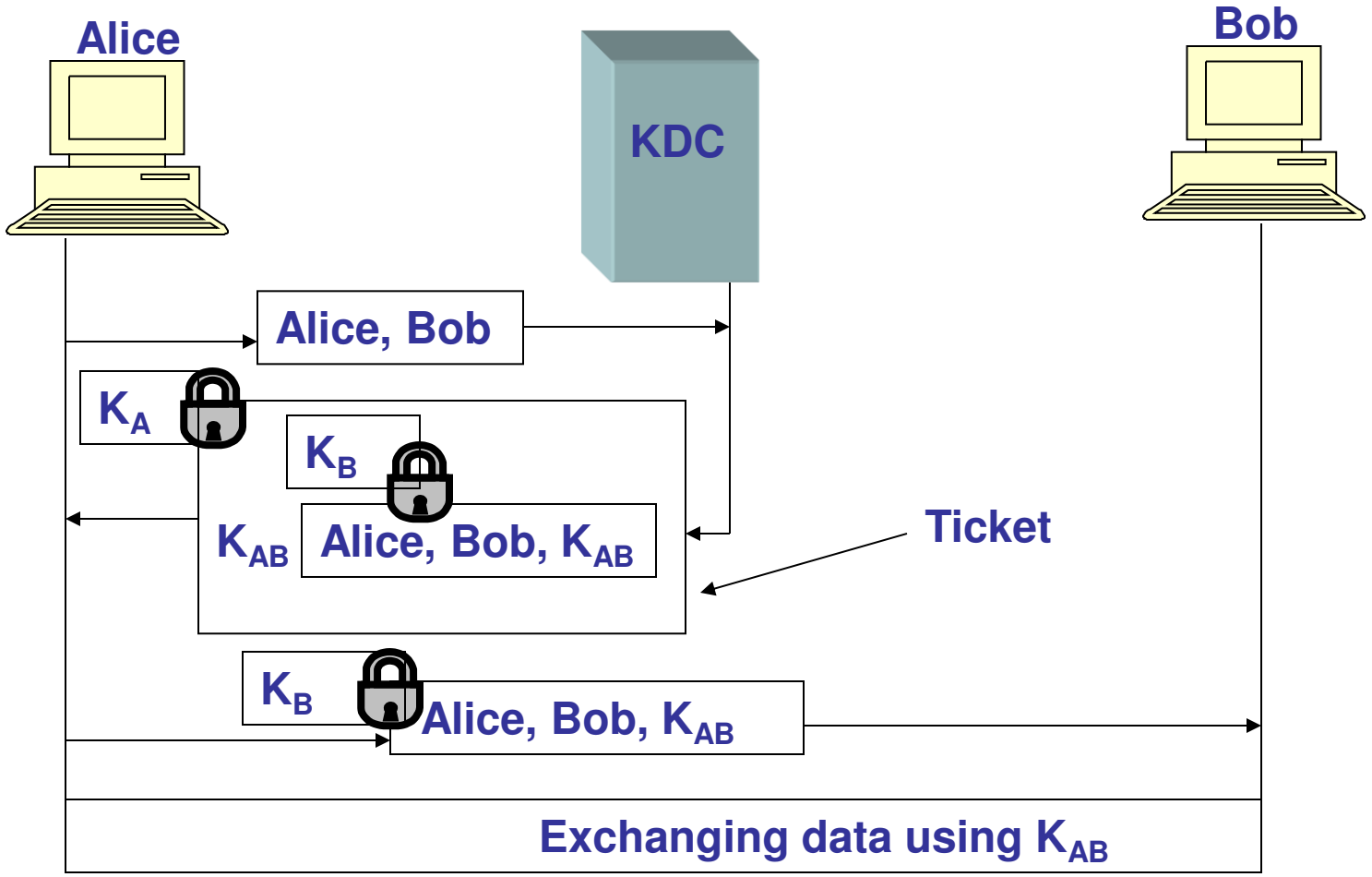
Key Management 2

Amit Konar
Math and Computer Sc., UMSL

Why Key Distribution Center?

The flaw in the previous protocol is the sending of R1 and R2 as plaintexts, which can be intercepted by any intruder. Any private correspondence between two parties should be encrypted using a symmetric key. But the two parties need to have a symmetric key before they establish communication. This can be realized by a **trusted third party**. This is the idea of behind a **key distribution center**.

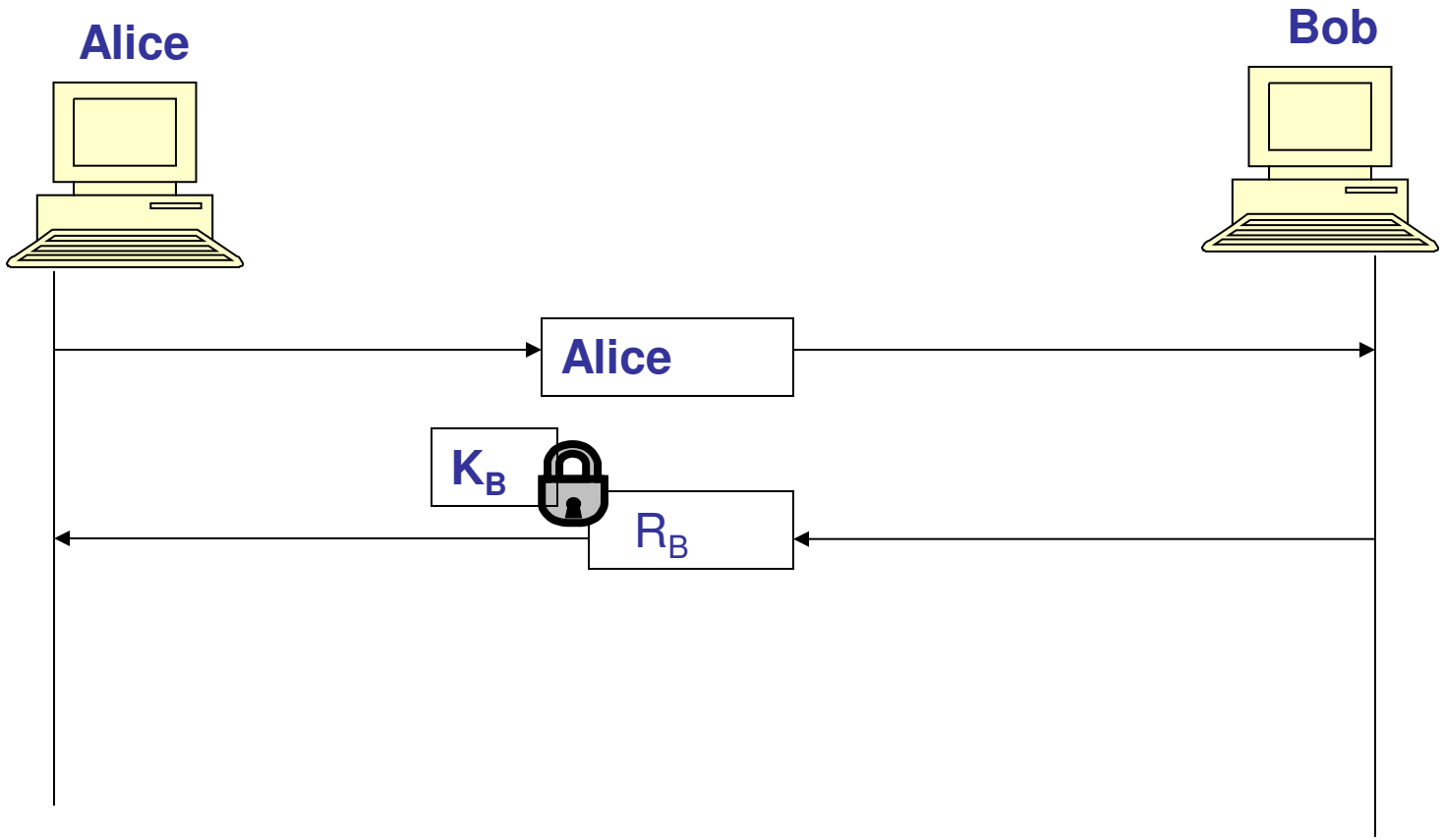
Alice and Bob manages two secure keys K_A and K_B, how do they get **Session key K_{AB}?**



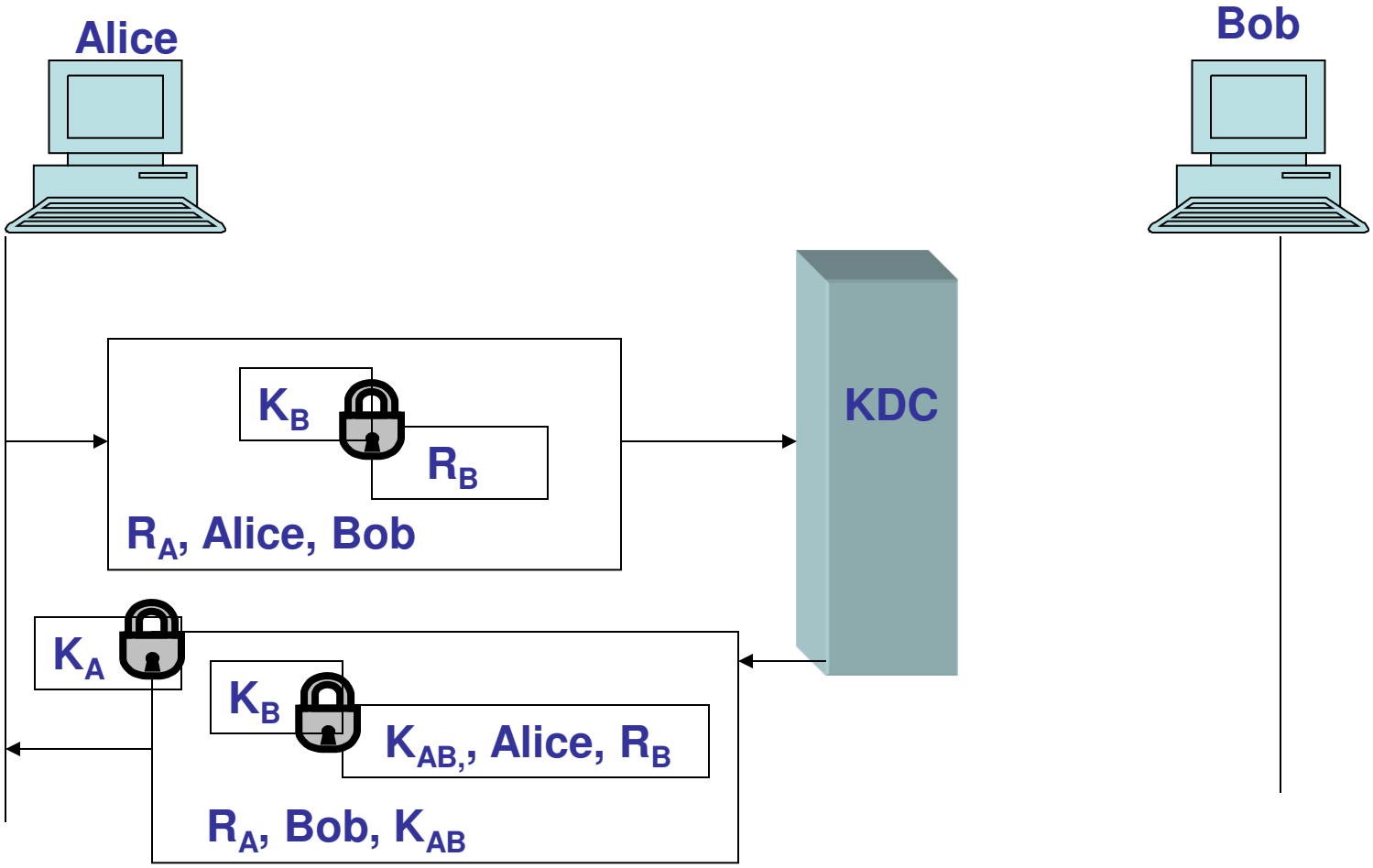
Can Eve use replay attack?

Yes, She can save the message in step 3 (i.e. encrypted Alice, Bob, K_{AB}) and and data and replay them later for some personal benefits, such as claiming payment twice, once normal and once for replay.

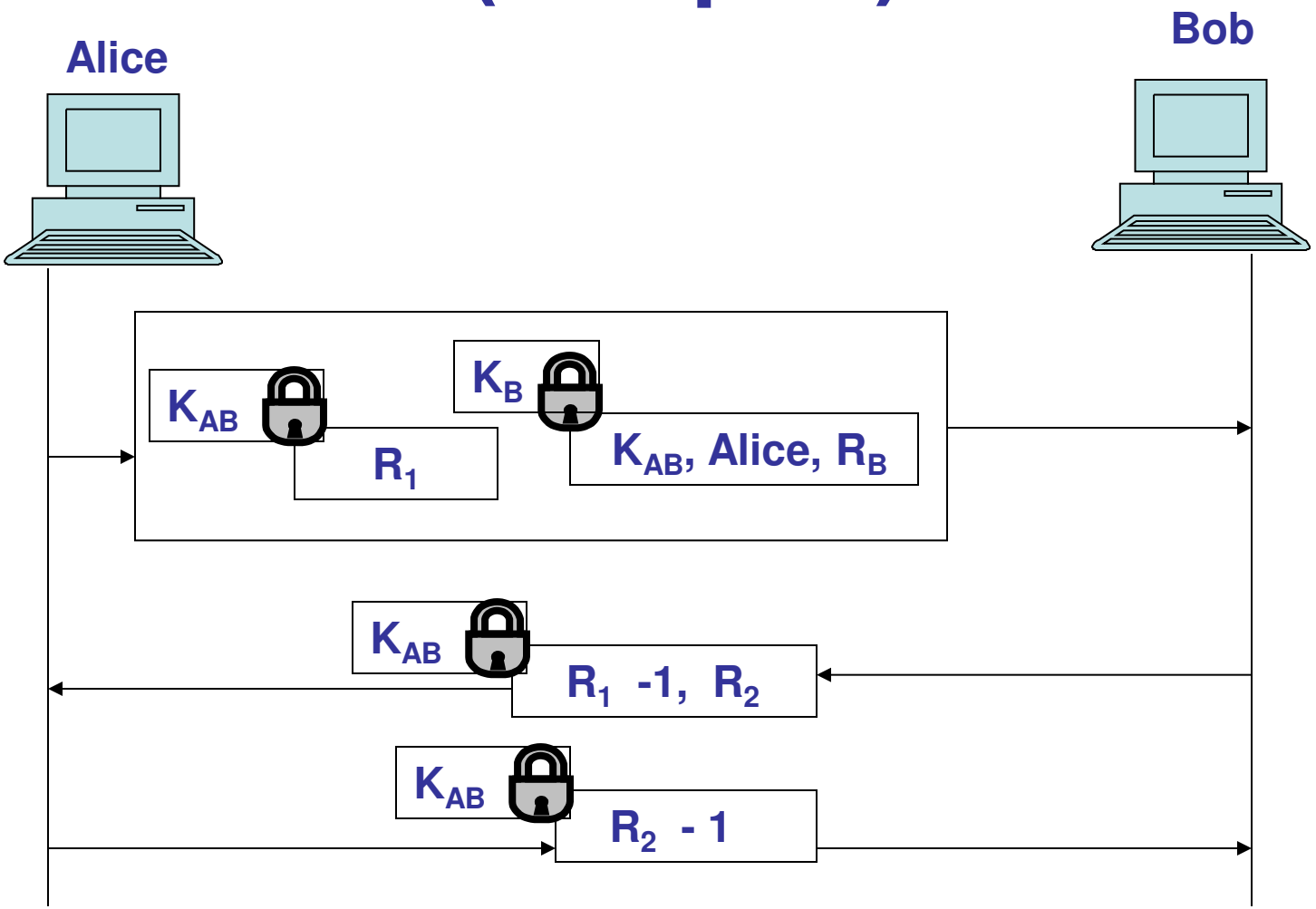
Needham-Schroeder Protocol



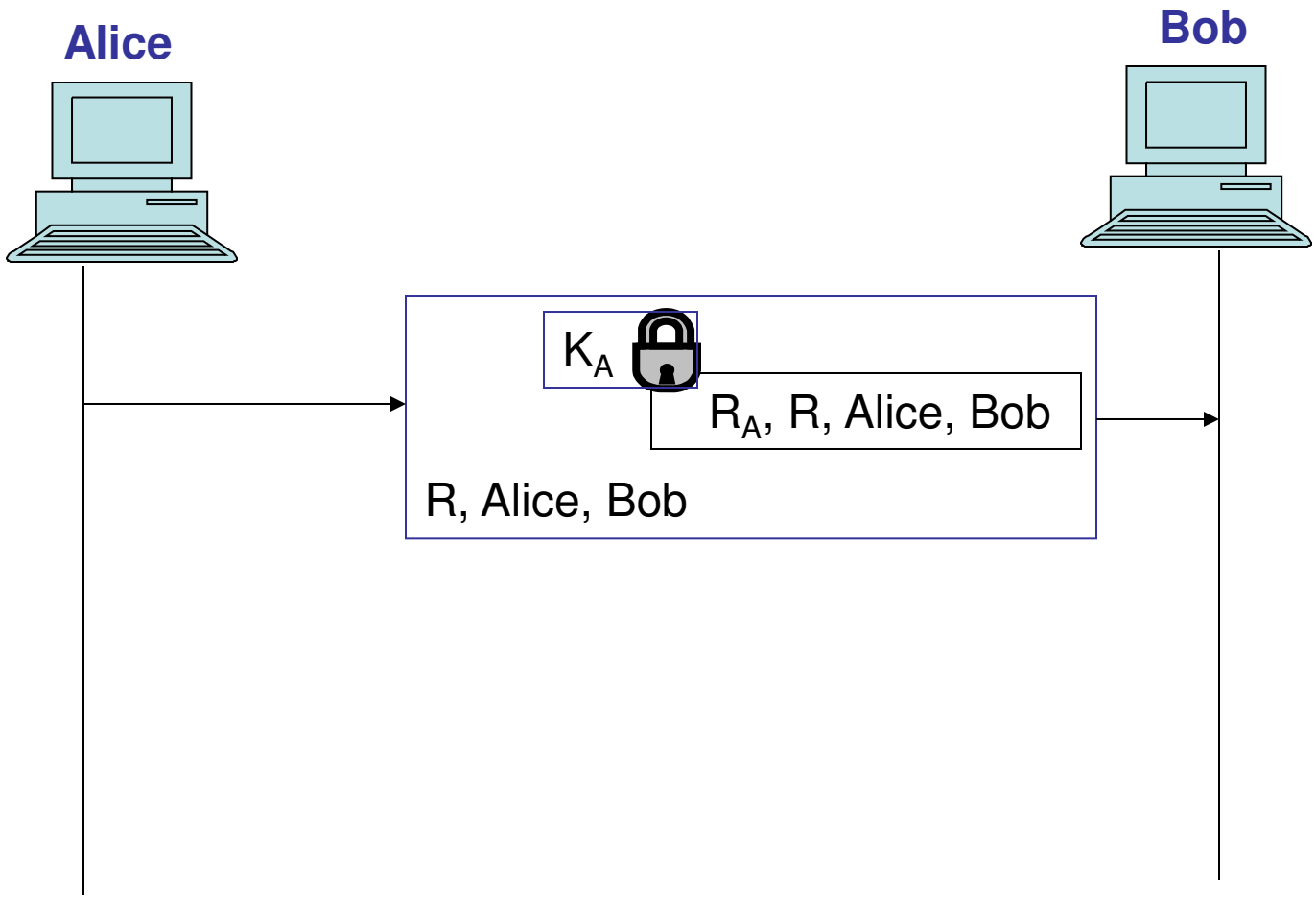
Needham-Schroeder Protocol (Contd.)



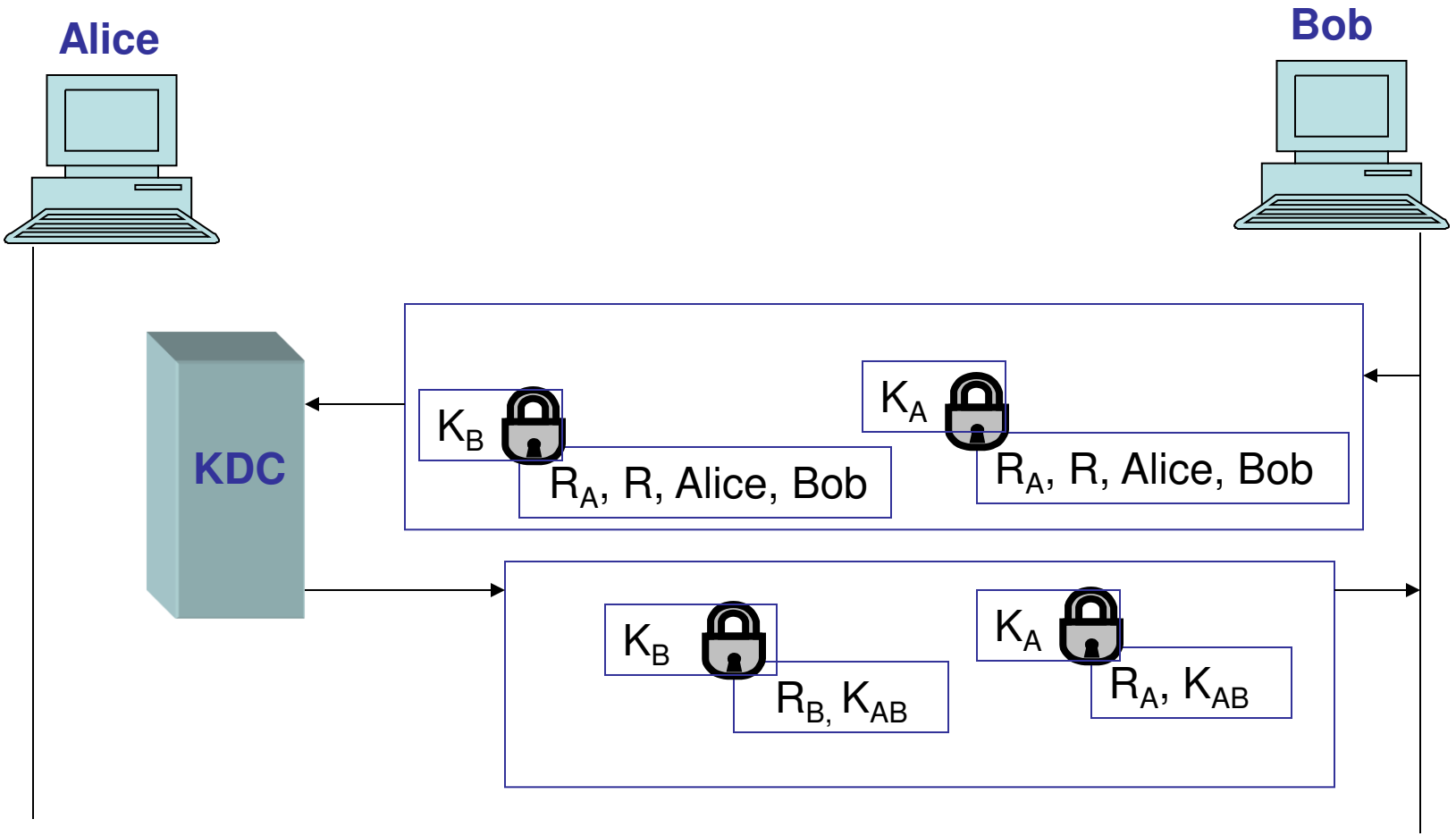
Needham-Schroeder Protocol (last part)



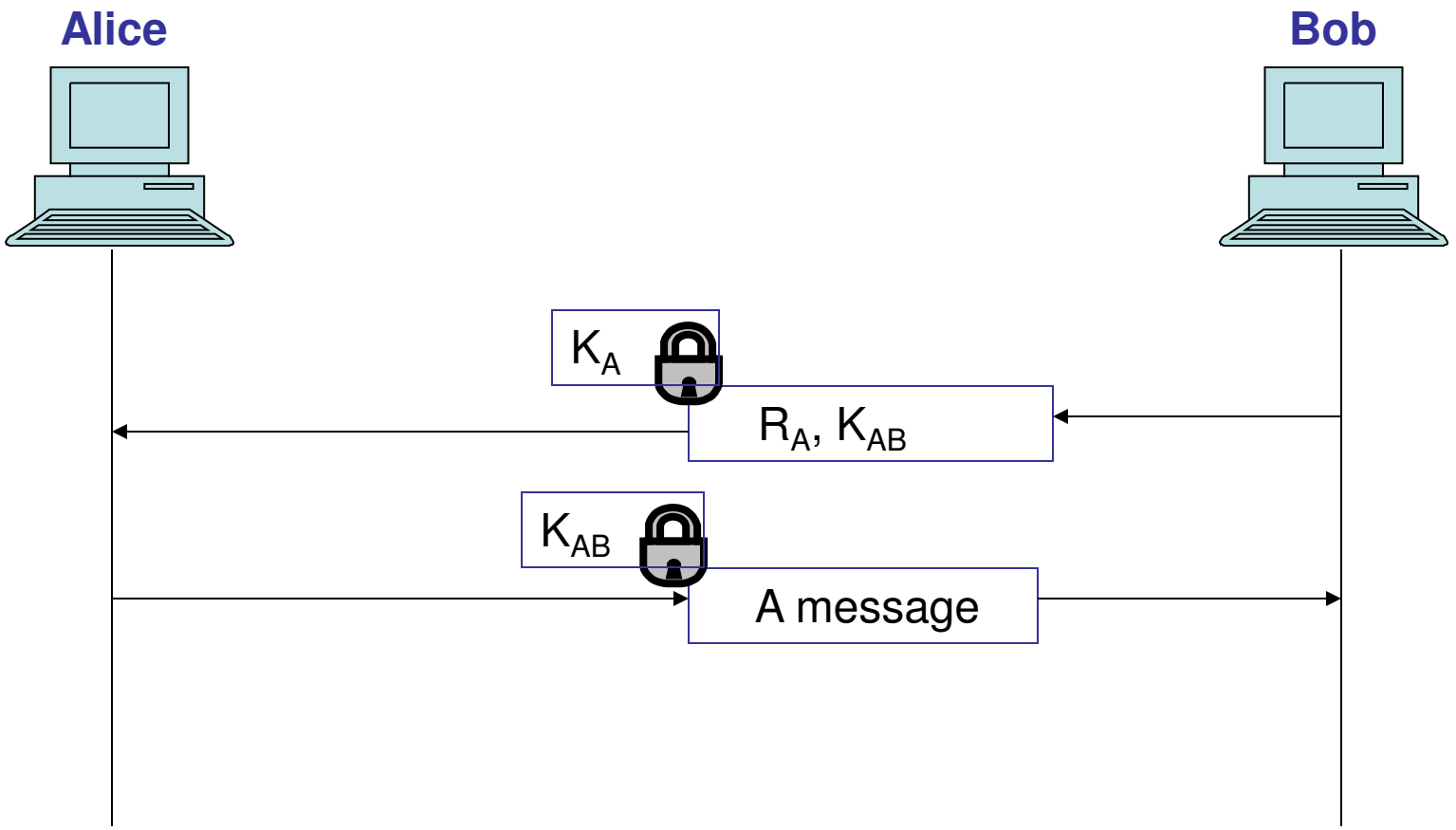
Otway-Rees Protocol



Otway-Rees Protocol (Contd.)



Otway-Rees Protocol (Contd.)



Public-Key Certification

In classical cryptography, two people need to know a symmetric shared key for secure communication. In public key cryptography, people need to know only the public key of the recipient. The public key is known to everyone. In public key cryptography, everyone shields his private key and advertises his/her public key.

Problem in advertising public key

Example: Suppose Bob sends his public key to Alice, Eve may intercept it, and send her (Eve's) own public key to Alice. Alice then may encrypt a message with this key, may send it to Bob. Eve again intercepts it, decrypts the message with her private key, and thus knows what alice has sent to Bob. Eve can even put her public key online and claim that this is Bob's public key.

How does Bob solve his problem?

Bob wants two things: 1) people should know his public key, and 2) no one should accept a public key forged as Bob's. To ensure this Bob can go to a **Certifying Authority (CA)** that binds a public key to an entity and issue a certificate. The CA does the following:

1) CA checks Bob's ID (using picture ID),

2) asks Bob his public key and writes it on the certificate,

3) creates a message digest from the certificate,

4) encrypts the digest with CA's private key (authentication)

Bob uploads the certificate and the encrypted message digest. The encrypted digest may be decrypted with CA's public key. A disgst is created from the plaintext and compared with the decrypted one.

If no mismatch, the certificate is valid, and no imposter has posed as Bob.

Why do we need X. 509 standardization?

The use of a Certification Authority has solved the problem of public-key fraud. But each certificate may have a different format. If Alice wants to use a program to automatically download different certificates and digests belonging to different people, the program may not be able to do so. The public key may be in the first line in one certificate, and in the third line in another. So, we need to standardize it. ITU has devised a protocol called X. 509.

Some fields of X. 509 and their meanings

Version- Version no. of X. 509

Serial No. – The unique identifier used by the CA

Signature- The certificate signature

Issuer- The name of the CA defined by X. 509

Validity period- start and end period that certificate is valid

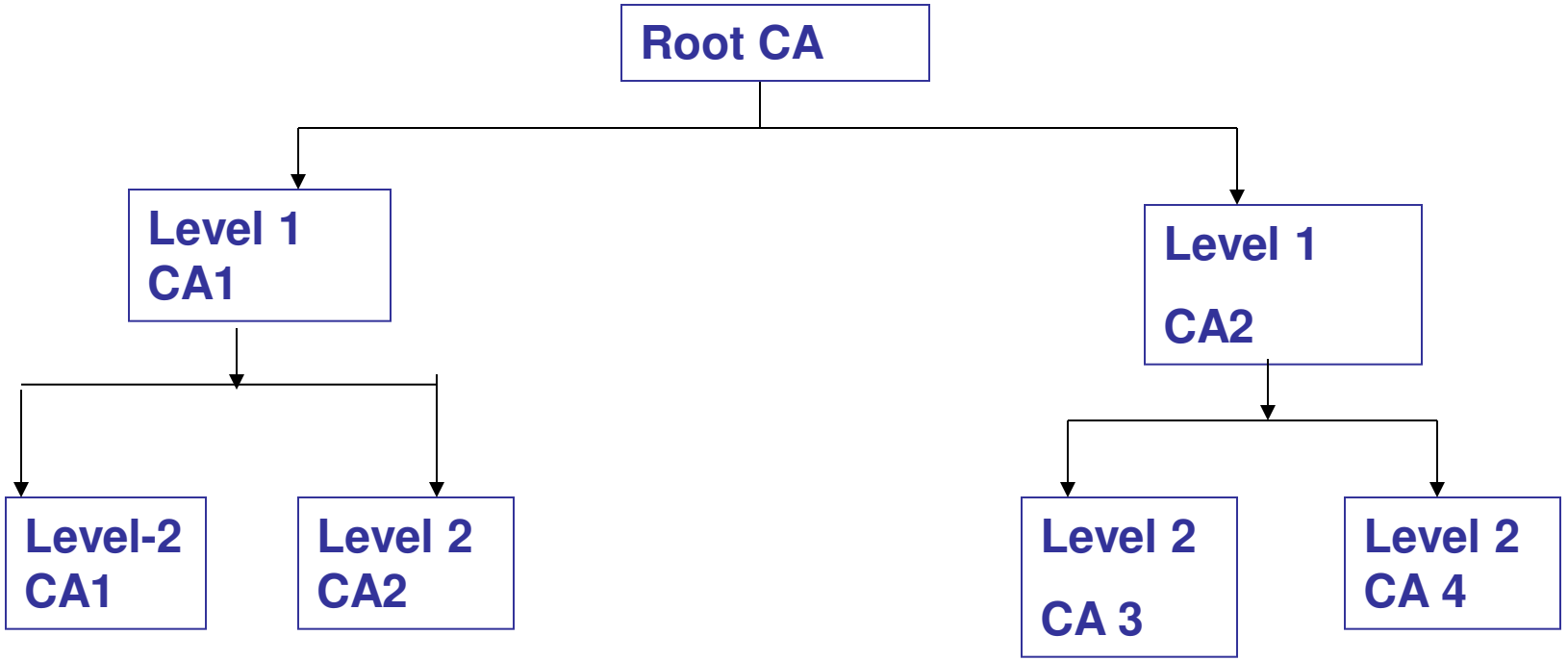
Subject name- the entity whose public key is being certified

Public key- the subject public key and the algorithm that uses it.

Public Key Infrastructure (PKI)

Normally servers acting as Certifying Authorities are organized in a hierarchical manner. The root CA is most trustworthy. Suppose, Alice needs Bob's certificate. She may find a CA somewhere to issue a certificate. But Alice may not trust it, and she can ask the next higher CA to certify the original Ca. The inquiry may go all the way to the root.

Public Key Infrastructure Hierarchy



The root CA can certify the performances of its next level Cas. Lev 1 CAs operate in large geographical areas, while level 2 Cas operate in smaller geographical areas.

Kerberos (History)

Kerberos is an **authentication protocol and at the same time a KDC**.

"Kerberos" is named after the three-headed dog in Greek mythology that guards the gates of Hades.

It was first designed at MIT. It has gone several versions. We discuss version 4, the most popular one and explain the difference between Version 4 and 5.

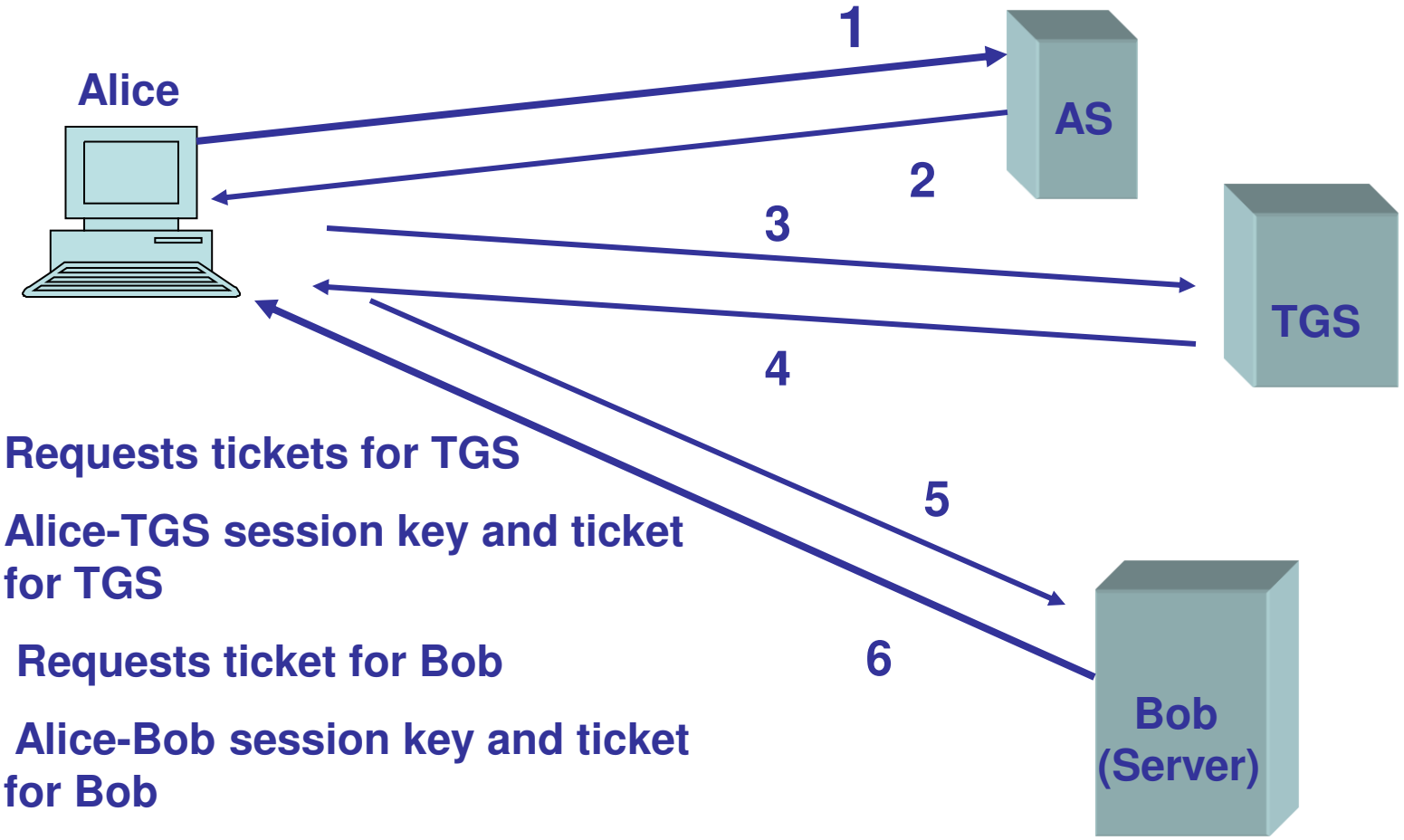
Servers used in Kerberos

Three servers are involved in Kerberos:

1) an authentication server (AS),
2) a ticket-granting server (TGS),
3) a real data server that provides services to others.

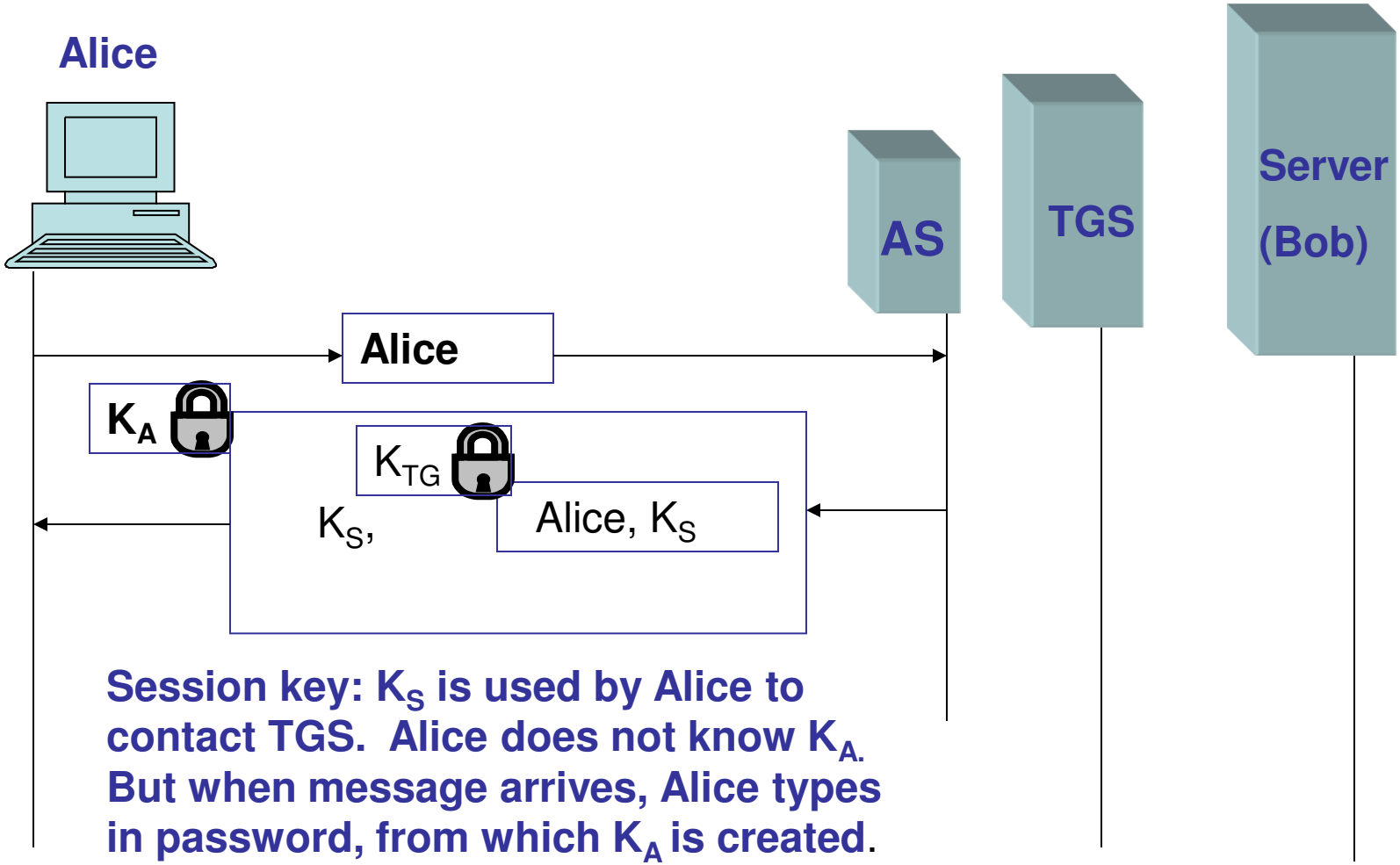
In our examples to follow, Bob is the real server and Alice is the user requesting service.

Kerberos' servers



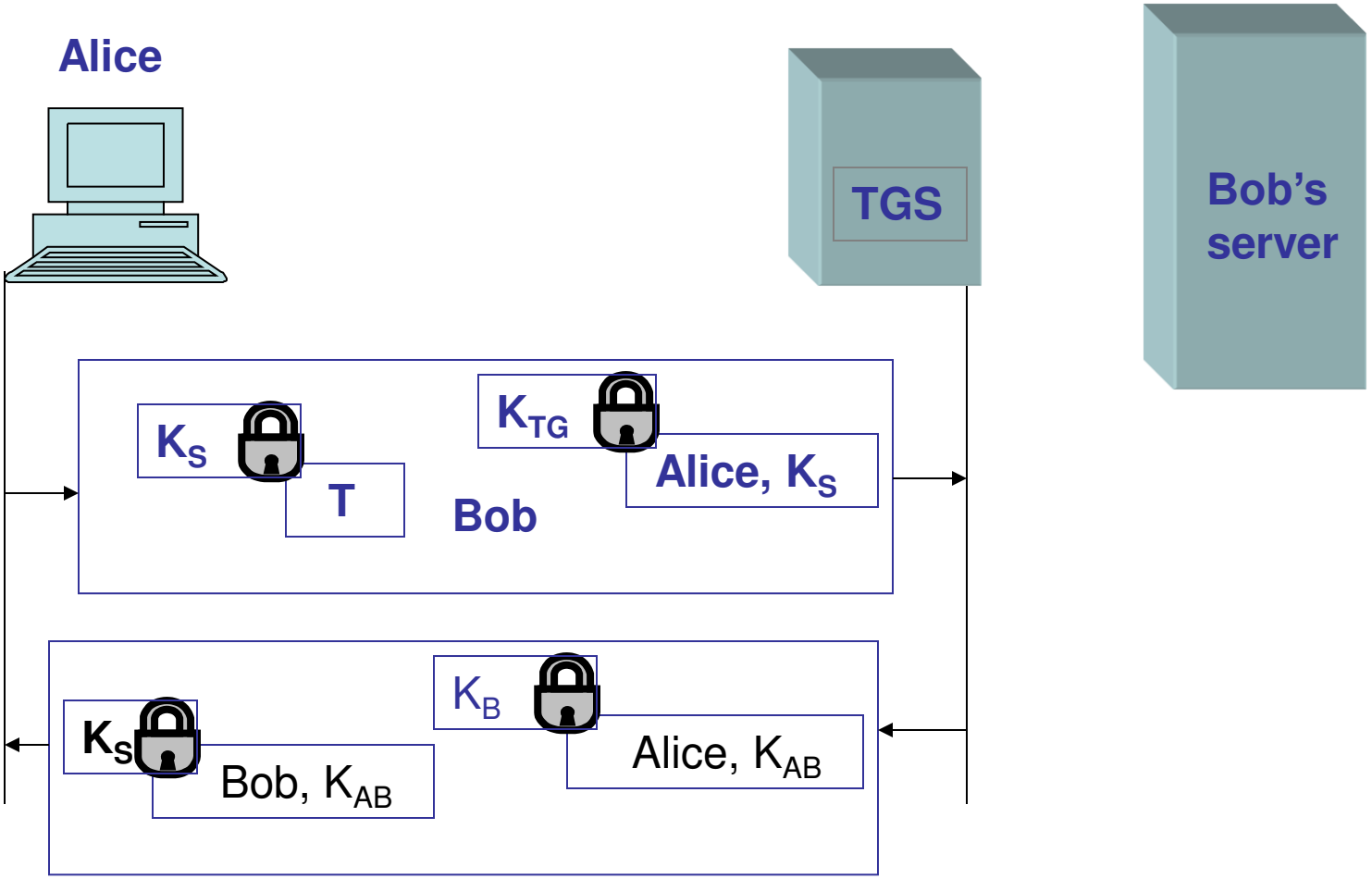
1. Requests tickets for TGS
2. Alice-TGS session key and ticket for TGS
3. Requests ticket for Bob
4. Alice-Bob session key and ticket for Bob
5. Request service
6. Provide service.

A client process (Alice) receives service from a process running on the real server Bob in 6 steps



Session key: K_S is used by Alice to contact TGS. Alice does not know K_A. But when message arrives, Alice types in password, from which K_A is created.

Kerberos Example (Contd.)



Kerberos Example (Contd.)