CS3235
Sixth set of lecture slides

Hugh Anderson

National University of Singapore
School of Computing

October, 2007
... and now if you could close the other 650 eyes and read the bottom line....
An English translation...

Before my bed, the moon is shining bright,
I think that it is frost upon the ground.
I raise my head and look at the bright moon,
I lower my head and think of home.
Outline

1 Administration
2 Error detection
   • CRC - Cyclic Redundancy Checks
   • MD5
3 Error Correction
   • BER - Bit Error Rate
   • Hamming codes
4 Encryption - symmetric key systems
   • Frequency analysis
   • Vigenère (polyalphabetic) cipher
   • Index of coincidence
   • DES - Data Encryption Standard
   • AES - Advanced Encryption Standard
Administration
  Error detection
  Error Correction
  Encryption - symmetric key systems

Project - what level is expected?

regurgitation < your work \leq research

- This could be write-your-own-software, compare-two-or-more-things, local-case-study, results-of-measurements, investigate-some-problem, and so on.

- 60-70% of what you present is YOUR work, or justification/confirmation of YOUR idea.

- Not at the level of original research, but more than just regurgitation. May include demo/CD.
Challenge number 2 - \( \min( \text{US}\$3253.00, \text{Coffee} ) \)

Here is a picture taken a few years ago:

![Picture](http://www.comp.nus.edu.sg/~cs3235/3253.tif)

The picture hides a problem.  
([http://www.comp.nus.edu.sg/~cs3235/3253.tif](http://www.comp.nus.edu.sg/~cs3235/3253.tif)) ... The problem goes away if you photocopy the picture...
Error detection

The history of human opinion is scarcely anything more than the history of human errors. [Voltaire]

Checking for errors...

- Transmit data:

  1 65 3 22 47 2

- Transmit data+checksum<sup>a</sup>:

  1 65 3 22 47 2 140

<sup>a</sup>checksum=fingerprint=message-digest
One-way parity

Use XOR to find the parity of each bit...

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Check: 0 1 1 1 0 1 1 0
# Two way parity

Both vertical and horizontal...

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>010000010</th>
<th>001100000</th>
<th>010001000</th>
<th>010001000</th>
<th>010001000</th>
<th>010001000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>001100000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
<td>010001000</td>
</tr>
<tr>
<td>Check:</td>
<td>0101101101101</td>
<td>0101101101101</td>
<td>0101101101101</td>
<td>0101101101101</td>
<td>0101101101101</td>
<td>0101101101101</td>
<td>0101101101101</td>
</tr>
</tbody>
</table>

Hugh Anderson  
CS3235 Sixth set of lecture slides
Simple check codes

The parity system is OK, but...

- Parity of bits - detects all 1 bit errors, but...
  - Horizontal and vertical parity - better, but ...
    - problems with repetitive errors

- Sum of values - problems with repetitive errors
- Want better level of error checking
Another scheme

- Treat the stream of transmitted bits as a representation of a polynomial with coefficients of 1:
  \[10110 = x^4 + x^2 + x^1 = F(x)\]

- Checksum bits are added to ensure that the final stream of bits is divisible by some other polynomial \(g(x)\).

- We can transform any stream \(F(x)\) into a stream \(T(x)\) which is divisible by \(g(x)\).

- If there are errors in \(T(x)\), they take the form of a difference bit string \(E(x)\) and the final received bits are \(T(x) + E(x)\).

- When the receiver gets a correct stream, it divides it by \(g(x)\) and gets no remainder.
The question is: How likely is that will also divide with no remainder?

- **Single bits?** - No a single bit error means that $E(x)$ will have only one term ($x^{1285}$ say). If the generator polynomial has $x^n + \ldots + 1$ it will never divide evenly.

- **Multiple bits?** - Various generator polynomials are used with different properties. Must have one factor of the polynomial being $x^1 + 1$, because this ensures all odd numbers of bit errors (1,3,5,7...).
Some common generators:

Used in systems all around us...

- **CRC-12** - $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$
- **CRC-16** - $x^{16} + x^{15} + x^2 + 1$
- **CRC-32** - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + 1$
- **CRC-CCITT** - $x^{16} + x^{12} + x^5 + 1$
Long division is easy!

We have met this sort of division before...

Generator \( g(x) : x^5 + x^2 + 1 \) (100101) and \( F(x) : 101101011 \). divide \( F(x) \) by \( g(x) \), remainder is appended to \( F(x) \) to get \( T(x) \):

\[
\begin{array}{c}
\text{1010.01000} \\
\hline
\text{100101 } \text{101101011.00000} \\
\text{100101} \\
\text{100001} \\
\text{100101} \\
\text{1001.00} \\
\text{1001.01} \\
\text{1000}
\end{array}
\]

\( T(x) = 10110101101000 \).
Long division is easy!

The division can be done with very simple hardware
When this stream is received, it is divided but now will have no remainder if the stream is received without errors.
Long division is easy!

### Step by step...

#### Input

<table>
<thead>
<tr>
<th>Input</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
</tbody>
</table>

#### (At end, feed in zeroes...)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case study: ethernet

Case study - use of CRC in ethernet

Ethernet is used for networking computers, principally because of its speed and low cost. The maximum size of an ethernet frame is 1514 bytes\(^a\), and a 32-bit FCS is calculated over the full length of the frame. The FCS used is:

\[
\text{CRC-32} - \quad x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + 1
\]

\(^a\)1500 bytes of data, a source and destination address each of six bytes, and a two byte type identifier. The frame also has a synchronizing header and trailer which is not checked by a CRC.
Md5SUM

Implementation of MD5 check algorithm is called md5sum

```
hugh@sf0:~[508]$ md5sum ss.c
550114bc3cc3359e55ba33abe8983a85  ss.c
hugh@sf0:~[509]$ cp ss.c XXX.c
hugh@sf0:~[510]$ md5sum XXX.c
550114bc3cc3359e55ba33abe8983a85  XXX.c
hugh@sf0:~[511]$
```

- Message digest, checksum, hash, digital fingerprint.
- A one-way function.
MD5 weaknesses

But how secure is it?

- Suspicion that MD5 may have cryptographic weaknesses.

Recent revelation (but few details beyond examples) at Crypto2004 of a generated MD5 collision:


Note that this does not reduce the effectiveness of MD5 (yet)
Simple error correction

Methods used to correct errors:

- Ignore errors, while acknowledging correct data. **ARQ** (for Automatic Repeat reQuest).
- Error correcting codes (for computer memory)
Code types

Classification of codes

- We can divide error correcting codes (ECC) into **continuous** and **block-based** types.
  - Convolutional encodings are used for continuous systems, and

- the common block-based codes are:
  - **Hamming** codes (for correcting single bit errors),
  - **Golay** codes (for correcting up to three bit errors), and
  - **Bose-Chaudhuri-Hocquenghem (BCH)** codes (for correcting block errors).
Combining error correcting codes

Different types of error correcting codes can be combined to produce composite codes.

For example, *Reed-Solomon* block-codes are often combined with convolutional codes to improve all-round performance. In this combined setup, the convolutional code corrects randomly distributed bit errors but not bursts of errors while the *Reed-Solomon* code corrects the burst errors.
BER and noise

How often do errors occur?

<table>
<thead>
<tr>
<th>System</th>
<th>(BER) Bit Error Rate (errors/bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiring of internal circuits</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>Memory chips</td>
<td>$10^{-14}$</td>
</tr>
<tr>
<td>Hard disk</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>Optical drives</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>Coaxial cable</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>Optical disk (CD)</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>Telephone System</td>
<td>$10^{-4}$</td>
</tr>
</tbody>
</table>
BER and noise

The channel capacity

We can determine the theoretical channel capacity $C$ knowing the SNR:

- BER is $0.01$, channel capacity $C \approx 0.92$ bits/symbol.
- BER is $0.001$, channel capacity $C \approx 0.99$ bits/symbol.
- BER is $0$, channel capacity $C = 1$ bits/symbol (perfect channel)

The theoretical maximum channel capacity is quite close to the perfect channel capacity, even if the BER is high.
Reducing BER

To reduce the bit error rate, we can

1. Increase the signal (power), or
2. Reduce the noise (often not possible), or
3. Use ECC.

The benefit of Error Correcting Codes is that they can improve the BER without increasing the transmitted power. This performance improvement is measured as a system gain.
Reducing BER

Example

Consider a system without ECC giving a BER of 0.001 with a S/N ratio of 30dB (1000:1). If we were to use an ECC codec, we might get the same BER of 0.001 with a S/N ratio of 20dB (100:1).

We say that the system gain due to ECC is 10dB (10:1).
Bad ECC scheme: repetition

A bad scheme is to just repeat the bits...

- Correct transmission errors by repeating bits\(^a\).

<table>
<thead>
<tr>
<th>Data:</th>
<th>0 1 0 0 1 1 1 1 1 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit:</td>
<td>0001110000000111111111111 ...</td>
</tr>
</tbody>
</table>

\(^a\)Note: there is no point in repeating bits *twice*. you must repeat three times, or 5 times, and then vote to decide the best value.

If we send three identical bits for every bit we wish to transmit, we can then use a voting system to determine the most likely bit. If our natural BER due to noise was 0.01, with three bits we would achieve a synthetic BER of 0.0001, but our channel capacity is reduced to about \(C = 0.31\) bits/symbol.
Bad ECC scheme: repetition

The error rate goes down slowly as the bits transmitted goes up. We can do better...

- We can see from this that the rate of transmission using repetition has to approach zero to achieve more and more reliable transmission.

However we know that the theoretical rate should be equal to or just below the channel capacity $C$. Convolutional and other encodings can achieve rates of transmission close to the theoretical maximum.
ECC scheme: Hamming

A better ECC scheme

- *Hamming* codes are block-based error correcting codes.

We add hamming bits to a string

Here we derive the inequality used to determine how many extra *hamming* bits are needed for an arbitrary bit string.
ECC scheme: Hamming

The hamming distance between two strings
The *hamming* distance is a measure of how FAR apart two bit strings are.

<table>
<thead>
<tr>
<th>A:</th>
<th>0 1 0 1 1 1 0 0 0 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:</td>
<td>0 1 1 1 1 1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>A XOR B:</td>
<td>0 0 1 0 0 0 1 0 0 0 1 0</td>
</tr>
</tbody>
</table>

(In this case: 3)
ECC scheme: Hamming

Detection and correction use different number of extra bits

If we had two bit strings $X$ and $Y$ representing two characters, and the *hamming* distance between any two codes was $d$, we could turn $X$ into $Y$ with $d$ single bit errors.

- If we had an encoding scheme (for say ASCII characters) and the minimum *hamming* distance between any two codes was $d + 1$, we could detect $d$ single bit errors\(^a\).

- We can correct up to $d$ single bit errors in an encoding scheme if the minimum *hamming* distance is $2d + 1$.

\(^a\)Because the code $d$ bits away from a correct code is not in the encoding.
ECC scheme: Hamming

Example encoding.
The hamming distance between any two codes is at least 2, and we use 3 bits to encode 4 symbols. (c.f. 8 symbols normally)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The minimum hamming distance between any two codes is 2. We can detect single bit errors.
ECC scheme: Hamming

Analysis of hamming codes

If we now encode \( m \) bits using \( r \) extra hamming bits to make a total of \( n = m + r \), we can count how many correct and incorrect hamming encodings we should have. With \( m \) bits we have \( 2^m \) unique messages - each with \( n \) illegal encodings, and:

\[
(n + 1)2^m \leq 2^n \\
(m + r + 1)2^m \leq 2^n \\
m + r + 1 \leq 2^{n-m} \\
m + r + 1 \leq 2^r
\]

We solve this inequality, and then choose \( R \), the next integer larger than \( r \).
ECC scheme: Hamming

Example:
If we wanted to encode 8 bit values \((m = 8)\) and be able to correct single bit errors:

\[
8 + r + 1 \leq 2^r \\
9 \leq 2^r - r \\
r \approx 3.5 \\
R = 4
\]
Encryption introduction

We call these systems symmetric key systems...

*I could have told her the truth - that the same calculation which had served me for deciphering the manuscript had enabled me to learn the word - but on a caprice it struck me to tell her that a genie had revealed it to me. This false disclosure fettered Madame d’Urfé to me. That day I became the master of her soul, and I abused my power.*

[Casanova]
Symmetric key systems

Alice uses a key to send to Bob, who uses the same key...

Alice

P
(Plaintext)

*

K_i

(Encrypted)

K_i[P]

Bob

P
(Plaintext)

*

K_i

(Harry–the–hacker)
Simple ciphers - transposition

Transposition ciphers

Transposition ciphers just **re-order the letters** of the original message. This is known as an **anagram**:

- *parliament* is an anagram of *partial men*
- *Eleven plus two* is an anagram of *Twelve plus one*

Perhaps you would like to see if you can unscramble “*age prison*”, or “*try open*”. 
Transposition

Detecting a transposition cipher

Detect a transposition cipher with the frequencies of the letters, and letter pairs.

If the frequency of single letters in ciphertext is correct, but the frequencies of letter pairs is wrong, then the cipher may be a transposition. This sort of analysis can also assist in unscrambling a transposition ciphertext, by arranging the letters in their letter pairs.
Simple ciphers - substitution

Rotation and Cæsar ciphers
Substitution cipher systems encode the input stream using a substitution rule.
The Cæsar cipher is an example of a simple substitution cipher system. Now we call it a rotation cipher, which can be \textit{cracked} in at most 25 attempts by just trying each of the 25 values in the keyspace. The original Cæsar cipher can be \textit{cracked} in 1 attempt.
Substitution

Random substitution - a monoalphabetic substitution cipher

<table>
<thead>
<tr>
<th>Code</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Q</td>
</tr>
<tr>
<td>B</td>
<td>V</td>
</tr>
<tr>
<td>C</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>W</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

If the mapping was more randomly chosen it is called a monoalphabetic substitution cipher, and the keyspace for encoding 26 letters would be

\[ 26! - 1 = 403, 291, 461, 126, 605, 635, 583, 999, 999. \]
Substitution

How safe is this cipher? (Not at all!)

- If we could decrypt 1,000,000 messages in a second, then the average time to find a solution by trying decryptions would be about 6,394,144,170,576 years!

We might be lulled into a sense of security by these big numbers, but of course this sort of cipher can be subject to frequency analysis.
Frequency analysis

Measure the frequency of the encrypted data to make good guesses at the original plaintext.

- In the English language, the most common letters are: "E T A O N I S H R D L U..." (from most to least common).

We may also look for **digrams** and **trigrams** (**th**, **the**).

1. Count occurrences of each character in as much ciphertext as you can
2. Arrange letters in frequency order
3. Replace letters by the standard ETAOINSHRDLU ones:
Frequency analysis

Example encrypted message

```
EV YQS CVV MIWK FRPC FRQF FRV IQFV WM
FIQSCKPCCPWS ACPSN IVJVFPFPWS RQC FW
QJJIWQYR ZVIW FW QYRPVDV KWIV QSB KWIV
IVTPQXTV FIQSCKPCCPWS. RWEVDVI EV LSWE
FRQF FRV FRVWIVFPYQT IQFV CRWATB ...
```

V occurs most often, F next and so on, so replace V with E, F with T...
Frequency Analysis

Example - first steps

```
EV YQS CVV MIWK FRPC FRQF FRV IQFV WM
-E -A- -EE F-O- THI- THAT THE -ATE OF

FIQSCKPCCPWS ACPSN IVJVFPFPWS RQC FW
T-A---I--IO- --I-- -E-ETITIO- HA- TO

QJJJIWQ YR ZVIW FW QYRPVDV KWIV QSB KWIV
A---OA -H -E-O TO A-HIE-E -O-E A-- -O-E
```
Vigenère

A polyalphabetic substitution cipher...

- The Vigenère cipher is a polyalphabetic substitution cipher invented around 1520.
- We use an encoding/decoding sheet, called a *tableau*, and a keyword or key sequence.
# Vigenère

## The tableau

|   | A | B | C | D | E | F | G | H | ...
|---|---|---|---|---|---|---|---|---|---|
| A | A | B | C | D | E | F | G | H | ...
| B | B | C | D | E | F | G | H | I | ...
| C | C | D | E | F | G | H | I | J | ...
| D | D | E | F | G | H | I | J | K | ...
| E | E | F | G | H | I | J | K | L | ...
| F | F | G | H | I | J | K | L | M | ...
| G | G | H | I | J | K | L | M | N | ...
| H | H | I | J | K | L | M | N | O | ...
|   |   |   |   |   |   |   |   |   |   |...
Vigenère

Sample encoding

If our keyword was **BAD**, then encoding **HAD A FEED** would result in

<table>
<thead>
<tr>
<th>Key</th>
<th>B</th>
<th>A</th>
<th>D</th>
<th>B</th>
<th>A</th>
<th>D</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>H</td>
<td>A</td>
<td>D</td>
<td>A</td>
<td>F</td>
<td>E</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>Cipher</td>
<td>I</td>
<td>A</td>
<td>G</td>
<td>B</td>
<td>F</td>
<td>H</td>
<td>F</td>
<td>D</td>
</tr>
</tbody>
</table>

If we can discover the length of the repeated key (in this case 3), and the text is long enough, we can just consider the cipher text to be a group of interleaved monoalphabetic substitution ciphers and solve accordingly.
Analysis

Index of coincidence

The index of coincidence is the probability that two randomly chosen letters from the cipher will be the same, and it can help us discover the length of a key.

\[
IC = \frac{1}{N(N-1)} \sum_{i=0}^{25} F_i(F_i - 1)
\]

where \( F_i \) is the frequency of the occurrences of symbol \( i \) and \( N \) is the length of the cipher.
Index of coincidence

Program to analyze data

<table>
<thead>
<tr>
<th>Period</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected IC</td>
<td>0.066</td>
<td>0.052</td>
<td>0.047</td>
<td>0.045</td>
<td>...</td>
</tr>
</tbody>
</table>

```perl
#!/usr/bin/perl
$skip=$ARGV[0] ;
@text=<stdin> ;
$all=join(”,@text) ;
$all =~ tr/a-z/A-Z/ ;
$all =~ tr/A-Z//cd ;
$header=substr($all,0,$skip) ;
$shifted = substr($all,$skip).$header ;
@alltxt=split(//,$all) ; @shiftxt=split(//,$shifted) ;
foreach $i(0..$#alltxt){
    if($alltxt[$i] eq $shiftxt[$i]) { $count++ ;}
}
printf("Index of Coincidence is: %2f\n",$count/$#alltxt) ;
```
History

- The ideas here were developed by William F. Friedman in his Ph.D.

Friedman also coined the words “cryptanalysis” and “cryptology”.

Friedman worked on the solution of German code systems during the first (1914-1918) world war, and later became a world-renowned cryptologist.
DES - Data Encryption Standard

DES

- DES was first proposed by IBM using 128 bit keys, but its security was reduced by NSA (the National Security Agency) to a 56 bit key.

At 1ms/GUESS. It would take $10^{80}$ years to solve 128 bit key encryption.

The DES Standard gave a business level of safety, and is a product cipher.
DES - Data Encryption Standard

DES

- The (shared) 56 bit key is used to generate 16 subkeys, which each control a sequenced P-box or S-box stage.

DES works on 64 bit messages called blocks.
If you intercept the key, you can decode the message.
However, there are about $10^{17}$ keys.
At the core of DES is the Feistel network

Each of the 16 stages (rounds) of DES uses a Feistel structure which encrypts a 64 bit value into another 64 bit value using a 48 bit key derived from the original 56 bit key.
DES modes of operation

ECB, CFB and CBC

- The US government specifically recommends not using the weakest simplest mode for messages, the Electronic Codebook (ECB) mode.

They recommend the stronger and more complex Cipher Feedback (CFB) or Cipher Block Chaining (CBC) modes. The CBC mode XORs the next 64-bit block with the result of the previous 64-bit encryption, and is more difficult to attack.
DES modes of operation

ECB and CBC

Electronic Code Book

- **msg**
- **DES**
- **Ctext**

Cipher Block Chaining

- **msg**
- **Initial vector**
- **DES**
- **Ctext**
- **DES**
- **Ctext**

Hugh Anderson  CS3235 Sixth set of lecture slides
DES software

How to program with DES

DES is available as a library on both UNIX and Microsoft-based systems. The Java library included DES. For C, there is typically a des.h file, which must be included in any C source using the DES library:

```c
#include "des.h"
// - Your calls
```

After initialization of the DES engine, the library provides a system call which can both encrypt and decrypt:

```c
int des_cbc_encrypt(clear, cipher, schedule, encrypt)
```

where encrypt determines if we are to encipher or decipher. The schedule contains the secret DES key.
Case study: Amoeba capabilities

Amoeba objects are identified by a capability string

A *capability* is long enough so that you can’t just make them up. If you have it, you have whatever the capability allows you.

To further prevent tampering, the capability is DES encrypted. The resultant bit stream may be used directly, or converted to and from an ASCII string with the *a2c* and *c2a* commands.
AES

The US Advanced Encryption Standard

- Symmetric block-based FAST data encryption standard.
- Adopted by US Govt in 2000
- Algorithm specified in code form
- Uses substitution, shifts, mixing
AES algorithm (Tutorial)

AES code

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4, Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
out = state
end
```