Key Management Issues

- Key management is the hardest part of cryptography
  - How should keys be generated so that they can not be easily guessed?
  - How to securely store keys so that they can not be easily stolen?
  - How could keys be delivered to their intended recipients securely?
  - How could two entities agree on, or establish, a key securely?
  - How are keys revoked and replaced?
  - ‘People’ and ‘management’ are at the centre of some of these issues
- For symmetrical ciphers - how to keep keys **secret**?
- For public-key ciphers - how to ensure private key secret and public keys **trust-worthy**?

Key Management Issues - Keys spaces

- Number of possible keys (key space) given various constraints:

	6-bytes	8-bytes
Lowercase letters(26)	$3.1*10^{8}$	$2.1*10^{11}$
Lowercase letters & digits (36)	$2.2*10^{9}$	$2.8*10^{12}$
Alphanumeric characters (62)	$5.7*10^{10}$	$2.2*10^{14}$
Printable characters (95)	$7.4*10^{11}$	$6.6*10^{15}$
ASCII characters (128)	$4.4*10^{12}$	$7.2*10^{16}$

MANCHESTER 1824

The University of Manchester

Key Management Issues - Keys spaces

- Exhaustive search (assume 10^6 attempts/second):

	6-bytes	8-bytes
Lowercase letters(26)	5 minutes	2.4 days
Lowercase letters & digits (36)	36 minutes	33 days
Alphanumeric characters (62)	16 hours	6.9 years
Printable characters (95)	8.5 days	210 years
ASCII characters (128)	51 days	2300 years

Key Management Issues - Keys spaces

- **Main points**
  - Giving various constraints on the input string can greatly reduce the number of possible keys (key space), making ciphertexts much easier to break!
  - Computer power increases all the time ...
    - If you expect your keys to stand up against brute-force attacks for 10 years, plan accordingly.

Key Management Issues – Key generation

- Good keys are random numbers.
- Users tend to choose less random keys.
- Which of these keys is more random (more difficult to guess) - *Barney1* or **9(hH/A*?
- Remember: a smart brute-force attacker doesn't try all possible keys in numeric order; he will try the obvious ones first.

- What is a random number?
  - Given an integer, $k>0$, and a sequence of numbers, $n_1, n_2, \ldots$, an observer can not predict n_k even if all of $n_1, \ldots, n_{k-1}$ are known.

Key Management Issues - Key generation

- Ordinary random number generation functions, e.g. `java.util.Random`, is <u>not</u> good enough for this purpose.
- Use a cryptographically secure pseudo-random-number generator, e.g. `SecureRandom` class in `java.security` package, or a reliably random source.

- Physical sources of random numbers
  - Based on nondeterministic physical phenomena, e.g. atmospheric noise,
  - stock market data, etc.

Key Management Issues - Key generation

- Some pseudo-random numbers are generated from a strong mixing function
  - that takes two or more inputs having some randomness (e.g. CPU load, arrival times of network packets), but produces an output each bit of which depends on some nonlinear function of all the bits of the inputs.
  - Cryptographic hashing functions and encryption algorithms (e.g. MD5, SHA3 and AES) are examples of the strong mixing function.

- For example, in a UNIX system, you may use the process state at a given time (date ; ps gaux) as the input to a MD5 function to generate a pseudorandom number, where 'ps gaux' lists all the information about all the processes on the system.

Key Management Issues - Key storage

- You must protect the key to the same degree as all the data it encrypts.
  - Why would one bother to go through all the trouble of trying to break the cipher system if he can recover the key because of sloppy key storage procedures?
  - Why would one spend $10 million building a cryptanalysis machine if he could spend $1000 bribing a clerk?

Key Management Issues - Key storage

- Attackers may defeat access control mechanisms, so should always encrypt the file containing keys.
- If possible, a key should not come out of the device generating the key.
- A key should never appear unencrypted outside the encryption device.
- Try not to store your key on a medium connected to the network.
- Key may be resident in memory, and attackers may be able to access it via, e.g. malware
  - Use a physical token to store the key (e.g. a smart card) and protect the token with a PIN number.
  - Card can be stolen, so splitting a key into two halves, store
    - one half in the machine, and
    - another half in the card.
- Splitting a key, K, into two halves (k_1, k_2): $k_2 = k_1$ xor K

Key Management Issues – more issues

- Key access to make sure only authorized users could gain access to some specific keys.
- Key updates to ensure key freshness (forward secrecy).
- Key deletions to discard any unrequired keys securely.
- Key usage auditing to ensure keys are used properly and securely.