Cryptography (hidden writing)

- Encryption
- Decryption

Cryptanalysis (finding meanings of hidden writings)

Introduction

Substitutions — substitute each char by another
- Cryptography
  1. Caesar Cipher - shift of 3 (A---D),(hello----khoor)
  2. Permutations – shuffling (hello (3,2,5,4,1)----(leolh))
  3. Key – if we use “key” (hello --- keyhello)
  4. Others...
- Cryptanalysis - using of language characteristics
  Frequency distribution
  highly likely letters and probable words
  repeated patterns
  others...

Transposition – letters in the message rearranged

Column transposition:

\[
\begin{array}{cccc}
  c_1 & c_2 & c_3 & c_4 \\
  c_5 & c_6 & c_7 & c_8 \\
  c_9 & c_{10} & \ldots \\
  c_1 & c_5 & c_9 & c_2 & c_6 & c_{10} & c_3 & c_7 & \ldots \\
\end{array}
\]

A lot others….

Cryptanalysis – more study of language patterns
- Diagrams (EN,RE,ER,NT,TH,…)
- Trigrams (AND,ENT,ION,FOR,OUR,…)

Confusion:
Good confusion is that with minimum relation between Plain and Cipher text

Diffusion:
Spread the information from the plaintext over the entire cipher text
Symmetric Key Systems:
- The same key for encryption and decryption
- Input divided into blocks (64 bits, 128, 192, 256…)
- The input goes through initial permutation
- Key reduction (from 64 bit to 56 bit – error correction bits)
- Break input into Left half & Right half
- The key shifted left by a number of bits and permuted
- The key is combined with the Right half

Rijndael (AES):
The input is divided into a matrix of (4XNb) bytes
Nb = block length divided by 32
The cipher key is divided into a matrix of (4XNr) bytes
Nr = Key length divided by 32

\[
\begin{array}{cccc}
\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \sigma_{0,3} \\
\sigma_{1,0} & \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\
\sigma_{2,0} & \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\
\sigma_{3,0} & \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \\
\end{array}
\]

\[
\begin{array}{cccc}
\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \sigma_{0,3} \\
\sigma_{1,0} & \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\
\sigma_{2,0} & \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\
\sigma_{3,0} & \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \\
\end{array}
\]

\[
\begin{array}{cccc}
\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \sigma_{0,3} \\
\sigma_{1,0} & \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\
\sigma_{2,0} & \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\
\sigma_{3,0} & \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \\
\end{array}
\]

\[
\begin{array}{cccc}
\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \sigma_{0,3} \\
\sigma_{1,0} & \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\
\sigma_{2,0} & \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\
\sigma_{3,0} & \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \\
\end{array}
\]

\[
\begin{array}{cccc}
\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \sigma_{0,3} \\
\sigma_{1,0} & \sigma_{1,1} & \sigma_{1,2} & \sigma_{1,3} \\
\sigma_{2,0} & \sigma_{2,1} & \sigma_{2,2} & \sigma_{2,3} \\
\sigma_{3,0} & \sigma_{3,1} & \sigma_{3,2} & \sigma_{3,3} \\
\end{array}
\]

Rijndael (AES):
- Consists of multiple rounds each round consists of 4 steps
- The number of rounds (Nr) depend on Nb and Nk as follows

<table>
<thead>
<tr>
<th>Nr</th>
<th>Nb = 4</th>
<th>Nb = 6</th>
<th>Nb = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nk = 4</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Nk = 6</td>
<td>12</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Nk = 8</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Step 1:
The substitution table S_box : operates on each input byte alone.
Rijndael continue ...

Step 2:
The shift row transformation: Row 0 is not shifted, Row 1 is shifted over \(C_1\) bytes, row 2 over \(C_2\) bytes and row 3 over \(C_3\) bytes.
The shift offsets \(C_1\), \(C_2\) and \(C_3\) depend on the block length \(Nb\).
The different values are:

<table>
<thead>
<tr>
<th>(i)</th>
<th>(C_1)</th>
<th>(C_2)</th>
<th>(C_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Rijndael continue ...

Step 3:
The mix column transformation: each column is represented as a polynomial of degree 3 \((b_3x^3+b_2x^2+b_1x+b_0)\) with \(b_i\) the byte \(i\) in the column and multiplied with \(C(x) = 3x^3+x^2+x+2\) modulo \(x^4+1\)

Rijndael continue ...

Step 4:
The round key addition: the round key is derived from the cipher key

- The total number of Round Key bits is equal to the block length multiplied by the number of rounds plus 1. (e.g., for a block length of 128 bits and 10 rounds, 1408 Round Key bits are needed).
- The Cipher Key is expanded into an Expanded Key.
- Round Keys are taken from this Expanded Key in the following way: the first Round Key consists of the first \(Nb\) words, the second one of the following \(Nb\) words, and so on.
Cryptanalysis.

Find the key (The only secret container)

Mainly Brute Force: $2^n$ different possibilities.

If Key length is not sufficiently large:

- **Key clustering**: 2 diff. Keys ---- same cipher
- **Differential cryptanalysis**: choose carefully 2 different texts and see the difference between their ciphers.

<table>
<thead>
<tr>
<th>Key length (symmetric in years)</th>
<th># of years to break</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>70</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key length (symmetric in years)</th>
<th># of years to break</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>2</td>
</tr>
<tr>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td>58</td>
<td>4</td>
</tr>
<tr>
<td>59</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>61</td>
<td>7</td>
</tr>
<tr>
<td>62</td>
<td>8</td>
</tr>
<tr>
<td>63</td>
<td>9</td>
</tr>
<tr>
<td>64</td>
<td>10</td>
</tr>
<tr>
<td>65</td>
<td>11</td>
</tr>
<tr>
<td>66</td>
<td>12</td>
</tr>
<tr>
<td>67</td>
<td>13</td>
</tr>
<tr>
<td>68</td>
<td>14</td>
</tr>
<tr>
<td>69</td>
<td>15</td>
</tr>
<tr>
<td>70</td>
<td>16</td>
</tr>
<tr>
<td>71</td>
<td>17</td>
</tr>
</tbody>
</table>

**Key length**

Performance: 200MHZ, Pentium, Linux

<table>
<thead>
<tr>
<th>Key/Block length</th>
<th>Speed</th>
<th># cycles for Romanian</th>
</tr>
</thead>
<tbody>
<tr>
<td>(128, 128)</td>
<td>27.6</td>
<td>360</td>
</tr>
<tr>
<td>(192, 128)</td>
<td>22.8</td>
<td>1125</td>
</tr>
<tr>
<td>(256, 128)</td>
<td>19.8</td>
<td>1295</td>
</tr>
</tbody>
</table>

Table 7: Performance figures for the cipher execution (Java)

<table>
<thead>
<tr>
<th>Key/block length</th>
<th>AES-CD (ANSI C)</th>
<th>Brian Gladman/Visual C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed (Mbit/sec)</td>
<td>cycles/block</td>
<td>cycles/block</td>
</tr>
<tr>
<td>(128, 128)</td>
<td>27.6</td>
<td>360</td>
</tr>
<tr>
<td>(192, 128)</td>
<td>22.8</td>
<td>1125</td>
</tr>
<tr>
<td>(256, 128)</td>
<td>19.8</td>
<td>1295</td>
</tr>
</tbody>
</table>

Table 6: Cipher (and inverse) performance
Power Consumption:

Depends on:
- # of dynamic Instructions (Architecture & Compiler)
- # of speculative Inst.
- # of committed Inst.
- # of Cache misses
- Implementation (HW/SW)
- General = # of dynamically committed inst. * XnJ
- Fine = sum over each power consumption component
- Sensor node architecture generally has no Speculation or memory hierarchy so use General

Problems:

- Two possible ways to disclose the key -- both ends.
- Distribution of keys are very difficult (to keep them secret)
- The # of keys increases with the square of the # of nodes

Public Key (Asymmetric) System

- Motivation — problems of symmetric.
- Two keys (only) per user: a public and private.
- Many Applications (Digital signature).
- Support Symmetric systems.

RSA Algorithm:

1. Select at random two (very) large prime numbers P & Q
2. Compute N = PQ
3. Compute O = (P-1)(Q-1)
4. Select a small odd integer E that is relatively prime to O
5. Compute D : \( D \cdot (1/E) \mod O = 1 \) (the multiplicative inverse of E)
6. Publish Public key \((Pu = \text{the pair (E,N)})\)
7. Keep Secret Private key \((Pr = \text{the pair (D,N)})\)
8. \( C = Pu(M) = M^E \mod N \)
9. \( Pr(C) = C^D \mod N \)
Example:

\[ P = 11 \]
\[ Q = 13 \]
\[ N = 11 \times 13 = 143 \]
\[ O = 10 \times 12 = 120 \]
\[ E = 11 \]
\[ D : D \times (1/11) \mod 120 = 1 \? D = 11 \]
\[ M = 7 \]
\[ C = 7^{11} \mod 143 = 106. \]
\[ M = 106 \times 11 \mod 143 = 7 \]

Cryptanalysis:

1. You Know : E,N
2. Want to know: D ?
3. The only way: is to know P & Q to compute O and then D
4. To Know P &Q : you have to factor N
5. The best Known factorization algorithms require: \((\text{EXP}(\text{SQRT}(\text{LN}(N)\times\text{LN}(N))))\) primitive operations

Cryptanalysis …

Asymmetric in hours

Cryptanalysis …

Asymmetric in years
Power Consumption:

1. # of primitive Bit operations = $C \cdot \text{Log}(N)^2$ (Encrypt)
2. # of primitive Bit operations = $C \cdot \text{Log}(N)^3$ (Decrypt)
3. $C$ depends on the detailed implementation and the architecture and the compiler

Summary

Key length (security life):
Brute force cryptanalysis of symmetric key cryptography using a key just enough to keep the data secure for 24 hours (until next refresh time), using a machine of 1000 GHZ primitive operations per second.

$\frac{2^\text{key}}{(1E+12 \cdot 60)} = 24$
which gives key length of 57 bits and we add extra 7 bits for error correction while transferring it through asymmetric system.
The Algorithm that will be used is based on AES standard (Rijndael).

Best known Algorithms for factoring large number ($n$), to be used to break the Asymmetric key system will require

$(\text{EXP}(\text{SQRT}(	ext{LN}(n) \cdot \text{LN}(n))))$ primitive operations. Using the same machine (1000 GHZ), we need the key length to be 940 bit, to secure the symmetric key for one year. Normally used 1024 bit key which provides security for 10 years.
The algorithm will be based on RSA standard.

Computation Power

for symmetric, depends on the Architecture, the compiler, and the specific implementation of the algorithm. For example Rijndael require 48780 clock cycles (768 bytes code length) on an Intel 8051 8-bit processor.

For Asymmetric we require approximately the key length raised to the third power primitive operations