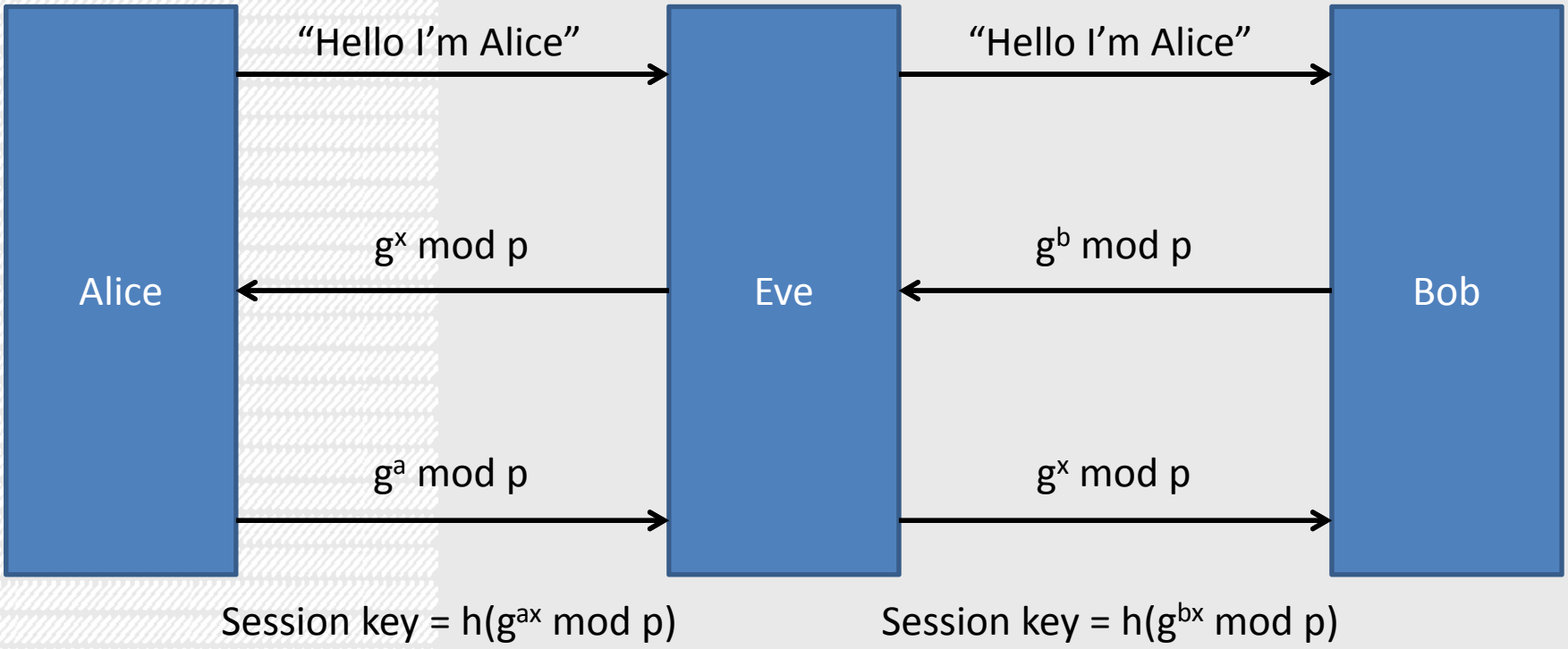
ELEC5616 COMPUTER & NETWORK SECURITY

Lecture 13:
Cryptographic Protocols I

MAN IN THE MIDDLE

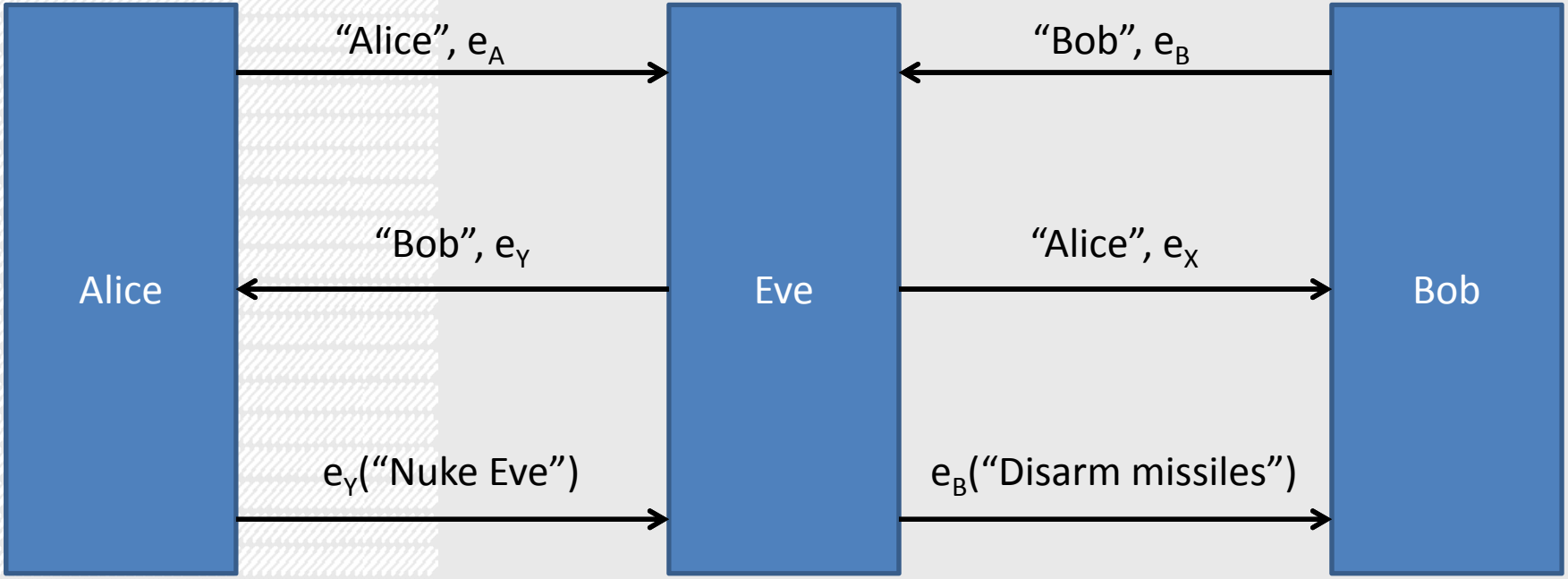
Problem with Diffie-Hellman



Eve owns the channel!

MAN IN THE MIDDLE

Problem with Key Exchange using Public Key Crypto



Eve owns the channel!

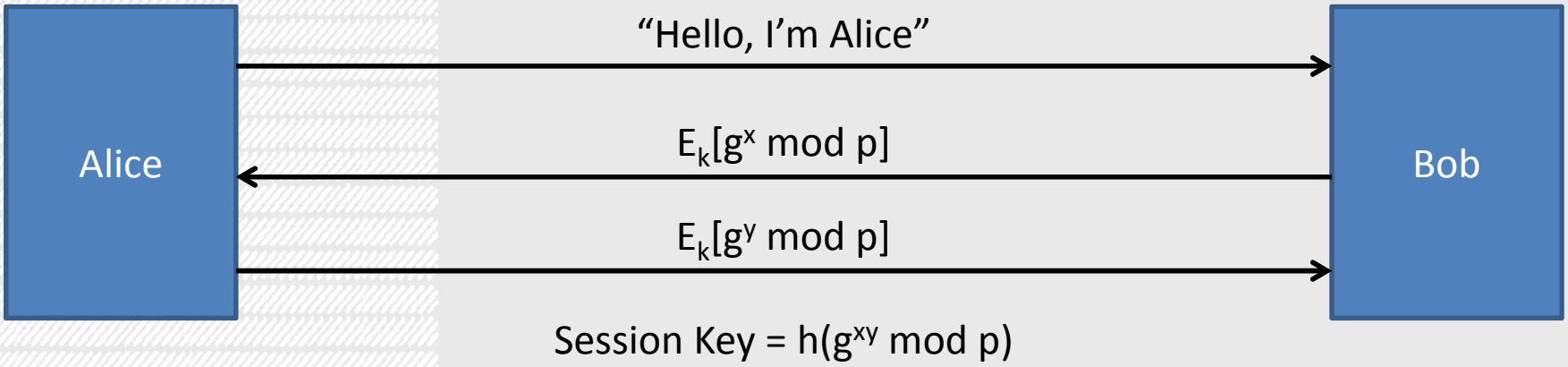
SESSION HIJACKING

Problem

Eve owns the channel!

EKE – ENCRYPTED KEY EXCHANGE

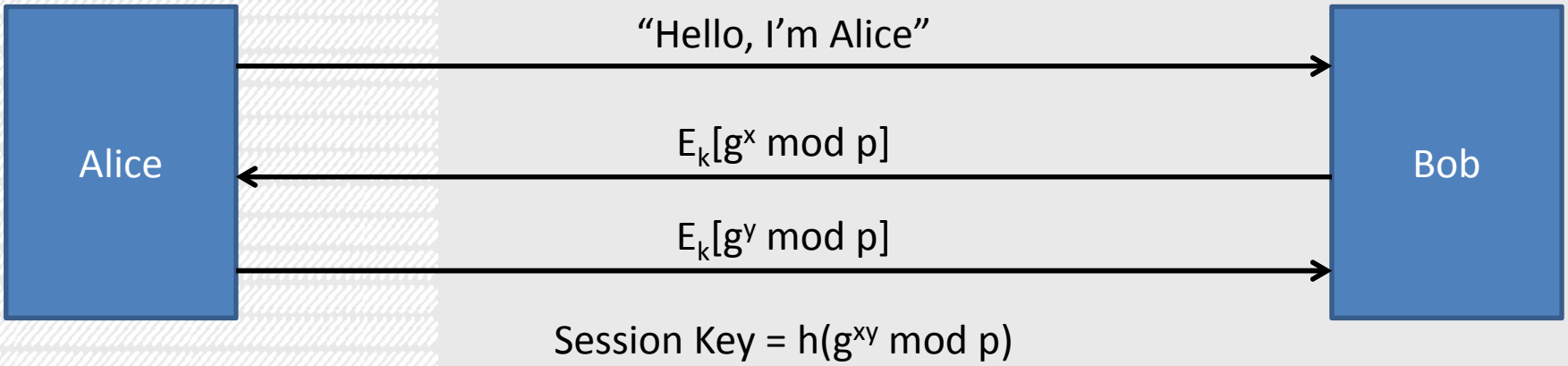
Problem



- A (possibly low entropy) shared **k** used to form high entropy shared secret $h(\mathbf{g^{xy}})$
- Prevents eavesdropping
- Prevents an active attack on Diffie-Hellman
- Forward secrecy
- What's the obvious problem?

EKE – ENCRYPTED KEY EXCHANGE

Problem



Public key cryptography would work well here except that we need a way for Alice to reliably obtain Bob's public key without directly contacting Bob...

DEFINITIONS

A protocol is said to have **perfect forward secrecy** if disclosure of long term keys does not compromise past (short term) session keys.

A protocol is vulnerable to a **known-key attack** if disclosure of past session keys allows an attacker to compromise future session keys (including actively impersonating)

NEEDHAM-SCHROEDER PROTOCOL (KEY EXCHANGE WITH TTP)

Alice and Bob want to setup a session key for communications.
We will denote encryption of m by key k as $\{m\}_k$
All parties share a key with Trent, a trusted third party.

Alice → Trent: "A", "B", r_A
Alice asks Trent to talk to Bob (r_A is a nonce)

Trent → Alice: $\{rA, "B", K_{AB}, \{K_{AB}, "A"\}K_{BT}\}K_{AT}$
Trent sends Alice a session key and ticket to give Bob

Alice → Bob: $\{K_{AB}, "A"\}K_{BT}$, $\{r_{A2}\}K_{AB}$
Alice sends to Bob the ticket (session key)

Bob → Alice: $\{r_{A2}-1, r_B\}K_{AB}$
Bob challenges Alice (rB is a nonce)

Alice → Bob: $\{r_B - 1\}K_{AB}$
–Alice responds

NEEDHAM-SCHROEDER PROTOCOL (KEY EXCHANGE WITH TTP)

Question: What happens if Eve gets a hold of K_{AT} somehow?

Alice → Trent: "A", "B", r_A

Trent → Alice: $\{r_A$, "B", K_{AB}, $\{K_{AB}$, "A"$\}K_{BT}\}K_{AT}$

Alice → Bob: $\{K_{AB}$, "A"$\}K_{BT}$, $\{r_{A2}\}K_{AB}$

Bob → Alice: $\{r_{A2}$-1, $r_B\}K_{AB}$

Alice → Bob: $\{r_B$ - 1$\}K_{AB}$

NEEDHAM-SCHROEDER PROTOCOL (KEY EXCHANGE WITH TTP)

Problem:

Bob has no guarantee K_{AB} is fresh (new)

– Old session keys are valuable as they do not expire

If Eve manages to get K_{AT} she can read all of Alice's messages and impersonate her to everyone else (e.g. Carol).

Alice informs everyone she has been issued a key for to revoke it.

Since Alice does not know if Eve impersonated her to someone she has never talked to (e.g. Carol), she has to get Trent to tell everyone "stop using all shared keys with Alice"

Thus **key revocation** is a **major problem**.

Needham-Shroeder's problem is that it assumes all users of the system are good guys and the goal is to keep bad guys from getting in ("eggshell model")

– This is the 'old school' model of security. It has been realised today that the most likely threats come from within (knowledgeable insiders).

PUBLIC KEY MANAGEMENT USING CERTIFICATION AUTHORITIES

A public key certificate binds a public key to its owner:

1. Alice sends her public key to the CA (Trent)
2. The CA produces a certificate for Alice
3. Alice sends her public key and certificate to Bob
4. Bob verifies the certificate using Alice's public key
5. Bob sends encrypted messages to Alice using the key

$$\mathbf{sign_{CA}[X.500: name, organisation, address, public key, expires \ldots]}$$

Everyone must be able to verify the CA's public key

– shipped with browsers, OS, published in the Australian

The CA is a trusted party

– it has the power to forge keys / signatures.

An example is the X.509 certificate for X.500 directory services

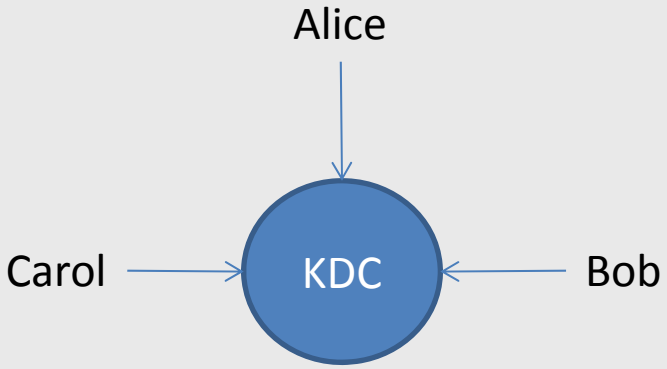
TRUST MODELS

Symmetric Keys

– TTP must be online (used every session)

– TTP is a juicy target (knows passwords)

– No forward secrecy



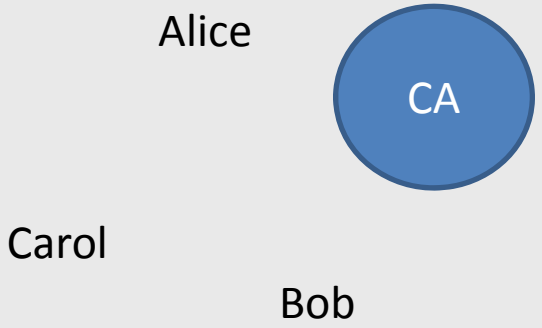
Asymmetric Keys

– TTP is offline (only used in key generation)

– TTP only knows public keys

– TTP has forward secrecy

– Not as fast (e.g. SSL/TLS, PGP, ...)



PUBLIC KEY CERTIFICATE GENERATION

Alice generates a public/private key pair

Alice sends the public key to the CA

The CA challenges Alice to see if she knows the private key

The CA generates a certificate and sends it to Alice

Note:

- The CA *never learns Alice's private key*
  - Important for forward secrecy

CERTIFICATE REVOCATION

Alice's certificate may need to be revoked

– Her private key is stolen

– She changes jobs

This is a major problem and is done in limited ways:

– Through users requiring daily certification-validation information
(slow, cumbersome)

– Use expiration date field

– Use of a certificate revocation list (CRL) which is circulated
(like bad credit cards)

REFERENCES

Handbook of Applied Cryptography

– read § 12 – 12.3.2 (ii)