Terminology...

Cipher: A map from the space of the plaintext to the space of the ciphertext.

Stream cipher: A cipher that enciphers the plaintext one character at a time.

Block cipher: A cipher that enciphers the plaintext in chunks of characters.

Mathematical Terminology

$P$: The plaintext.

$M$: The plaintext (message).

$C$: The ciphertext.

$E$: The encryption function.

$D$: The decryption function.

$$E(M) = C$$

$$D(C) = M$$

$$D(E(M)) = M$$

Terminology

Encryption: Disguising a message to hide its contents.

Plaintext: The message we want to hide.

Ciphertext: The encrypted message.

Encode: Converting the plaintext into a standard alphabet.

Decode: Converting the encoded message back into the plaintext.

Encipher: Converting the plaintext to the ciphertext.

Example: `uuencode` converts a binary file into ASCII text. Also, `encrypt`.

Decipher: Converting the ciphertext to the plaintext. Example: `uudecode` reconstitutes a unencoded file. Also, `decrypt`. 
**Terminology – Cryptanalysis**

**Ciphertext-only attack**
- **We have:** the ciphertext of several messages that have been encrypted with the same key.
- **We recover:** the plaintexts, or the key.

**Known-plaintext attack**
- **We have:** the ciphertexts and corresponding plaintexts of several messages.
- **We recover:** the key(s) or an algorithm to decrypt new messages encrypted with the same key.

**Terminology – Keys**

**$K$:** The key, used by the encryption and decryption functions.

**Keyspace:** The range of possible values of the key.

\[
E_K(M) = C
\]

\[
D_K(C) = M
\]

\[
D_K(E_K(M)) = M
\]

**Terminology – Cryptanalysis...**

**Chosen-plaintext attack**
- **We have:** the ciphertext of several messages that have been encrypted with the same key; we get to choose the plaintexts.
- **We recover:** the key(s) or an algorithm to decrypt new messages encrypted with the same key.

**Rubber-hose cryptanalysis**
- **We have:** access to a person who can be threatened, blackmailed, tortured, ... AKA, **purchase-key attack.**
- **We recover:** Everything!

**Terminology – Keys...**

- Some algorithms use different keys for encryption and decryption:

\[
E_{K_1}(M) = C
\]

\[
D_{K_2}(C) = M
\]

\[
D_{K_2}(E_{K_1}(M)) = M
\]

**Cryptosystem:** An algorithm, all possible plaintexts, ciphertexts, and keys.
**Monoalphabetic Substitution Ciphers**

**Caesar Cipher:** Add 3 to the ASCII value of each character, mod 26:

\[ A \rightarrow D, B \rightarrow E, X \rightarrow A, \ldots \]

**ROT13:** Unix utility used on usenet. Adds 13 mod 26 to each letter.

\[ P = \text{ROT13} (\text{ROT13} (P)) \]

- These methods are simple to break: use the fact that different letters in the English alphabet occur with different frequencies.

**Transposition Ciphers**

- In a transposition cipher the original characters of the plaintext are not changed, but simply moved around in the ciphertext. Letter frequencies don’t change.
- In a simple columnar transposition cipher we write the plaintext horizontally in a fixed width table, and read it off vertically.

The plaintext `attack at dawn` could be enciphered into `actwtk atufaamw`, using this table:

\[
\begin{array}{c c c c c}
\text{a} & \text{t} & \text{t} & \text{a} \\
\text{c} & \text{k} & \text{a} \\
\text{t} & \text{d} & \text{a} \\
\text{w} & \text{n} \\
\end{array}
\]

**Substitution Ciphers**

- In a monoalphabetic cipher each character of the plaintext is mapped to a corresponding character of the ciphertext:

\[ A \rightarrow 9, B \rightarrow 11, \ldots \]

- In a homophonic cipher each character of the plaintext is mapped to several characters of the ciphertext:

\[ A \rightarrow \{9, 10, 11\}, B \rightarrow \{3, 1, 8\}, \ldots \]

- In a polygram cipher blocks of characters in the plaintext are mapped to blocks of characters in the ciphertext:

\[ ARF \rightarrow RTW, ING \rightarrow PWQ, \ldots \]

**Substitution Ciphers...**

- In a polyalphabetic cipher you have several keys, each one used to encrypt one letter of the plaintext. We recycle keys when we run out of them:

<table>
<thead>
<tr>
<th>K_1</th>
<th>K_2</th>
<th>K_3</th>
<th>K_1</th>
<th>K_2</th>
<th>K_3</th>
<th>K_1</th>
<th>K_2</th>
<th>K_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>t</td>
<td>t</td>
<td>a</td>
<td>c</td>
<td>k</td>
<td>a</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>v</td>
<td>d</td>
<td>x</td>
<td>t</td>
<td>d</td>
<td>r</td>
<td>p</td>
<td>d</td>
</tr>
</tbody>
</table>

The number of keys is called the period.

- In a running-key cipher (AKA book cipher) one text is used to encrypt another.
**DES**

- *Data Encryption Standard.* Designed by IBM and “certified” by NSA. Widely used in industry. The NSA is rumored to be able to break it in 3–15 minutes.
- A block cipher. Blocks are 64 bits, keys 56 bits:

  64-bit plaintext \[\rightarrow\] 56-bit key \[\rightarrow\] 64-bit ciphertext

DES is a symmetric algorithm; the same key and algorithm is used for encryption and decryption.

**Confusion and Diffusion**

- DES is a combination of two basic principles:
  - **confusion:** Confusion scrambles up the letters of the plaintext so that it has no direct relationship to the ciphertext. Substitution ciphers do this.
  - **diffusion:** Diffusion spreads the plaintext out over the ciphertext. Transposition (AKA permutation) ciphers do this.
- DES has 16 rounds. Each round consists of a substitution followed by a permutation.

**Exclusive-OR**

\[
\begin{array}{c|c|c}
0 \oplus 0 & a \oplus a & = 0 \\
0 \oplus 1 & a \oplus b \oplus b & = a \\
1 \oplus 0 & a \oplus a & = a \\
1 \oplus 1 & 0 & &
\end{array}
\]

- Since xor-ing the same value twice gives us the original, we get a simple symmetric algorithm:

  \[
P \oplus K = C
  \]

  \[
  C \oplus K = P
  \]

**One-Time Pads**

- The **pad** is a large, non-repeating set of random key letters.
- To encrypt, add each plaintext letter to the next letter on the pad, mod 26:

<table>
<thead>
<tr>
<th>r</th>
<th>a</th>
<th>n</th>
<th>d</th>
<th>o</th>
<th>m</th>
<th>p</th>
<th>a</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>t</td>
<td>t</td>
<td>a</td>
<td>c</td>
<td>k</td>
<td>a</td>
<td>t</td>
<td>d</td>
</tr>
<tr>
<td>s</td>
<td>u</td>
<td>h</td>
<td>e</td>
<td>r</td>
<td>w</td>
<td>q</td>
<td>u</td>
<td>h</td>
</tr>
</tbody>
</table>

Decryption is done the same.

- This is provably secure, provided you have a truly random set of pad letters and never reuse the pad.
**DES – The Rounds**

1. Split 64-bit input into $L_0$ and $R_0$, each 32 bits.
2. Round 1:
   \[ L_1 = R_0 \]
   \[ R_1 = L_0 \oplus f(R_0, K_1) \]
3. Round $i$:
   \[ L_i = R_{i-1} \]
   \[ R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \]
4. Round 16:
   \[ L_{16} = R_{15} \]
   \[ R_6 = L_{15} \oplus f(R_{15}, K_{16}) \]
5. Return the 64-bit ciphertext $L_{16}R_{16}$.

**DES Overview – The Rounds**

Slide 11–16

Slide 11–17

**DES – The $f$ Function**

- $f$ is the function that combines the data (the right 32-bit half, $R$) with the key during each round:
  1. Get 48 bits of the key,
  2. Expand $R$ to 48 bits using the expansion permutation,
  3. XOR the expanded $R$ and the compressed key,
  4. Send the result through 8 S-boxes using the S-Box Substitution to get 32 new bits,
  5. Permute the result using the P-Box Permutation,
- The result of the $f$ function is XORed with the left half ($L$) to get the new right half.

**DES – The $f$ Function**

Slide 11–18

Slide 11–19
DES – Key Manipulation

- Each round, 48 bits of the 56-bit key is extracted and used by the function. Also, a new version of the key is generated to be used in the next round:
  1. Split the key into two halves, each 28 bits.
  2. Shift each half by 1 or 2 steps.
  3. Compress the two halves into 48 bits using the Compression Permutation.
  4. Merge the two shifted halves into the key for the next round.

Slide 11–20

DES – Expansion Permutation

- The expansion permutation takes the 32 bits of $R_{i}$ and permutes these bits, and, by copying some of them, expands $R_{i}$ into 48 bits.
- The table below says that bit 32 of $R_{i}$ moves into position 1, bit 1 moves to 2, 2 to 3, 4 to 5, 5 to 6, 6 to 7, 7 to 8, 8 to 9.

<table>
<thead>
<tr>
<th>Round</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Slide 11–21

DES – Tables, Tables, Tables...

- The various permutations, shifts, and substitutions used by the algorithm are all specified using tables.
- The shift table expresses how much (1 or 2 steps) the key should be circularly shifted each round. Because of this shift, a different part of the key will be used in each round.

Slide 11–22

DES – Expansion Permutation

- The expansion permutation takes the 32 bits of $R_{i}$ and permutes these bits, and, by copying some of them, expands $R_{i}$ into 48 bits.
- The table below says that bit 32 of $R_{i}$ moves into position 1, bit 1 moves to 2, 2 to 3, 4 to 5, 5 to 6, 6 to 7, 7 to 8, 8 to 9.

<table>
<thead>
<tr>
<th>Round</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
**Slide 11–24**

- **DES – S-Box Substitution**
  - The expanded right (R_{e-1}) and the compressed key (K) are XORed together. The 48-bit result is then passed through 8 S-Boxes (Substitution Boxes) to form 32 bits.
  - Each of the 8 S-Boxes is different. Each takes 6 bits of input and produces 4 bits of output.

**Slide 11–25**

- **DES – S-Box Substitution Tables**
  - Each S-Box is given by a table consisting of 4 rows of 16 numbers. Each number is a 4-bit quantity.
  - The input to each S-Box is 6 bits.
  - The S-Box table is indexed by taking the outer 2 bits (b1 and b0) and forming a number between 0 and 3. This is used to get the row number. The middle 4 bits (b2 ... b6) get the column number.
  - The S-Boxes are what gives DES its security.

**Slide 11–26**

**Slide 11–27**
**Symmetric Encryption Protocol**

- Now that we have access to a symmetric cryptosystem like DES, how do we use it?
- We have to describe a protocol that shows how each party uses the cryptosystem to solve a communication/security problem.
  1. Alice and Bob agree on a cryptosystem.
  2. Alice and Bob agree on a key.
  3. Alice encrypts her plaintext, getting a ciphertext.
  4. Alice sends the ciphertext to Bob.
  5. Bob decrypts the message using the same cryptosystem and key.

**Symmetric Encryption Protocol – Attacks**

- What can an attacker do?
- If Eve listens in on the communication between Alice and Bob she will get a sequence of ciphertext messages. She can use these to launch a ciphertext-only attack.
- Eve could also try to listen in to the first two parts of the protocol, where Alice and Bob decide on a key and cryptosystem to use.
- Eve could also sit in the middle, intercept Alice’s messages, and substitute her own messages encrypted with the key she has discovered.

**DES – P-Box Permutation Tables**

- The final operation in the f function is the P-Box.
- It’s a strict permutation of the 32 bits that come out of the S-Box.
- Bit number 16 moves into position 1, bit 7 into 2, 20 into 3, etc:

| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 | 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 | 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

**DES – Decryption**

- The same algorithm we just described for encryption can also be used for decryption.
- However, we have to use the keys in the reverse order.
- That is, during decryption we use the keys
  \[ K_{16}, K_{15}, \ldots, K_2, K_1. \]
**Public Key Protocol**

- Key-management is the main problem with symmetric algorithms. Bob and Alice have to somehow agree on a key to use.
- In public key cryptosystems there are two keys, a public one used for encryption and a private one for decryption.
  1. Alice and Bob agree on a public key cryptosystem.
  2. Bob sends Alice his public key.
  3. Alice encrypts her plaintext using Bob's public key and sends it to Bob.
  4. Bob decrypts the message using his private key.

**RSA**

- RSA is the best known public-key cryptosystem. Its security is based on the (believed) difficulty of factoring large numbers.

1. Select two random primes $p$ and $q$.
2. Compute $n = pq$.
3. Compute $\phi(n) = (p-1)(q-1)$.
4. Select a small odd integer $e$ relatively prime with $\phi(n)$.
5. Compute $d = e^{-1} \pmod{\phi(n)}$.
6. $P = (e, n)$ is the RSA public key.
7. $S = (d, n)$ is the RSA private key.

**A Hybrid Protocol**

- In practice, public key cryptosystems are not used to encrypt messages — they are simply too slow.
- Instead, public key cryptosystems are used to encrypt keys for symmetric cryptosystems. These are called session keys, and are discarded once the communication session is over.

1. Bob generates a session key $K$ and sends it to Bob.
2. Bob encrypts the message using his private key to get the session key $K$.
3. Both Alice and Bob communicate by encrypting their messages using $K$.

To encrypt a message $M$, do:

$$P(M) = M^e \pmod{n}$$

To decrypt a ciphertext $C$, do:

$$S(C) = C^d \pmod{n}$$
Modular Arithmetic...

- The inverse of 4 is $\frac{1}{4}$. Modular inverses are harder.
- To find the the inverse of 4 modulo 7 we want to compute:
  
  $$4 \times x = 1 \pmod{7}$$

  which is the same as finding integers $x$ and $k$ such that

  $$4x = 7k + 1$$

- This is also written: $4^{-1} = x \pmod{n}$.
- Sometimes inverses exist, sometimes not:

  $$5^{-1} = 3 \pmod{14}$$  
  $$2^{-1} = ? \pmod{14}$$

RSA Example

1. Select two primes: $p = 47$ and $q = 71$.
2. Compute $n = pq = 3337$.
3. Compute $\phi(n) = (p - 1)(q - 1) = 3220$.
4. Select $e = 79$.
5. Compute

   $$d = e^{-1} \pmod{\phi(n)}$$

   $$= 79^{-1} \pmod{3220}$$

   $$= 1019$$

6. $P = (79, 3337)$ is the RSA public key.
7. $S = (1019, 3337)$ is the RSA private key.

Modular Arithmetic

- $a \equiv b \pmod{n}$ if $a = b + kn$ for some integer $k$.

  $$(a + b) \pmod{n} = ((a \pmod{n}) + (b \pmod{n})) \pmod{n}$$
  $$(a - b) \pmod{n} = ((a \pmod{n}) - (b \pmod{n})) \pmod{n}$$
  $$(a \times b) \pmod{n} = ((a \pmod{n}) \times (b \pmod{n})) \pmod{n}$$

  $$23 \equiv 11 \pmod{12}$$
  $$23 \equiv 2 \pmod{7}$$
  $$(10 + 13 \pmod{7})$$

  $$(10 \pmod{7}) + (13 \pmod{7}) \pmod{7}$$

  $$= 3 + 6 \pmod{7} = 2$$

Modular Arithmetic...

- Modular exponentiation is an important operation in cryptography:

  $$a^x \pmod{n} = (a \times a \times \cdots \times a) \pmod{n}$$
  $$a^8 \pmod{n} = (a \times a \times a \times a \times a \times a \times a \times a) \pmod{n}$$

  $$= ((a^2 \pmod{n})^2 \pmod{n}) \pmod{n}$$
Encrypt $M = 688232687966683$.

1. Break up $M$ into 3-digit blocks:
   
   $$m = (688, 232, 687, 966, 668, 003)$$
   
   Note the padding at the end.

2. Encrypt each block:
   
   $$c_1 = m_1^e \mod n$$
   
   $$= 688^{79} \mod 3337$$
   
   $$= 1570$$
   
   We get:
   
   $$c = (1570, 2756, 2091, 2276, 2423, 158)$$

3. Decrypt each block:
   
   $$m_1 = c_1^d \mod n$$
   
   $$= 1570^{1019} \mod 3337$$
   
   $$= 688$$

---

**Digital Signatures**

- Often, Bob will want to make sure that the document he gets from Alice in fact originated with her.
- In the non-digital world, Alice would sign the document. We can do the same with digital signatures.

1. Alice encrypts her document with her private key, thereby signing it.
2. Alice sends the signed document to Bob.
3. Bob decrypts the document using Alice's public key, thereby verifying her signature.

---

**One Way Hash Functions**

- Public key algorithms are too slow to sign large documents.
- A better protocol is to use a one way hash function.

1. Alice computes a one-way hash of her document.
2. Alice encrypts the hash with her private key, thereby signing it.
3. Alice sends the encrypted hash and the document to Bob.
4. Bob decrypts the hash Alice sent him, and compares it against a hash he computes himself of the document. If they are the same, the signature is valid.

---

**Software**

- An RSA demonstration applet: http://cisnet.bazarch.org/cmy.edu/holocrzak/classes/9444/rsademod#overview
- An RSA demonstration applet: http://www-fs.informatik.uni-marburg.de/~reichard/crypto/English/4.1/applet_e.html
- A modular exponentiation applet: http://www.math.umn.edu/~garrett/crypto/a01/FastPow.html
**Software – PGP**

- We can extract Sally’s public key from the keyfile:

  > pgp -kx sally sally.pgp
  > cat sally.pgp

  ----BEGIN PGP PUBLIC KEY BLOCK-----
  Version: PGP 6.5.8
  mQGiBDxZiKMRBAdptM3zA2n0wuQwSc8bWy... SEKU5724Fcd8nOLD5/KAQkw01Fo4BzTn8QR... r5o20x7b+ZnwDhkdddFEnDrWz2vzNL4GF9... /EmDUnqStgd3aL82DQ8Kv0EAIjDGcUepG... XAz6ZCrz7kMC97277t7xG73gVy61MLnxCU1... 
  ----END PGP PUBLIC KEY BLOCK-----

**Software – PGP**

- We can encrypt a message using Sally’s public key:

  > pgp -ea message sally

  Key for user ID: Sally Student <sally@student.org>
  1024-bit key, Key ID 68CBA84E, created 2002/01/31
  Transport armor file: message.asc

  > cat message.asc

  ----BEGIN PGP MESSAGE-----
  Version: PGP 6.5.8
  qANQR1DBw64Da/7A1bGYfWQQ/A9mrK0... ChLql0s6oh04+uLMsrJqQnH5Jz+FZZH... f619C+sXQIGS7djHSqgCIjC6qK4AyV... Og+cvAP+K7LA7ke+83LScVCLrOlujL... 
  ----END PGP MESSAGE-----

**Software – des**

- cat message
  
  Attack at dawn

  > des -k 12345678 -e message ciphertext
  > od ciphertext
  0000000 036367 147372 026370 003617 051751 024226 132155 011603 0000020

  > des -k 12345678 -d ciphertext
  Attack at dawn

**Software – PGP**

- pgp is a public domain implementation of RSA.
- First you have to generate a public and private key-pair:

  > pgp -kg

  Choose 1, 2, 3, or enter desired number of bits: 1
  Generating a 1024-bit DSS/DH key.

  Enter a user ID for your public key: Sally Student \\
  <sally@student.org>
  Enter pass phrase: go 2 Taco Hell
  Enter same pass phrase again: go 2 Taco Hell

  > pgp -kvv

  Type bits/keyID Date User ID
  sec 1024/68CBA84E 2002/01/31 Sally Student <sally@student.org>
Software – PGP...

- Sally can now decrypt the message using her private key:

  > pgp message.asc  
  Key for user ID: Sally Student <sally@student.org>  
  Enter pass phrase: go 2 Taco Hell  
  cat message  
  > pgp message.asc  
  Attack at dawn

---BEGIN PGP SIGNATURE-----
Version: PGP 6.5.8
iQA/AwUBPmcQL018iJoy6OeEQlmWwCeMoW...
-----END PGP SIGNATURE-----
File is signed. Good signature from "Sally Student <sally@student.org>". Signature made 2002/01/31 19:34 GMT

Readings and References

- The information in this lecture is taken from Applied Cryptography by Bruce Schneier.
- More on RSA: http://www.hobnob.com/mcs225/
- Peter Wayner. Disappearing Cryptography.