Chapter 7

Lecture 7 - Encryption

Mid semester Test

- ✔ 9th October 2003
- ✔ LT27, 14:30
- ✔ MCQ, closed book
- ✔ Covers everything up to the lecture before...

Last session

- Information flow
- Simple error detection
- Simple error correction

This session

- Finish on error correction
- Encryption
  - Symmetric keys
    * DES
  - Public keys
    * RSA

Key points from last week

✔ Error detection vs Error correction

✔ Mathematical analysis

✔ Error rate, noise, channel capacity

✔ Theoretical vs actual channel capacity

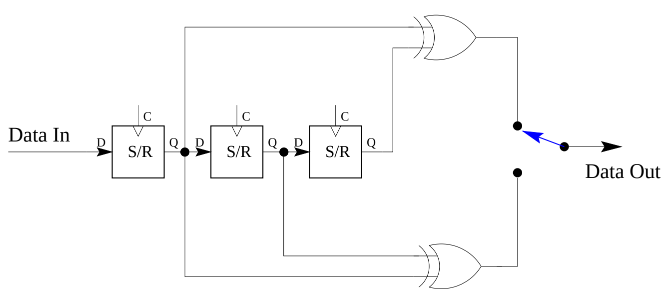
This session

- Finish on error correction
- Encryption
  - Symmetric keys
    * DES
  - Public keys
    * RSA

Convolutional codes

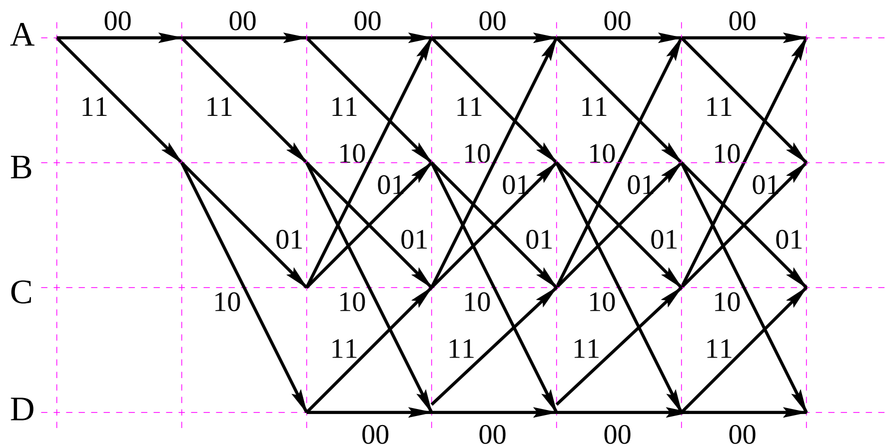
- ✔ Convolutional codes operate continuously and so are especially useful in data transmission systems.
- ✔ The convolutional *encoder* operates on a continuous stream of data using a shift-register to produce a continuous encoded output stream.

Convolutional codes



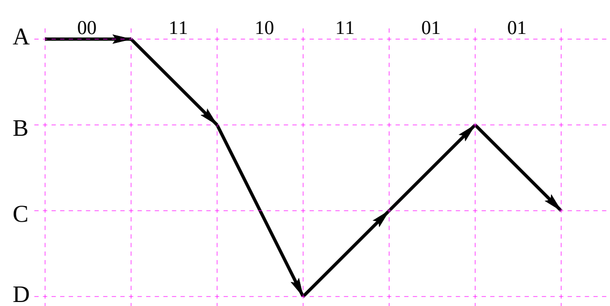
Received bit sequence can be examined for the *most likely correct* output sequence

Trellis diagram



Most likely path

If we were to input the sequence **011010**, we would get the following trace through the trellis, with the bit sequence output as **001110110101**:



Convolutional codes

- ✔ Determine the *most likely* path, even with large numbers of bit errors.
- ✔ A convolutional encoding can often reduce errors by a factor of 10^2 to 10^3.

Viterbi decoding

- ✔ The ***Viterbi*** algorithm tries to find the most likely received data sequence, by keeping track of the four *most likely* paths through the trellis.
- ✔ For each path, a running count of the *hamming* distance between the received sequence and the path is maintained.
- ✔ The most likely received string is the one with the lowest hamming distance.

This session

- Finish on error correction
- Encryption
  - Symmetric keys
    - DES
  - Public keys
    - RSA

Encryption and authentication

Security and Cryptographic systems act to reduce failure of systems due to the following threats:

Interruption - attacking the availability of a service (Denial of Service).

Interception - attacks confidentiality.

Modification - attacks integrity.

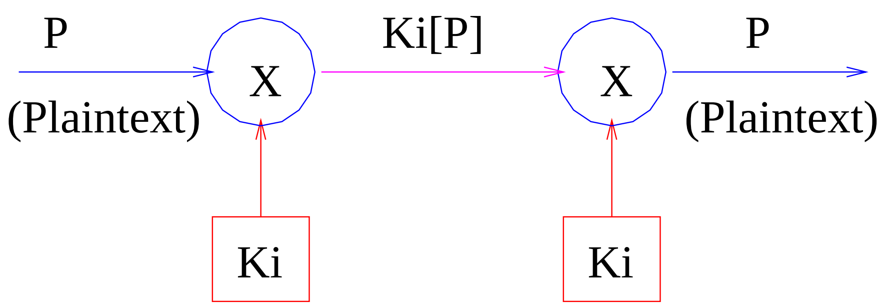
Fabrication - attacks authenticity. Note that you may not need to decode a signal to fabricate it - you might just record and replay it.

Encoding and deciphering

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.

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*".

Transposition

- ✔ 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

- ✔ Substitution cipher systems encode the input stream using a substitution rule.
- ✔ The Cæsar 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.

Substitution

Code	Encoding
A	Q
B	V
C	X
D	W
...	...

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**	**...**
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

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

Key	B	A	D	B	A	D	B	A
Text	H	A	D	A	F	E	E	D
Cipher	I	A	G	B	F	H	F	D

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

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

where F_i is the frequency of the occurences of symbol i and N is the length of the cipher.

Index of coincidence

```
#!/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) ;
```

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

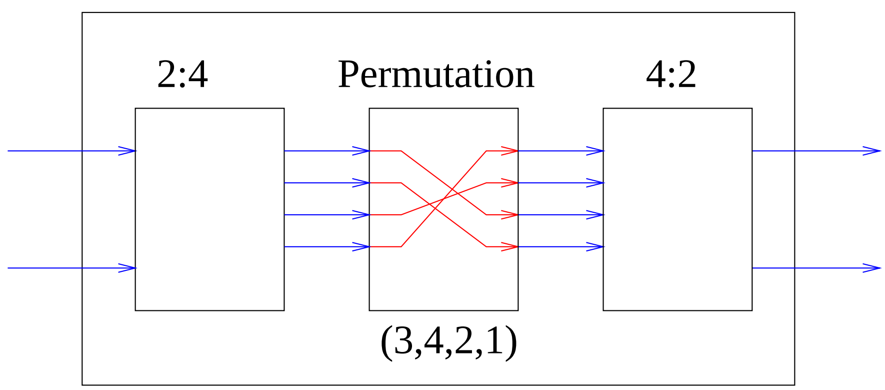
Index of coincidence

- ✔ 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.

This session

- Finish on error correction
- Encryption
  - Symmetric keys
    - DES
  - Public keys
    - RSA

S-box



S-boxes and P-boxes

- ✔ The S-box (Substitution-Box) is a hardware device which encodes *n* bit numbers to other *n* bit numbers and can be represented by a permutation.
- ✔ A P-box is just a simple permutation box.
- ✔ If you use an S-box and a P-box at once, you have a product cipher which is generally harder to decode.

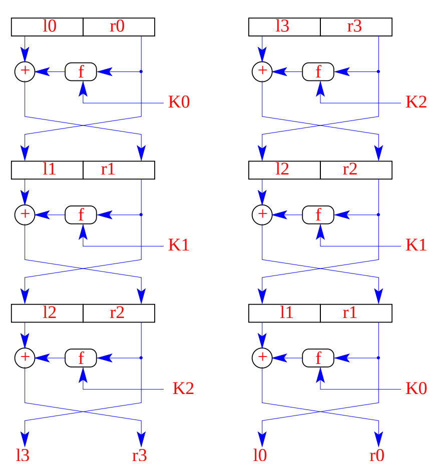
DES - Data Encryption Standard

- ✔ 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

- ✔ 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.

Feistel

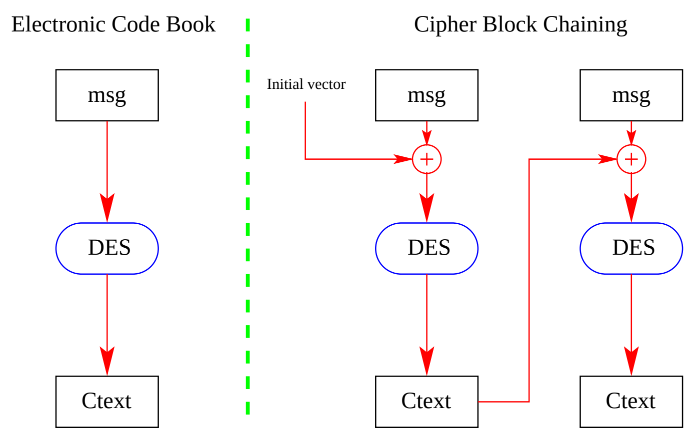


Each of the 16 stages (rounds) of DES uses a Feistel structure which encrypts a 64 bit value into another 64 bit value using a 48 bit key derived from the original 56 bit key.

DES modes of operation

- ✔ 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



DES software

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

```
#include “des.h”
//
// - Your calls
```


DES software

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

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

where the *encrypt* parameter determines if we are to encipher or decipher.

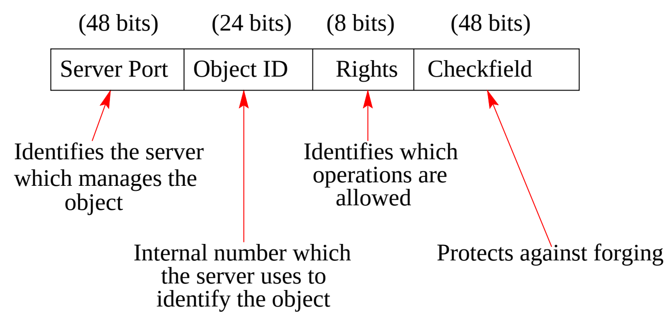
The *schedule* contains the secret DES key.

Case study: Amoeba capabilities

- ✔ All Amoeba objects are identified by a *capability* string which is encrypted using DES encryption. A *capability* is long enough so that you can't just make them up.

- ✔ If you have the string, you have whatever the capability allows you. If you want to give someone some access to a file, you can give them the capability string. They place this in their directory, and can *see* the file.

Case study: Amoeba capabilities



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.

This session

- Finish on error correction
- Encryption
  - Symmetric keys
    - DES
  - Public keys
    - RSA

Public key systems

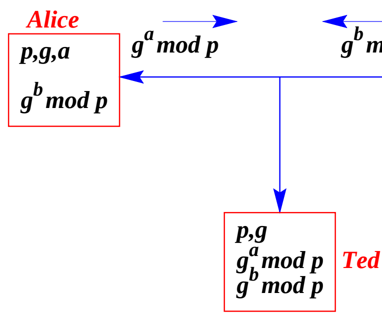
✔ In 1976 Diffie and Hellman published the paper "*New Directions in Cryptography*", which first introduced the idea of *public* key cryptography.

✔ Public key cryptography relies on the use of enciphering functions which are not *realistically* invertible unless you have a deciphering key.

✔ For example, we have the discrete logarithm problem in which it is relatively easy to calculate $n = g^k \textbf{ mod } p$ given g, k and p, but difficult to calculate k in the same equation, given g, n and p.

Diffie-Hellman key agreement

Two separated users *create* and *share* a secret key. A third party is not realistically able to calculate the shared key.



Knowledge different

- All participants know two system parameters p, and g
- Alice and Bob each have a secret value (Alice has a and Bob has b)
- Alice and Bob each calculate and exchange a public key ($g^a \bmod p$ for Alice and $g^b \bmod p$ for Bob).
- Ted knows g, p, $g^a \bmod p$ and $g^b \bmod p$, but not a or b.

Diffie-Hellman key agreement

Both Alice and Bob can now calculate the value $g^{ab} \bmod p$.

1. Alice calculates $(g^b \textbf{ mod } p)^a \textbf{ mod } p = (g^b)^a \textbf{ mod } p$.

2. Bob calculates $(g^a \textbf{ mod } p)^b \textbf{ mod } p = (g^a)^b \textbf{ mod } p$.

And of course $(g^b)^a \textbf{ mod } p = (g^a)^b \textbf{ mod } p = g^{ab} \textbf{ mod } p$ which is the shared key.

Diffie-Hellman key agreement

Ted has a much more difficult problem. It is difficult to calculate $g^{ab} \textbf{ mod } p$ without knowing either a or b. The algorithmic run-time of the (so-far best) algorithm for doing this is in

$$O(e^{c\sqrt{r \log r}})$$

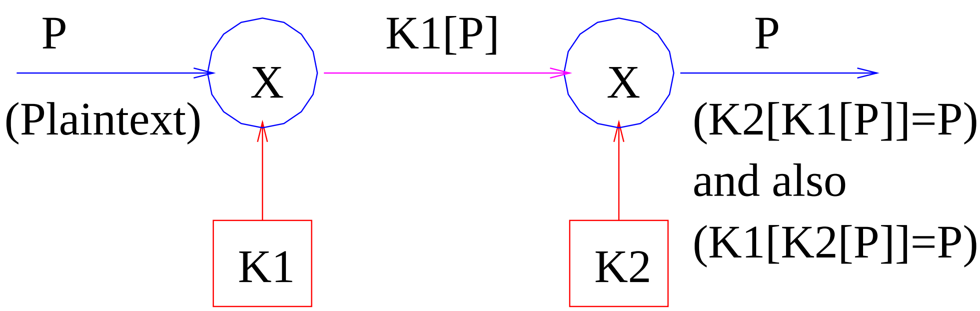
where c is small, but ≥ 1, and r is the number of bits in the number.

Diffie-Hellman key agreement

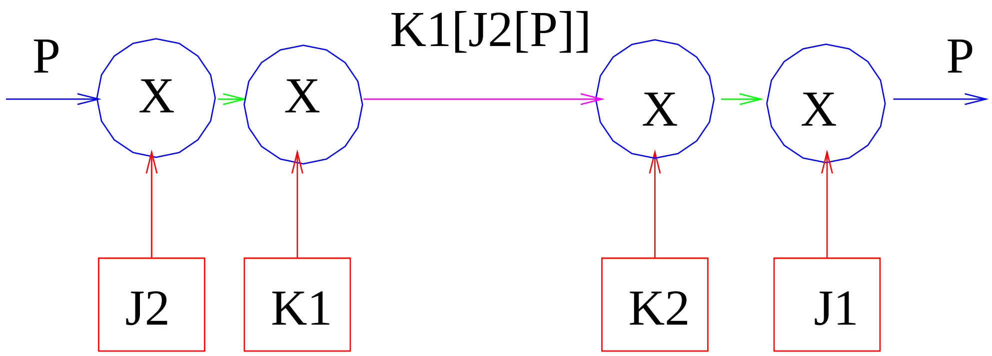
By contrast, the enciphering and deciphering process may be done in $O(r)$:

Bit size	Enciphering	Discrete logarithm solution
10	10	23
100	100	1,386,282
1,000	1,000	612,700,000,000,000,000,000,000

Encryption



Authentication



This session

- Finish on error correction
- Encryption
  - Symmetric keys
    * DES
  - Public keys
    * RSA

RSA (Rivest, Shamir, Adelman)

This public key system relies on the difficult problem of trying to find the complete factorization of a large composite[9] integer whose prime factors[10] are not known.

[9] An integer larger than 1 is called *composite* if it has at least one divisor larger than 1.

[10] The *Fundamental Theorem of Arithmetic* states that any integer N (greater than 0) may be expressed uniquely as the product of prime numbers.

RSA hacks

Two RSA-encrypted messages have been cracked:

- The inventors of RSA published a 129-digits (430 bits) RSA public key. In 1994, it was factored with 5000 MIPS-years of computing time.
- A year later, a 384-bit PGP key was cracked. It needed 1300 MIPS-years to factor the key in three months.

Note that these efforts each only cracked a single RSA key.

RSA hacks

If you happen to be able to factor the following number, please tell Hugh - we can split US$200,000[11]!

```
2519590847565789349402718324004839857142928212620403202777713783604366202
0707595556264018525880784406918290641249515082189298559149176184502808489
1200728449926873928072877767359714183472702618963750149718246911650776133
7985909570009733045974880842840179742910064245869181719511874612151517265
4632282216869987549182422433637259085141865462043576798423387184774447920
7399342365848238242811981638150106748104516603773060562016196762561338441
4360383390441495263443219011465754445417842402092461651572335077870774981
7125772467962926386356373289912154831438167899885040445364023527381951378
636564391212010397122822120720357
```

[11] US$150,000 for me, US$50,000 for you...

RSA coding algorithms

Below are outlined the four processes needed for RSA encryption:

1. Creating a public key
2. Creating a secret key
3. Encrypting messages
4. Decoding messages

To create public key K_p

1. Select two different large primes P and Q.
2. Assign $x = (P-1)(Q-1)$. (Does this ring a bell?)
3. Choose E relative prime to x. (This must satisfy condition for K_s given later)
4. Assign $N = P * Q$.
5. K_p is N concatenated with E.

To create private (secret) key K_s

1. Choose D: $D * E$ **mod** $x = 1$.
   (a) (i.e. multiplicative inverses)
   (b) another way: $DE = k(P-1)(Q-1) + 1$
2. K_s is N concatenated with D.

To encode plain text m

1. Pretend m is a number.
2. Calculate $c = m^E \textbf{ mod } N$.

To decode c back to m

1. Calculate $m = c^D \textbf{ mod } N$.
2. ....WHY?....

...Why?...

$$
\begin{aligned}
c^D \text{ mod } N &= m^{ED} \text{ mod } N \\
&= m^{k(P-1)(Q-1)+1} \text{ mod } PQ \\
&= m * m^{k(P-1)(Q-1)} \text{ mod} PQ
\end{aligned}
$$

- $m^{P-1} \text{ mod } P = 1$, so $(m^{(P-1)})^{k(Q-1)} \text{ mod} P = 1$
- $m^{Q-1} \text{ mod } Q = 1$, and so (tutorial) $(m^{(P-1)})^{k(Q-1)} \text{ mod} PQ = 1$.

$$c^D \text{ mod } N = m^{ED} \text{ mod } N$$

RSA code

```
#!/usr/bin/perl -sp0777i<X+d*lMLa^*lN%0]dsXx++lMlN/dsM0<j]dsj
$/=unpack('H*',$_);$_=`echo 16dio\U$k"SK$/SM$n\EsN0p[lN*1
lK[d2%Sa2/d0$^Ixp"|dc`;s/\W//g;$_=pack('H*',/((..)*)$/)
```

and then

- `echo "squeamish ossifrage" | ./rsa.perl -k=10001 -n=1967cb529 > msg.rsa`
- `./rsa.perl -d -k=ac363601 -n=1967cb529 < msg.rsa`

Testing large numbers for primality

RSA requires us to generate large prime numbers, but there is no algorithm for constructing arbitrarily large prime numbers. Instead we use statistical testing methods to determine primality.

Quiz! Is $162,259,276,829,213,363,391,578,010,288,127$ prime[12]?

After choosing a large random (odd) number p, we can quickly see if p is divisible by 2, 3 and so on (say all primes up to 1000). If our number p passes this, then we can perform some sort of statistical primality test.

[12] Note that this is only a 33 digit number, and we typically use prime numbers with hundreds of digits.

Lehmann test

1. Choose a random number w(for witness) less than p
2. If $w^{(p-1)/2} \not\equiv \pm 1 \textbf{ mod } p$ then p is not prime
3. If $w^{(p-1)/2} \equiv \pm 1 \textbf{ mod } p$ then the likelihood is less than 0.5 that p is not prime

Repeat the test over and over, say n times. The likelihood of a false positive will be less than $\frac{1}{2^n}$. Other tests, such as the Rabin-Miller test may converge more quickly.

Primes is in P!

- ✔ Group at the Indian Institute of Technology have discovered the unexpected result that testing a number for primality can be done in polynomial time, rather than using probabilistic tests as just shown.
- ✔ This is unlikely to affect the effectiveness of public key systems.
- ✔ The paper is only 7 pages long and is beautifully written...

Case study: PGP

- ✔ PGP (Pretty Good Privacy) is a public key encryption package to protect E-mail and data files.
- ✔ It lets you communicate securely with people you've never met, with no secure channels needed for prior exchange of keys.
- ✔ PGP can be used to append digital signatures to messages, as well as encrypt the messages, or do both.

Case study: PGP

- ✔ It uses various schemes including patented ones like IDEA and RSA.
- ✔ The patent on IDEA allows non-commercial distribution, and the RSA patent has expired.
- ✔ However there are also commercial versions of PGP.
- ✔ PGP can use, for example, 2048 bit primes, and it is considered unlikely that PGP with this level of encryption can be broken.

Uses of encryption

1. Generating encrypted passwords with 1-way functions
2. Checking integrity by appending digital signature
3. Checking the authenticity of a message.
4. Encrypting timestamps with messages to prevent replay attacks.
5. Exchanging a key.