CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 4
January 20, 2010

Automated cryptanalysis

Problem set 1 is to explore a simple automated strategy for breaking a Caesar cipher using letter frequency analysis.

Basically, the idea is to generate all 26 possible decryptions of a given ciphertext c and choose the message that is most likely according to the message distribution.

The method (explained in the assignment) requires generating random numbers according to an arbitrary distribution over the 26-letter alphabet.

Choosing uniformly from a finite set

The system-provided function `rand()` produces a pseudo-random integer in the set $\{0, \ldots, \texttt{RAND\_MAX}\}$.

To get a number in the set $\{0, \ldots, n-1\}$, one typically computes `rand()%n`.

Unfortunately, the result is not quite uniformly distributed unless n divides $\texttt{RAND\_MAX} + 1$.

Handout 3 describes how to fix this problem.

Generating a random double

To find an approximately uniformly-distributed double x in the semi-open interval $[0, 1)$, one wants to compute

`rand()/(RAND_MAX + 1)`.

Unfortunately, this can cause integer overflow, since `RAND_MAX+1` is typically too large to represent as an `int`.

Again, see Handout 3 for hints on how to solve this problem.

Choosing from an arbitrary finite distribution

Now, suppose P is a probability distribution over the finite set $\{0, \ldots, n-1\}$.

We'd like to generate a random $k \in \{0, \ldots, n-1\}$ distributed according to P.

- Partition the unit interval into n semi-open subintervals, where the i^{th} subinterval has length $P(i)$.
- Generate a random $x \in [0, 1)$.
- Find the subinterval $P(k)$ that contains x and return k.

How to efficiently find k from x requires designing an appropriate data structure to represent P.

Hill cipher

Encrypts groups of letters at once to mask letter frequencies.

Based on linear algebra.

- The key is, say, a non-singular 3×3 matrix K.
- The message m is divided into vectors m_i of 3 letters each.
- Encryption is just the matrix-vector product $c_i = Km_i$.
- Decryption uses the matrix inverse, $m_i = K^{-1}c_i$.

Unfortunately, the Hill cipher succumbs to a known plaintext attack. Given three linearly independent vectors m_1, m_2, and m_3 and the corresponding ciphertexts $c_i = Km_i$, $i = 1, 2, 3$, it is straightforward to solve for K.

Polyalphabetic ciphers

Another way to strengthen monoalphabetic ciphers is to use different substitutions for different letter positions.

- Choose, say, 10 different alphabet permutations $k_1, \ldots, k_{10}$.
- Use k_1 for the first letter of m, k_2 for the second letter, etc.
- Repeat this sequence after every 10 letters.

While this is much harder to break than monoalphabetic ciphers, letter frequency analysis can still be used.

Every 10th letter is encrypted using the same permutation, so the submessage consisting of just those letters still exhibits normal English language letter frequencies.

Transposition

Methods discussed so far are based on letter substitution.

Another technique is to rearrange the letters of the plaintext.

- Example: Write a plaintext message into a matrix by rows and read it out by columns.

When used in combination with substitution techniques, transposition can be quite effective.

Most practical symmetric cryptosystems are built by composing many stages of substitutions and transpositions.

Composition of Ciphers

Let (E', D') and (E'', D'') be ciphers.
Their *composition* is the cipher (E, D) with keys of the form $k = (k'', k')$, where

$$E_{(k'',k')}(m) = E''_{k''}(E'_{k'}(m))$$

$$D_{(k'',k')}(c) = D'_{k'}(D''_{k''}(m)).$$

Can express this using *functional composition*.
$h = f \circ g$ is the function such that $h(x) = f(g(x))$.

Using this notation, we can write $E_{(k'',k')} = E''_{k''} \circ E'_{k'}$ and $D_{(k'',k')} = D'_{k'} \circ D''_{k''}$.

Security of composition

Composition might or might not give a stronger cipher.
Can be difficult to analyze.

Practical symmetric cryptosystems such as DES and AES are built as a composition of simpler systems.

Each component offers little security by itself, but when composed, the layers obscure the message to the point that it is difficult for an adversary to recover.

The trick is to find ciphers that successfully hide useful information from a would-be attacker when used in concert.

Double Encryption

Double encryption is when a cryptosystem is composed with itself. Each message is encrypted twice using two different keys k' and k'', so $E^2_{(k'',k')} = E_{k''} \circ E_{k'}$ and $D^2_{(k'',k')} = D_{k'} \circ D_{k''}$.

(E, D) is the *underlying* or *base* cryptosystem and (E^2, D^2) is the *doubled* cryptosystem. $\mathcal{M}$ and $\mathcal{C}$ are unchanged, but $\mathcal{K}^2 = \mathcal{K} \times \mathcal{K}$.

The size of the keyspace is squared, resulting in an apparent doubling of the effective key length and making a brute force attack much more costly.

However, *it does not always increase the security* of a cryptosystem as much as one might naïvely think, for other attacks may become possible.

Example: Double Caesar

Consider Double Caesar, the Caesar cipher composed with itself.

There are now $26^2 = 676$ possible key pairs (k'', k'). One might hope that double Caesar is more resistant to a brute force attack.

Unfortunately, still only 26 possible distinct encryption functions and only 26 possible decryptions of each ciphertext.

This is because $E^2_{(k'',k')} = E_k$ for $k = (k' + k'') \bmod 26$.

Any attack on the Caesar cipher will work equally well on the Double Caesar cipher.

To the attacker, there is no difference between the two systems. Eve neither knows nor cares how Alice actually computed the ciphertext; all that matters to Eve is probabilistic relationships between plaintexts and ciphertexts.

Group property

Let (E, D) be a cryptosystem for which $\mathcal{M} = \mathcal{C}$.
Each E_k is then a *permutation* on $\mathcal{M}$.[1]
The set of all permutations on $\mathcal{M}$ forms a *group*.[2]

Definition

(E, D) is said to have the *group property* if the set of possible encryption functions $\mathcal{E} = \{E_k \mid k \in \mathcal{K}\}$ is closed under functional composition $\circ$.

That is, if $k', k'' \in \mathcal{K}$, then there exists $k \in \mathcal{K}$ such that

$$E_k = E_{k''} \circ E_{k'}.$$

[1] A *permutation* is one-to-one and onto function.

[2] A *group* has an associative binary operation with an identity element, and each element has an inverse.

Cryptosystems with group property

We've seen that the Caesar cipher has the group property.

When $\mathcal{E}$ is closed under composition, then $(\mathcal{E}, \circ)$ is a subgroup of all permutations on $\mathcal{M}$. In this case, double encryption adds no security against a brute force attack.

Even though the key length has doubled, the number of distinct encryption functions has not increased, and the double encryption system will fall to a brute force attack on the original cryptosystem.

Birthday Problem

The *birthday problem* is to find the probability that two people in a set of randomly chosen people have the same birthday.

This probability is greater than 50% in any set of at least 23 randomly chosen people.[3].

23 is far less than the 253 people that are needed for the probability to exceed 50% that at least one of them was born on a specific day, say January 1.

[3] See Wikipedia, "Birthday problem".

Birthday attack on a cryptosystem

A *birthday attack* is a known plaintext attack on a cryptosystem that reduces the number of keys that must be tried to roughly the square root of what a brute force attack needs.

For example, if the original key length was 56 (as is the case with DES), then only about $\sqrt{2^{56}} = 2^{28}$ keys need to be tried.

Any cryptosystem with the group property is subject to a birthday attack.

How a birthday attack works

Assume (m, c) is a known plaintext-ciphertext pair, so $E_{k_0}(m) = c$ for Alice's secret key k_0.

- Choose 2^{28} random keys k_1 and encrypt m using each.
- Choose another 2^{28} random keys k_2 and decrypt c using each.
- Look for a match (non-empty intersection) between these two sets.
- Suppose one is found for k_1 and k_2, so $E_{k_1}(m) = u = D_{k_2}(c)$. It follows that $E_{k_2}(E_{k_1}(m)) = c$, so we have succeeded in finding a key pair (k_2, k_1) that works for the pair (m, c).

There is a key k such that $E_k = E_{k_2} \circ E_{k_1}$ since we are assuming the group property, so $E_k(m) = c$.

How a birthday attack works (cont.)

Alice's key k_0 also has $E_{k_0}(m) = c$. If it happens that $E_k = E_{k_0}$, then we have broken the cryptosystem.

We do not need to find k itself since we can compute E_k from E_{k_1} and E_{k_2} and D_k from D_{k_1} and D_{k_2}.

Assuming reasonable randomness properties, there are unlikely to be many distinct keys k such that $E_k = E_{k_0}$, so with significant probability we have cracked the system. (For Caesar, there is only one such k.)

Using additional plaintext-ciphertext pairs, we can increase our confidence that we have found the correct key pair.
Repeat this process if we have not yet succeeded.

I've glossed over many assumptions and details, but that's the basic idea.

Weakenss of the birthday attack

The drawback to the birthday attack (from the attacker's perspective) is that it requires a lot of storage in order to find a matching element.

Nevertheless, if DES were a group, this attack could be carried out in about a gigabyte of storage, easily within the storage capacity of modern workstations.

(We will see later that DES is not a group.)

Data encryption standard (DES)

The Data Encryption Standard is a block cipher that operates on 64-bit blocks and uses a 56-bit key.

It became an official Federal Information Processing Standard (FIPS) in 1976. It was officially withdrawn as a standard in 2005 after it became widely acknowledged that the key length was too short and it was subject to brute force attack.

Nevertheless, triple DES (with a 112-bit key) is approved through the year 2030 for sensitive government information.

The new Advanced Encryption Standard (AES), based on the Rijndael algorithm, became an official standard in 2001. AES supports key sizes of 128, 192, and 256 bits and works on 128-bit blocks.