EXAMPLE 17.13. (homophonic substitution cipher) Consider $A = \{a, b\}$, $H(a) = \{00, 10\}$, and $H(b) = \{01, 11\}$. The plain text message block ab encrypts to one of the following: 0001, 0011, 1001, 1011. Observe that the codomain of the encryption function (for messages of length two) consists of the following pairwise disjoint sets of four-element bitstrings:

$aa \rightarrow \{0000, 0010, 1000, 1010\}$
$ab \rightarrow \{0001, 0011, 1001, 1011\}$
$ba \rightarrow \{0100, 0110, 1100, 1110\}$
$bb \rightarrow \{0101, 0111, 1101, 1111\}$

Any 4-bitstring uniquely identifies a codomain element, and hence a plain text message.

Often the symbols do not occur with equal frequency in plain text messages. With a simple substitution cipher this non-uniform frequency property is reflected in the cipher text as illustrated in Example 17.12. A homophonic cipher can be used to make the frequency of occurrence of cipher text symbols more uniform, at the expense of data expansion. Decryption is not as easily performed as it is for simple substitution ciphers.

Polyalphabetic substitution ciphers

DEFINITION 17.16. A polyalphabetic substitution cipher is a block cipher with block length t over an alphabet $\mathcal{A}$ having the following properties:

(i) the key space $\mathcal{K}$ consists of all ordered sets of t permutations $(p_1, p_2, ..., p_t)$, where each permutation p_i is defined on the set $\mathcal{A}$;

(ii) encryption of the message $m = (m_1 m_2 \cdots m_t)$ under the key $e = (p_1, p_2, ..., p_t)$ is given by $E_e(m) = (p_1(m_1)p_2(m_2)\cdots p_t(m_t))$; and

(iii) the decryption key associated with $e = (p_1, p_2, ..., p_t)$ is $d = (p_1^{-1}, p_2^{-1}, ..., p_t^{-1})$.

EXAMPLE 17.14. (Vigenère cipher) Let $\mathcal{A} = \{A, B, C, ..., X, Y, Z\}$ and $t = 3$. Choose $e = (p_1, p_2, p_3)$, where p_1 maps each letter to the letter three positions to its right in the alphabet, p_2 to the one seven positions to its right, and p_3 ten positions to its right. If

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

then

$c = E_e(m) =$ *WOSVJSSOOUPCFLBWHSQSIQVDVLMXYO*

Polyalphabetic ciphers have the advantage over simple substitution ciphers that symbol frequencies are not preserved. In the example above, the letter E is encrypted to both O and L. However, polyalphabetic ciphers are not significantly more difficult to cryptanalyze, the approach being similar to the simple substitution cipher. In fact, once the block length t is determined, the cipher text letters can be divided into t groups (where group $i, 1 \leq i \leq t$, consists of those cipher text letters derived using permutation p_i), and a frequency analysis can be done on each group.

Transposition ciphers

Another class of symmetric-key ciphers is the simple transposition cipher, which simply permutes the symbols in a block.

DEFINITION 17.17. Consider a symmetric-key block encryption scheme with block length t. Let $\mathcal{K}$ be the set of all permutations on the set $\{1, 2, ..., t\}$. For each $e \in \mathcal{K}$ define the encryption function

$$E_e(m) = (m_{e(1)} m_{e(2)} \cdots m_{e(t)})$$

where $m = (m_1 m_2 \cdots m_t) \in \mathcal{M}$, the message space. The set of all such transformations is called a *simple transposition cipher*. The decryption key corresponding to e is the inverse permutation $d = e^{-1}$. To decrypt $c = (c_1 c_2 \cdots c_t)$, compute $D_d(c) = (c_{d(1)} c_{d(2)} \cdots c_{d(t)})$.

A simple transposition cipher preserves the number of symbols of a given type within a block, and thus is easily cryptanalyzed.

Composition of ciphers

In order to describe product ciphers, the concept of composition of functions is introduced. Compositions are a convenient way of constructing more complicated functions from simpler ones.

Composition of functions

DEFINITION 17.18. Let $\mathcal{S}, \mathcal{T}$, and $\mathcal{U}$ be finite sets and let $f : \mathcal{S} \to \mathcal{T}$ and $g : \mathcal{T} \to \mathcal{U}$ be functions. The composition of g with f, denoted $g \circ f$ (or simply gf), is a function from $\mathcal{S}$ to $\mathcal{U}$ and defined by $(g \circ f)(x) = g(f(x))$ for all $x \in \mathcal{S}$.

Composition can be easily extended to more than two functions. For functions $f_1, f_2, ..., f_t$, one can define $f_t \circ \cdots \circ f_2 \circ f_1$, provided that the domain of f_t equals the codomain of f_{t-1}^{-1} and so on.

Compositions and involutions

Involutions were introduced as a simple class of functions with an interesting property: $E_k(E_k(x)) = x$ for all x in the domain of E_k; that is, $E_k \circ E_k$ is the identity function.

Remark (*composition of involutions*) The composition of two involutions is not necessarily an involution, as illustrated in Figure 8. However, involutions may be composed to get somewhat more complicated functions whose inverses are easy to find. This is an important feature for decryption. For example if $E_{k_1}, E_{k_2}, ..., E_{k_t}$ are involutions then the inverse of $E_k = E_{k_1} E_{k_2} \cdots E_{k_t}$ is $E_k^{-1} = E_{k_t} E_{k_{t-1}} \cdots E_{k_1}$, the composition of the involutions in the reverse order.

Product ciphers

Simple substitution and transposition ciphers individually do not provide a very high level of security. However, by combining these transformations it is possible to obtain strong ciphers. Some of the most practical and effective symmetric-key systems are product ciphers. One example of a product cipher is a composition of $t \geq 2$ transformations $E_{k_1} E_{k_2} \cdots E_{k_t}$ where each $E_{k_i}, 1 \leq i \leq t$, is either a substitution or a transposition cipher. For the purpose of this introduction, let the composition of a substitution and a transposition be called a round.

EXAMPLE 17.15. (product cipher) Let $\mathcal{M} = \mathcal{C} = \mathcal{K}$ be the set of all binary strings of length six. The number of elements in $\mathcal{M}$ is $2^6 = 64$.

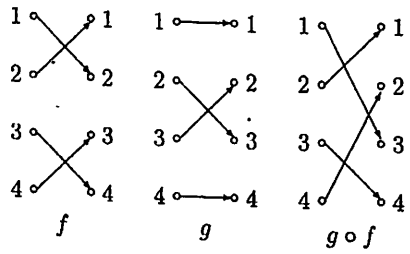

FIGURE 8. The composition $g \circ f$ of involutions g and f is not an involution.

Let $m = (m_1 m_2 \cdots m_6)$ and define

$$E_k^{(1)}(m) = m \oplus k, \text{where } k \in \mathcal{K},$$

$$E^{(2)}(m) = (m_4 m_5 m_6 m_1 m_2 m_3).$$

Here, $\oplus$ is the exclusive-OR (XOR) operation defined as follows: $0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$. $E_k^{(1)}$ is a polyalphabetic substitution cipher and $E^{(2)}$ is a transposition cipher (not involving the key). The product $E_k^{(1)} E^{(2)}$ is a round. While here the transposition cipher is very simple and is not determined by the key, this need not be the case.

Remark (*confusion and diffusion*) A substitution in a round is said to add confusion to the encryption process whereas a transposition is said to add diffusion. Confusion is intended to make the relationship between the key and cipher text as complex as possible. Diffusion refers to rearranging or spreading out the bits in the message so that any redundancy in the plain text is spread out over the cipher text. A round then can be said to add both confusion and diffusion to the encryption. Most modern block cipher systems apply a number of rounds in succession to encrypt plain text.

Stream ciphers

Stream ciphers form an important class of symmetric-key encryption schemes. They are, in one sense, very simple block ciphers having block length equal to one. What makes them useful is the fact that the

encryption transformation can change for each symbol of plain text being encrypted. In situations where transmission errors are highly probable, stream ciphers are advantageous because they have no error propagation. They can also be used when the data must be processed one symbol at a time (e.g., if the equipment has no memory or buffering of data is limited).

DEFINITION 17.19. Let $\mathcal{K}$ be the key space for a set of encryption transformations. A sequence of symbols $e_1 e_2 e_3 \cdots e_i \in \mathcal{K}$, is called a keystream.

DEFINITION 17.20. Let $\mathcal{A}$ be an alphabet of q symbols and let E_e be a simple substitution cipher with block length 1 where $e \in \mathcal{K}$. Let $m_1 m_2 m_3 \cdots$ be a plain text string and let $e_1 e_2 e_3 \cdots$ be a keystream from $\mathcal{K}$. A stream cipher takes the plain text string and produces a cipher text string $c_1 c_2 c_3 \cdots$ where $c_i = E_{e_i}(m_i)$. If d_i denotes the inverse of e_i, then $D_{d_i}(c_i) = m_i$ decrypts the cipher text string.

A stream cipher applies simple encryption transformations according to the keystream being used. The keystream could be generated at random, or by an algorithm which generates the keystream from an initial small keystream (called a seed), or from a seed and previous cipher text symbols. Such an algorithm is called a keystream generator.

The Vernam cipher

A motivating factor for the Vernam cipher was its simplicity and ease of implementation.

DEFINITION 17.21. The *Vernam Cipher* is a stream cipher defined on the alphabet $\mathcal{A} = \{0, 1\}$. A binary message $m_1 m_2 \cdots m_t$ is operated on by a binary key string $k_1 k_2 \cdots k_t$ of the same length to produce a cipher text string $c_1 c_2 \cdots c_t$ where

$$c_i = m_i \oplus k_i, 1 \leq i \leq t.$$

If the key string is randomly chosen and never used again, the Vernam cipher is called a one-time system or a one-time pad.

To see how the Vernam cipher corresponds to Definition 0.5.1, observe that there are precisely two substitution ciphers on the set $\mathcal{A}$. One is simply the identity map E_0 which sends 0 to 0 and 1 to 1; the other E_1 sends 0 to 1 and 1 to 0. When the keystream contains a 0, apply E_0 to the corresponding plain text symbol; otherwise, apply E_1. If the key string is reused there are ways to attack the system. For example, if $c_1 c_2 \cdots c_t$ and $c_1' c_2' \cdots c_t'$ are two cipher text strings produced by the same keystream $k_1 k_2 \cdots k_t$ then

$$c_i = m_i \oplus k_i; c_i' = m_i' \oplus k_i$$

and $ci \oplus c_i' = mi \oplus m_i'$. The redundancy in the latter may permit cryptanalysis.

The one-time pad can be shown to be theoretically unbreakable. That is, if a cryptanalyst has a cipher text string $c_1c_2\cdots c_t$ encrypted using a random key string which has been used only once, the cryptanalyst can do no better than guess at the plain text being any binary string of length t (i.e., t-bit binary strings are equally likely as plain text). It has been proven that to realize an unbreakable system requires a random key of the same length as the message. This reduces the practicality of the system in all but a few specialized situations. Reportedly until very recently the communication line between Moscow and Washington was secured by a one-time pad. Transport of the key was done by trusted courier.

The key space

The size of the key space is the number of encryption/decryption key pairs that are available in the cipher system. A key is typically a compact way to specify the encryption transformation (from the set of all encryption transformations) to be used. For example, a transposition cipher of block length t has $t!$ encryption functions from which to select. Each can be simply described by a permutation which is called the key. It is a great temptation to relate the security of the encryption scheme to the size of the key space. The following statement is important to remember.

Fact A necessary, but usually not sufficient, condition for an encryption scheme to be secure is that the key space be large enough to preclude exhaustive search.

For instance, the simple substitution cipher in Example 17.12 has a key space of size $26! \approx 4\times10^26$. The polyalphabetic substitution cipher of Example 1.31 has a key space of size $(26!)^3 \approx 7 \times 10^79$. Exhaustive search of either key space is completely infeasible, yet both ciphers are relatively weak and provide little security.

0.6. Digital signatures. A cryptographic primitive which is fundamental in authentication, authorization, and nonrepudiation is the digital signature. The purpose of a digital signature is to provide a means for an entity to bind its identity to a piece of information. The process of signing entails transforming the message and some secret information held by the entity into a tag called a signature. A generic description follows.

Nomenclature and set-up

- $\mathcal{M}$ is the set of messages which can be signed.

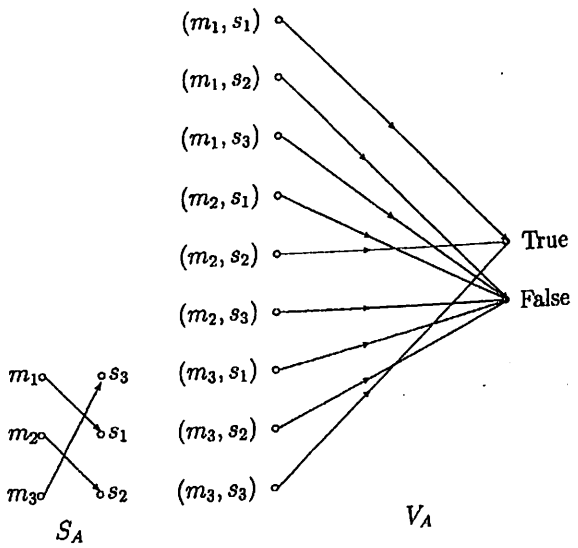

FIGURE 9. A signing and verification function for a digital signature scheme.

- $\mathcal{S}$ is a set of elements called signatures, possibly binary strings of a fixed length.
- S_A is a transformation from the message set $\mathcal{M}$ to the signature set $\mathcal{S}$, and is called a signing transformation for entity A. The transformation S_A is kept secret by A, and will be used to create signatures for messages from $\mathcal{M}$.
- V_A is a transformation from the set $\mathcal{M}\times\mathcal{S}$ to the set $\{true; false\}$. V_A is called a verification transformation for A's signatures, is publicly known, and is used by other entities to verify signatures created by A.

DEFINITION 17.22. The transformations S_A and V_A provide a digital signature scheme for A. Occasionally the term digital signature mechanism is used.

EXAMPLE 17.16. (digital signature scheme) $\mathcal{M} = \{m_1, m_2, m_3\}$ and $\mathcal{S} = \{s_1, s_2, s_3\}$. The left side of Figure 9 displays a signing function S_A from the set $\mathcal{M}$ and, the right side, the corresponding verification function V_A.

Signing procedure

Entity A (the signer) creates a signature for a message $m \in \mathcal{M}$ by doing the following:

1. Compute $s = S_A(m)$.
2. Transmit the pair (m, s). s is called the signature for message m.

Verification procedure

To verify that a signature s on a message m was created by A, an entity B (the verifier) performs the following steps:

1. Obtain the verification function V_A of A.
2. Compute $u = V_A(m, s)$.
3. Accept the signature as having been created by A if $u = true$, and reject the signature if $u = false$.

Remark (concise representation) The transformations S_A and V_A are typically characterized more compactly by a key; that is, there is a class of signing and verification algorithms publicly known, and each algorithm is identified by a key. Thus the signing algorithm S_A of A is determined by a key k_A and A is only required to keep k_A secret. Similarly, the verification algorithm V_A of A is determined by a key l_A which is made public.

Remark (handwritten signatures) Handwritten signatures could be interpreted as a special class of digital signatures. To see this, take the set of signatures S to contain only one element which is the handwritten signature of A, denoted by s_A. The verification function simply checks if the signature on a message purportedly signed by A is s_A.

An undesirable feature is that the signature is not message-dependent. Hence, further constraints are imposed on digital signature mechanisms as next discussed.

Properties required for signing and verification functions

There are several properties which the signing and verification transformations must satisfy.

(a) s is a valid signature of A on message m if and only if $V_A(m, s) = true$.

(b) It is computationally infeasible for any entity other than A to find, for any $m \in \mathcal{M}$, an $s \in \mathcal{S}$ such that $V_A(m, s) = true$.

Figure 9 graphically displays property (a). There is an arrowed line in the diagram for V_A from (m_i, s_j) to true provided there is an arrowed line from m_i to s_j in the diagram for S_A. Property (b) provides the security for the method - the signature uniquely binds A to the message which is signed. No one has yet formally proved that digital signature schemes satisfying (b) exist (although existence is widely believed to be true); however, there are some very good candidates.

0.7. Authentication and identification. Authentication is a term which is used (and often abused) in a very broad sense. By itself it has little meaning other than to convey the idea that some means has been provided to guarantee that entities are who they claim to be, or that information has not been manipulated by unauthorized parties. Authentication is specific to the security objective which one is trying to achieve. Examples of specific objectives include access control, entity authentication, message authentication, data integrity, nonrepudiation, and key authentication. For the purposes of this chapter, it suffices to give a brief introduction to authentication by describing several of the most obvious applications. Authentication is one of the most important of all information security objectives. Until the mid 1970s it was generally believed that secrecy and authentication were intrinsically connected. With the discovery of hash functions and digital signatures, it was realized that secrecy and authentication were truly separate and independent information security objectives. It may at first not seem important to separate the two but there are situations where it is not only useful but essential. For example, if a two-party communication between Alice and Bob is to take place where Alice is in one country and Bob in another, the host countries might not permit secrecy on the channel; one or both countries might want the ability to monitor all communications. Alice and Bob, however, would like to be assured of the identity of each other, and of the integrity and origin of the information they send and receive. The preceding scenario illustrates several independent aspects of authentication. If Alice and Bob desire assurance of each other's identity, there are two possibilities to consider.

1. Alice and Bob could be communicating with no appreciable time delay. That is, they are both active in the communication in "real time".
2. Alice or Bob could be exchanging messages with some delay. That is, messages might be routed through various networks, stored, and forwarded at some later time.

In the first instance Alice and Bob would want to verify identities in real time. This might be accomplished by Alice sending Bob some challenge, to which Bob is the only entity which can respond correctly. Bob could perform a similar action to identify Alice. This type of authentication is commonly referred to as entity authentication or more simply identification. For the second possibility, it is not convenient to challenge and await response, and moreover the communication path may be only in one direction. Different techniques are now required to

authenticate the originator of the message. This form of authentication is called data origin authentication.

Identification

DEFINITION 17.23. An identification or entity authentication technique assures one party (through acquisition of corroborative evidence) of both the identity of a second party involved, and that the second was active at the time the evidence was created or acquired.

Typically the only data transmitted is that necessary to identify the communicating parties. The entities are both active in the communication, giving a timeliness guarantee.

EXAMPLE 17.17. (identification) A calls B on the telephone. If A and B know each other then entity authentication is provided through voice recognition. Although not foolproof, this works effectively in practice.

EXAMPLE 17.18. (identification) Person A provides to a banking machine a personal identification number (PIN) along with a magnetic stripe card containing information about A. The banking machine uses the information on the card and the PIN to verify the identity of the card holder. If verification succeeds, A is given access to various services offered by the machine.

Example 17.17 is an instance of mutual authentication whereas Example 17.18 only provides unilateral authentication. There are exist numerous mechanisms and protocols devised to provide mutual or unilateral authentication.

Data origin authentication

DEFINITION 17.24. Data origin authentication or message authentication techniques provide to one party which receives a message assurance (through corroborative evidence) of the identity of the party which originated the message.

Often a message is provided to B along with additional information so that B can determine the identity of the entity who originated the message. This form of authentication typically provides no guarantee of timeliness, but is useful in situations where one of the parties is not active in the communication.

EXAMPLE 17.19. (need for data origin authentication) A sends to B an electronic mail message (e-mail). The message may travel through

various network communications systems and be stored for B to retrieve at some later time. A and B are usually not in direct communication. B would like some means to verify that the message received and purportedly created by A did indeed originate from A.

Data origin authentication implicitly provides data integrity since, if the message was modified during transmission, A would no longer be the originator.

0.8. Public-key cryptography. The concept of public-key encryption is simple and elegant, but has far-reaching consequences.

Public-key encryption

Let $\{E_e : e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{D_d : d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations (E_e, D_d) and suppose that each pair has the property that knowing E_e it is computationally infeasible, given a random cipher text $c \in \mathcal{C}$, to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d. (Of course e and d are simply means to describe the encryption and decryption functions, respectively.) E_e is being viewed here as a trapdoor one-way function with d being the trapdoor information necessary to compute the inverse function and hence allow decryption. This is unlike symmetric-key ciphers where e and d are essentially the same. Under these assumptions, consider the two-party communication between Alice and Bob illustrated in Figure 10. Bob selects the key pair (e, d). Bob sends the encryption key e (called the public key) to Alice over any channel but keeps the decryption key d (called the private key) secure and secret. Alice may subsequently send a messagem to Bob by applying the encryption transformation determined by Bob's public key to get $c = E_e(m)$. Bob decrypts the cipher text c by applying the inverse transformation D_d uniquely determined by d.

Notice how Figure 10 differs from Figure 7 for a symmetric-key cipher. Here the encryption key is transmitted to Alice over an unsecured channel. This unsecured channel may be the same channel on which the cipher text is being transmitted. Since the encryption key e need not be kept secret, it may be made public. Any entity can subsequently send encrypted messages to Bob which only Bob can decrypt. Figure 11 illustrates this idea, where A_1, A_2, and A_3 are distinct entities. Note that if A_1 destroys message m_1 after encrypting it to c_1, then even A_1 cannot recover m_1 from c_1. As a physical analogue, consider a metal box with the lid secured by a combination lock. The combination is

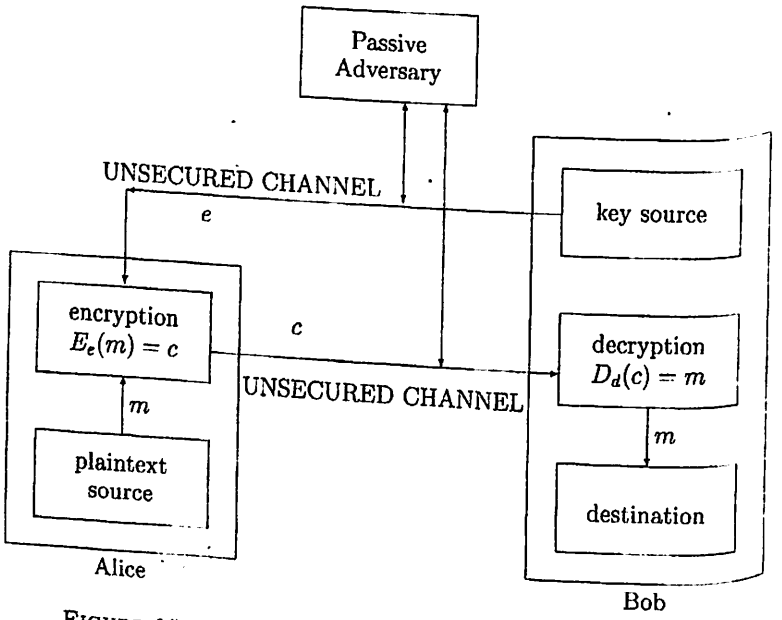

FIGURE 10. Encryption using public-key techniques.

known only to Bob. If the lock is left open and made publicly available then anyone can place a message inside and lock the lid. Only Bob can retrieve the message. Even the entity which placed the message into the box is unable to retrieve it. Public-key encryption, as described here, assumes that knowledge of the public key e does not allow computation of the private key d. In other words, this assumes the existence of trapdoor one-way functions.

DEFINITION 17.25. Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively. The encryption method is said to be a public-key encryption scheme if for each associated encryption/decryption pair (e, d), one key e (the public key) is

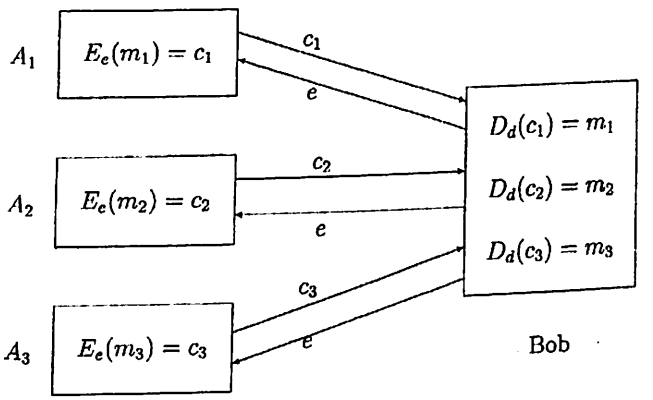

FIGURE 11. Schematic use of public-key encryption.

made publicly available, while the other d (the private key) is kept secret. For the scheme to be secure, it must be computationally infeasible to compute d from e.

Remark (*private key vs. secret key*) To avoid ambiguity, a common convention is to use the term private key in association with public-key cryptosystems, and secret key in association with symmetric-key cryptosystems. This may be motivated by the following line of thought: it takes two or more parties to share a secret, but a key is truly private only when one party alone knows it.

There are many schemes known which are widely believed to be secure public-key encryption methods, but none have been mathematically proven to be secure independent of qualifying assumptions. This is not unlike the symmetric-key case where the only system which has been proven secure is the one-time pad.

The necessity of authentication in public-key systems

It would appear that public-key cryptography is an ideal system, not requiring a secure channel to pass the encryption key. This would imply that two entities could communicate over an unsecured channel without ever having met to exchange keys. Unfortunately, this is not the case. Figure 12 illustrates how an active adversary can defeat the system (decrypt messages intended for a second entity) without breaking the encryption system. This is a type of impersonation and is an

example of protocol failure. In this scenario the adversary impersonates entity B by sending entity A a public key e_0 which A assumes (incorrectly) to be the public key of B. The adversary intercepts encrypted messages from A to B, decrypts with its own private key d_0, re-encrypts the message under B's public key e, and sends it on to B. This highlights the necessity to authenticate public keys to achieve data origin authentication of the public keys themselves. A must be convinced that she is encrypting under the legitimate public key of B. Fortunately, public-key techniques also allow an elegant solution to this problem.

Digital signatures from reversible public-key encryption

This section considers a class of digital signature schemes which is based on public-key encryption systems of a particular type. Suppose E_e is a public-key encryption transformation with message space $\mathcal{M}$ and cipher text space $\mathcal{C}$. Suppose further that $\mathcal{M} = \mathcal{C}$. If D_d is the decryption transformation corresponding to E_e then since E_e and D_d are both permutations, one has

$$D_d(E_e(m)) = E_e(D_d(m)) = m; \forall m \in \mathcal{M}$$

A public-key encryption scheme of this type is called reversible. Note that it is essential that $\mathcal{M} = \mathcal{C}$ for this to be a valid equality for all $m \in \mathcal{M}$; otherwise, $D_d(m)$ will be meaningless for $m \notin \mathcal{C}$.

Construction for a digital signature scheme

1. Let $\mathcal{M}$ be the message space for the signature scheme.
2. Let $\mathcal{C} = \mathcal{M}$ be the signature space $\mathcal{S}$.
3. Let (e, d) be a key pair for the public-key encryption scheme.
4. Define the signing function S_A to be D_d. That is, the signature for a message $m \in \mathcal{M}$ is $s = D_d(m)$.
5. Define the verification function V_A by

$$V_A(m, s) = \begin{cases} true; & \text{if } E_e(s) = m, \\ false; & \text{otherwise.} \end{cases}$$

The signature scheme can be simplified further if A only signs messages having a special structure, and this structure is publicly known. Let $\mathcal{M}'$ be a subset of $\mathcal{M}$ where elements of $\mathcal{M}'$ have a well-defined special structure, such that $\mathcal{M}'$ contains only a negligible fraction of messages from the set. For example, suppose that $\mathcal{M}$ consists of all binary strings of length $2t$ for some positive integer t. Let $\mathcal{M}'$ be the subset of $\mathcal{M}$ consisting of all strings where the first t bits are replicated in the last t positions (e.g., 101101 would be in $\mathcal{M}'$ for $t = 3$). If A only signs messages within the subset $\mathcal{M}'$; these are easily recognized by a

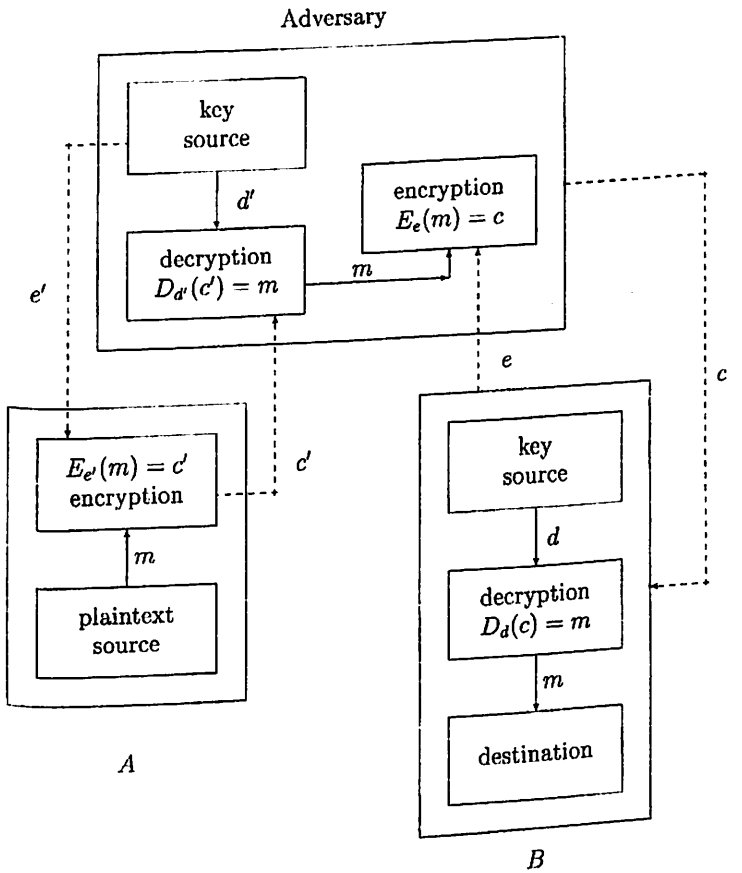

FIGURE 12. An impersonation attack on a two-party communication.

verifier. Redefine the verification function V_A as

$$V_A(s) = \begin{cases} true; & \text{if } E_e(s) \in \mathcal{M}', \\ false; & \text{otherwise.} \end{cases}$$

Under this new scenario A only needs to transmit the signature s since the message $m = E_e(s)$ can be recovered by applying the verification function. Such a scheme is called a digital signature scheme with message recovery. Figure 13 illustrates how this signature function is used.

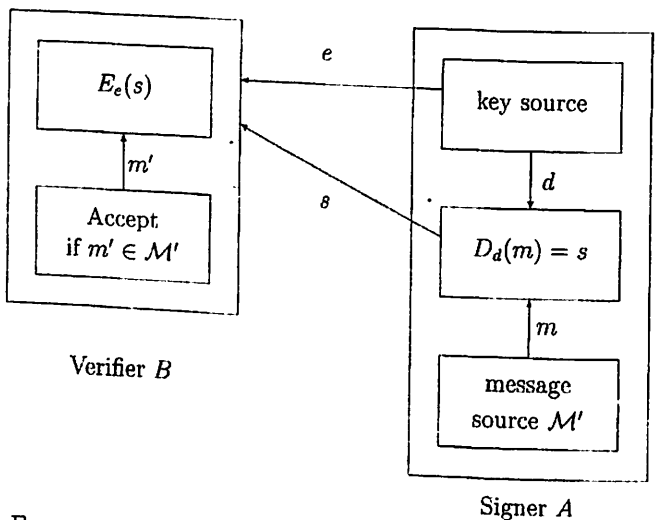

FIGURE 13. A digital signature scheme with message recovery.

The feature of selecting messages of special structure is referred to as selecting messages with redundancy.

The modification presented above is more than a simplification; it is absolutely crucial if one hopes to meet the requirement of property (b) of signing and verification functions. To see why this is the case, note that any entity B can select a random element $s \in \mathcal{S}$ as a signature and apply E_e to get $u = E_e(s)$, since $\mathcal{S} = \mathcal{M}$ and E_e is public knowledge. B may then take the message $m = u$ and the signature on m to be s and transmits (m, s). It is easy to check that s will verify as a signature created by A for m but in which A has had no part. In this case B has forged a signature of A. This is an example of what is called existential forgery. (B has produced A's signature on some message likely not of B's choosing.) If $\mathcal{M}'$ contains only a negligible fraction of messages from $\mathcal{M}$, then the probability of some entity forging a signature of A in this manner is negligibly small.

Remark (*digital signatures vs. confidentiality*) Although digital signature schemes based on reversible public-key encryption are attractive, they require an encryption method as a primitive. There are

situations where a digital signature mechanism is required but encryption is forbidden. In such cases these digital signature schemes are inappropriate.

Digital signatures in practice

For digital signatures to be useful in practice, concrete realizations of the preceding concepts should have certain additional properties. A digital signature must

1. be easy to compute by the signer (the signing function should be easy to apply);
2. be easy to verify by anyone (the verification function should be easy to apply); and
3. have an appropriate lifespan, i.e., be computationally secure from forgery until the signature is no longer necessary for its original purpose.

Resolution of disputes

The purpose of a digital signature (or any signature method) is to permit the resolution of disputes. For example, an entity A could at some point deny having signed a message or some other entity B could falsely claim that a signature on a message was produced by A. In order to overcome such problems a trusted third party (TTP) or judge is required. The TTP must be some entity which all parties involved agree upon in advance. If A denies that a message m held by B was signed by A, then B should be able to present the signature s_A for m to the TTP along with m. The TTP rules in favor of B if $V_A(m, sA) = true$ and in favor of A otherwise. B will accept the decision if B is confident that the TTP has the same verifying transformation V_A as A does. A will accept the decision if A is confident that the TTP used V_A and that S_A has not been compromised. Therefore, fair resolution of disputes requires that the following criteria are met.

Requirements for resolution of disputed signatures

1. S_A and V_A have properties (a) and (b) of page 124.
2. The TTP has an authentic copy of V_A.
3. The signing transformation S_A has been kept secret and remains secure.

These properties are necessary but in practice it might not be possible to guarantee them. For example, the assumption that S_A and V_A have the desired characteristics given in property 1 might turn out to be false for a particular signature scheme. Another possibility is that A claims falsely that S_A was compromised. To overcome these problems requires an agreed method to validate the time period for which A will accept responsibility for the verification transformation. An analogue of this situation can be made with credit card revocation. The holder of

a card is responsible until the holder notifies the card issuing company that the card has been lost or stolen.

Symmetric-key vs. public-key cryptography

Symmetric-key and public-key encryption schemes have various advantages and disadvantages, some of which are common to both. This section highlights a number of these and summarizes features pointed out in previous sections.

(i) Advantages of symmetric-key cryptography

1. Symmetric-key ciphers can be designed to have high rates of data throughput. Some hardware implementations achieve encrypt rates of hundreds of megabytes per second, while software implementations may attain throughput rates in the megabytes per second range.

2. Keys for symmetric-key ciphers are relatively short.

3. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions, and computationally efficient digital signature schemes, to name just a few.

4. Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyze, but on their own weak, can be used to construct strong product ciphers.

5. Symmetric-key encryption is perceived to have an extensive history, although it must be acknowledged that, notwithstanding the invention of rotor machines earlier, much of the knowledge in this area has been acquired subsequent to the invention of the digital computer, and, in particular, the design of the Data Encryption Standard in the early 1970s.

(ii) Disadvantages of symmetric-key cryptography

1. In a two-party communication, the key must remain secret at both ends.

2. In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP (Definition 17.31.

3. In a two-party communication between entities A and B, sound cryptographic practice dictates that the key be changed frequently, and perhaps for each communication session.

4. Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP.

(iii) Advantages of public-key cryptography

1. Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).

2. The administration of keys on a network requires the presence of only a functionally trusted TTP (Definition 17.32) as opposed to an unconditionally trusted TTP. Depending on the mode of usage, the TTP might only be required in an "off-line" manner, as opposed to in real time.

3. Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).

4. Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.

5. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

(iv) Disadvantages of public-key encryption

1. Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes.

2. Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

3. No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key encryption schemes found to date have their security based on the presumed difficulty of a small set of number-theoretic problems.

4. Public-key cryptography does not have as extensive a history as symmetric-key encryption, being discovered only in the mid 1970s.

Summary of comparison

Symmetric-key and public-key encryption have a number of complementary advantages. Current cryptographic systems exploit the strengths of each. An example will serve to illustrate. Public-key encryption techniques may be used to establish a key for a symmetric-key system being used by communicating entities A and B. In this scenario A and B can take advantage of the long term nature of the public/private keys of the public-key scheme and the performance efficiencies of the symmetric-key scheme. Since data encryption is frequently the most time consuming part of the encryption process, the public-key scheme for key establishment is a small fraction of the total encryption process between A and B. To date, the computational performance of public-key encryption is inferior to that of symmetric-key

encryption. There is, however, no proof that this must be the case. The important points in practice are:

1. public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key mangement; and
2. symmetric-key cryptography is efficient for encryption and some data integrity applications.

Remark (*key sizes: symmetric key vs. private key*) Private keys in public-key systems must be larger (e.g., 1024 bits for RSA) than secret keys in symmetric-key systems (e.g., 64 or 128 bits) because whereas (for secure algorithms) the most efficient attack on symmetric key systems is an exhaustive key search, all known public-key systems are subject to "shortcut" attacks (e.g., factoring) more efficient than exhaustive search. Consequently, for equivalent security, symmetric keys have bitlengths considerably smaller than that of private keys in public-key systems, e.g., by a factor of 10 or more.

0.9. Hash functions.

One of the fundamental primitives in modern cryptography is the cryptographic hash function, often informally called a one-way hash function. Asimplified definition for the present discussion follows.

DEFINITION 17.26. A hash function is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values.

For a hash function which outputs n-bit hash-values (e.g., $n = 128$ or 160) and has desirable properties, the probability that a randomly chosen string gets mapped to a particular n-bit hash-value (image) is 2^{-n}. The basic idea is that a hash-value serves as a compact representative of an input string. To be of cryptographic use, a hash function h is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value (i.e., two colliding inputs x and y such that $h(x) = h(y)$), and that given a specific hash-value y, it is computationally infeasible to find an input (pre-image) x such that $h(x) = y$. The most common cryptographic uses of hash functions are with digital signatures and for data integrity. With digital signatures, a long message is usually hashed (using a publicly available hash function) and only the hash-value is signed. The party receiving the message then hashes the received message, and verifies that the received signature is correct for this hash-value. This saves both time and space compared to signing the message directly, which would typically involve splitting the message into appropriate-sized blocks and signing each block individually. Note here that the inability to find

two messages with the same hash-value is a security requirement, since otherwise, the signature on one message hash-value would be the same as that on another, allowing a signer to sign one message and at a later point in time claim to have signed another. Hash functions may be used for data integrity as follows. The hash-value corresponding to a particular input is computed at some point in time. The integrity of this hash-value is protected in some manner. At a subsequent point in time, to verify that the input data has not been altered, the hash-value is recomputed using the input at hand, and compared for equality with the original hash-value. Specific applications include virus protection and software distribution. A third application of hash functions is their use in protocols involving a priori commitments, including some digital signature schemes and identification protocols. Hash functions as discussed above are typically publicly known and involve no secret keys. When used to detect whether the message input has been altered, they are called modification detection codes (MDCs). Related to these are hash functions which involve a secret key, and provide data origin authentication as well as data integrity; these are called message authentication codes (MACs).

0.10. Protocols and mechanisms.

DEFINITION 17.27. A cryptographic protocol (protocol) is a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective.

Remark (*protocol vs. mechanism*) As opposed to a protocol, a mechanism is a more general term encompassing protocols, algorithms (specifying the steps followed by a single entity), and non-cryptographic techniques (e.g., hardware protection and procedural controls) to achieve specific security objectives.

Protocols play a major role in cryptography and are essential in meeting cryptographic goals. Encryption schemes, digital signatures, hash functions, and random number generation are among the primitives which may be utilized to build a protocol.

EXAMPLE 17.20. (a simple key agreement protocol) Alice and Bob have chosen a symmetric-key encryption scheme to use in communicating over an unsecured channel. To encrypt information they require a key. The communication protocol is the following:

1. Bob constructs a public-key encryption scheme and sends his public key to Alice over the channel.
2. Alice generates a key for the symmetric-key encryption scheme.

3. Alice encrypts the key using Bob's public key and sends the encrypted key to Bob.

4. Bob decrypts using his private key and recovers the symmetric (secret) key.

5. Alice and Bob begin communicating with privacy by using the symmetric-key system and the common secret key.

This protocol uses basic functions to attempt to realize private communications on an unsecured channel. The basic primitives are the symmetric-key and the public-key encryption schemes. The protocol has shortcomings including the impersonation attack, but it does convey the idea of a protocol.

Often the role of public-key encryption in privacy communications is exactly the one suggested by this protocol - public-key encryption is used as a means to exchange keys for subsequent use in symmetric-key encryption, motivated by performance differences between symmetric-key and public-key encryption.

Protocol and mechanism failure

DEFINITION 17.28. A protocol failure or mechanism failure occurs when a mechanism fails to meet the goals for which it was intended, in a manner whereby an adversary gains advantage not by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself.

EXAMPLE 17.21. (mechanism failure) Alice and Bob are communicating using a stream cipher. Messages which they encrypt are known to have a special form: the first twenty bits carry information which represents a monetary amount. An active adversary can simply XOR an appropriate bitstring into the first twenty bits of cipher text and change the amount. While the adversary has not been able to read the underlying message, she has been able to alter the transmission. The encryption has not been compromised but the protocol has failed to perform adequately; the inherent assumption that encryption provides data integrity is incorrect.

EXAMPLE 17.22. (forward search attack) Suppose that in an electronic bank transaction the 32- bit field which records the value of the transaction is to be encrypted using a public-key scheme. This simple protocol is intended to provide privacy of the value field - but does it? An adversary could easily take all 2^32 possible entries that could be plain text in this field and encrypt them using the public encryption function. (Remember that by the very nature of public-key encryption this function must be available to the adversary.) By comparing each of the 2^32 cipher texts with the one which is actually encrypted in the transaction, the adversary can determine the plain text. Here the public-key encryption function is not compromised, but rather the way it is used. A closely related attack which applies directly to authentication for access control purposes is the dictionary attack.

Remark (*causes of protocol failure*) Protocols and mechanisms may fail for a number of reasons, including:

1. weaknesses in a particular cryptographic primitive which may be amplified by the protocol or mechanism;

2. claimed or assumed security guarantees which are overstated or not clearly understood; and

3. the oversight of some principle applicable to a broad class of primitives such as encryption.

Example 17.21 illustrates item 2 if the stream cipher is the one-time pad, and also item 1.

Example 17.22 illustrates item 3.

Remark (*protocol design*) When designing cryptographic protocols and mechanisms, the following two steps are essential:

1. identify all assumptions in the protocol or mechanism design; and

2. for each assumption, determine the effect on the security objective if that assumption is violated.

0.11. Key establishment, management, and certification. This section gives a brief introduction to methodology for ensuring the secure distribution of keys for cryptographic purposes.

DEFINITION 17.29. Key establishment is any process whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use.

DEFINITION 17.30. Key management is the set of processes and mechanisms which support key establishment and the maintenance of ongoing keying relationships between parties, including replacing older keys with new keys as necessary.

Key establishment can be broadly subdivided into key agreement and key transport. Many and various protocols have been proposed to provide key establishment. For the purpose of this chapter only a brief overview of issues related to key management will be given. Simple architectures based on symmetric key and public-key cryptography along with the concept of certification will be addressed. A major issue when using symmetric-key techniques is the establishment of pairwise secret

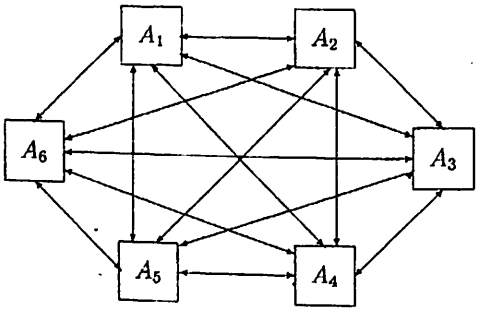

FIGURE 14. Keying relationships in a simple 6-party network.

keys. This becomes more evident when considering a network of entities, any two of which may wish to communicate. Figure 14 illustrates a network consisting of 6 entities. The arrowed edges indicate the 15 possible two-party communications which could take place. Since each pair of entities wish to communicate, this small network requires the secure exchange of $\binom{6}{2} = 15$ key pairs. In a network with n entities, the number of secure key exchanges required is $\binom{n}{2} = \frac{n(n-1)}{2}$.

The network diagram depicted in Figure 14 is simply the amalgamation of 15 two party communications as depicted in Figure 7. In practice, networks are very large and the key management problem is a crucial issue. There are a number of ways to handle this problem. Two simplistic methods are discussed; one based on symmetric-key and the other on public-key techniques.

Key management through symmetric-key techniques

One solution which employs symmetric-key techniques involves an entity in the network which is trusted by all other entities. This entity is referred to as a trusted third party (TTP). Each entity A_i shares a distinct symmetric key k_i with the TTP. These keys are assumed to have been distributed over a secured channel. If two entities subsequently wish to communicate, the TTP generates a key k (sometimes called a session key) and sends it encrypted under each of the fixed keys as depicted in Figure 15 for entities A_1 and A_6.

Advantages of this approach include:

1. It is easy to add and remove entities from the network.
2. Each entity needs to store only one long-term secret key.

Disadvantages include:

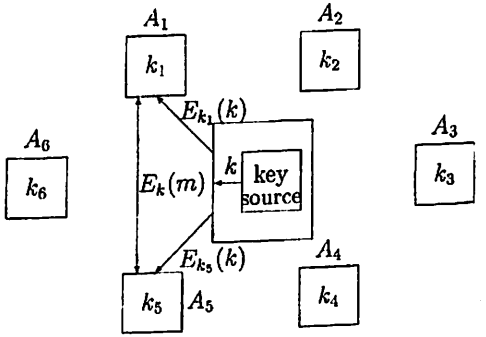

FIGURE 15. Key management using a trusted third party (TTP).

1. All communications require initial interaction with the TTP.
2. The TTP must store n long-term secret keys.
3. The TTP has the ability to read all messages.
4. If the TTP is compromised, all communications are insecure.

Key management through public-key techniques

There are a number of ways to address the key management problem through public-key techniques. For the purpose of this chapter a very simple model is considered. Each entity in the network has a public/private encryption key pair. The public key along with the identity of the entity is stored in a central repository called a public file. If an entity A_1 wishes to send encrypted messages to entity A_6, A_1 retrieves the public key e_6 of A_6 from the public file, encrypts the message using this key, and sends the cipher text to A_6. Figure 16 depicts such a network.

Advantages of this approach include:

1. No trusted third party is required.
2. The public file could reside with each entity.
3. Only n public keys need to be stored to allow secure communications between any pair of entities, assuming the only attack is that by a passive adversary. The key management problem becomes more difficult when one must take into account an adversary who is active (i.e. an adversary who can alter the public file containing public keys). Figure 17 illustrates how an active adversary could compromise the

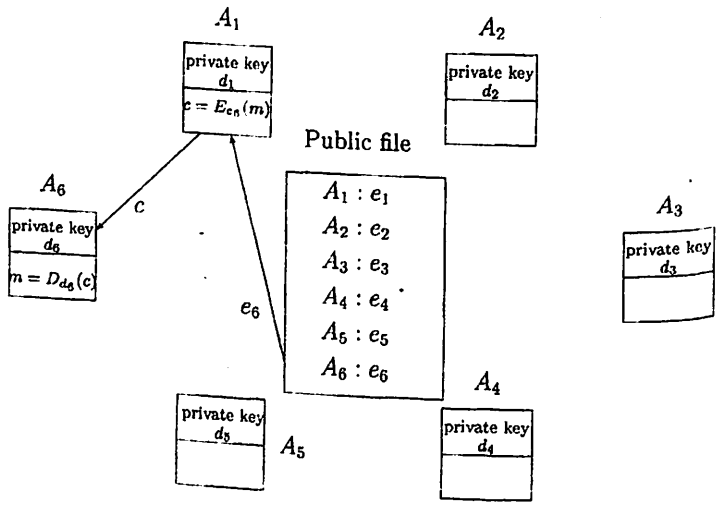

FIGURE 16. Key management using public-key techniques.

key management scheme given above. In the figure, the adversary alters the public file by replacing the public key e_6 of entity A_6 by the adversary's public key e

Any message encrypted for A_6 using the public key from the public file can be decrypted by only the adversary. Having decrypted and read the message, the adversary can now encrypt it using the public key of A_6 and forward the cipher text to A_6. A_1 however believes that only A_6 can decrypt the cipher text c.

To prevent this type of attack, the entities may use a TTP to certify the public key of each entity. The TTP has a private signing algorithm S_T and a verification algorithm V_T assumed to be known by all entities. The TTP carefully verifies the identity of each entity, and signs a message consisting of an identifier and the entity's authentic public key. This is a simple example of a certificate, binding the identity of an entity to its public key. Figure 18 illustrates the network under these conditions. A_1 uses the public key of A_6 only if the certificate signature verifies successfully.

Advantages of using a TTP to maintain the integrity of the public file include:

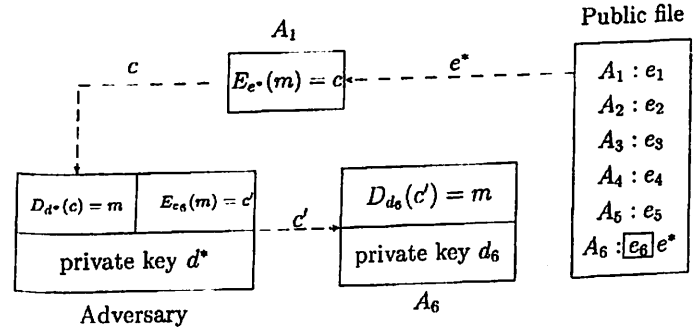

FIGURE 17. An impersonation of A_6 by an active adversary with public key e.

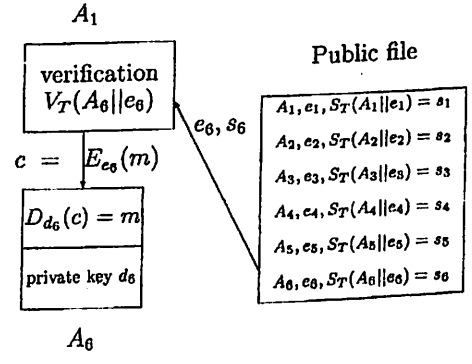

FIGURE 18. Authentication of public keys by a TTP. || denotes concatenation.

1. It prevents an active adversary from impersonation on the network.

2. The TTP cannot monitor communications. Entities need trust the TTP only to bind identities to public keys properly.

3. Per-communication interaction with the public file can be eliminated if entities store certificates locally.

Even with a TTP, some concerns still remain:

1. If the signing key of the TTP is compromised, all communications become insecure.

2. All trust is placed with one entity.

Trusted third parties and public-key certificates

A trusted third party has been used in this chapter. The trust placed on this entity varies with the way it is used, and hence motivates the following classification.

DEFINITION 17.31. A TTP is said to be unconditionally trusted if it is trusted on all matters. For example, it may have access to the secret and private keys of users, as well as be charged with the association of public keys to identifiers.

DEFINITION 17.32. A TTP is said to be functionally trusted if the entity is assumed to be honest and fair but it does not have access to the secret or private keys of users.

Public-key certificates

The distribution of public keys is generally easier than that of symmetric keys, since secrecy is not required. However, the integrity (authenticity) of public keys is critical. A public-key certificate consists of a data part and a signature part. The data part consists of the name of an entity, the public key corresponding to that entity, possibly additional relevant information (e.g., the entity's street or network address, a validity period for the public key, and various other attributes). The signature part consists of the signature of a TTP over the data part. In order for an entity B to verify the authenticity of the public key of an entity A, B must have an authentic copy of the public signature verification function of the TTP. For simplicity, assume that the authenticity of this verification function is provided to B by noncryptographic means, for example by B obtaining it from the TTP in person. B can then carry out the following steps:

1. Acquire the public-key certificate of A over some unsecured channel, either from a central database of certificates, from A directly, or otherwise.

2. Use the TTP's verification function to verify the TTP's signature on A's certificate.

3. If this signature verifies correctly, accept the public key in the certificate as A's authentic public key; otherwise, assume the public key is invalid.

Before creating a public-key certificate for A, the TTP must take appropriate measures to verify the identity of A and the fact that the public key to be certificated actually belongs to A. One method is to require that A appear before the TTP with a conventional passport as

proof of identity, and obtain A's public key from A in person along with evidence that A knows the corresponding private key. Once the TTP creates a certificate for a party, the trust that all other entities have in the authenticity of the TTP's public key can be used transitively to gain trust in the authenticity of that party's public key, through acquisition and verification of the certificate.

0.12. Pseudorandom numbers and sequences. Random number generation is an important primitive in many cryptographic mechanisms. For example, keys for encryption transformations need to be generated in a manner which is unpredictable to an adversary. Generating a random key typically involves the selection of random numbers or bit sequences. Random number generation presents challenging issues. A brief introduction is given here. Often in cryptographic applications, one of the following steps must be performed:

(i) From a finite set of n elements (e.g., $\{1, 2, ..., n\}$), select an element at random.

(ii) From the set of all sequences (strings) of length m over some finite alphabet A of n symbols, select a sequence at random.

(iii) Generate a random sequence (string) of symbols of length m over a set of n symbols. It is not clear what exactly it means to select at random or generate at random. Calling a number random without a context makes little sense. Is the number 23 a random number? No, but if 49 identical balls labeled with a number from 1 to 49 are in a container, and this container mixes the balls uniformly, drops one ball out, and this ball happens to be labeled with the number 23, then one would say that 23 was generated randomly from a uniform distribution. The probability that 23 drops out is 1 in 49 or $\frac{1}{49}$. If the number on the ball which was dropped from the container is recorded and the ball is placed back in the container and the process repeated 6 times, then a random sequence of length 6 defined on the alphabet $A = \{1, 2, ..., 49\}$ will have been generated. What is the chance that the sequence 17, 45, 1, 7, 23, 35 occurs? Since each element in the sequence has probability $\frac{1}{49}$ of occuring, the probability of the sequence 17, 45, 1, 7, 23, 35 occurring is

$$\frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} \times \frac{1}{49} = \frac{1}{13841287201}.$$

There are precisely 13841287201 sequences of length 6 over the alphabet A. If each of these sequences is written on one of 13841287201 balls and they are placed in the container (first removing the original 49 balls) then the chance that the sequence given above drops out is the same as if it were generated one ball at a time. Hence, (ii) and (iii)

above are essentially the same statements. Finding good methods to generate random sequences is difficult.

EXAMPLE 17.23. (random sequence generator) To generate a random sequence of 0's and 1's, a coin could be tossed with a head landing up recorded as a 1 and a tail as a 0. It is assumed that the coin is unbiased,which means that the probability of a 1 on a given toss is exactly $\frac{1}{2}$. This will depend on how well the coin is made and how the toss is performed. This method would be of little value in a system where random sequences must be generated quickly and often. It has no practical value other than to serve as an example of the idea of random number generation.

EXAMPLE 17.24. (random sequence generator) A noise diode may be used to produce random binary sequences. This is reasonable if one has some way to be convinced that the probability that a 1 will be produced on any given trial is $\frac{1}{2}$. Should this assumption be false, the sequence generated would not have been selected from a uniform distribution and so not all sequences of a given length would be equally likely. The only way to get some feeling for the reliability of this type of random source is to carry out statistical tests on its output. If the diode is a source of a uniform distribution on the set of all binary sequences of a given length, it provides an effective way to generate random sequences.

Since most true sources of random sequences (if there is such a thing) come from physical means, they tend to be either costly or slow in their generation. To overcome these problems, methods have been devised to construct pseudorandom sequences in a deterministic manner from a shorter random sequence called a seed. The pseudorandom sequences appear to be generated by a truly random source to anyone not knowing the method of generation. Often the generation algorithm is known to all, but the seed is unknown except by the entity generating the sequence. A plethora of algorithms has been developed to generate pseudorandom bit sequences of various types. Many of these are completely unsuitable for cryptographic purposes and one must be cautious of claims by creators of such algorithms as to the random nature of the output.

0.13. Classes of attacks and security models. Over the years, many different types of attacks on cryptographic primitives and protocols have been identified. The discussion here limits consideration to attacks on encryption and protocols. In this chapter the roles of

an active and a passive adversary were discussed. The attacks these adversaries can mount may be classified as follows:

1. A passive attack is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data.

2. An active attack is one where the adversary attempts to delete, add, or in some other way alter the transmission on the channel. An active attacker threatens data integrity and authentication as well as confidentiality.

A passive attack can be further subdivided into more specialized attacks for deducing plain text from cipher text.

Attacks on encryption schemes

The objective of the following attacks is to systematically recover plain text from cipher text, or even more drastically, to deduce the decryption key.

1. A cipher text-only attack is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plain text by only observing cipher text. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.

2. A known-plain text attack is one where the adversary has a quantity of plain text and corresponding cipher text. This type of attack is typically only marginally more difficult to mount.

3. A chosen-plain text attack is one where the adversary chooses plain text and is then given corresponding cipher text. Subsequently, the adversary uses any information deduced in order to recover plain text corresponding to previously unseen cipher text.

4. An adaptive chosen-plain text attack is a chosen-plain text attack wherein the choice of plain text may depend on the cipher text received from previous requests.

5. A chosen-cipher text attack is one where the adversary selects the cipher text and is then given the corresponding plain text. One way to mount such an attack is for the adversary to gain access to the equipment used for decryption (but not the decryption key, which may be securely embedded in the equipment). The objective is then to be able, without access to such equipment, to deduce the plain text from (different) cipher text.

6. An adaptive chosen-cipher text attack is a chosen-cipher text attack where the choice of cipher text may depend on the plain text received from previous requests.

Most of these attacks also apply to digital signature schemes and message authentication codes. In this case, the objective of the attacker is to forge messages or MACs, respectively.

Attacks on protocols

The following is a partial list of attacks which might be mounted on various protocols. Until a protocol is proven to provide the service intended, the list of possible attacks can never be said to be complete.

1. known-key attack. In this attack an adversary obtains some keys used previously and then uses this information to determine new keys.

2. replay. In this attack an adversary records a communication session and replays the entire session, or a portion thereof, at some later point in time.

3. impersonation. Here an adversary assumes the identity of one of the legitimate parties in a network.

4. dictionary. This is usually an attack against passwords. Typically, a password is stored in a computer file as the image of an unkeyed hash function. When a user logs on and enters a password, it is hashed and the image is compared to the stored value. An adversary can take a list of probable passwords, hash all entries in this list, and then compare this to the list of true encrypted passwords with the hope of finding matches.

5. forward search. This attack is similar in spirit to the dictionary attack and is used to decrypt messages. An example of this method was cited in Example 17.22.

6. interleaving attack. This type of attack usually involves some form of impersonation in an authentication protocol.

Models for evaluating security

The security of cryptographic primitives and protocols can be evaluated under several different models. The most practical security metrics are computational, provable, and ad hoc methodology, although the latter is often dangerous. The confidence level in the amount of security provided by a primitive or protocol based on computational or ad hoc security increases with time and investigation of the scheme. However, time is not enough if few people have given the method careful analysis.

(i) Unconditional security

The most stringent measure is an information-theoretic measure - whether or not a system has unconditional security. An adversary is assumed to have unlimited computational resources, and the question is whether or not there is enough information available to defeat the system. Unconditional security for encryption systems is called perfect secrecy. For perfect secrecy, the uncertainty in the plain text, after observing the cipher text, must be equal to the a priori uncertainty about the plain text - observation of the cipher text provides no information whatsoever to an adversary. A necessary condition for a

symmetric-key encryption scheme to be unconditionally secure is that the key be at least as long as the message. The one-time pad is an example of an unconditionally secure encryption algorithm. In general, encryption schemes do not offer perfect secrecy, and each cipher text character observed decreases the theoretical uncertainty in the plain text and the encryption key. Public-key encryption schemes cannot be unconditionally secure since, given a cipher text c, the plain text can in principle be recovered by encrypting all possible plain texts until c is obtained.

(ii) Complexity-theoretic security

An appropriate model of computation is defined and adversaries are modeled as having polynomial computational power. (They mount attacks involving time and space polynomial in the size of appropriate security parameters.) A proof of security relative to the model is then constructed. An objective is to design a cryptographic method based on the weakest assumptions possible anticipating a powerful adversary. Asymptotic analysis and usually also worst-case analysis is used and so care must be exercised to determine when proofs have practical significance. In contrast, polynomial attacks which are feasible under the model might, in practice, still be computationally infeasible. Security analysis of this type, although not of practical value in all cases, may nonetheless pave the way to a better overall understanding of security. Complexity-theoretic analysis is invaluable for formulating fundamental principles and confirming intuition. This is like many other sciences, whose practical techniques are discovered early in the development, well before a theoretical basis and understanding is attained.

(iii) Provable security

A cryptographic method is said to be provably secure if the difficulty of defeating it can be shown to be essentially as difficult as solving a well-known and supposedly difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms. Thus, "provable" here means provable subject to assumptions. This approach is considered by some to be as good a practical analysis technique as exists. Provable security may be considered part of a special sub-class of the larger class of computational security considered next.

(iv) Computational security

This measures the amount of computational effort required, by the best currently-known methods, to defeat a system; it must be assumed here that the system has been well-studied to determine which attacks are relevant. A proposed technique is said to be computationally secure if the perceived level of computation required to defeat it (using the

best attack known) exceeds, by a comfortable margin, the computational resources of the hypothesized adversary. Often methods in this class are related to hard problems but, unlike for provable security, no proof of equivalence is known. Most of the best known public-key and symmetrickey schemes in current use are in this class. This class is sometimes also called practical security.

(v) Ad hoc security

This approach consists of any variety of convincing arguments that every successful attack requires a resource level (e.g., time and space) greater than the fixed resources of a perceived adversary. Cryptographic primitives and protocols which survive such analysis are said to have heuristic security, with security here typically in the computational sense. Primitives and protocols are usually designed to counter standard attacks. While perhaps the most commonly used approach (especially for protocols), it is, in some ways, the least satisfying. Claims of security generally remain questionable and unforeseen attacks remain a threat.

Perspective for computational security

To evaluate the security of cryptographic schemes, certain quantities are often considered.

DEFINITION 17.33. The work factor W_d is the minimum amount of work (measured in appropriate units such as elementary operations or clock cycles) required to compute the private key d given the public key e, or, in the case of symmetric-key schemes, to determine the secret key k.

More specifically, one may consider the work required under a cipher text-only attack given n cipher texts, denoted $W_d(n)$. If W_d is t years, then for sufficiently large t the cryptographic scheme is, for all practical purposes, a secure system. To date no public-key system has been found where one can prove a sufficiently large lower bound on the work factor W_d. The best that is possible to date is to rely on the following as a basis for security.

DEFINITION 17.34. The historical work factor W_d is the minimum amount of work required to compute the private key d from the public key e using the best known algorithms at a given point in time.

The historical work factor W_d varies with time as algorithms and technology improve. It corresponds to computational security, whereas W_d corresponds to the true security level, although this typically cannot be determined.

How large is large?

Reference	Magnitude
Seconds in a year	$\approx 3 \times 10^7$
Age of our solar system (years)	$\approx 6 \times 10^9$
Seconds since creation of solar system	$\approx 2 \times 10^{17}$
Clock cycles per year, 50 MHz computer	$\approx 1:6 \times 10^{15}$
Binary strings of length 64	$2^{64} \approx 1.8 \times 10^{19}$
Binary strings of length 128	$2^{128} \approx 3.4 \times 10^{38}$
Binary strings of length 256	$2^{256} \approx 1.2 \times 10^{77}$
Number of 75-digit prime numbers	$\approx 5.2 \times 10^{72}$
Electrons in the universe	$\approx 8.37 \times 10^{77}$

TABLE 2. Reference numbers comparing relative magnitudes.

We described how the designer of an encryption system tries to create a scheme for which the best approach to breaking it is through exhaustive search of the key space. The key space must then be large enough to make an exhaustive search completely infeasible. An important question then is "How large is large?". In order to gain some perspective on the magnitude of numbers, Table 2 lists various items along with an associated magnitude.

Some powers of 10 are referred to by prefixes. For example, high-speed modern computers are now being rated in terms of teraflops where a teraflop is 10^{12} floating point operations per second. Table 3 provides a list of commonly used prefixes.

Prefix	Symbol	Magnitude
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	k	10^{3}
hecto	h	10^{2}
deca	da	10
deci	d	10^{-1}
centi	c	10^{-2}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}
atto	a	10^{-18}

TABLE 3. Prefixes used for various powers of 10.

Арсланов Марат Зуфарович

ЛЕКЦИИ ПО ТЕОРИИ ИНФОРМАЦИЙ

Arslanov Marat Zufarovich

A COURSE IN INFORMATION THEORY

Подписано в печать 20.01.2009

Формат 60 x 84 1/16. Бумага тип. Ризограф.
Усл.печ.15,7 л. Уч.-изд. 15,9 л. Тираж 500.

Заказ № 700

Цена договорная

Издание Казахской головной архитектурно-строительной академии
Издательский дом КазГАСА «Строительство и архитектура»
480043, г.Алматы, ул. К.Рыскулбекова, 28

1991 *Mathematics Subject Classification.* 94A20

Key words and phrases. Information theory, coding theory

ABSTRACT. This is a text for a course on information theory and coding theory.

This work was typeset with LaTeX, using the $\mathcal{AMS}$-LaTeX and $\mathcal{AMS}$Fonts packages of the American Mathematical Society.