COMP 200: Elements of Computer Science
Fall 2004
Lecture 31: November 14, 2004

Introduction to Cryptography

On the Board

Final Projects — get started
Today's lecture & Wednesday's lecture — Chapter 12

Encryption

As digital communication becomes widespread, we need ways to provide secure messages. Many applications require security — sending a credit card number to a web site, electronic voting for SA President, digital contract negotiation, sending military orders via the web (launch missile).

The problem is not new. Since the earliest days of remote commerce (outside one's village), merchants and generals have needed ways to send and receive secure messages. Early mystics used codes to write down their secrets — only a member of the inner group could decode them.

A Simple First Step — Substitution Codes

The easiest encryption algorithms to derive are substitution codes. They are (literally) child's play. Each letter is assigned a substitute. The simplest substitution codes are *linear substitution codes*, where a letter x is replaced with the letter $x + k$ modulo (*alphabet size*).

```
len ← length(message)
ciphertext ← empty string of length len

for i ← 1 to len
    ciphertext[i] ← alphabet[(message[i] + k ) modulo (alphabet size)]
```

To decrypt a cyphertext, we need to know k. We can run the inverse algorithm, subtracting k rather than adding it.

This simple crypto-scheme has a number of critical elements. It has a message, a ciphertext, and a key (k). To pass messages between two individuals, we need to arrange for them both to know the key.

Another way to envision writing this code uses a *translation table*. We can construct a vector that has *alphabet size* entries. In entry i, we record the

letter for *i+k* modulo *alphabet size*. Now, we can rewrite the code so that it avoids most of the arithmetic in that inner loop (lowering the cost per letter but not changing its asymptotic, or big-O complexity. The code with a translation table might be

```
len ← length(message)
ciphertext ← empty string of length len

for i ← 1 to len
    ciphertext[i] ← TranslationTable[message[i]]
```

This version of the algorithm uses more space (the table) to simplify the code and make the implementation run faster — a classic time-space tradeoff. To initialize the Translation Table, we can write a little loop:

```
for i ← 1 to alphabet size
   TranslationTable[i] ← (i + k ) modulo (alphabet size)
```

With this scheme, we still need to transmit the key to our partner, but the encryption and decryption run slightly faster.

Linear substitution codes have a major problem. They work well for children, but they only admit *alphabet size – 1* codes. With manual translation, cracking a linear substitution code would take some time (but not much). With a digital computer, it is no harder than reading the plaintext — well, from a Big-O perspective it is the same. For English, we have to try a constant 25 translations.

The translation-table scheme, however, points to an improvement over linear substitution codes. Any *one-to-one mapping* between the plaintext alphabet and the ciphertext alphabet will work. Any assignment of letters to positions works; the number of such assignments is (*alphabet size*)! In general, this may require two tables — one to encode and one to decode. They are (from an algebraic perspective) functional inverses and have the property that, given one table, constructing the other is simple.

An ancient substitution code that has appeared in recent literature (the DaVinci code, along with myriad books — scholarly and otherwise on the Dead Sea scrolls, the Knights Templar, and similar subjects) is the Atbash code (google it to see some of the better theories on the Templars and the reemergence of the divine feminine). The Atbash code is a substitution code that replaces the first letter with the last (and vice-versa), the second letter with the second-to-last letter (and vice-versa), and so on.

Cracking Substitution Codes

Unfortunately, substitution codes can be cracked in a (relatively) straightforward way. If we analyze enough text in the underlying language, we can determine the frequency of use for each character. Looking at the distribution of letters in the ciphertext, we can make educated guesses about the high-probability letters. With work, we can use probability to crack such codes — look for three letter words beginning with the high-probability letter "t". Look for single letter words. Digits (0 — 9) occur as either single letter words or in clusters with other digits (976, 100, and so on). Eventually, these codes fall.

(Ask any good "Wheel of Fortune" player for the English-language probabilities! In a game show, short words or phrases can be chosen to defy probabilistic attacks — zephyr is a good choice. In a coded message of any length, the normal frequency distribution begins to assert itself. You could supply your correspondent with a thesaurus of probability-defying alternatives, but that is just another form of encryption…)

In a substitution code, the *key* is the entire translation table. To communicate in secure fashion, two people must agree on a translation table. To send an encrypted message to a new party, you must find a secure way to transmit the table to them. In the digital world of the Internet, we need to address this shortcoming.

Single-use Ciphers

We can view the Atbash code as a positional substitution, with the table supplying a series of numbers that we add to the corresponding character.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
z	y	x	w	v	u	t	s	r	q	p	o	n	m	l	k	j	i	h	g	f	e	d	c	b	a
25	23	21	19	17	15	13	11	9	7	5	3	1	25	23	21	19	17	15	13	11	9	7	5	3	1

Here, the translation uses a fixed, letter-specific number to translate every occurrence of a given letter. 'g' always becomes 't', with offset 13. To defeat probabilistic attacks, we can make the translation numbers random and position-sensitive — that is, choose the translation offset as a function of the letter's position in the message rather than as a function of the letter itself. We write down a long list of numbers (from 1 to *alphabet size -1*), make sure that both parties have the same list of numbers, and start encoding. As long as both parties see the same set of messages, their number lists stay synchronized, and they have secure messaging.

This scheme has been widely used for decades. To make the codes truly secure, the lists should be drawn carefully to avoid any repeatable patterns — patterns that can be attacked with statistical analysis. With a good generator of random numbers, and some way to securely transmit the list of numbers (a *one-way pad*), this encryption system defies systematic attack. The easiest way to break a single-use cipher is steal the pad — espionage rather than mathematics.

Single-use ciphers work in the world of espionage. A CD-rom can hold a huge list of numbers; CD's are fairly innocuous. (In fact, you could record the numbers in space left over on a music CD and arrange to send the innocent looking Avril Lavigne CD to your correspondent.)

In the world of the Internet, they have serious drawbacks.

- The amount of data that must be transmitted between two parties to establish secure communication is large.
- Prior arrangement is required for any secure transaction — the sender and receiver must have a common one-way pad.
- The message must be self-identifying (in terms of sender's identity) for the receiver to know which one-way pad applies.
- Single-use ciphers make no provision for validating and verifying messages — imagine the contractual disputes that can arise from an encrypted order to Amazon.com. ("I got a message for 100,000 copies of that book. You owe me the money!")

Encryption in the Internet Age

The Internet age, with widespread secure commerce transacted casually among strangers, poses some unusual challenges. A suitable encryption technology must:

- Enable encrypted messages between total strangers (you and Amazon)
- Be secure against computational attacks (use every PC on campus to find the professor's Visa card number)
- Provide for digital signatures — a mechanism that lets the receiver know, with *complete* certainty, the sender's identify. (No spoofing)

These properties led to the development of an idea called *public-key encryption*.