Chapter 5

Lecture 5 - Preliminaries

Assignment 1

✔ Form your groups,
✔ Select your project,
✔ Email me with your proposal for approval

Assignment 2

✔ More detailed, in-depth study
✔ Not necessarily a software development project, but may be...
  ✔ Comparison,
  ✔ research,
  ✔ idea,
  ✔ program

✔ Literature review - not a survey :

✔ Formal paper - format in either word.zip, latex2e.zip in http://www.comp.nus.edu.sg/~cs3235/2003-semesterI/

✔ Heading, Author(s), abstract (?), Introduction, body of paper, Summary/Conclusion, References

✔ 5-50 pages, 5-50 references
The challenge...

Factorize:

3: 493
9: 256548911
19: 2399557933351623797
30: 2103049505718271668932997799
40: 52192541030037024328412332570126942904289
50: 34140838649750908282137081148636105533382866525339
59: 26120139124694484823073438898153152744842206694017897257399
68: 341138210618403034089080607020126290000049367328046381220764154001
77: 1974025104076746055742338911550894097073749540855591099939961368040398403222619
87: 436557784371554569114262417700283001153114796149237290780255686606068785
114: 3289007170346419389826843211747469690855951410750551704185508113661646418795579
129: 376913201530163994815348263715558901305038078694463768491362396267582834411351930

The challenge...

Done! (blue)

3: 493
9: 256548911
19: 2399557933351623797
40: 52192541030037024328412332570126942904289
50: 34140838649750908282137081148636105533382866525339
59: 26120139124694484823073438898153152744842206694017897257399
68: 341138210618403034089080607020126290000049367328046381220764154001
77: 1974025104076746055742338911550894097073749540855591099939961368040398403222619
90: 42605577884371554569114262417700283001153114796149237290780255686606068785
114: 3289007170346419389826843211747469690855951410750551704185508113661646418795579
129: 376913201530163994815348263715558901305038078694463768491362396267582834411351930

And the winners are:

✔ Grand overall winner in the my-computer-is-faster-than-your-computer challenge, for factorizing the largest number (#9): **Toh Kaiyang**

✔ For being first with a factorization (17*29=493): **Francis** (again)

✔ Honourable mention:
  ✓ **Zhang Yanchun** (wrote a program)
  ✓ **Nicolas Bally, Chia Boon Peen, Xue Mingqiang, Lee Pei Ru, Rémi Fontan, Weng Wei**

So... what's the deal?

✔ It is easy to multiply numbers... but hard to factorize (example in bc)

✔ Factorizing software:
  ✓ [ecm-gmp](http://webloria.loria.fr/~zimmerma/records/ecmnet.html)
  ✓ [pari](http://pari.math.u-bordeaux.fr/download.html)
The graph...

RSA challenge...

There's gold in them-thar hills:
http://www.rsasecurity.com/rsalabs/node.asp?id=2094

This session

- Security models
  - Access control
  - Covert channels
  - Formal models - BLP, Biba

- Error detection
  - checksums
  - CRC

- Encryption intro

Preliminaries - security models

The sciences do not try to explain, they hardly even try to interpret, they mainly make models.  [J. von Neumann]
Preliminaries - security models

Definition: a range of formal policies for specifying the security of a system in terms of a (mathematical) model.

✔ access control matrix
✔ Bell-LaPadula
✔ Biba
✔ Clark-Wilson

Security model approach

✔ Have a model
✔ Determine properties
✔ Verify implementations

Access control matrix

Rows of the matrix are subjects, columns are objects:

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Object 1</th>
<th>Object 2</th>
<th>Object 3</th>
<th>Object 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>read</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>write</td>
<td></td>
<td>read</td>
<td>execute</td>
</tr>
<tr>
<td>s3</td>
<td>read</td>
<td>write</td>
<td></td>
<td>execute</td>
</tr>
<tr>
<td>s4</td>
<td>read</td>
<td>write</td>
<td>write</td>
<td>read</td>
</tr>
</tbody>
</table>

s4 cannot read f1. But subjects may collude...

This session

- Security models
  - Access control
  - Covert channels
  - Formal models - BLP, Biba
- Error detection
  - checksums
  - CRC
- Encryption intro
The confinement problem is one of preventing a system from leaking (possibly partial) information. Sometimes a system can have an unexpected path of transmission of data, termed a covert channel, and through the use of this covert channel information may be leaked either by a malicious program, or by accident.

Consider the set of permissions on a file. An unscrupulous program could modify these permissions cyclically to transmit a very-low data-rate message to another unscrupulous program.

We categorize covert channels into two:

1. **Storage channels**: using the presence or absence of objects

2. **Timing channels**: the speed of events

We can attempt to identify covert channels by building a shared resource matrix, determining which processes can read and write which resources.

- Governing body may keep secret individual information, but release cumulative information

- For example: Today’s average temperature of SOC staff by nationality:
Attacks on databases

✓ OK - doesn’t release any sensitive information, but
✓ what if another part of the database released the numbers of SOC staff by nationality...

<table>
<thead>
<tr>
<th></th>
<th>Singaporean</th>
<th>Malaysian</th>
<th>PRC</th>
<th>Poland</th>
<th>German</th>
<th>Australian</th>
<th>New Zealand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>23</td>
<td>12</td>
<td>14</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

✓ By inference you can deduce that the temperature of a particular individual is too high!

CS3235 - Hugh Anderson's notes. Page number: 260

This session

- Security models
  - Access control
  - Covert channels
  - Formal models - BLP, Biba
- Error detection
  - checksums
  - CRC
- Encryption intro

CS3235 - Hugh Anderson's notes. Page number: 261

Bell-LaPadula, confidentiality

✓ Military style to assure confidentiality services.
✓ Security levels in a (total) ordering formalizing a policy which restricts information flow from a higher security level to a lower security level.
✓ Lower-level subjects from accessing higher-level objects.

Bell-LaPadula, levels

1. Top secret (T)
2. Secret (S)
3. Confidential (C)
4. Unclassified (U)

where $T > S > C > U$. Access operations visualized using an access control matrix, and are drawn from \{read, write\}.

CS3235 - Hugh Anderson's notes. Page number: 262
BLP security property

The clearance classification for a subject \( s \in S \) or object \( o \in O \) is denoted \( L(s) = l_s \) or \( L(o) = l_o \). We might then assume we can use this to construct a first simple security property:

- **No read-up-1**: \( s \) can read \( o \) if and only if \( l_o \leq l_s \), and \( s \) has read access in the access control matrix.

This single property is insufficient to ensure the restriction we need for the security policy.

BLP Trojan Horse property

Consider the case when a low security subject creates a high security object (say a program) which then reads a high security file, copying it to a low security one. This behaviour is commonly called a Trojan Horse. A second property is needed:

- **No write-down-1**: \( s \) can write \( o \) if and only if \( l_s \leq l_o \), and \( s \) has write access in the access control matrix.

These two properties can be used to enforce our security policy, but with a severe restriction. For example, how does any subject write down without invalidating a security policy?

BLP extended

A security category \( c \in C \) is used to classify objects in the model, with any object belonging to a set of categories. Each pair \( (l \times c) \) is termed a security level, and forms a lattice.

We define a relation between security levels:

- The security level \( (l, c) \) dominates \( (l', c') \) (written \( (l, c) \text{ dom} (l', c') \)) iff \( l' \leq l \), and \( c' \subseteq c \).

A subject \( s \) and object \( o \) then belong to one of these security levels.
BLP extended

The new properties are:

- **No read-up-2:** \( s \) can read \( o \) if and only if \( s \) dom \( o \), and \( s \) has read access in the access control matrix.

- **No write-down-2:** \( s \) can write \( o \) if and only if \( o \) dom \( s \), and \( s \) has write access in the access control matrix.

BLP security

- A system is considered secure in the current state if all the current accesses are permitted by the two properties.

- A transition from one state to the next is considered secure if it goes from one secure state to another secure state.

- The basic security theorem states that if the initial state of a system is secure, and if all state transitions are secure, then the system will always be secure.

BLP limits

BLP is a static model, not providing techniques for changing access rights or security levels\(^9\).

However the model does demonstrate initial ideas into how to model, and how to build security systems that are provably secure.

Biba model, integrity

- **Trustworthiness** of data and programs - assurance for integrity services.

- Levels like clean or dirty (in reference to database entries).

- Biba model is a kind of dual for Bell-LaPadula. integrity vs confidentiality.

---

\(^9\)You might want to explore the Harrison-Ruzo-Ullman model for this capability.
Biba levels

✔ The integrity levels $I$ are ordered as for the security levels.

✔ Function $i : O \rightarrow I$ (i : $S \rightarrow I$) which returns the integrity level of an object (subject).

Biba properties

The properties/rules for the main (static) Biba model are:

- **No read-down:** $s$ can read $o$ iff $i(s) \leq i(o)$.
- **No write-up:** $s$ can write $o$ iff $i(o) \leq i(s)$.
- **No invoke-up:** $s_1$ can execute $s_2$ iff $i(s_2) \leq i(s_1)$.

Biba - dynamic

Biba models can also handle dynamic integrity levels, where the level of a subject reduces if it accesses an object at a lower level (in other words it has got dirty). The low-watermark policies are:

- **No write-up:** $s$ can write $o$ iff $i(o) \leq i(s)$.
- **Subject lowers:** if $s$ reads $o$ then $i'(s) = \min(i(s), i(o))$.
- **No invoke-up:** $s_1$ can execute $s_2$ iff $i(s_2) \leq i(s_1)$.

Biba - ring

Finally, we have a ring policy,

- **All read:** $s$ can read $o$ regardless.
- **No write-up:** $s$ can write $o$ if and only if $i(o) \leq i(s)$.
- **No invoke-up:** $s_1$ can execute $s_2$ if and only if $i(s_2) \leq i(s_1)$.

Each of these policies have an application in some area.
Transactions defined through certification rules. The Clark-Wilson model has the following terminology:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDI</td>
<td>Constrained Data Item (data subject to control)</td>
</tr>
<tr>
<td>UDI</td>
<td>Unconstrained Data Item (data not subject to control)</td>
</tr>
<tr>
<td>IVP</td>
<td>Integrity Verification Procedures (for testing correct CDIs)</td>
</tr>
<tr>
<td>TP</td>
<td>Transformation Procedures (for transforming the system)</td>
</tr>
</tbody>
</table>

This session

- Security models
  - Access control
  - Covert channels
  - Formal models - BLP, Biba
- Error detection
  - checksums
  - CRC
- Encryption intro

Error detection

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

Simple check codes

- Transmit data:
  - 1 65 3 22 47 2
- Transmit data+checksum\(^{10}\):
  - 1 65 3 22 47 2 140

[Example check program]

\(^{10}\) checksum=fingerprint=message-digest
### One-way parity

<table>
<thead>
<tr>
<th></th>
<th>A</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></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>
<td></td>
</tr>
<tr>
<td></td>
<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></td>
</tr>
<tr>
<td></td>
<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></td>
<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></td>
<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>
<tr>
<td><strong>Check:</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Two-way parity

<table>
<thead>
<tr>
<th></th>
<th>A</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>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<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></td>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td><strong>Check:</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### Simple check codes

- ✔ 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

### This session

- **Security models**
  - Access control
  - Covert channels
  - Formal models - BLP, Biba
- **Error detection**
  - checksums
  - CRC
- **Encryption intro**
Cyclic redundancy check codes

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 composite 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 \( T(x) + E(x) \) 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:

- **CRC-12** - \( x^{12} + x^{11} + x^3 + x^2 + x^1 + 1 \)
- **CRC-16** - \( x^{16} + x^{15} + x^2 + 1 \)
- **CRC-32** - \( x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^7 + x^0 \)
- **CRC-CCITT** - \( x^{16} + x^{12} + x^5 + 1 \)

CS3235 - Hugh Anderson's notes. Page number: 284

CS3235 - Hugh Anderson's notes. Page number: 286

CS3235 - Hugh Anderson's notes. Page number: 287
Long division is easy!

Generator \( g(x) : x^5 + x^2 + 1 \) (100101) and \( F(x) : 101101011 \).

divide \( F(x) \) by \( g(x) \), and the remainder is appended to \( F(x) \) to give \( T(x) \):

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

\( T(x) = 10110101101000 \).

When this stream is received, it is divided but now will have no remainder if the stream is received without errors.

<table>
<thead>
<tr>
<th>Input data</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Initial state</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>First bit</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Second bit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Third bit</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

CS3235 - Hugh Anderson's notes.
Case study: 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\(^{11}\), and a 32-bit FCS is calculated over the full length of the frame.

The FCS used is:

- **CRC-32** \(x^{32}+x^{26}+x^{24}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^6+x^4+x^2+1\)

\(^{11}\)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.

This session

- Security models
  - Access control
  - Covert channels
  - Formal models - BLP, Biba
- Error detection
  - checksums
  - CRC
- Encryption intro

Encryption introduction

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]

We call these systems symmetric key systems...
Symmetric key systems

Simple ciphers - transposition

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”.

Simple ciphers - substitution

Substitution cipher systems encode the input stream using a substitution rule.

- The Caesar cipher is an example of a simple substitution cipher system, but it can be cracked in at most 25 attempts by just trying each of the 25 values in the keyspace.

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.
Substitution

<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

- If we could decrypt 1,000,000 messages in a second, then the average time to find a solution 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

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), and we may use the frequency of the encrypted data to make good guesses at the original plaintext.

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

Vigenère

- 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

| 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 |
| A | 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 |
| B | 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 |
| C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| D | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| E | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

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

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

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

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

Index of coincidence

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

Show analysis using shifts of 1...2...3...
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.