Table of Contents

[Key Management Issues 2](#_Toc530523198)

[Key spaces 2](#_Toc530523199)

[Key Generation 3](#_Toc530523200)

[Key Storage 3](#_Toc530523201)

[Session Key Establishment 3](#_Toc530523202)

[Key agreement 3](#_Toc530523203)

[Key Transportation 3](#_Toc530523204)

[Entity and key authentication 3](#_Toc530523205)

[Key Freshness 4](#_Toc530523206)

[Diffie-Hellman Protocol 4](#_Toc530523207)

[Initial Condition 4](#_Toc530523208)

[Protocol 4](#_Toc530523209)

[4](#_Toc530523210)

[Man-in-the-middle attack vulnerability 4](#_Toc530523211)

[5](#_Toc530523212)

[Approach (1) solution 5](#_Toc530523213)

[Approach (2) solution 5](#_Toc530523214)

[Approach 1 VS Approach 2 6](#_Toc530523215)

[6](#_Toc530523216)

[Needham-Schroeder Protocol 6](#_Toc530523217)

# Key Management Issues

* This involves issues relating to **storing**, **strengthening keys**, **delivery**, **agreement of keys** and **revoking and replacement** of keys.
* Need to keep **symmetrical keys** secret.
* Need to confirm that **public keys** belong to who we think it they belong to.

## Key spaces

* This table shows that as the **key space increases** in size the **number of possible keys also increases** in size.
* Increases difficulty of revealing key by **brute-force.**
* **Increasing the number of bytes** of the key **increases the number of possible keys** also.
* Keep in mind that **computing power approximately doubles every 18 months** when choosing a key.
* **Making passwords random e.g. 430in6!23 rather than common and guessable e.g. D.O.B** also increases difficulty of revelation by brute-force.



## Key Generation

* Good keys are random numbers.
* Java.util.Random
* Cryptographically secure pseudo random number generator: SecureRandom class in java.security package.
* **Random number definition**: Given an integer, k >0, and a sequence of numbers, n1, n2, ..., an observer cannot predict nk even if all of n1, ..., nk-1 are known. The previous number/numbers cannot determine the next.
* **Physical sources of random numbers: atmospheric noise, stock market data.** These are non-deterministic.
* **Pseudo random numbers** can be generated from a **strong mixing function. Mixing function** takes 2 or more random inputsbased on **CPU load, arrival times of packets etc.** Each bit of the output then depends on some non-linear function of all the bits in the inputs.
* **Cryptographic hashing functions and encryption algorithms** are examples of **strong mixing functions.**

## Key Storage

* Try not to store key on a medium connected to the network.
* Key should never be unencrypted outside the encryption device.
* Use a physical token to store the key (e.g. smart card) and protect the token with a PIN number.
* Card can be stolen therefore store one half on the key in the machine and one half on the card.

## Session Key Establishment

* **A symmetric key is more likely to be broken** as the more times it is used, the more times it is brought into the memory which can be hacked. And the more ciphertexts are produced using it.
* Thus, create a **symmetric key for one session** only which is known as a **session key**.
* Using session keys, prevents the long-term storage of possibly many symmetric keys.

### Key agreement

* **Both parties contribute to the derivation of the shared secret** so that not party can pre-determine the resulting value. This is namely the **Diffie-Hellman protocol. What is the problem with one party generating the secret key?**

### Key Transportation

* **Without public-key cipher**: Session keys are generated and distributed with the help of **a third party**. Known as **Needham-Schroeder protocol**.
* **With the use of a public key cipher: One party** creates a secret key (session key), and securely transfers it to the other party using the **recipient’s public key.**

### Entity and key authentication

* Make sure no other party could gain access to the key.
* Confirm that the other party knows about the key by producing a one-way hash of the key or, encrypting some known data (e.g. nonce) with the key.
* Key freshness: make sure the key is fresh i.e. not been used before.

### Key Freshness

* Check if the key has not been used before.

# Diffie-Hellman Protocol

* Key-agreement protocol
* Public key algorithm
* Allows two parties who have **never met before** to exchange messages in public and **collectively generate a key** that is private to them.
* Security is based on the difficulty of calculating **discrete logs** in the finite field.
* Given integers y and g and prime number n, compute x such that y = g^x mod n

## Initial Condition

* Alice and Bob, agree on two large integers **g** and **n.**
* G and n do not have to be a secret.
* Alice has a private key Xa and public key Ya.
* Bob has a private key Xb and public key Yb.

## Protocol

# 

* This protocol resists passive attacks because calculating a **discrete logarithm** is a **computationally hard problem**.

## Man-in-the-middle attack vulnerability

* The receiver of the other half of the key does not know who it is coming from, so this is vulnerable to a **man-in-the-middle-attack.**

# 

* Alice and Bob actually share a key with Eve, rather than with each other.
* But, if Alice and Bob do not share a key then how do they encrypt the message? I guess they cannot.

## Approach (1) solution

## Approach (2) solution



## Approach 1 VS Approach 2

# 

# Needham-Schroeder Protocol