

## Topic 8: Key Management

Understand crypto key management issues and their distribution methods

---

*Source: Stallings book, Chapter 14*

*Some slides are based on the slides by Lawrie Brown prepared  
for the text book*

## Overview

- Part 1
  - Key Management Issues
- Part 2
  - Symmetric Key Establishment: Diffie-Hellman (DH) algorithm/protocol
- Part 3
  - Symmetric Key Distribution using symmetric key encryption
- Part 4
  - Symmetric Key Distribution using public-key encryption
  - A summary and conclusion

## 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?
- For symmetrical ciphers - how to keep keys **secret**?
- For public-key ciphers - how to ensure public keys are **trust-worthy**?

## Key Management Issues - Keys spaces

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

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

## 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, therefore 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

- When people choose their keys, they generally choose poor ones.
  - Which of these two keys are better (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.
- Good keys are random numbers.
- What is a random number?
  - Given an integer,  $k > 0$ , and a sequence of numbers,  $n_1, n_2, \dots$ , an observer can not predict  $n_k$  even if all of  $n_1, \dots, n_{k-1}$  are known.

## Key Management Issues - Key generation

- Ordinary random number generation functions, e.g. `java.util.Random`, may not be 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, SHA and DES) are all 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

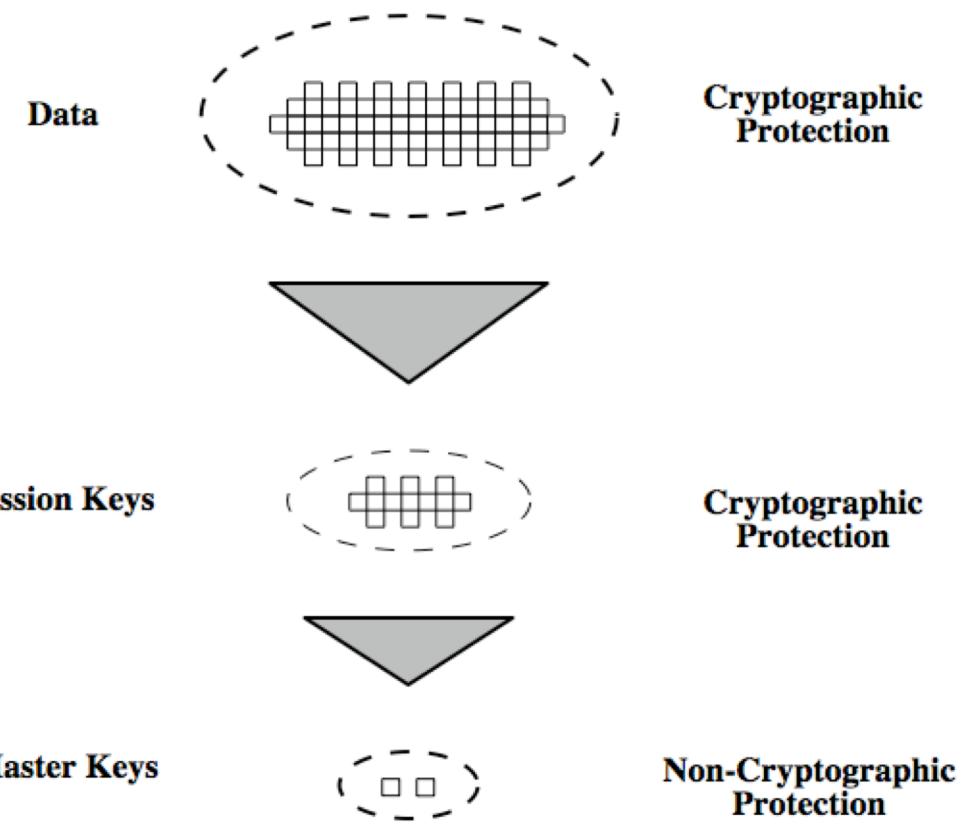
- Attackers may defeat access control mechanisms, so encrypt the file containing key
  - Ideally, a key should never appear unencrypted outside the encryption device.
  - Try not to store your key on a medium connected to the network.
- Key may be resident in memory, so attackers may be able to read if they could get access to the machine
  - 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 ( $k_1, k_2$ ):  $k_2 = k_1 \text{ xor } K$

## Part 2 Overview

- Symmetric Key (Shared Secret) Establishment:
  - Diffie-Hellman (DH) protocol

# Key Hierarchy

- Typically have a key hierarchy
  - Master key/secret (key encryption key)
    - used to establish/distribute session keys
  - Session key (data encryption key)
    - used to encrypt data/message
    - for one logical session only



## Session Keys

- More often a symmetric key is used, more likely it may be compromised.
- Generate and use a symmetric (secret) key for one session only  
→ **session key**.
- Using different session keys in different sessions can
  - limit available ciphertexts for cryptanalysis.
  - limit exposure (both in time period and amount of data) in an event of key compromise.
- To avoid long-term storage of a large number of secret keys, we only generate them when they are needed.

## Session Key Establishment

### □ Session key establishment solutions

#### ➤ Key agreement (exchange) protocols

- A shared secret (master or session secret) is derived by the parties as a function of information contributed by each, such that no party can predetermine the resulting value - Diffie-Hellman (DH) protocol.

#### ➤ Key transportation protocols

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

## Session Key Establishment

- There are other issues that should be considered

- **Key secrecy and entity/key authentication**

- Assurance: no other party (outsiders - apart from the entities involved) could gain access to the established session key.
    - The session key is established with the intended entities.
    - Key confirmation: asking the other entity (possibly unidentified) to demonstrate that he has the knowledge of the key by
      - producing a one-way hash value of the key; or
      - encrypting some known data (e.g. nonce) with the key.

- **Key freshness**

- Assurance: the key is fresh, i.e. not used before.

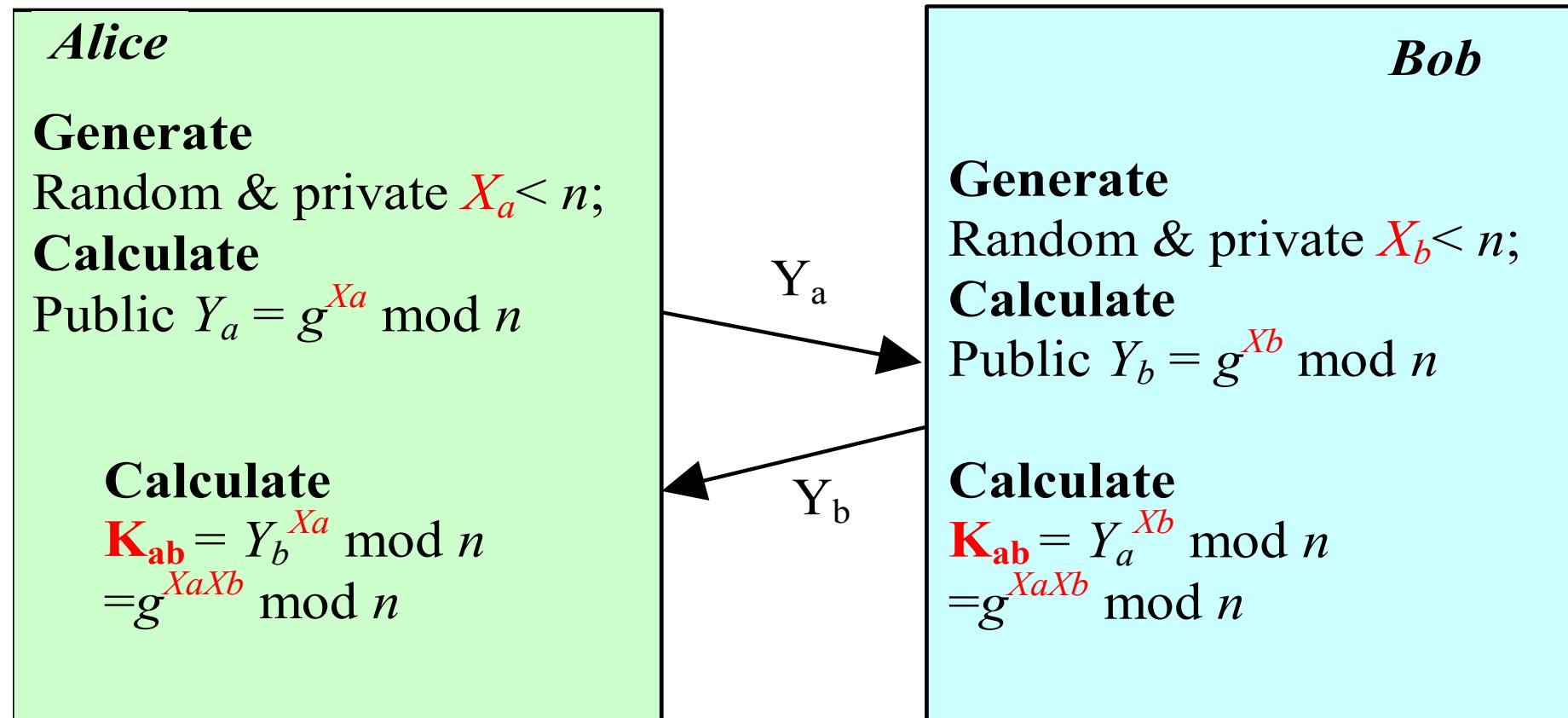
## Diffie-Hellman Algorithm/Protocol

- DH was the 1st public-key algorithm ever invented - back in 1976.
- **DH key exchange** protocol allows two parties who have never met before to exchange messages **in public** and collectively generate a key that is private to them, and none of the parties could predetermine the key.
- Its security is based on the **difficulty of calculating discrete logarithms** in a finite field.
  - Given integers  $y$  and  $g$  and prime number  $n$ , compute  $x$  such that  $y = g^x \text{ mod } n$ .
  - This is computationally infeasible if  $n$  is sufficiently large.

## Diffie-Hellman Algorithm/Protocol

- Assuming two parties, *Alice* and *Bob*, take part in the exchange.
- Initial condition
  - *Alice* and *Bob* agree on two large integers,  $\mathbf{g}$  and  $\mathbf{n}$ ;
  - $n$  - prime number that serves as the modulus.
  - $g$  - random number that serves as the basis, with  $1 < g < n$ .
  - $g$  and  $n$  do not have to be secret.
- Definition
  - *Alice* has private key  $\mathbf{X}_a$ , and public key  $Y_a$ .
  - *Bob* has private key  $\mathbf{X}_b$ , and public key  $Y_b$ .

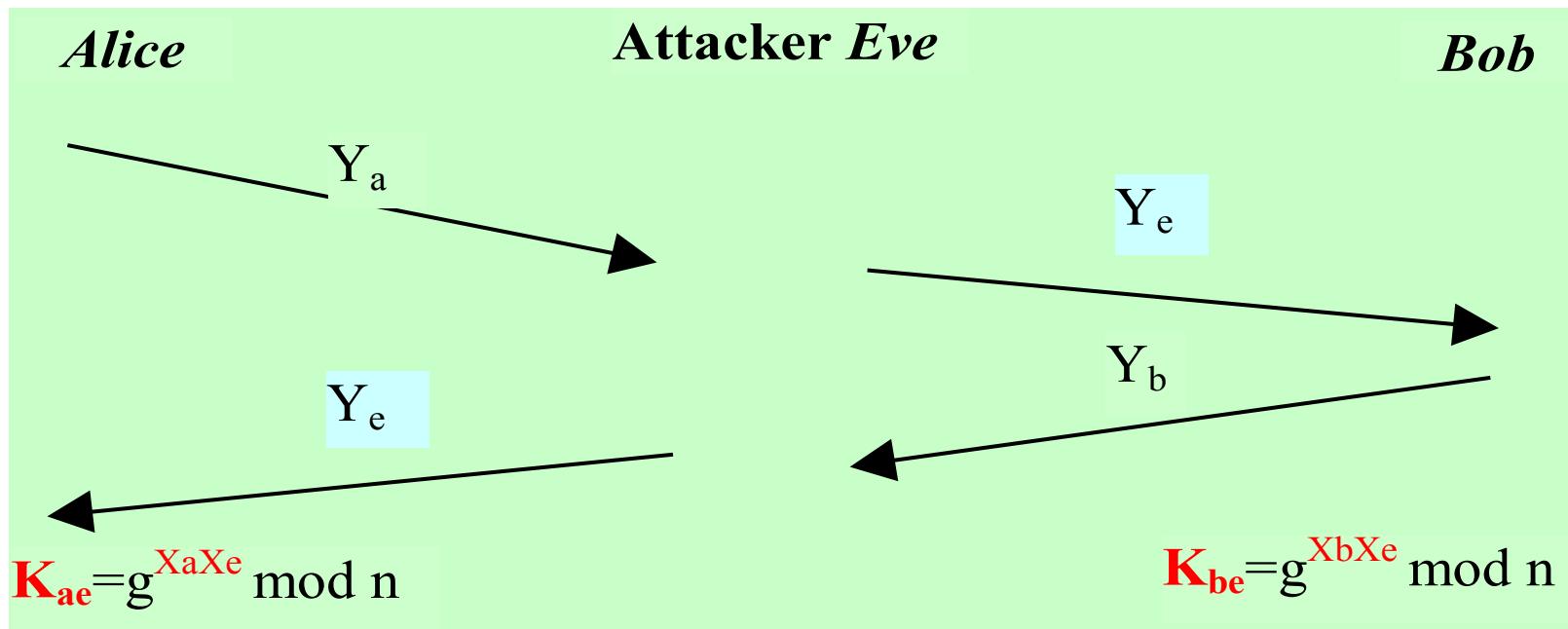
# Diffie-Hellman Algorithm/Protocol



## Diffie-Hellman Protocol

- It resists passive attacks such as eavesdropper, as calculating a discrete logarithm is a computationally hard problem.
- There is **one problem** - neither party knows who it shares the secret with! So it is vulnerable to active, **man-in-the-middle attacks**, as to be illustrated shortly.

## Diffie-Hellman Protocol - Man-in-the-middle attack



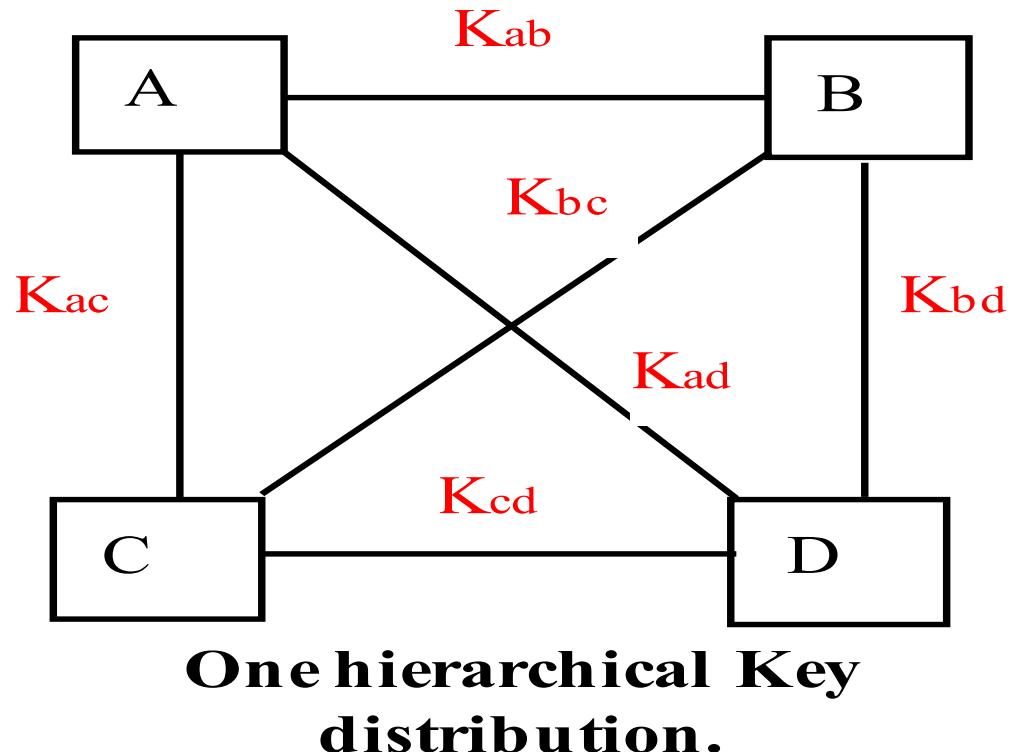
- *Alice (Bob)* thought she shares a key with *Bob (Alice)*, but actually with *Eve*.
- So the attacker *Eve* can intercept and read any messages encrypted **without** been detected by *Alice* and *Bob* .

## Part 3 Overview

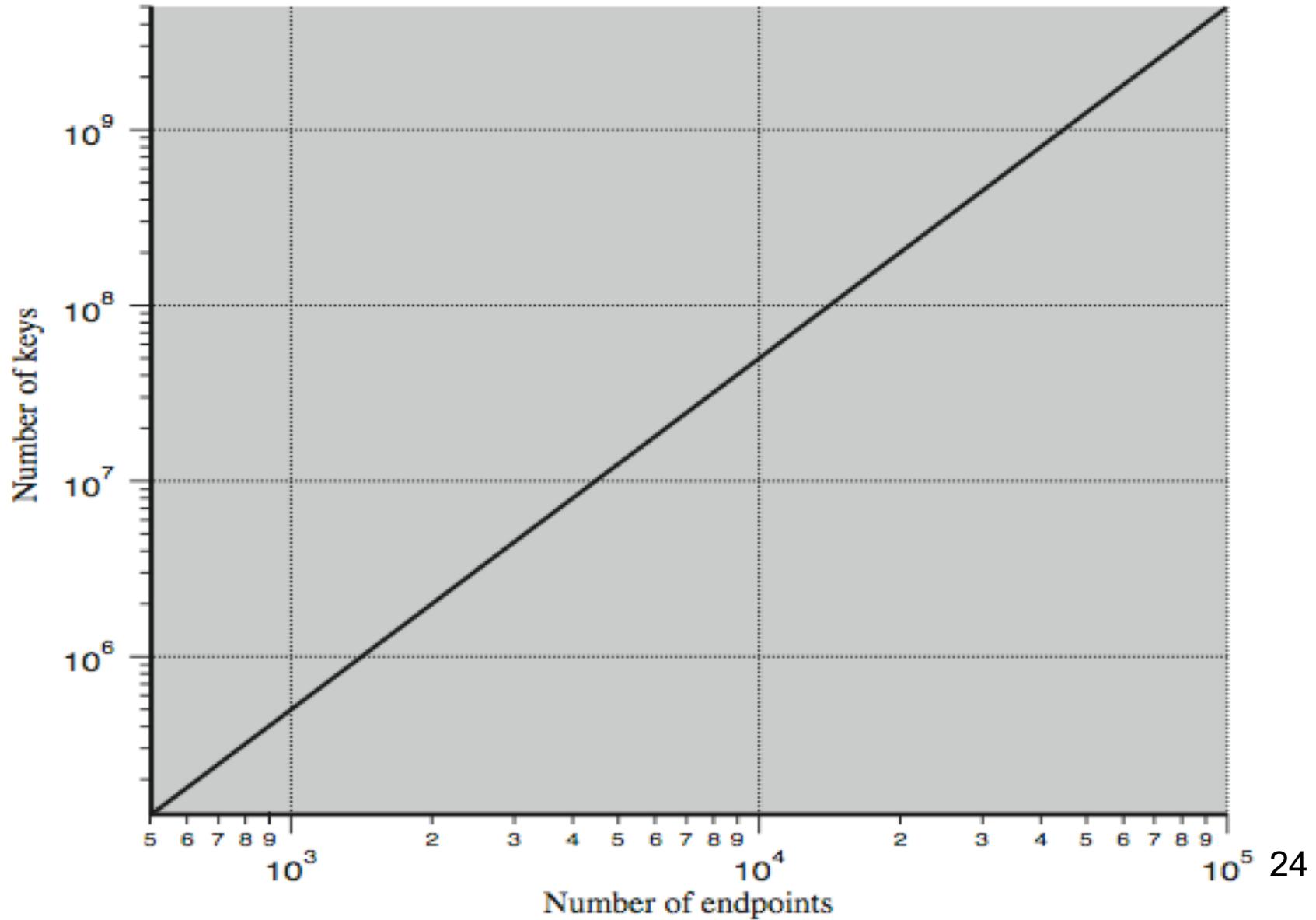
- Symmetric key distribution using symmetric key encryption - Needham-Schroeder Protocol.
- This protocol is widely used in single sign on (SSO) solutions, e.g. window domain authentication, Kerberos.

## Distribution without use of PKC - Approach One

- *Approach One*: Given  $n$  users (parties/nodes) to communicate to each other, the system needs  $n(n-1)/2$  keys.
- As  $n$  increases, the number of keys becomes untenable for everyone.
- The  $n^2$  problem!

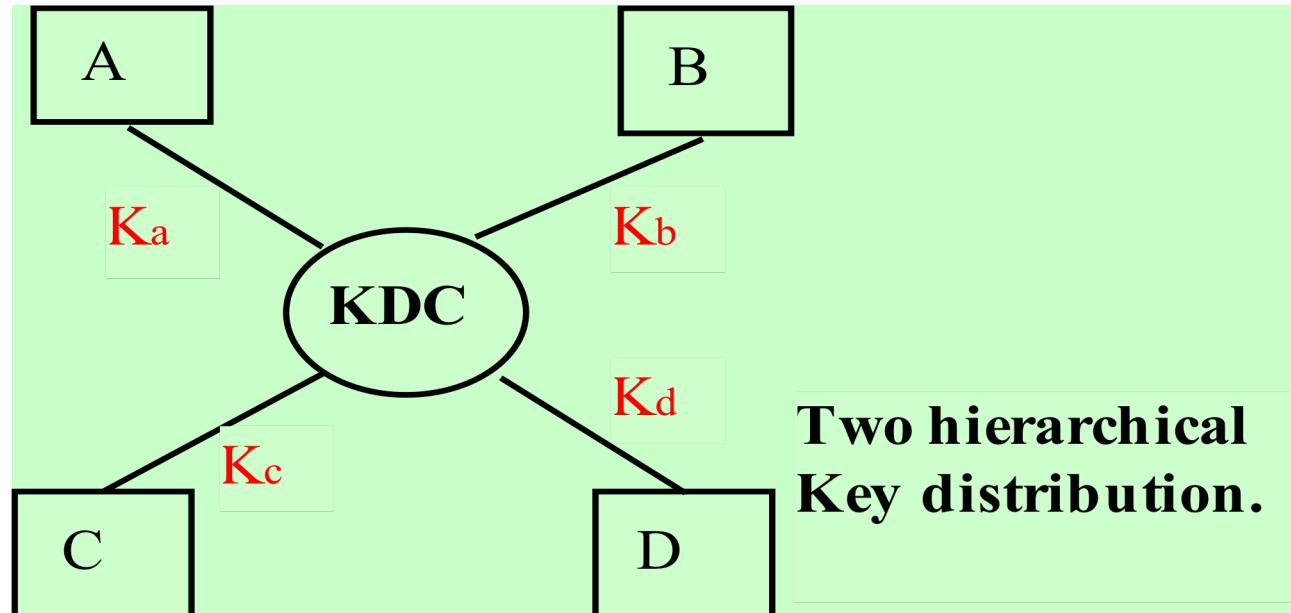


# Distribution without use of PKC - Scalability problem



## Distribution without use of PKC - Approach Two

- *Approach Two:* use a key distribution centre (*KDC*) or security server.
  - A key hierarchy, e.g. two hierarchical approach - *master keys* (*long-term keys*) and *session keys* (*valid just for one session*).



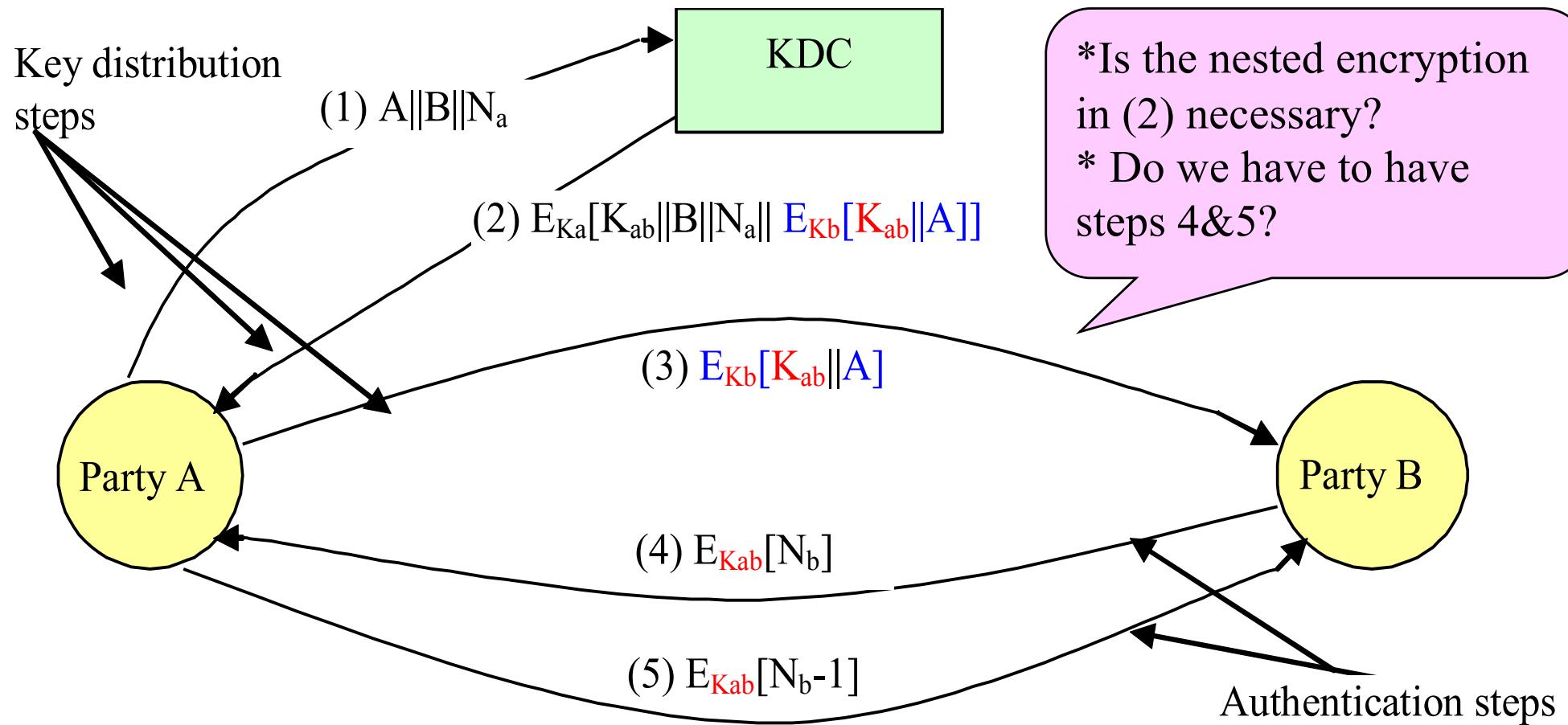
## Distribution without use of PKC – Approach Two

- A unique **master key**, shared between a pair of user/*KDC*, is for session key distribution.
- A session key is to secure a particular session.
- **Benefit** of using Approach Two
  - Reduces the scale of the problem - reduces the  $n^2$  problem to an  $n$  problem, thus making the system more scalable.
- **But:**
  - The need to trust the intermediaries - KDC.
    - KDC has enough information to impersonate anyone to anyone. If it is compromised, all the resources in the system are vulnerable.
  - KDC is a single point of failure.
  - KDC may be a performance bottleneck.

## Distribution without use of PKC - Needham-Schroeder Protocol

- The Needham-Schroeder is a [key distribution protocol](#).
- It uses the Approach two. That is:
  - both parties,  $A$  and  $B$ , shares a secret key with the KDC,  $K_a$  and  $K_b$ ;
  - $A$  and  $B$  wishes to establish a secure communication channel, i.e. establish a shared one-time session key  $K_{ab}$ , for use between  $A$  and  $B$  in this session.
- $N_a, N_b$  are nonces (random challenges), generated by  $A$  and  $B$  respectively, to keep messages fresh.

## Distribution without use of PKC - Needham-Schroeder Protocol



## Distribution without use of PKC - Needham-Schroeder Protocol

- (1)  $A$  sends a request to  $KDC$  for a session key to establish a secure channel with  $B$ .
- (2)  $KDC$  generate a random number  $K_{ab}$ , and replies with the response containing
  - session key  $K_{ab}$ .
  - original request enables  $A$  matching the response with the request.
  - an item (the session key and  $A$ 's identity) which only  $B$  can view.
- (3)  $A$  forwards the item to  $B$ .  
*At this point, the session key is securely delivered to  $A$  and  $B$ , and they may begin secure communication.*
- (4)  $B$  sends a nonce  $N_b$  to  $A$  encrypted using the new session key.
- (5)  $A$  responds with  $N_b-1$ .

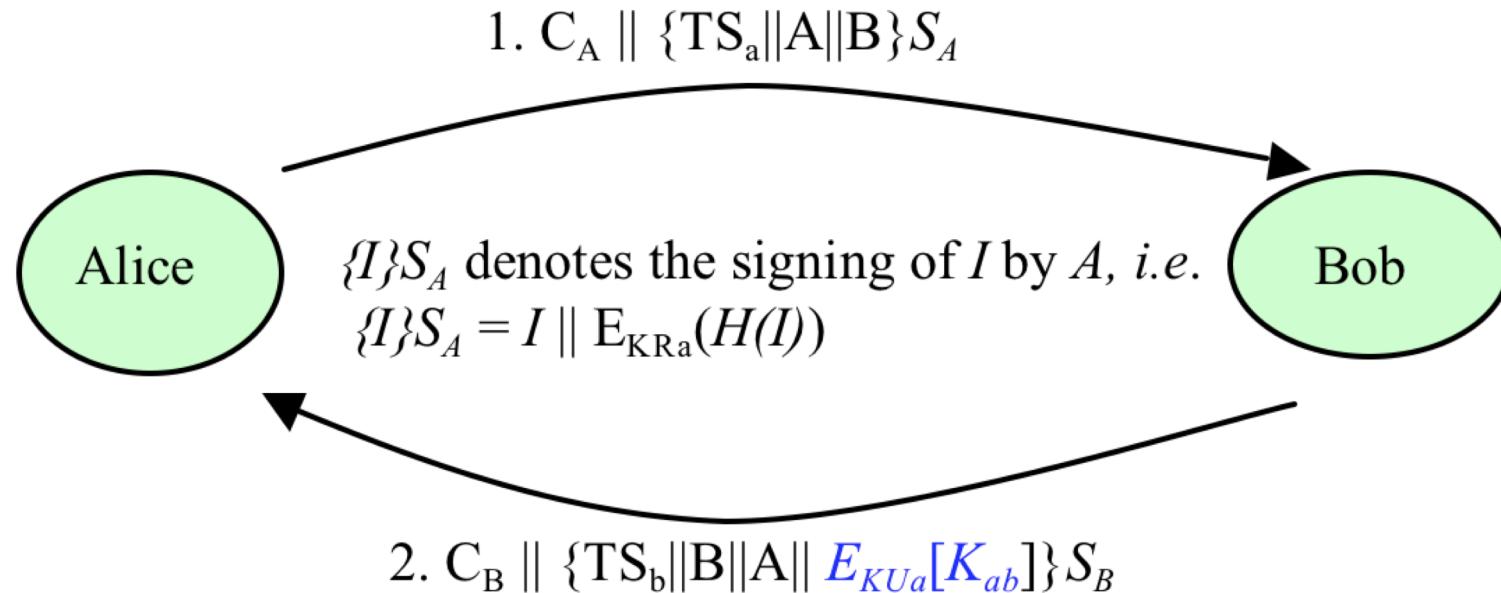
*Steps (4) & (5) assure  $B$  that the message received in (3) was not a replay, i.e. to authenticate  $A$ .*

## Part 4 Overview

- Symmetric Key Distribution using public-key encryption
- A summary and conclusion

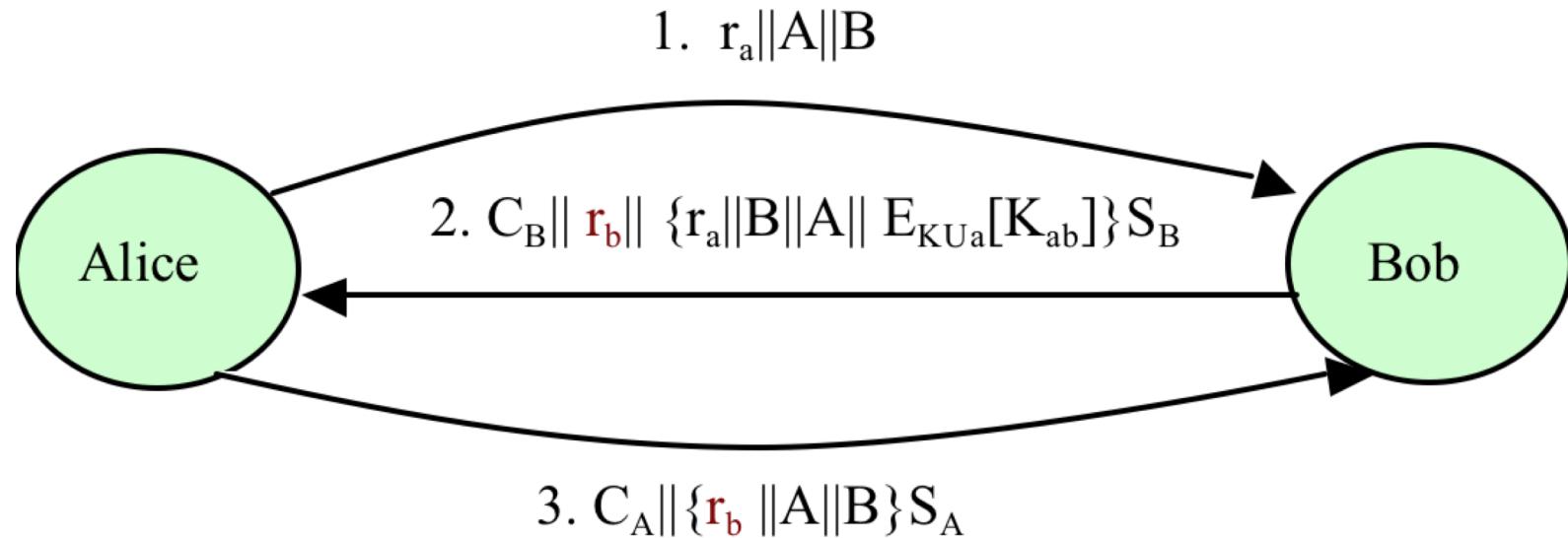
## Distribution using PKC – Two passes

- Secret key distribution with mutual authentication using public key cryptosystem + **timestamps**.



## Distribution using PKC (2) - Three passes

- Symmetrical key distribution with mutual authentication using digital signatures + **nonces** (random numbers).



## A summary of secret key establishment protocols

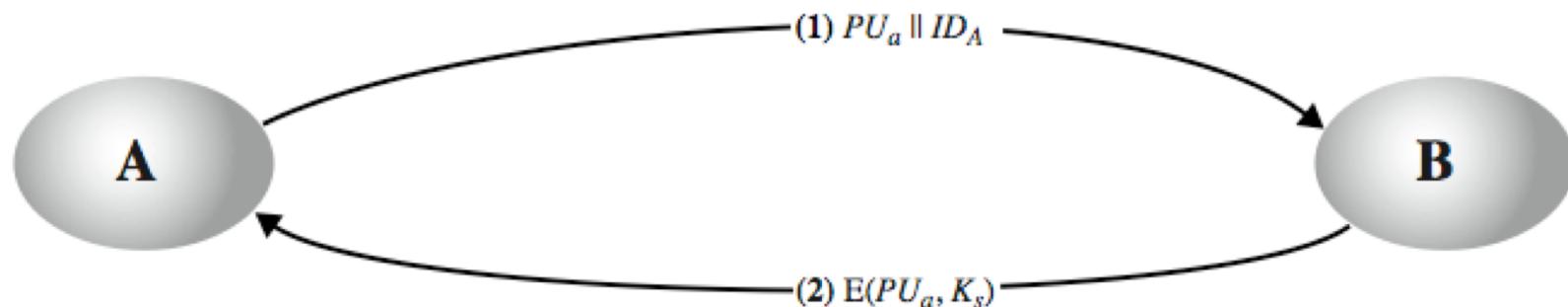
Protocols	ThirdParty	Timestamps	EntityAuth	messages
Diffie-Hellman	No	No	None	2
Needham-Schroeder protocol	KDC (online)	No	Symmetric encryption	5
X.509 (2 pass)	CA (offline)	Yes	mutual	2
X.509 (3 pass)	CA (offline)	No, but with nonce	mutual	3

## Exercise Question – E8.1

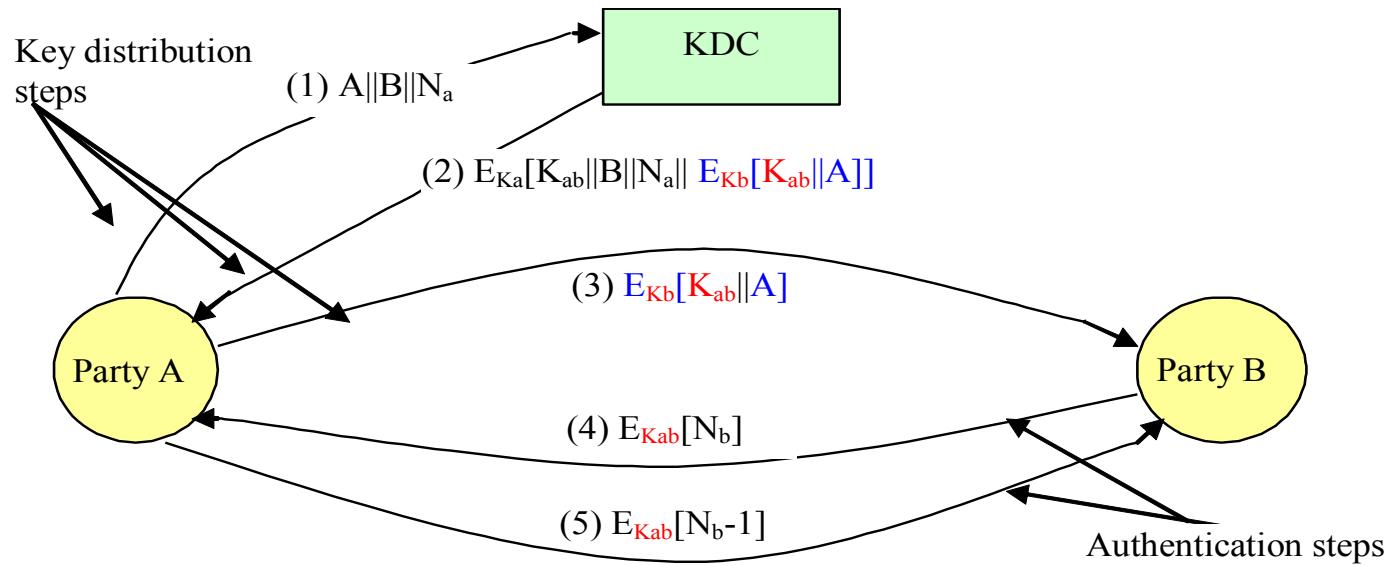
Assuming that Alice is to send a message,  $M$ , to Bob.  $M$  is encrypted with a shared key established using the DH protocol. Explain whether Eve could access this message  $M$ . If so, explain how, and propose a solution to address this vulnerability.

## Exercise Question – E8.2

- The following is an extremely simple protocol proposed for symmetric key distribution. It is assumed that A and B has never met before (or there is no key established prior to this communication).
  - Identify as many problems/flaws as you can.
  - Modify the protocol to fix the problems/flaws you have identified.



## Exercise Question – E8.3



This is the Needham-Schroeder protocol. Answer the following questions:

- What are the benefits for A to forward the session key to B (i.e. step 3), rather than letting KDC to directly send the session key to B?
- TRY to identify two application areas of the Needham-Schroeder protocol and to elaborate the benefits of using the Needham-Schroeder protocol in these application areas.

## Conclusions

- Key management encompasses a number of critical issues to the effective use of cryptosystems.
- A number of protocols exist to support symmetrical key distribution and agreement.
  - Key transport protocols
    - One party creates or otherwise obtains a secret value, and securely transfers it to the other party.
  - Key agreement protocols
    - A shared secret is derived by the parties using information contributed by each, such that no party can predetermine the resulting value.
- Key agreement/distribution protocols can be vulnerable to security attacks, such as the man-in-the-middle and replay attacks, so they should be used with care.