Some Mathematical Techniques in Cryptography and Related Fields

Charles Blair
c-blair@uiuc.edu

Contents

1 Subset-Sum (Knapsack) problems and their uses

1.1 Subset-sum problems are hard

A subset of the numbers

$$267 \quad 493 \quad 869 \quad 961 \quad 1000 \quad 1153 \quad 1246 \quad 1598 \quad 1766 \quad 1922$$

adds up to 5842. Spend a few minutes trying to find such a subset. Whether you succeed or not, I hope you are convinced the task is not trivial.

This is an example of a *subset-sum* problem. In general, we are given n natural numbers a_i and a target number T and asked to find a $S \subset \{1, \ldots, n\}$ with

$$\sum_{i \in S} a_i = T \qquad (*)$$

A seemingly simpler problem is the *subset-sum decision* problem. For a given a_i and T, decide whether there is an S for which $(*)$ holds, without being required to identify such an S. However, it can be proved that the decision problem is just as difficult as the subset-sum problem in this sense:

Theorem 1 *Suppose we had a method of solving the subset-sum decision problem. Then we could solve a subset-sum problem using the assumed method n times.*

(the n is not particularly important— the main thing is that the number of uses of the method is not too large.)

Proof: Begin by using the method to decide if T is a sum of the a_i— if not, we can stop immediately. Then use the method to determine if $(*)$ holds for some $S \subset \{2, \ldots, n\}$. If the answer is yes, we ignore a_1 for the rest of the analysis. If the answer is no, we know we must have $1 \in S$. In this second case, we continue by using the method to decide whether there is $S \subset \{3, \ldots, n\}$ with

$$\sum_{i \in S} a_i = T - a_1$$

A yes answer means we can assume $2 \notin S$, otherwise $2 \in S$.

The idea of this proof is more important than the specific result. We show that one problem is as difficult as another by showing that a method of solving the supposedly easier problem can be used to solve another problem. This involves constructing one or several easier problems whose solution answers the hard problem.

Using more elaborate versions of the techniques in Theorem 1, it can be shown that a method of solving the subset-sum decision problem could be used to solve many other problems, including:

- Factoring
- The Travelling Salesman Problem
- Any Integer Programming Problem

Don't worry if you are not familiar with the details of these problems. The important point is that they are all well-known problems for which many people have been unable to find efficient solution methods, which makes it unlikely that there is a method which solves all subset-sum decision problems efficiently.

The discussion above makes it plausible that some subset-sum problems are difficult. Further, there is some evidence that the "typical" subset-sum problem is not easy. V. Chvatal[1] has shown that if the a_i and T are randomly chosen, then with high probability (i) there will be no S for which $(*)$ holds and (ii) certain simple ways of proving this will not work.

1.2 Uses of the subset-sum problem

1.2.1 Computer passwords

A computer needs to verify a user's identity before allowing him or her access to an account. The simplest system would have the machine keep a copy of the password in an internal file, and compare it with what the user types. A drawback is that anyone who sees the internal file could later impersonate the user.

I believe this alternative is actually implemented on some systems: the computer generates a large number (say 500) of a_i. They are stored in the internal file. A password is a subset of $\{1, \ldots, 500\}$. (in practice, there is a program to convert a normal sequence-of-symbols password to such a subset.)

[1] "Hard Knapsack Problems," *Operations Research*, vol. 28, pp 1402–1411

Instead of having the password for the user, the computer keeps the total associated with the appropriate subset. When the user types in the subset, the computer tests whether the total is correct. It does not keep a record of the subset. Thus impersonation is possible only if somebody can reconstruct the subset knowing the a_i and the total.

1.2.2 Message verification

A sender (S) wants to send messages to a receiver (R). Keeping the message secret is not important. However, R wants to be sure that the message he is receiving is not from an imposter and has not been tampered with. S and R agree on a set of a_i (say 500) and a set of totals T_j (say 200). These numbers may be publicly known, but only S knows which subsets of the a_i correspond to which T_j. The message sent by S is a subset of size 100 of $\{1, \ldots, 200\}$. He does this by sending 100 subsets of the a_i corresponding to the message he wants to send.

2 Encryption Systems

An encryption system is a procedure which takes the original message (*plaintext*) and a small piece of information arranged in advance between sender and receiver (the *key*) and creates an encoded version of the message (the *ciphertext*).

Usually one assumes the person trying to break the code is only working with the ciphertext. However, there are situations in which both plaintext and ciphertext of a previously encoded message are available. For example, I often keep encrypted versions of examinations on a mainframe computer, only decoding them just before having them printed, and deleting the plaintext file immediately afterward. If a student was able to look at my files, he could keep a copy of the encoded test and compare this with the test he took. This may be very useful in decoding future tests.

A more demanding standard is that a code may be safe against a *chosen-plaintext attack*. We are imagining that the encryption is done by a machine, and that unauthorized persons may have access to the machine, and may type in any message to see how it is encrypted.

Example: Suppose the machine takes each 60-character line and writes the characters in a different order. If we could do a chosen-plaintext attack, we would use a message with all the characters different (all the small letters, all the capital letters, and some digits, for example) to find out immediately what permutation was being used.

Public-Key encryption carries the idea of a code that is safe from chosen-plaintext attack one step further. The "bad guy" may not only type messages on the encryption machine and note the encrypted version, he is allowed to take the machine apart and see exactly how it works.

In the next section, we will present some material from number theory, which can be used to construct a public-key system.

3 Introduction to Number Theory

3.1 Congruences

The congruence $a \equiv b \bmod n$ ("a is congruent to b mod n") says that, when divided by n, a and b have the same remainder.

$$100 \equiv 34 \bmod 11 \qquad -6 \equiv 10 \bmod 8$$

In the second congruence, we are using $-6 = 8(-1) + 2$. We always have $a \equiv b \bmod n$ for some $0 \leq b \leq n-1$, and we are usually concerned with that b. If $a \equiv b \bmod n$ and $c \equiv d$, we can add or multiply

$$a + c \equiv b + d \bmod n \qquad ac \equiv bd \bmod n$$

Division does not always work: $6 \equiv 18 \bmod 12$ but $3 \not\equiv 9 \bmod 12$.

3.2 The Greatest Common Divisor

For a and b, the number (a, b) is the largest number which divides a and b evenly.

$$(56, 98) = 14 \qquad (76, 190) = 38$$

Theorem 2 *For any a, b there are integers x, y with $ax + by = (a, b)$*

Proof: The equation can be solved by making a sequence of simplifying substitutions:

$$30x + 69y = 3$$

$$\begin{aligned}30x' + 9y &= 3 \quad [x' = x + 2y] \\ 3x' + 9y' &= 3 \quad [y' = y + 3x'] \\ 3x'' + 0y' &= 3 \quad [x'' = x' + 3y']\end{aligned}$$

It is easy to see that $x'' = 1$, $y' = 0$ is a solution to the final equation and we get a solution to the original equation by working backwards:

$$x' = x'' - 3y' = 1 \quad y = y' - 3x' = -3 \quad x = x' - 2y = 7$$

We could also solve an equation like $30x + 69y = 15$ by multiplying our solution: $y = -15$, $x = 35$. It should be clear that the equation will have no solution in integers if 15 is replaced by something that is not a multiple of $(30, 69) = 3$.

All other integer solutions of $30x+69y = 15$ may be obtained by changing the first solution:

$$y = -15 + \frac{30}{3}t \quad x = 35 - \frac{69}{3}t \quad \text{for } t \text{ integer}$$

If we do the process illustrated on the previous page for any equation $ax+by = (a, b)$, we eventually get one of the coefficients as zero and the other as (a, b). [In fact, this process is usually presented as "Euclid's algorithm for finding the greatest common divisor."]

It is important that this process is feasible [on a computer] even if a and b are several hundred digits long. It is easy to show that the larger of the two coefficients decreases by at least $1/2$ every two equations, hence that in twenty equations the larger coefficient has decreased by $2^{-10} < 10^{-3}$, so a 600-digit number would not require more than 4000 equations. [this analysis can be improved]

We pointed out earlier that division does not work with congruences. An important application of Theorem 2 is that it does work for prime numbers.

Corollary 3 *If p is a prime number, $ar \equiv as$ mod p and $a \not\equiv 0$, then $r \equiv s$.*

Proof: Since p is a prime, $(a, p) = 1$, so Theorem 2 says there are integer x, y with $ax + py = 1$. Hence

$$ax \equiv 1 \text{ mod } p \quad \text{and} \quad r \equiv (1)r \equiv axr \equiv xar \equiv xas \equiv s \text{ mod } p$$

Corollary 4 *If p is a prime number and $a \not\equiv 0$ mod p, then for any b, there is y with $ay \equiv b$ mod p.*

Proof: We showed in the preceding proof that there is x with $ax \equiv 1$ mod p. Let $y = bx$.

Corollary 5 (The "Chinese Remainder Theorem") *If $(p, q) = 1$, then for any a, b, there is an n with*

$$n \equiv a \text{ mod } p \quad \text{and} \quad n \equiv b \text{ mod } q$$

Proof: Theorem 2 implies there are integers x, y such that

$$px + a = qy + b \quad \text{so let } n = px + a$$

3.3 Powers modulo a prime

The sequence

$$a \quad a^2 \quad a^3 \ldots \quad \text{mod } p$$

has many applications in cryptography. Before looking at theoretical properties, the example below (done using a pocket calculator) should make clear that it is practical to compute these numbers, even when many digits are involved.

Suppose we want to compute 432^{678} mod 987. The basic trick is to start with a number and keep squaring:

$$432^2 = 186624 \equiv 81 \quad 432^4 \equiv 81^2 \equiv 639 \quad 432^8 \equiv 639^2 \equiv 690 \ldots 432^{512} \equiv 858$$

Since $678 = 512 + 128 + 32 + 4 + 2$,

$$432^{678} \equiv (81)(639) \ldots (858) \equiv 204 \qquad \text{(I hope!)}$$

Calculations with exponents involve not-too-many multiplications. If the numbers have several hundred digits, however, it is necessary to design special subroutines to do the multiplications (see Knuth, volume 2).

Let us look at the sequence of powers of 2 mod 11:

$$2\ 4\ 8\ 5\ 10\ 9\ 7\ 3\ 6\ 1$$

Each number from 1 to 10 appears in the sequence.

Theorem 6 *Let p be a prime. There is an a such that for every $1 \leq b \leq p-1$, there is $1 \leq x \leq p-1$ such that $a^x \equiv b$ mod p.*

It is not always the case that $a = 2$. The powers of 2 mod 7 are 2, 4, 1 after which the sequence repeats and we never get 3, 5, or 6.

The proof of Theorem 6 is long, and can be found in number theory books. For now, we want to look at some of its consequences.

Corollary 7 *Let a be as in Theorem 6. Then $a^{p-1} \equiv 1$ mod p.*

Proof: We know that $a^d \equiv 1$ for some $1 \leq d \leq p-1$. If $d < p-1$, the sequence of powers of a would start repeating before we got all the numbers: $a^{d+1} \equiv a$, $a^{d+2} \equiv a^2$, etc.

Corollary 8 (Fermat's Theorem) *For any $b \not\equiv 0$, $b^{p-1} \equiv 1$ mod p.*

Proof: Let a be as in Theorem 6. Using Corollary 7

$$b^{p-1} \equiv a^{x(p-1)} \equiv \left(a^{p-1}\right)^x \equiv 1$$

Corollary 9 *If $x \equiv y$ mod $(p-1)$, then $b^x \equiv b^y$ mod p*

Proof: For some integer r, $y = r(p-1) + x$ and by Corollary 8

$$b^y \equiv \left(b^{p-1}\right)^r b^x \equiv b^x \bmod p$$

Lemma 10 *Let $b \not\equiv 0$, d the smallest positive number such that $b^d \equiv 1$. Then for any $e > 0$ with $b^e \equiv 1$ d divides e evenly. In particular, by Corollary 8, d divides $p-1$ evenly.*

Proof: If d does not divide e, then $e = dr + s$ for some $0 < s < d$, but

$$b^s \equiv b^e \left(b^d\right)^{-r} \equiv 1$$

would contradict the definition of d.

4 A public key system as hard as factoring

The system we present here is due to Michael Rabin. It is a modification of the famous system of Rivest, Shamir, and Adleman.[2] The person who wishes to receive messages announces a number n publicly, with $n = pq$ for primes p, q that are kept secret. For technical reasons, we assume $p, q \equiv 3 \bmod 4$.

The message being transmitted is converted to a number M between 1 and n, and the encoding function is simply:

$$f(M) \equiv M^2 \bmod n$$

Strictly speaking, there is a problem with this encoding function, in that there are four different M which will all give the same $f(M)$. (example: let $n = 133 = 7(19)$. The encoding of $M = 15$ is $92 \equiv 225 \bmod 133$, but $M = 34$, 99, or 118 also have $f(M) = 92$.[3]

However, this is not serious from a practical point of view. We would be implementing this with p, q at least one hundred digits long, and it is very unlikely that more than one of the four numbers would correspond to an intelligible message. The key facts about this system are:

1. If p, q are known, it is easy to compute all the M_i given $f(M)$.
2. If we are given n, $f(M)$, and all the M_i, we can calculate p, q.

The problem of devising an efficient algorithm for factoring large numbers n has been unsolved in spite of intense efforts for several hundred years, so the second point provides substantial reassurance about the strength of the system. (note, by the way, the similarity to the situation in Theorem 1. Just as before, we will prove that an efficient method of solving one problem gives an efficient method of solving another one)

proof of 1: Since $p \equiv 3 \bmod 4$, there is an integer k with $4k = p + 1$. If $x \equiv (f(M))^k \bmod p$, then using Corollary 8:

$$x^2 \equiv \left(\left(M^2\right)^k\right)^2 \equiv M^{4k} \equiv M^2 \bmod p$$

[2] *Communications of the Association for Computing Machinery* 21, 120–126. See also Knuth, *The Art of Computer Programming*, Vol. 2 (second edition), 386–389.

[3] This is not as mysterious as it may look. Note that $15 \equiv 1 \bmod 7$ and $15 \equiv 15 \bmod 19$, while $118 \equiv -1 \bmod 7$ and $118 \equiv -15 \bmod 19$.

Similarly if $y \equiv (f(M))^L \mod q$ $[4L = q+1]$, then $y^2 \equiv M^2 \mod q$. The M_i are given from Corollary 4 by

$$\begin{array}{llll} M_1 \equiv x \mod p & M_2 \equiv x \mod p & M_3 \equiv -x \mod p & M_4 \equiv -x \mod p \\ M_1 \equiv y \mod q & M_2 \equiv -y \mod q & M_3 \equiv y \mod q & M_4 \equiv -y \mod q \end{array}$$

proof of 2: If we know the M_i, there will be two like M_1 and M_3 above with $M_1 + M_3 \equiv 0 \mod p$ and $M_1 + M_3 \not\equiv 0 \mod q$. Thus we can calculate $(M_1 + M_3, n)$ to obtain p.

5 Probabilistic Encryption

We have presented an encryption function f such that the message M presumably cannot be computed from the encoding $f(M)$. A further concern arises as to whether, even if the adversary cannot identify M exactly, he may be able to obtain some partial information about M, for example tell whether M is an even number, a square, a power of 2, etc.

During World War II, the German general staff often began secret messages with several paragraphs that did not change from one message to the next. This was very useful to the English cryptanalysts. It would be nice to know that such patterns will not occur in the systems we use.

Probabilistic encryption[4] is a system designed to avoid these problems. Instead of $f(M)$ being a single number, the calculation of $f(M)$ involves the sender doing some things randomly during the calculation, so that M has many different encryptions. Indeed, the probability should be very close to 1 that if the same message is sent twice, the encryptions should be different.

As with the Rabin system, the receiver publically announces $n = pq$, while keeping p and q secret. In this system, the crucial assumption is that, for a given number k, it is computationally infeasible for somebody who does not know p, q to tell whether there is an x with $x^2 \equiv k \mod n$, in other words, whether k is a square mod n (like factoring, the problem of finding an efficient algorithm for this task has been unsolved for a long time). In addition to announcing n, the receiver announces a single number w which is guaranteed not to be a square.

[4]The idea is due to Goldwasser and Micali, *Journal of Computer & System Sciences* 28, pp. 270–299. See also *Primality and Cryptography*, by E. Kranakis]

The sender converts his message into a sequence of 0's and 1's. Corresponding to each 0, the sender chooses a random number $1 < r < n$, and transmits $r^2 \bmod n$. (for an application, n would be at least 100 digits long, so the chance of sending an easily detectable "real" perfect square would be negligible) Corresponding to each 1, $wr^2 \bmod n$ is transmitted (this is certain not to be a perfect square). A separate random r is used for each bit of the message, leading to an enormous number of different possible encryptions.

The facts that allow the receiver to distinguish squares from non-squares are:

1. a is a square mod n if and only if it is a square mod p and a square mod q.

2. Let $h = \frac{p-1}{2}$. If a is a square mod p, $a^h \equiv 1 \bmod p$. If a is not a square, $a^h \equiv -1$.

6 An *Unbreakable* Encryption System

Yes, there really is such a thing. Further, it does not depend on the advanced mathematics of the preceding sections.

Assume for simplicity that your message is 500 characters long, with each character being one of 120 possibilities (including a lot of possible symbols). Arrange in advance with the sender (this is *not* a public-key system) a table of 500 random numbers, each between 1 and 120. To encode your message, shift each character by the amount specified by the corresponding entry in the table. Thus,if the mumber in the table was 5, you would replace an "e" by a "j."

What makes this system unbreakable? There is no way for the bad guys to guess what the random table looks like. For *any* 500-character plaintext, there is some table of random numbers which would generate the actual ciphertext, and one table is as likely as another.

This system is called a *one-time pad.* It is crucial that you don't use the same table to generate two different random messages. If you did, it would become possible to do analysis based on letter frequencies, etc.

One-time pads seem practical in situations where one agent is communicating with a central command. They become less attractive if several agents may need to communicate with each other. The one-time feature is lost if X and Y inadvertently use the same page to talk as W and Z are using. Also capture of X's equipment makes it possible to overhear a conversation between Y and Z.

7 A recent pseudo-random number generator

The preceding two sections both made use of random numbers, so I will describe a random number generator with some desirable properties here. Before I do so, perhaps a cautionary tale about a system-supplied RNG is in order:

> I wrote a program to play backgammon. The RNG supplied by C was used to roll the dice. I soon discovered that it *never* gave the same number on two consecutive rolls!

The following simple generator was proposed in the 1989 *Foundations of Computer Science*.

Choose $1 \leq a_i \leq 2^k$ randomly, for $1 \leq i \leq n$, with $n < k < 1.5n$. Choose $S \subset \{1, \ldots n\}$ randomly. Add the a_i corresponding to S mod 2^k to obtain a sequence of k bits. Use the first $k - n$ bits as output from the generator. The remaining n bits give you a new S, for which you obtain a new sum. At each step, you get $k - n$ random bits.

The authors of the paper (Impagliazzo and Naor) prove that any method of predicting the output of this generator could be used to provide an efficient solution method for the subset-sum problem.