Digital Signatures

Digital Signatures

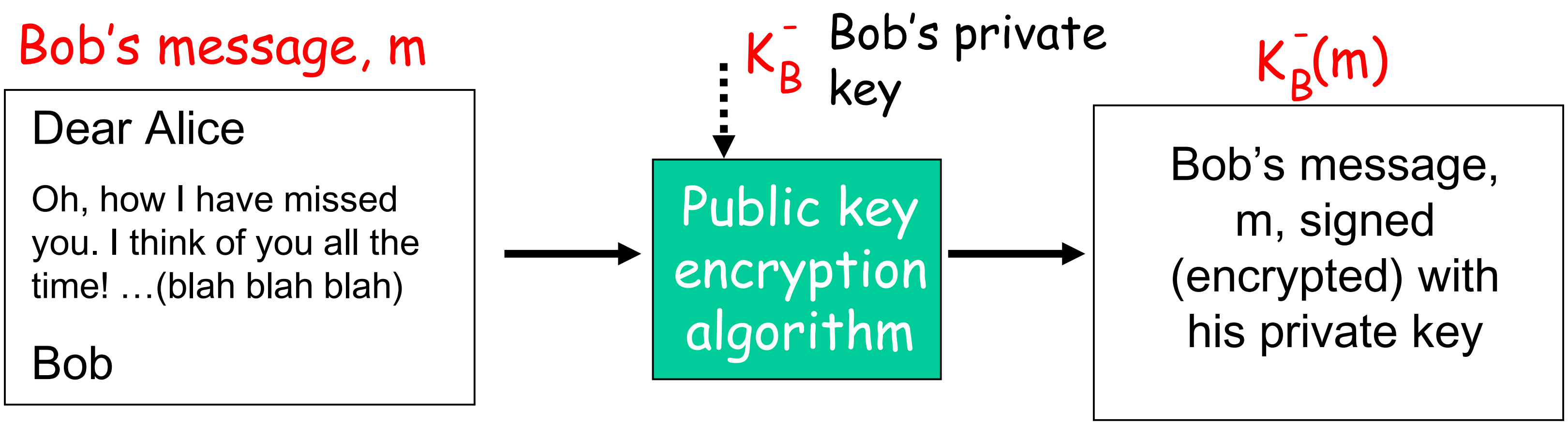
Cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- verifiable, nonforgeable: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital Signatures

Simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Digital Signatures (more)

- Suppose Alice receives msg m, digital signature $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

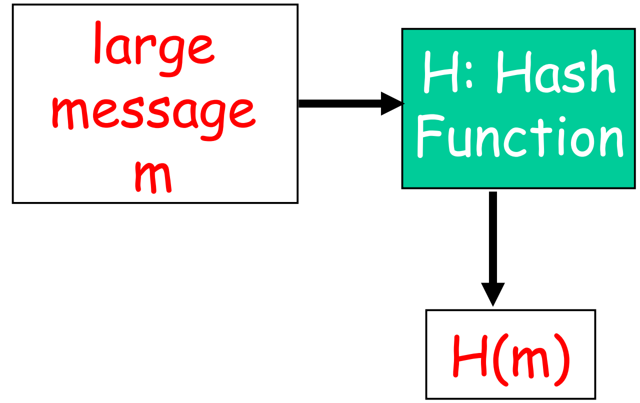
Alice thus verifies that:

- ✓ Bob signed m.
- ✓ No one else signed m.
- ✓ Bob signed m and not m'.

Non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Message Digests



Computationally expensive to public-key-encrypt long messages

Goal: fixed-length, easy-to-compute digital "fingerprint"

- apply hash function H to *m*, get fixed size message digest, *H(m)*.

Hash function properties:

- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest x, computationally infeasible to find m such that x = H(m)

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- ✓ produces fixed length digest (16-bit sum) of message
- ✓ is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

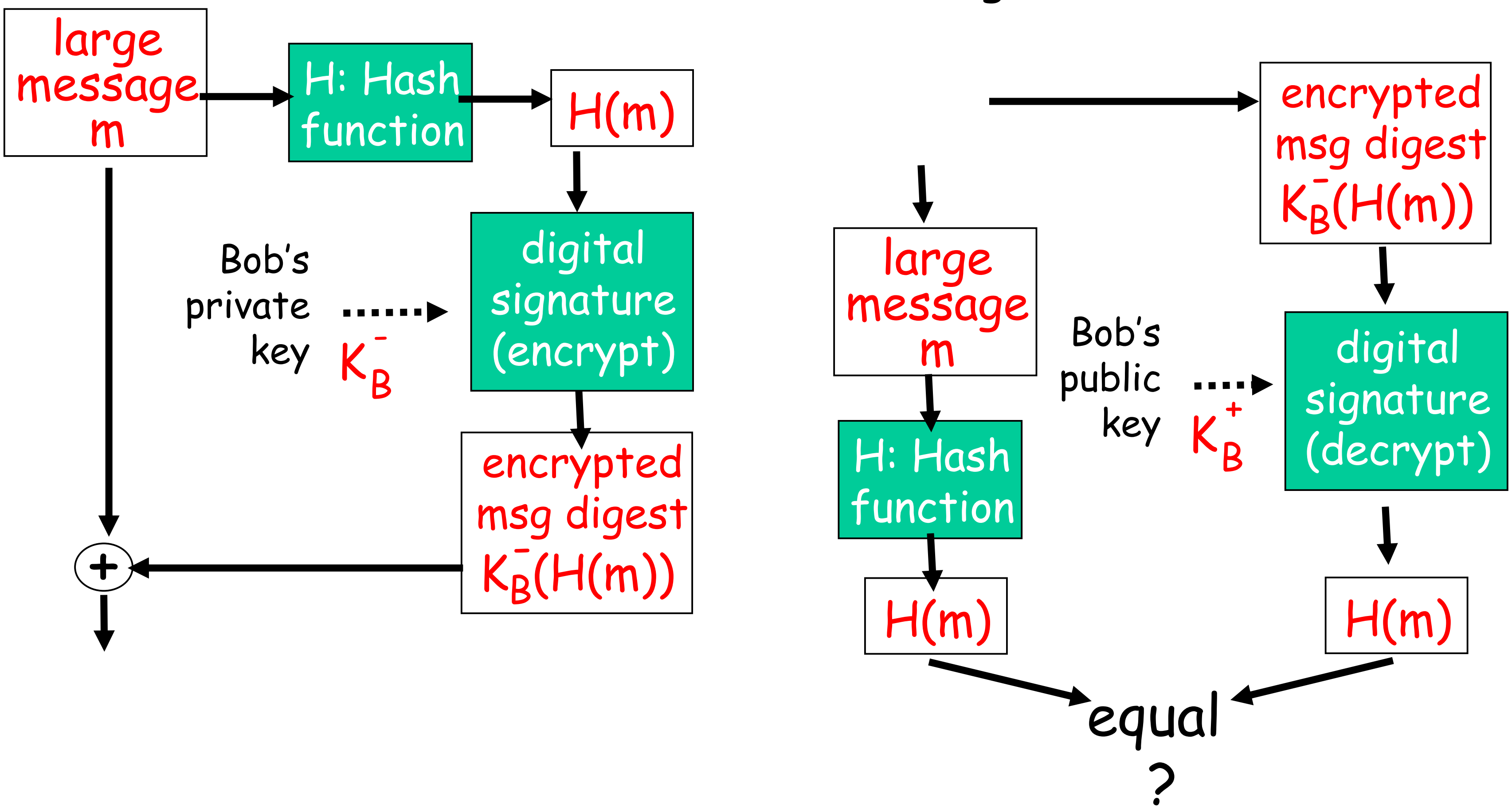
message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature and integrity of digitally signed message:

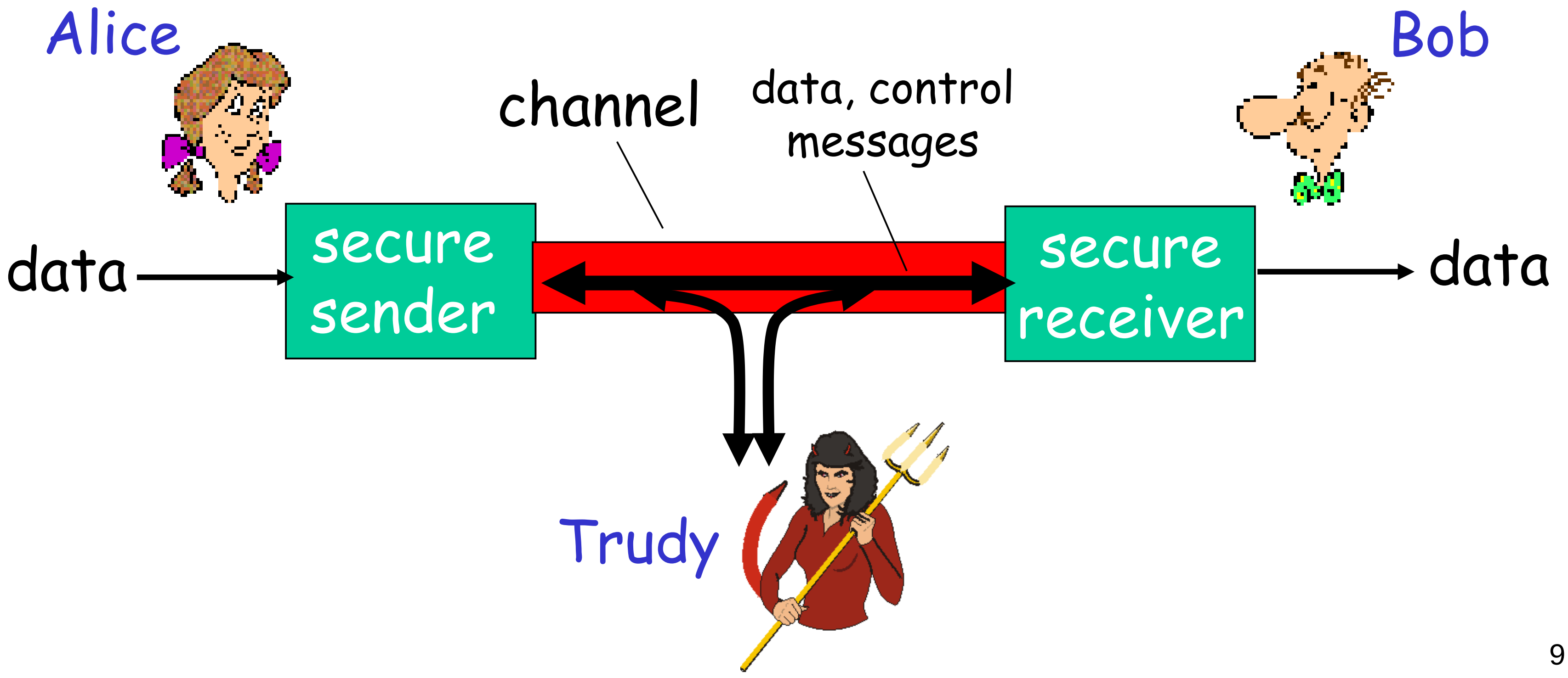


Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages



Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates

Authentication

There are bad guys (and girls) out there!

Q: What can a "bad guy" do?

A: a lot!

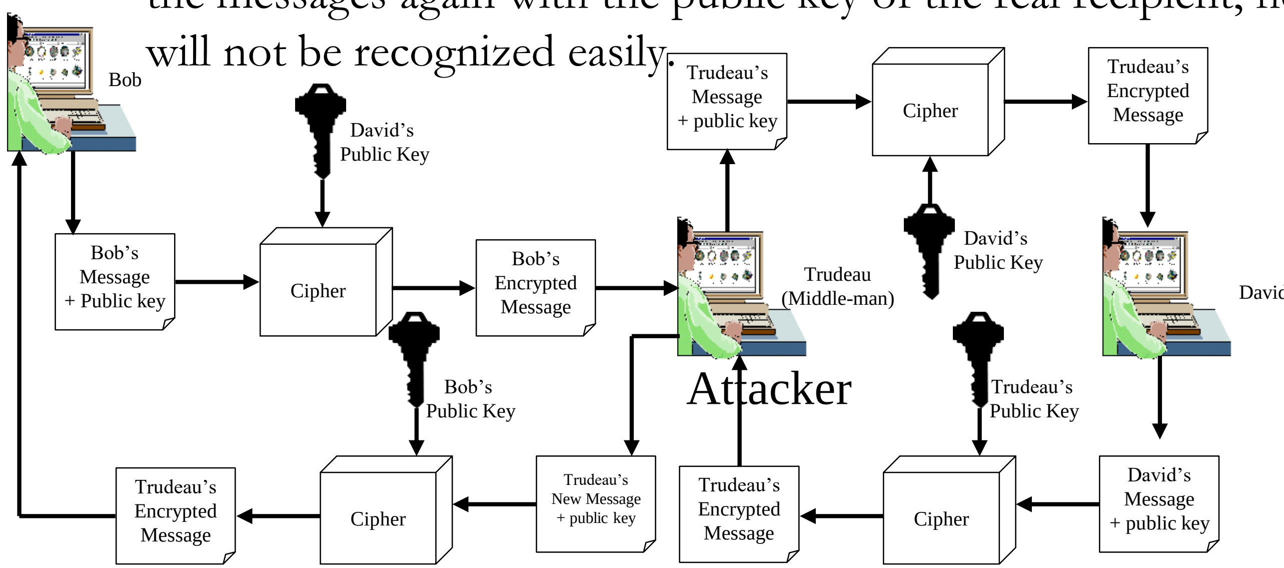
- *eavesdrop:* intercept messages
- actively *insert* messages into connection
- *impersonation:* can fake (spoof) source address in packet (or any field in packet)
- *hijacking:* "take over" ongoing connection by removing sender or receiver, inserting himself in place
- *denial of service*: prevent service from being used by others (e.g., by overloading resources)

more on this later ……

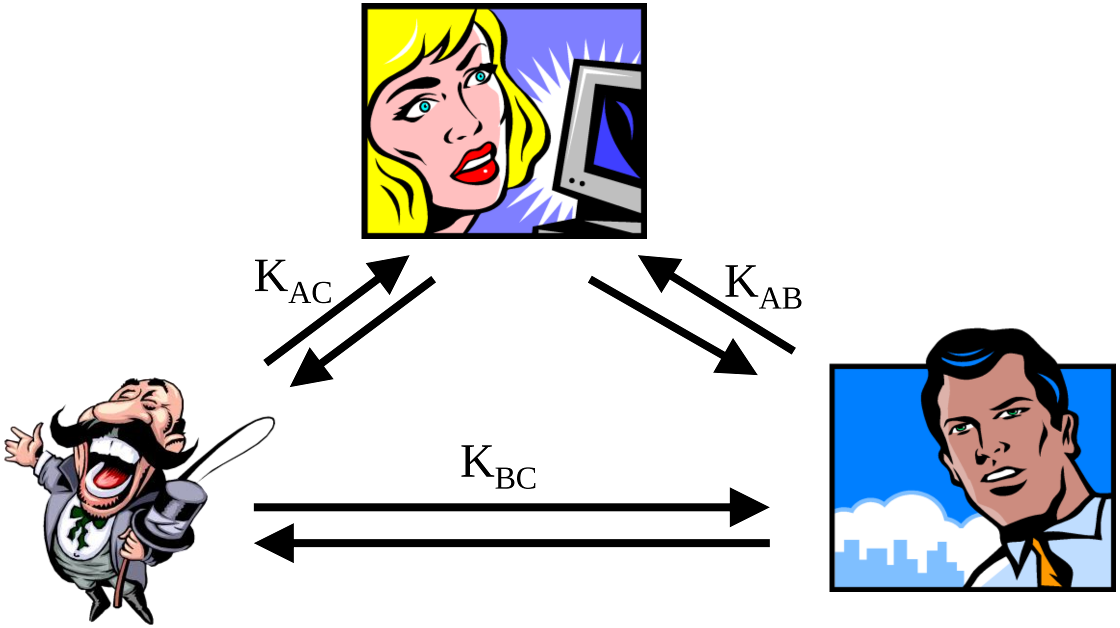
Asymmetric Encryption

Man-in-the-middle Attack

- Hacker could generate a key pair, give the public key away and tell everybody, that it belongs to somebody else. Now, everyone believing it will use this key for encryption, resulting in the hacker being able to read the messages. If he encrypts the messages again with the public key of the real recipient, he will not be recognized easily.



The Key Management Problem



Authentication

Goal: Bob wants Alice to "prove" her identity to him

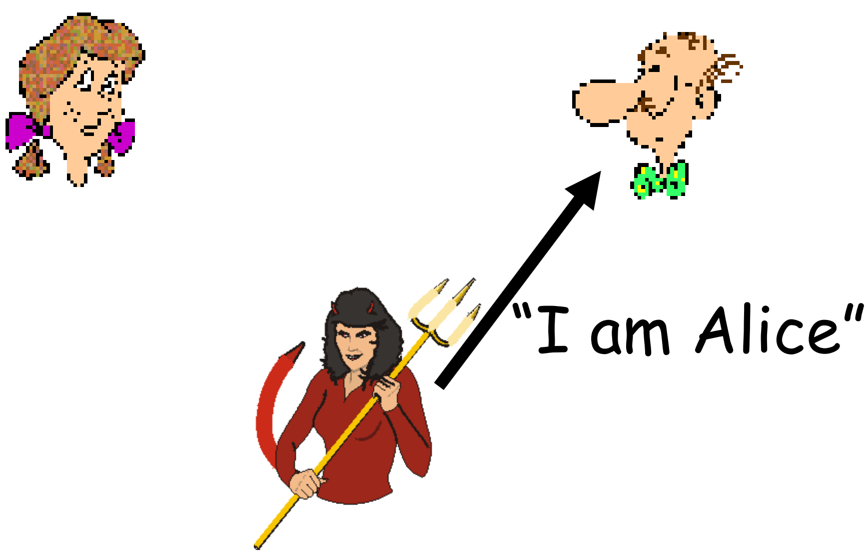
Protocol ap1.0: Alice says "I am Alice"

Failure scenario??

Authentication

Goal: Bob wants Alice to “prove” her identity to him

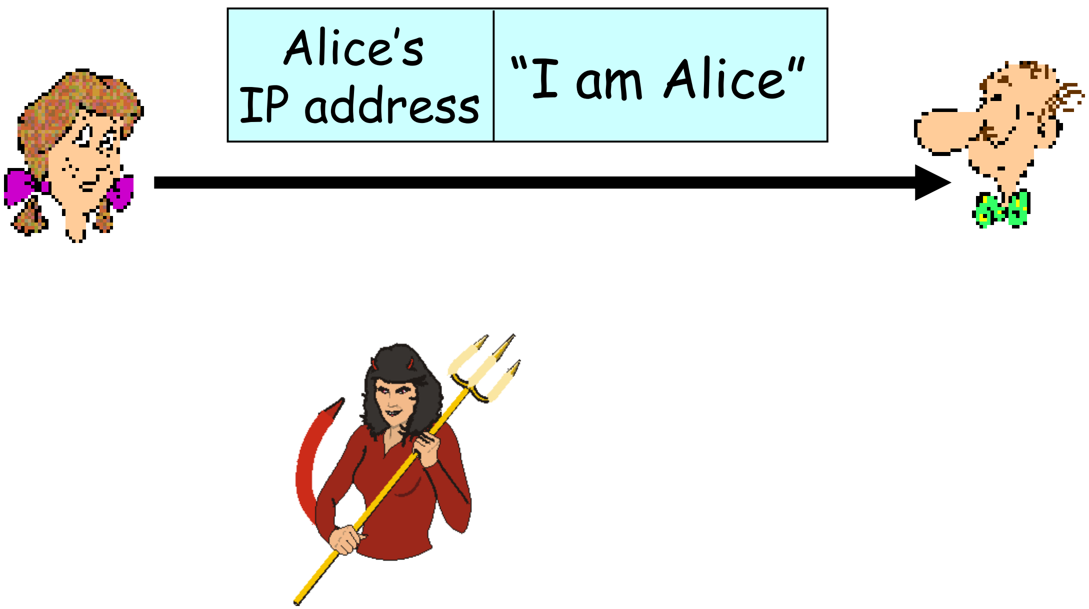
Protocol ap1.0: Alice says “I am Alice”



in a network,
Bob can not “see”
Alice, so Trudy simply
declares
herself to be Alice

Authentication: another try

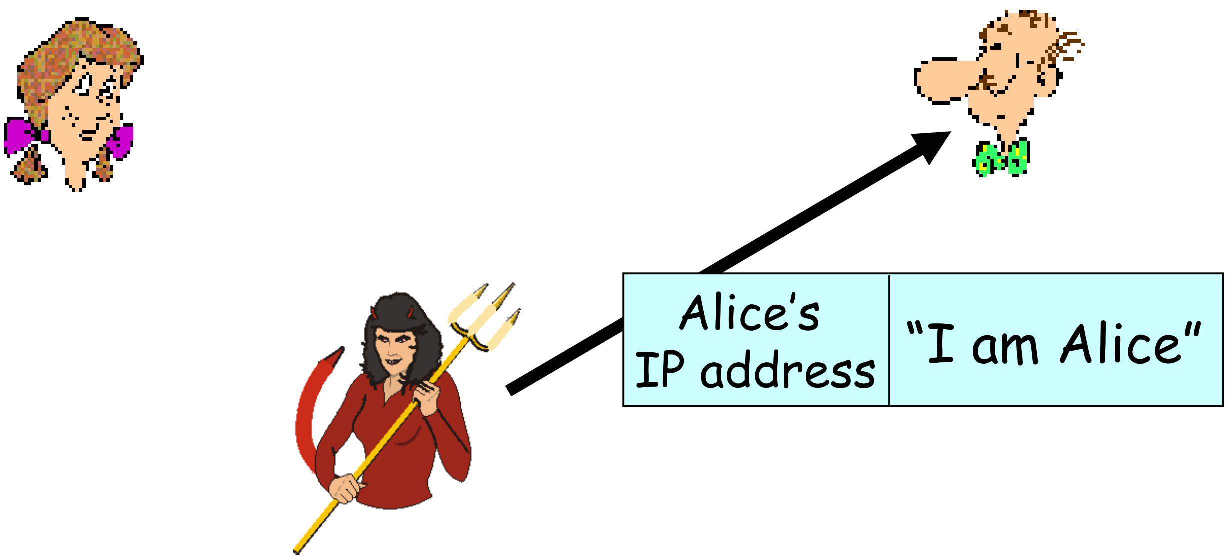
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Failure scenario??

Authentication: another try

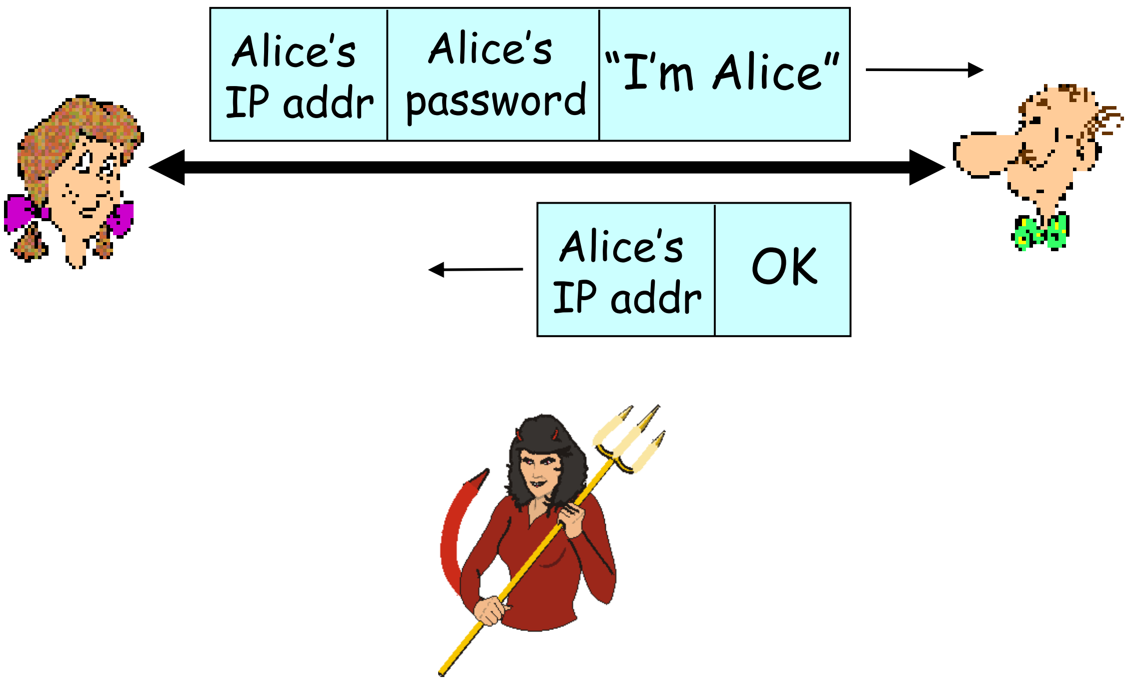
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Trudy can create a packet "spoofing" Alice's address

Authentication: another try

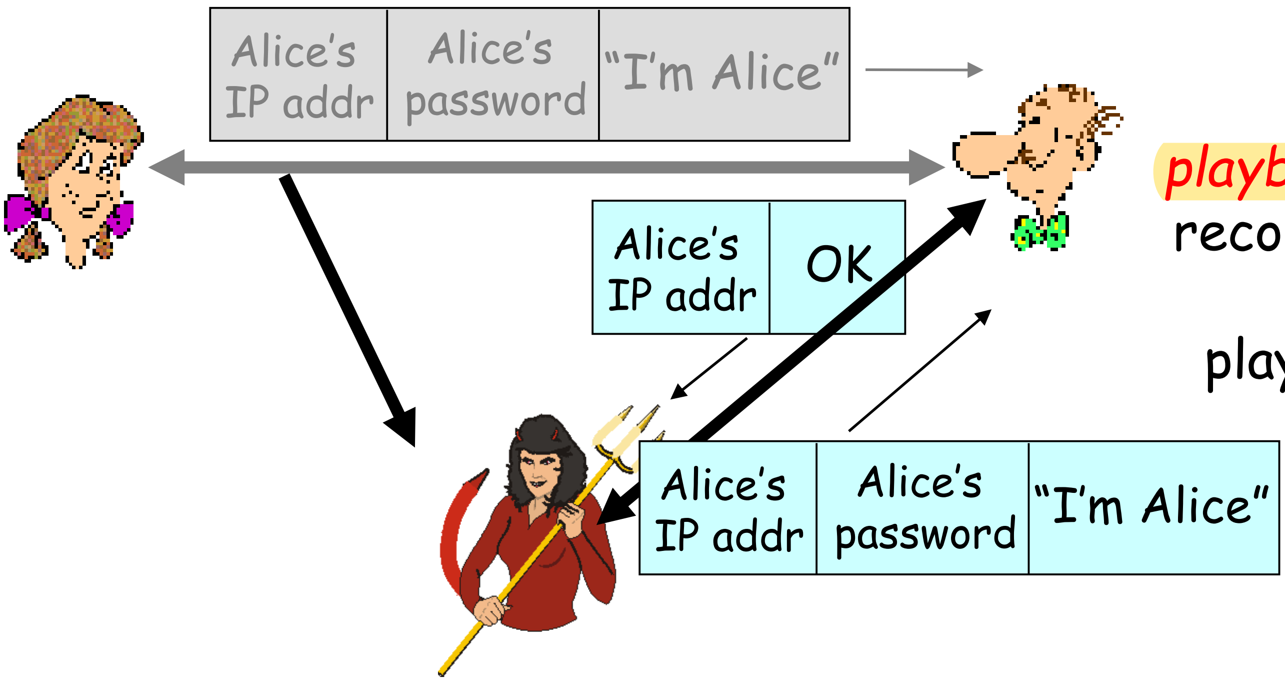
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

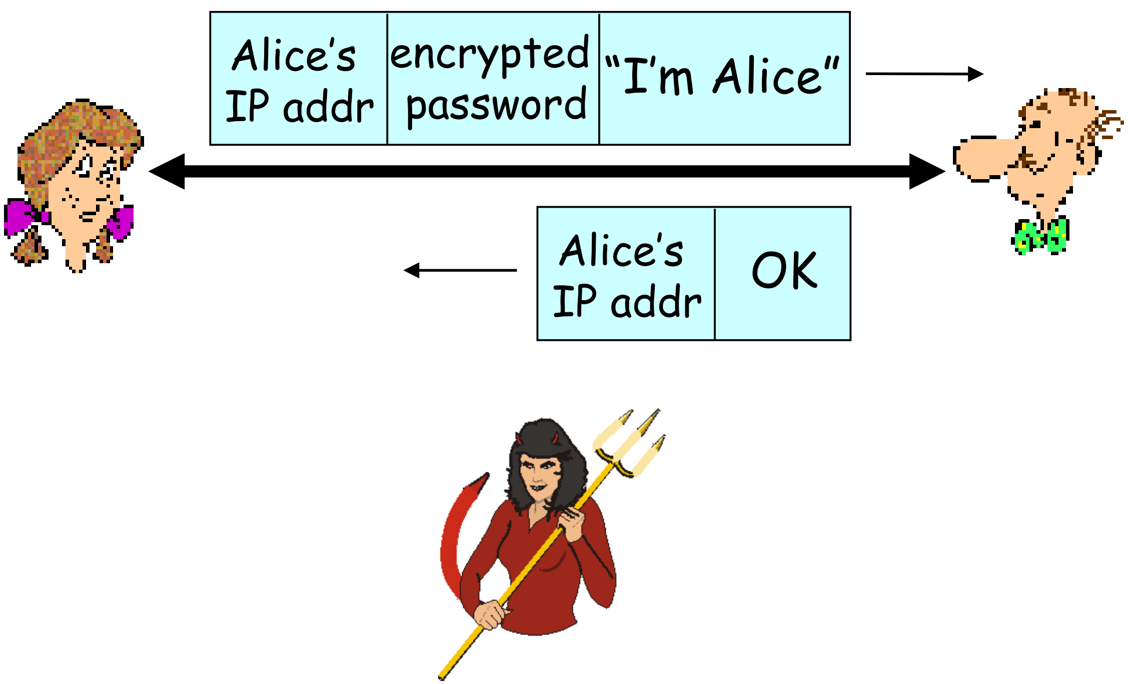
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

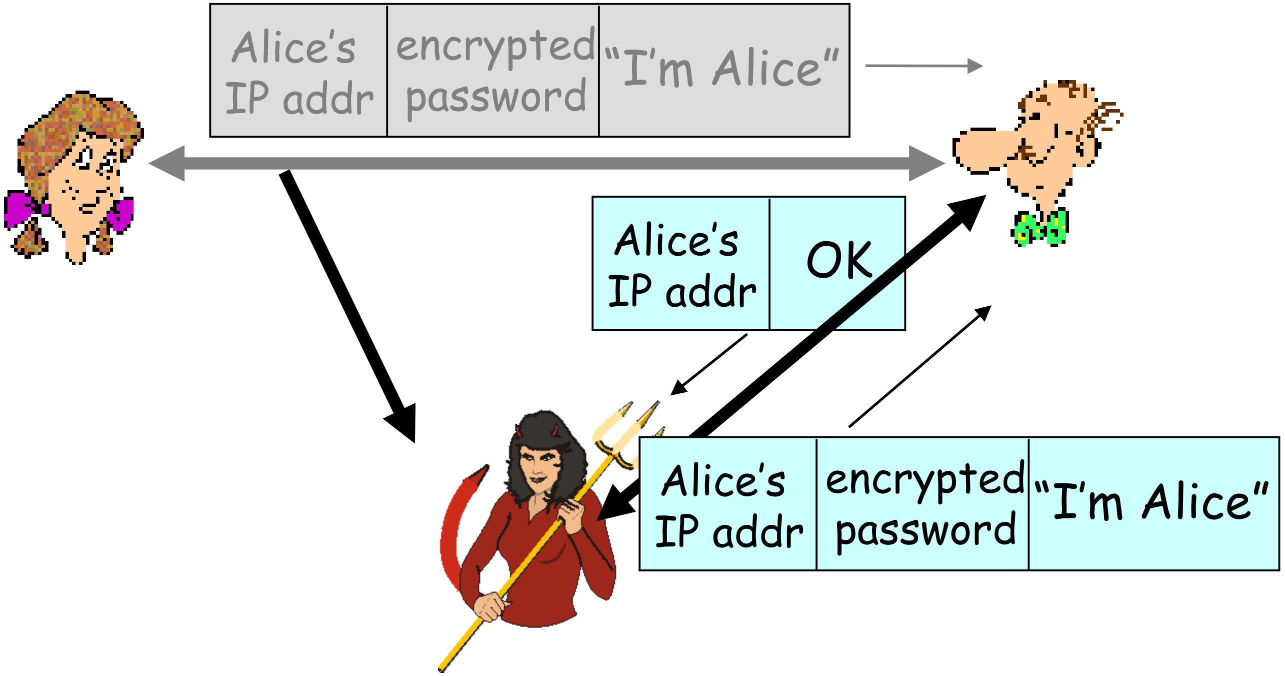
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



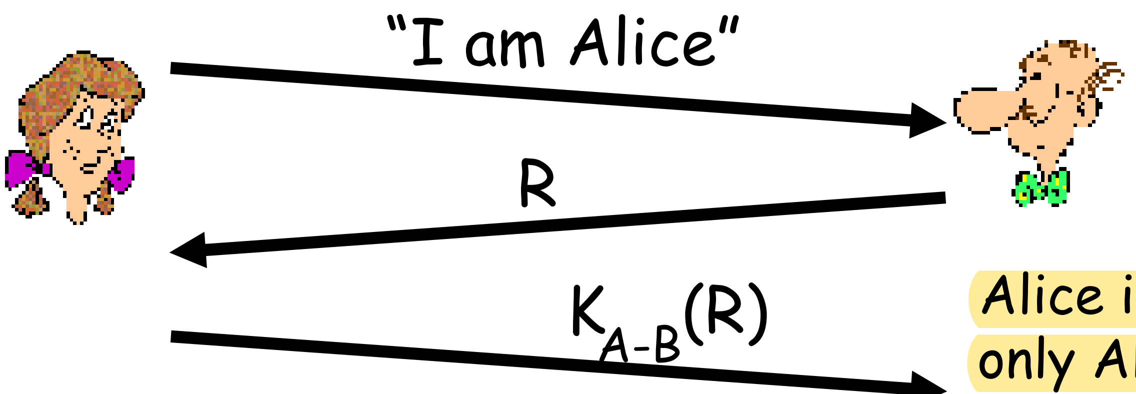
record and playback still works!

Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once –in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

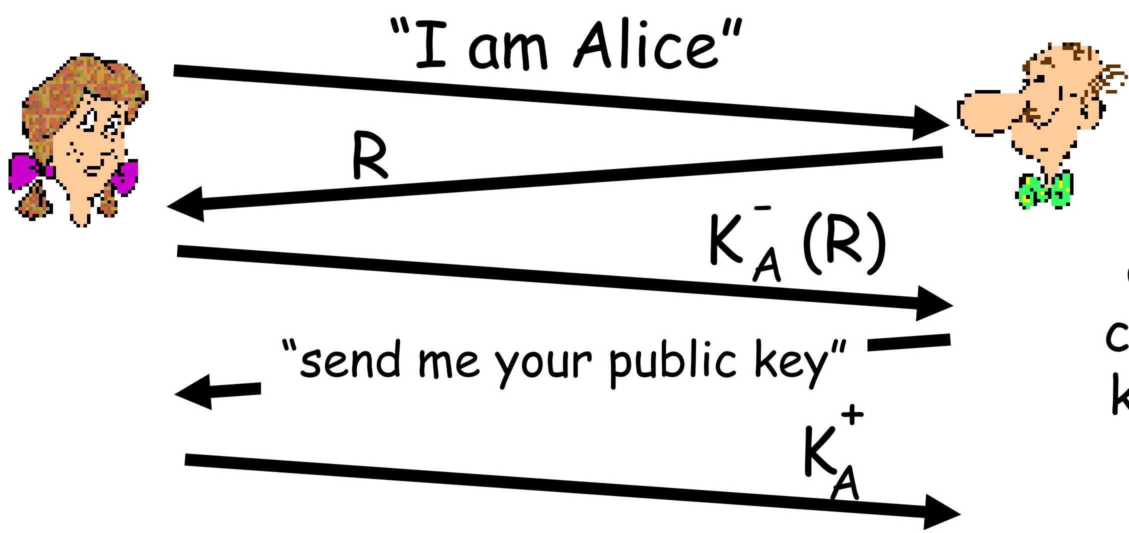
Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

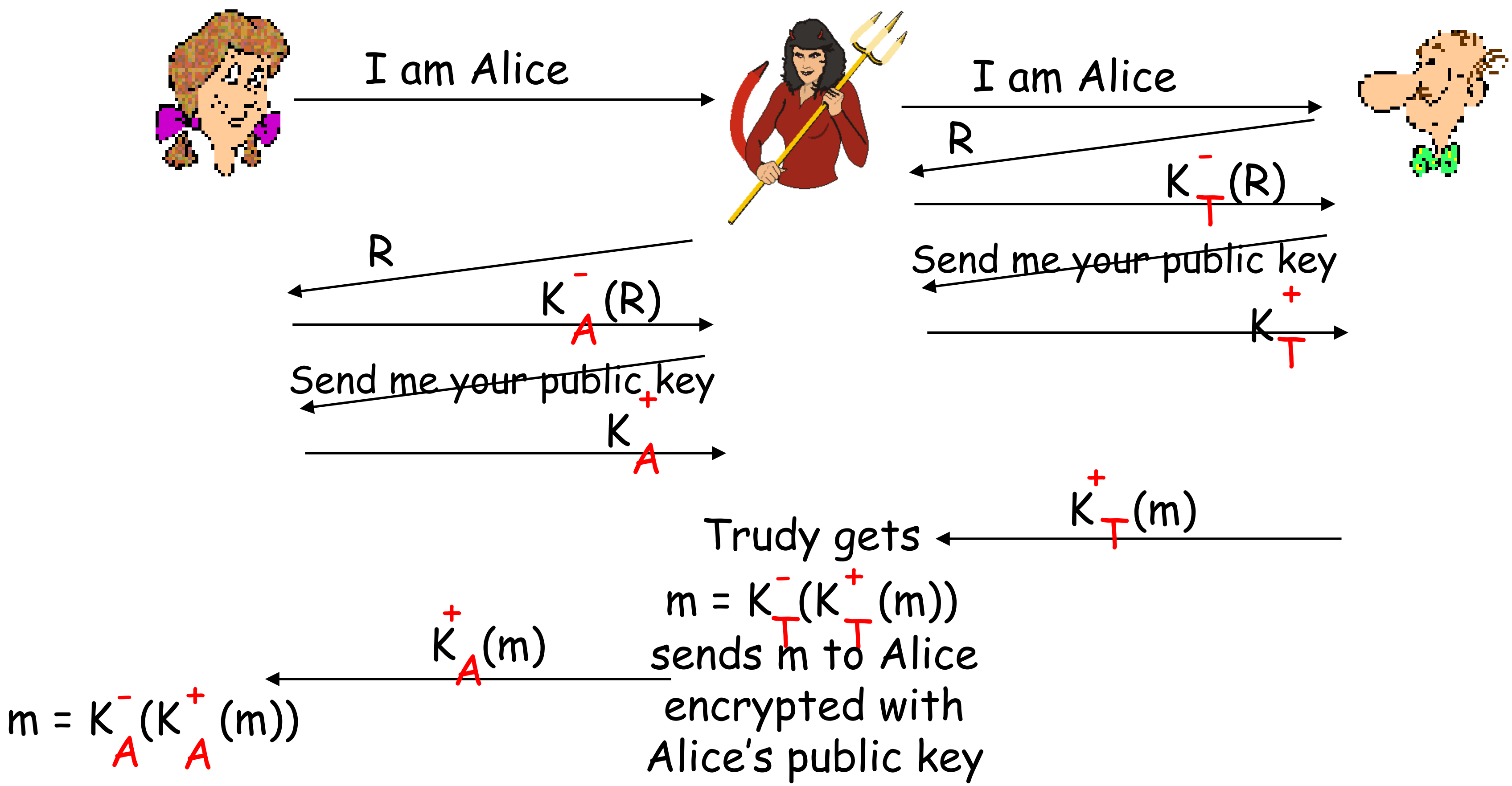
$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

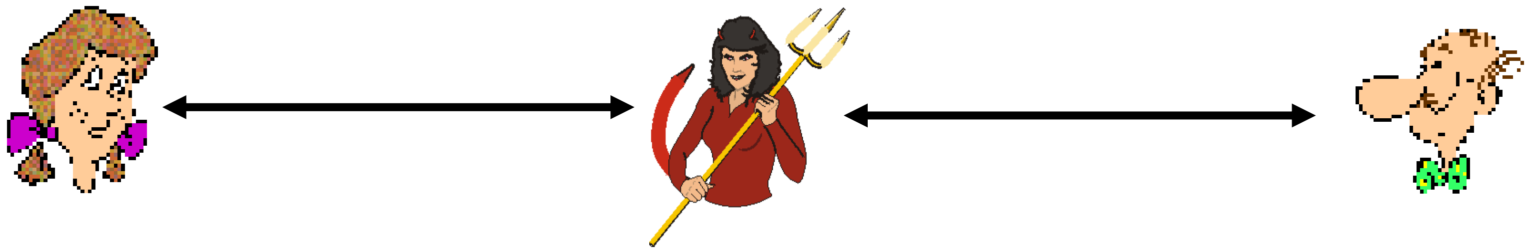
ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



ap5.0: security hole

Man (woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Difficult to detect:

- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation)
- problem is that Trudy receives all messages as well!

Authentication + Message integrity

Key Distribution and certification

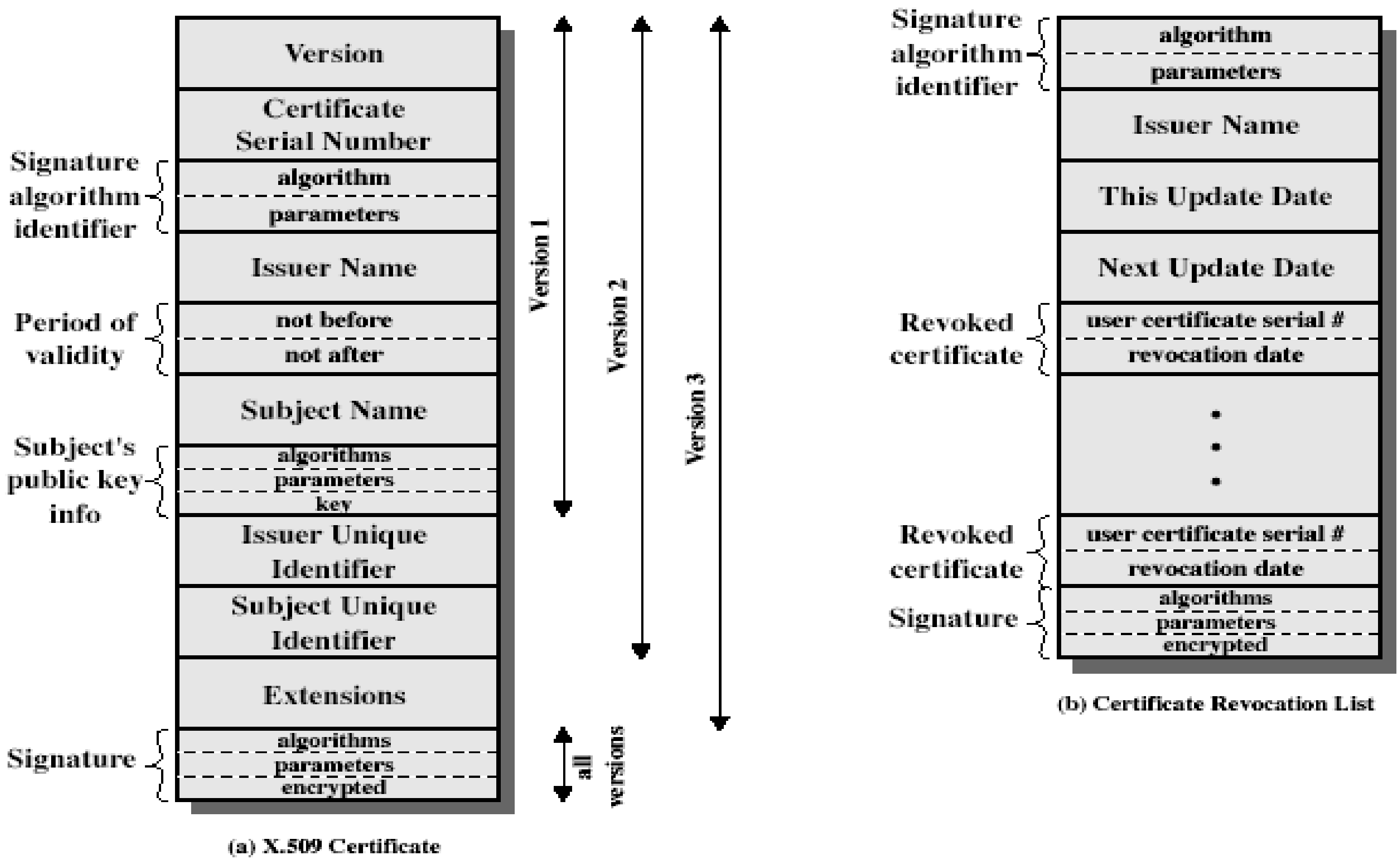
X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining some info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user
  - signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardized, but RSA recommended

X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version (1, 2, or 3)
  - serial number (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer X.500 name (CA)
  - period of validity (from - to dates)
  - subject X.500 name (name of owner)
  - subject public-key info (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- notation `CA<<A>>` denotes certificate for A signed by CA

X.509 Certificates



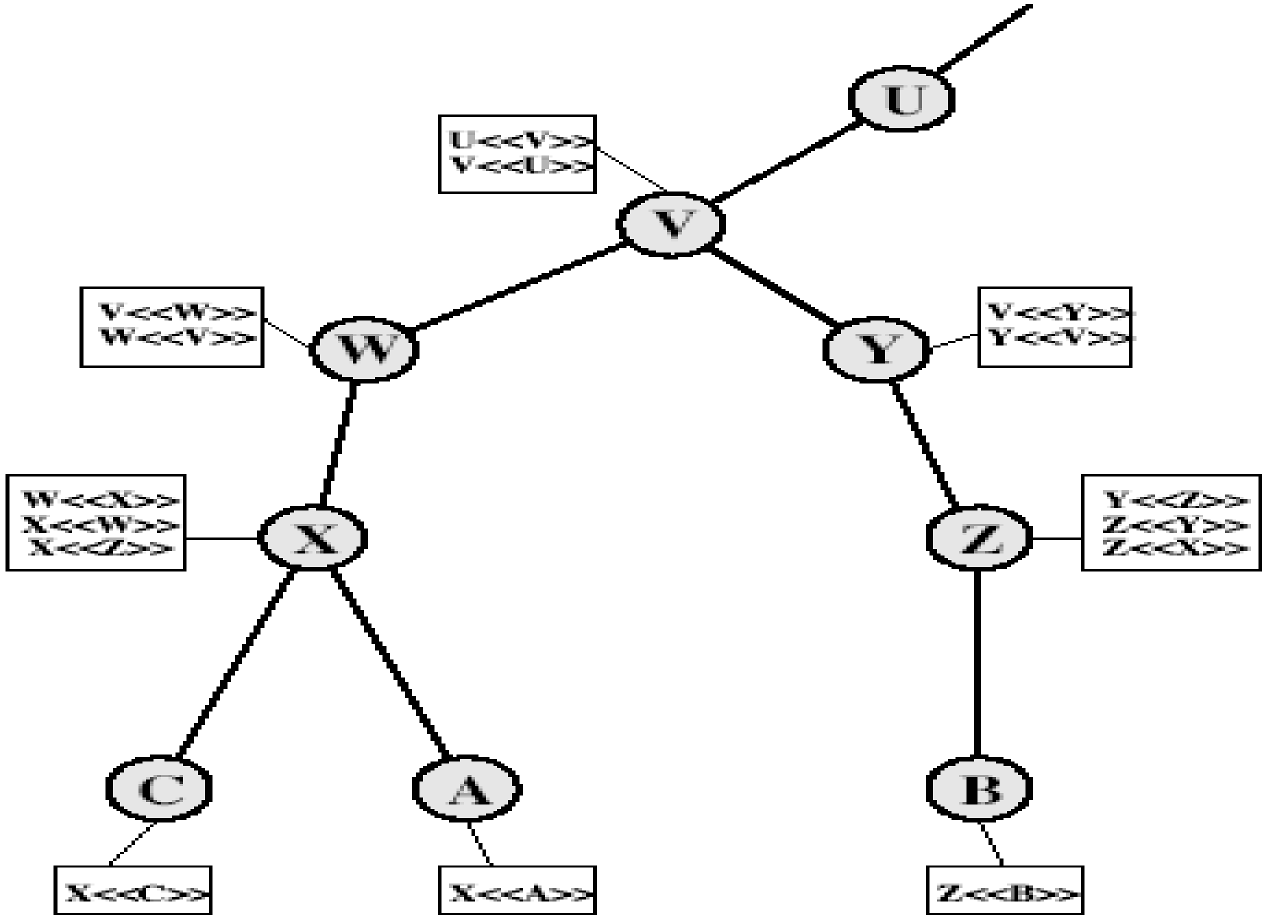
Obtaining a Certificate

- any user with access to the public key of the CA can verify the user public key that was certified
- only the CA can modify a certificate without being detected
- cannot be forged, certificates can be placed in a public directory

CA Hierarchy

- if both users share a common CA then they are assumed to know its public key
- otherwise CA's must form a hierarchy
- use certificates linking members of hierarchy to validate other CA's
  - each CA has certificates for clients (forward) and parent (backward)
- each client trusts parents certificates
- enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA Hierarchy Use



Digital Certificates

- The certification Authority
- How do you generate a public/private key?
- How do you inform everyone?
- How do others know that the key sent by you is actually sent by you?
- Classes of certificates
- Certification Revocation List
- Online Certificate Validation Protocol

How digital Certificates Work

- Let us say that A wants to send his credit card details to B. and A wants to verify that B is actually B.
- A will ask for digital certificate.
- B will send this certificate to A. As we know certificate will contain B's identity, public key etc.
- A can now send the message, encrypting it with public key of B to B.
- B will decrypt it with its private key.

A Digital Certificate include

1. Certificate owner’s identifying information

1. Certificate owner’s public key

1. Validity Date

1. Serial Number of the certificate

1. Name of the certificate issuer

1. Digital Signature of the issuer

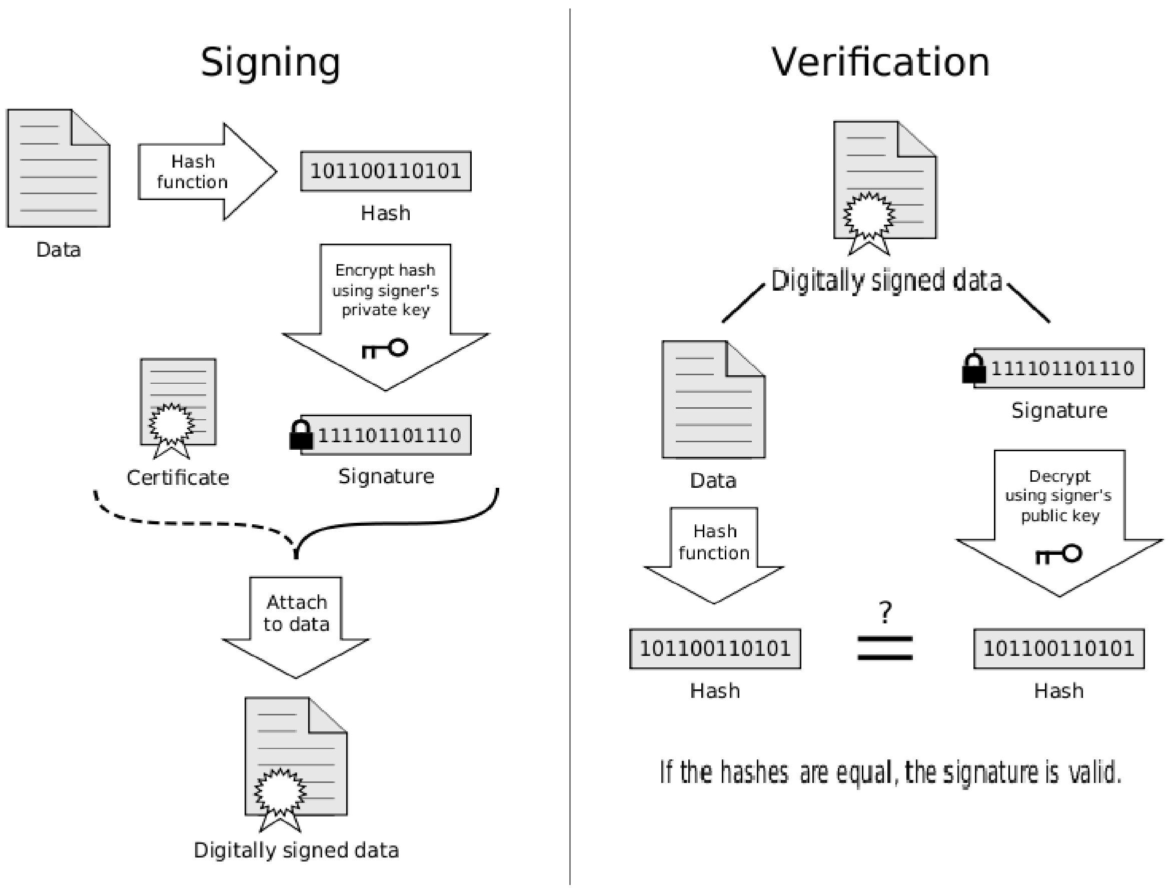
A Digital Certificate

Data:

Version: v3 (0x2)
Serial Number: 3 (0x3)
Signature Algorithm: PKCS #1 MD5 With RSA Encryption
Issuer: OU=Ace **Certificate** Authority, O=Ace Industry, C=US
Validity:
    Not Before: Fri Oct 17 18:36:25 1997
    Not After: Sun Oct 17 18:36:25 1999
Subject: CN=Jane Doe, OU=Finance, O=Ace Industry, C=US
Subject **Public Key** Info:
    Algorithm: PKCS #1 RSA Encryption
    Public Key:
        Modulus:
        43:7d:45:6d:71:4e:17:3d:f0:36:4b:5b:7f:a8
    Public Exponent: 65537 (0x10001)

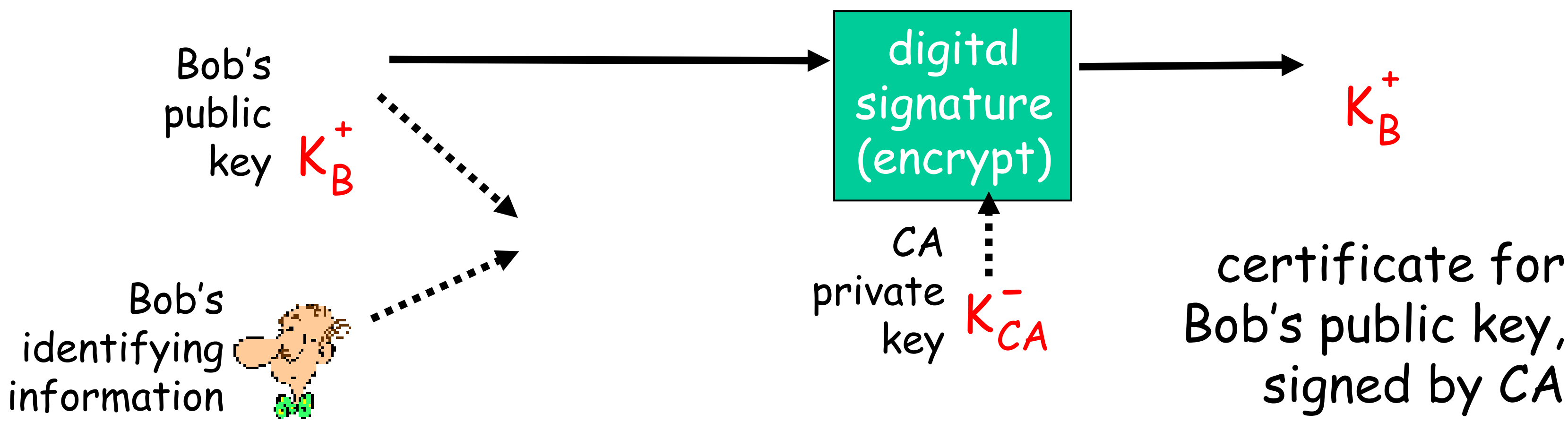
Signature

Algorithm: PKCS #1 MD5 With RSA Encryption
Signature:
6d:23:af:f3:d3:b6:7a:df:90:df:cd:7e:18:6c


Signing
Data
Hash function
101100110101
Hash
Encrypt hash using signer's private key
Certificate
111101101110
Signature
Attach to data
Digitally signed data
Verification
Digitally signed data
Data
111101101110
Signature
Hash function
Decrypt using signer's public key
101100110101
Hash
?
101100110101
Hash
If the hashes are equal, the signature is valid.

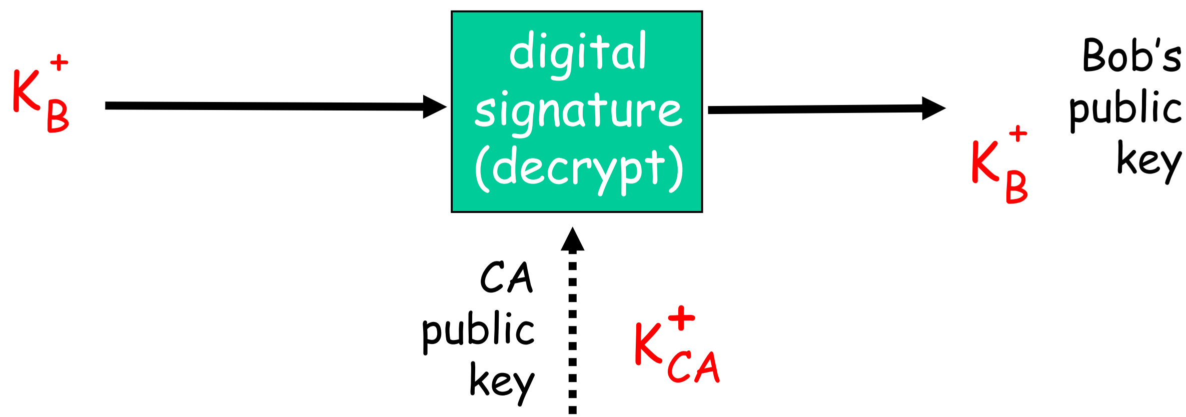
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



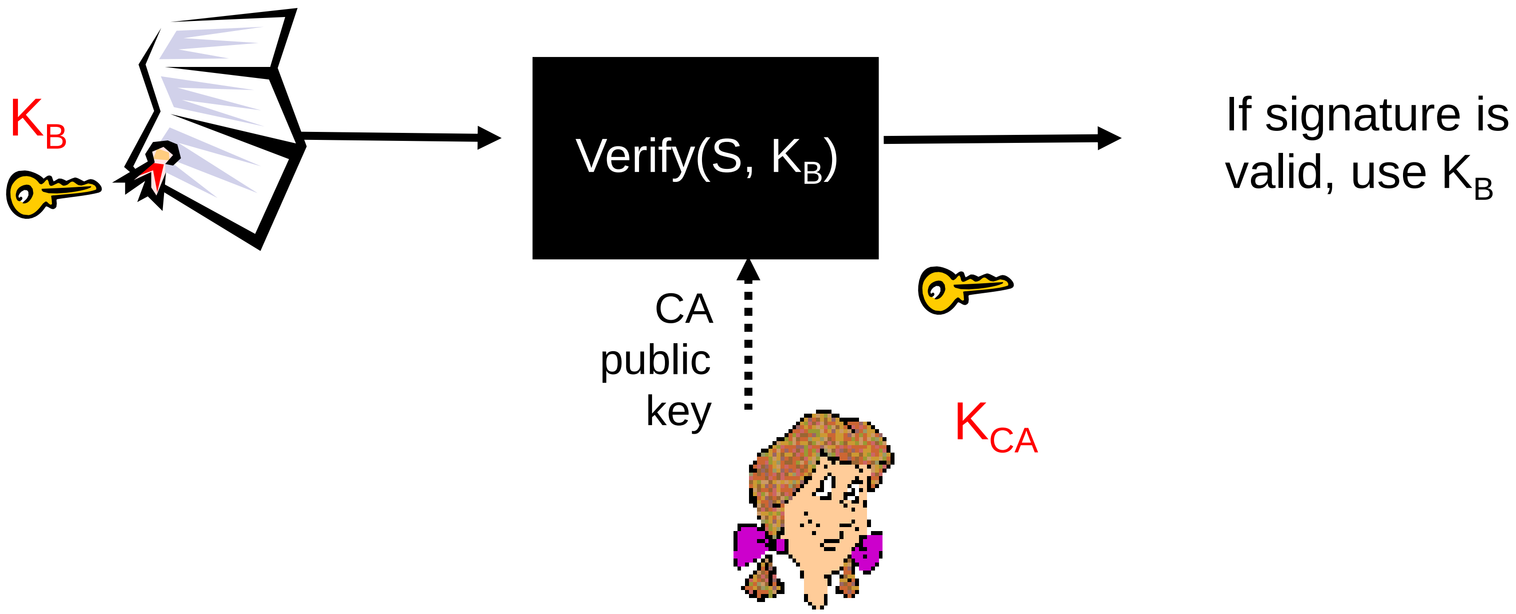
Certification Authorities

- When Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



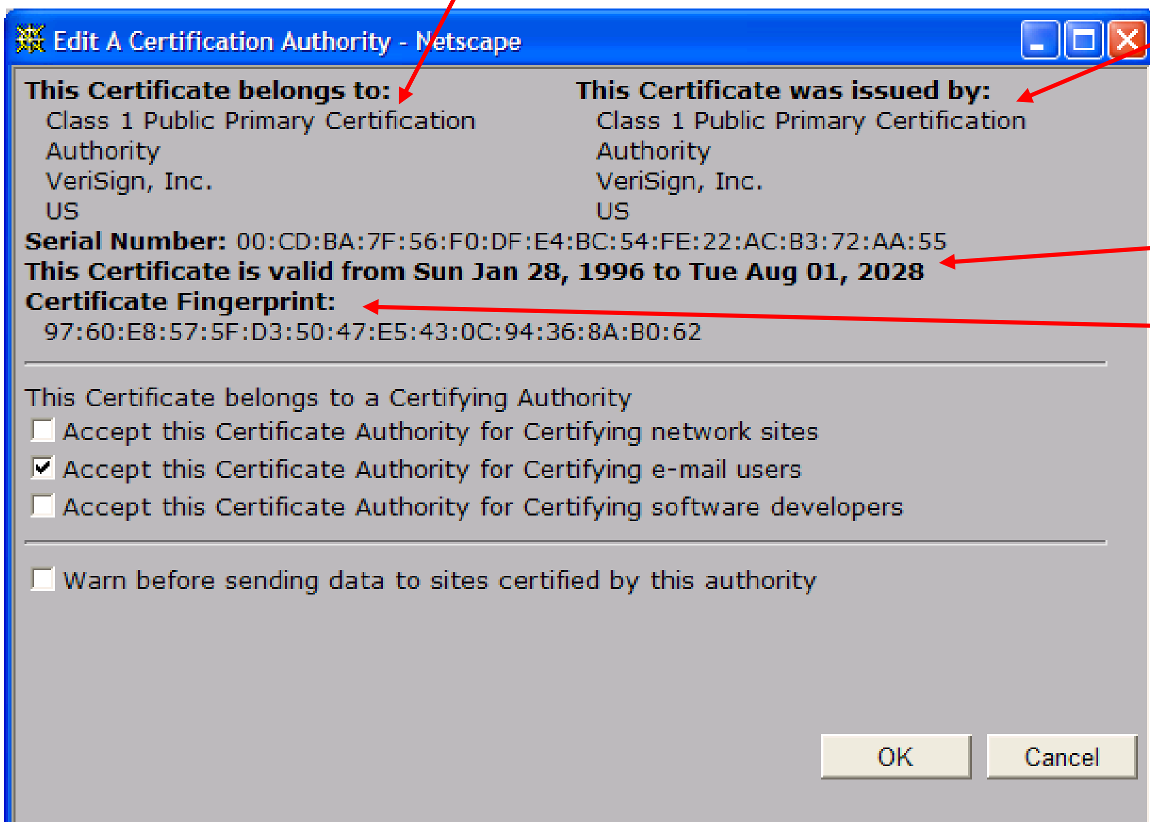
Certification Authorities

- When Alice wants Bob's public key:
  - Gets Bob's certificate (Bob or elsewhere).
  - Use CA's public key to verify the signature within Bob's certificate, then accepts public key



A certificate contains:

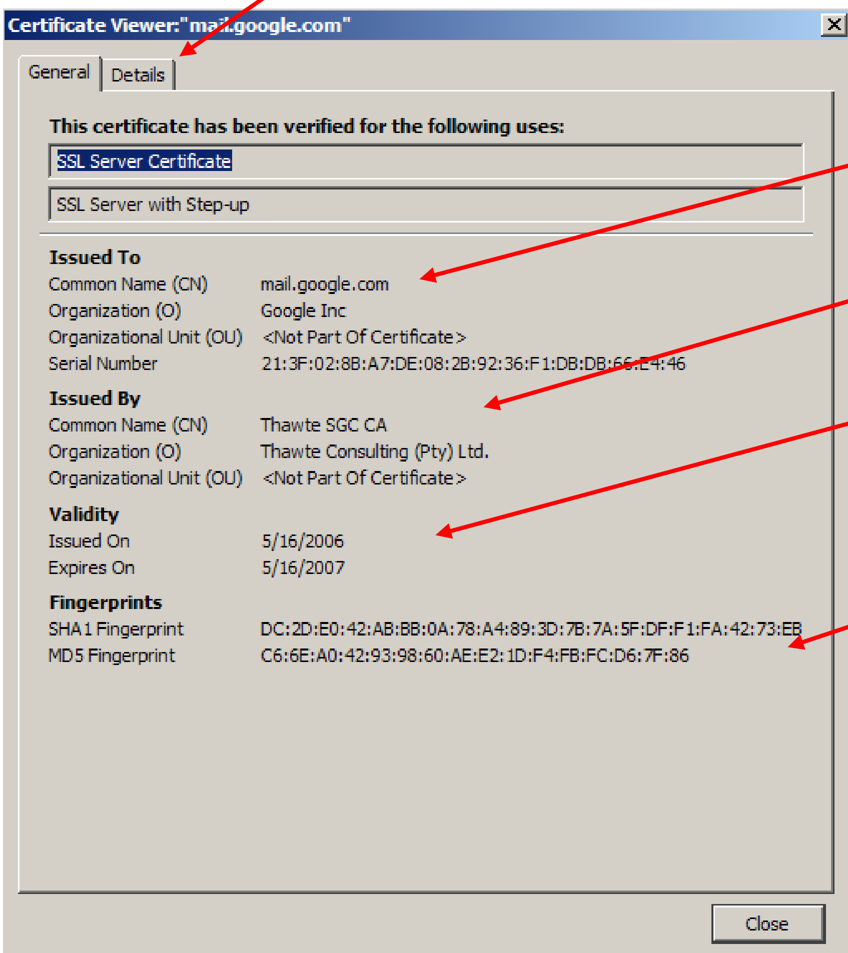
- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer

Certificate Contents

- info algorithm and key value itself (not shown)



- Cert owner
- Cert issuer
- Valid dates
- Fingerprint of signature

Certificate Revocation

- certificates have a period of validity
- may need to revoke before expiration, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
- CAs maintain list of revoked certificates
  - the Certificate Revocation List (CRL)
- users should check certificates with CA's CRL

X.509 Version 3

- has been recognized that additional information is needed in a certificate
  - email/URL, policy details, usage constraints
- rather than explicitly naming new fields a general extension method was defined
- extensions consist of:
  - extension identifier
  - criticality indicator
  - extension value

Authentication Procedures

- X.509 includes three alternative authentication procedures:
  - One-Way Authentication
  - Two-Way Authentication
  - Three-Way Authentication
- all use public-key signatures

Nonce

- a nonce is a parameter that varies with time. A nonce can be a time stamp, a visit counter on a Web page, or a special marker intended to limit or prevent the unauthorized replay or reproduction of a file.

Nonce

- from RFC 2617:
  - For applications where no possibility of replay attack can be tolerated the server can use one-time nonce values which will not be honored for a second use. This requires the overhead of the server remembering which nonce values have been used until the nonce time-stamp (and hence the digest built with it) has expired, but it effectively protects against replay attacks.

One-Way Authentication

- One message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality (message hasn't been sent multiple times)
- message must include timestamp, nonce, B's identity and is signed by A

Two-Way Authentication

- Two messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B

Three-Way Authentication

- 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
- has reply from A back to B containing a signed copy of nonce from B
- means that timestamps need not be checked or relied upon

Key distribution

Trusted Intermediaries

Symmetric key problem:

- How do two entities establish shared secret key over network?

Solution:

- trusted key distribution center (KDC) acting as intermediary between entities

Public key problem:

- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Solution:

- trusted certification authority (CA)

The best of both worlds

- The client generates a one time symmetric session key with the help of certain cryptography algorithms.

- The client then encrypts the original clear text message with one-time symmetric key to produce ciphertext.

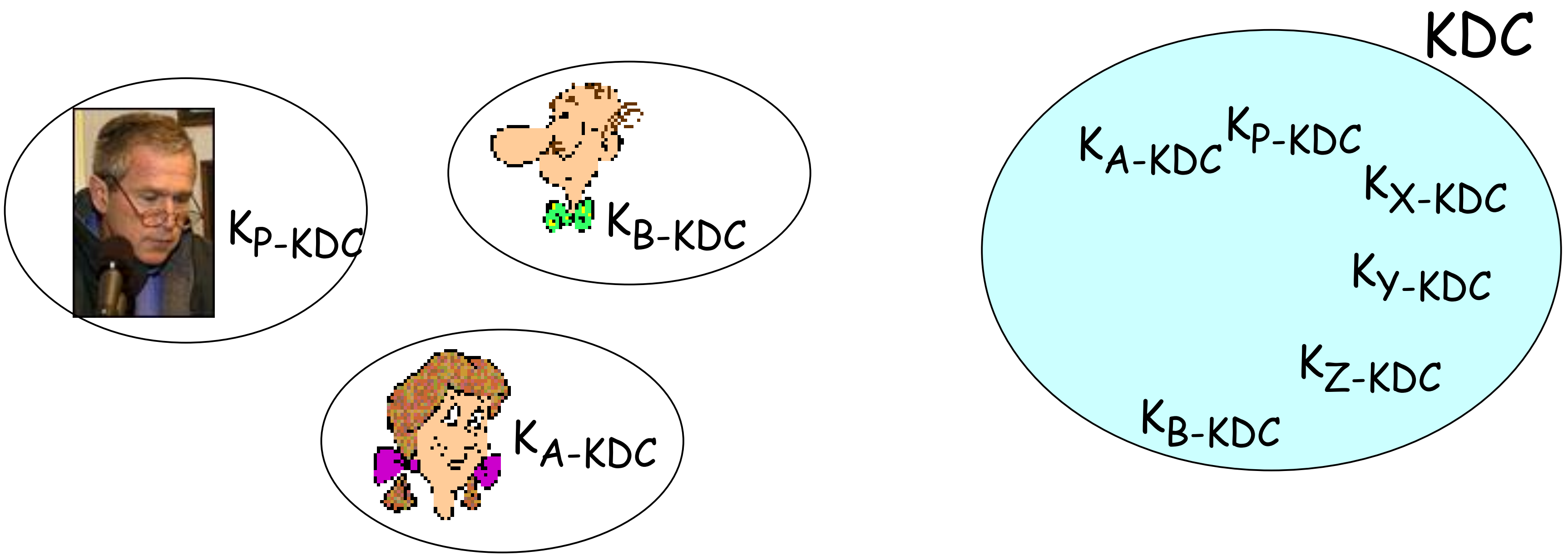
- The client takes key one-time symmetric key and and encrypts it with server's public key **(key wrapping).**

The best of both worlds

- The encrypted symmetric key + ciphertext message is encrypted again with server’s public key and sent to the server **(digital envelope)**.

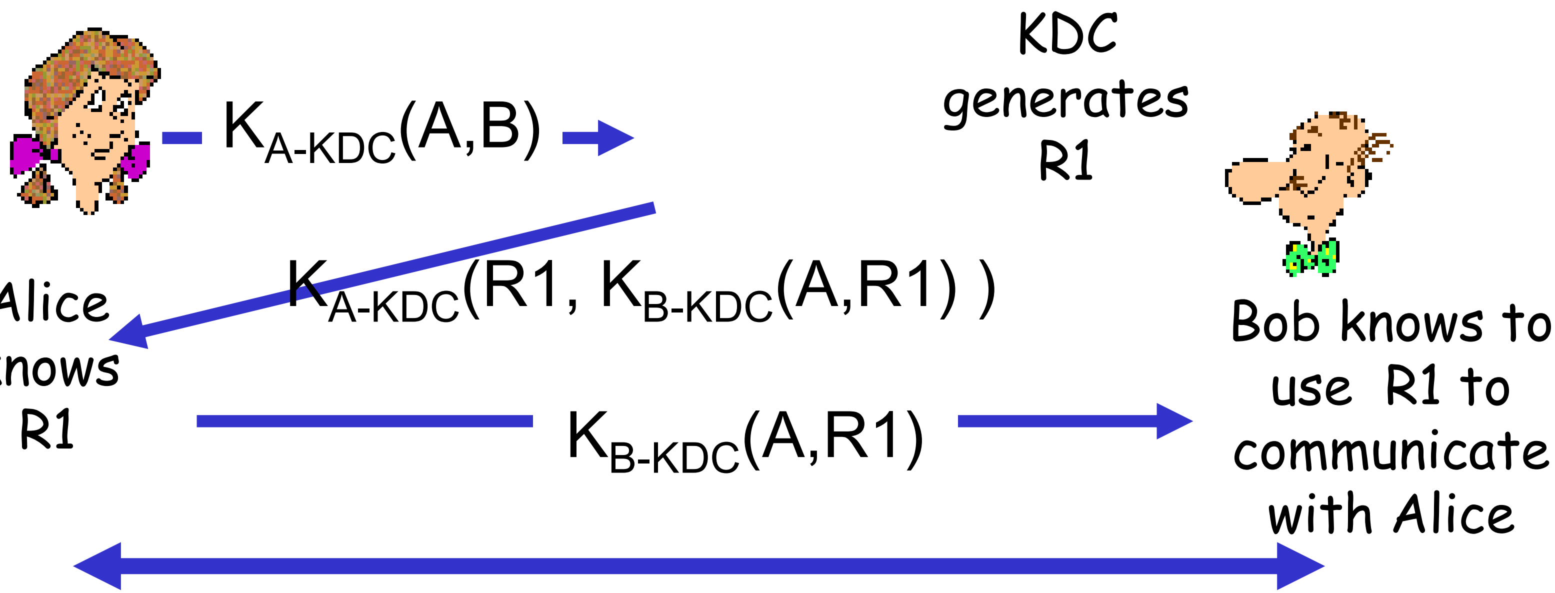
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key.
- KDC: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC.



Key Distribution Center (KDC)

Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



Alice and Bob communicate: using R1 as *session key* for shared symmetric encryption

Kerberos

- Identity
- Password
- Plaintext or Cleartext

Authentication Applications

- will consider authentication functions
- developed to support application-level authentication & digital signatures
- will consider Kerberos – a private-key authentication service
- then X.509 directory authentication service

Kerberos

- trusted key server system from MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
- two versions in use: 4 & 5

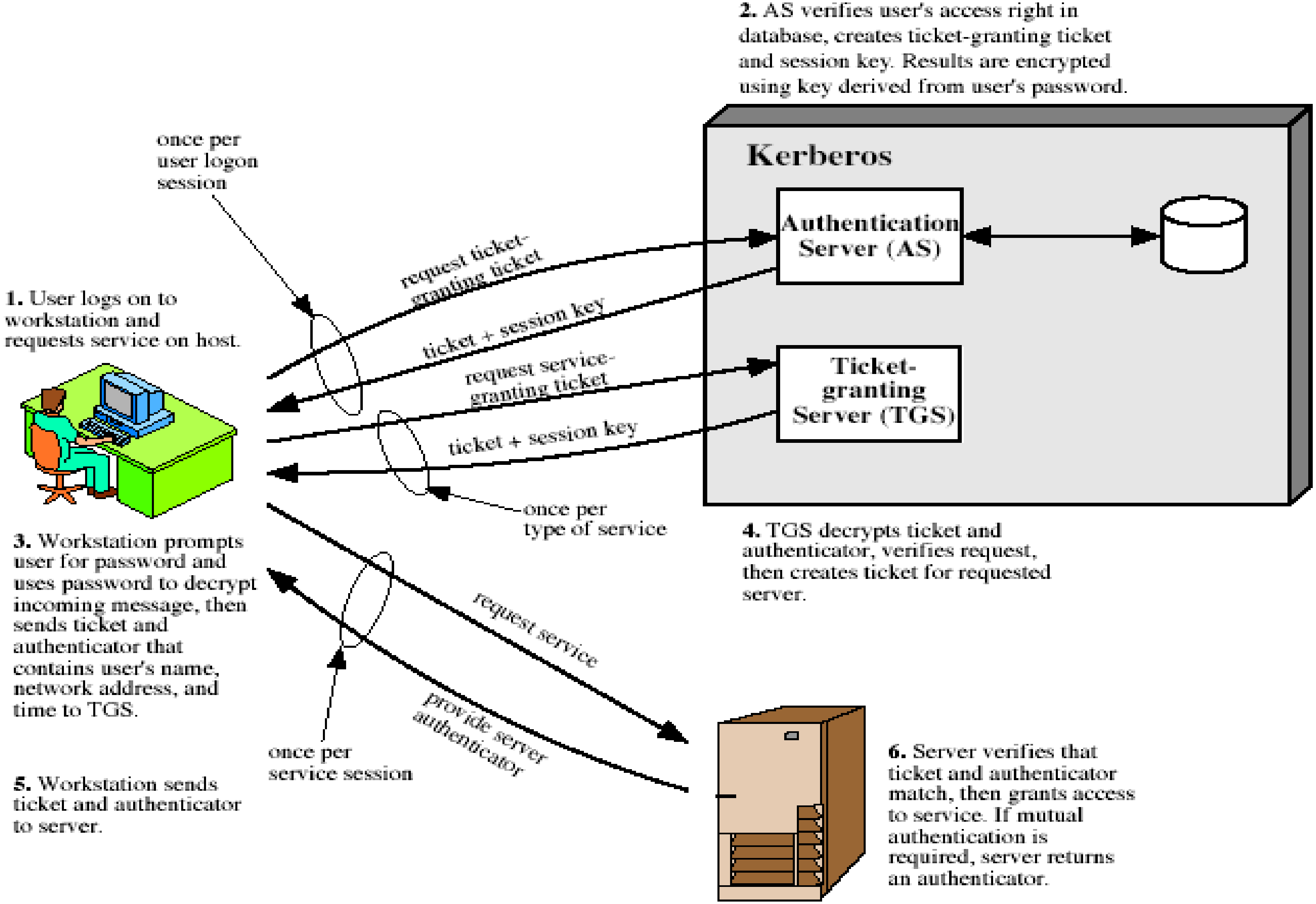
Kerberos Requirements

- first published report identified its requirements as:
  - security-an eavesdropper shouldn't be able to get enough information to impersonate the user
  - reliability- services using Kerberos would be unusable if Kerberos isn't available
  - transparency-users should be unaware of its presence
  - scalability- should support large number of users
- implemented using a 3rd party authentication scheme using a protocol proposed by Needham-Schroeder (NEED78)

Kerberos 4 Overview

- a basic third-party authentication scheme
  - uses DES buried in an elaborate protocol
- Authentication Server (AS)
  - user initially negotiates with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket-granting ticket TGT)
- Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT

Kerberos 4 Overview



Kerberos Realms

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- if have multiple realms, their Kerberos servers must share keys and trust

Kerberos Version 5

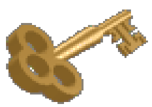
- developed in mid 1990's
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption algorithm, network protocol, byte order, ticket lifetime, authentication forwarding, inter-realm authentication
  - and technical deficiencies
    - double encryption, non-standard mode of use, session keys, password attacks
- specified as Internet standard RFC 1510

Step 0

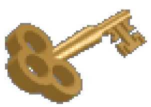
What KDC know and does

K_A K_B

I am Asha I want to talk to Bharat



K_A



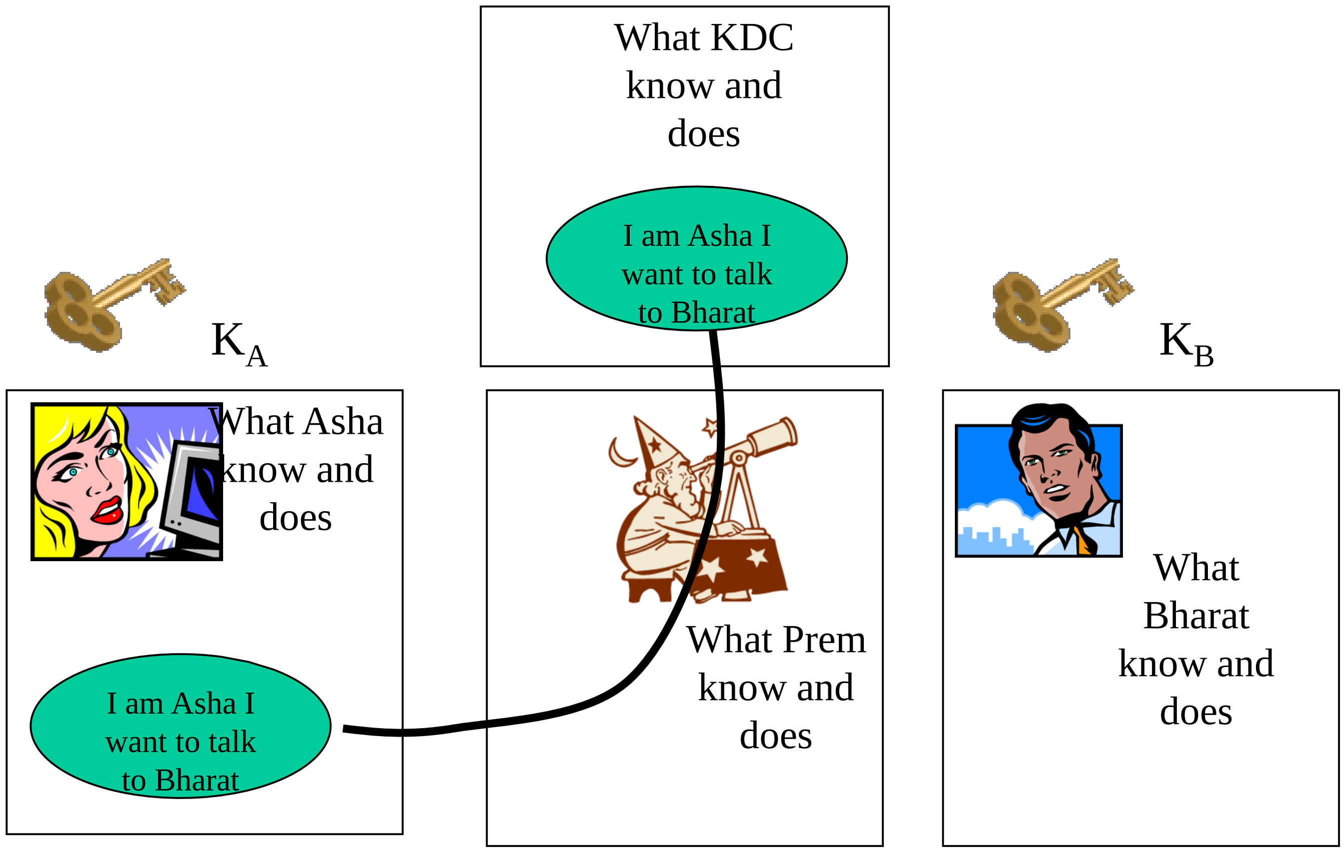
K_B

What Asha know and does

What Prem know and does

What Bharat know and does

Step 1



What KDC know and does
I am Asha I want to talk to Bharat
K_A
K_B
What Asha know and does
I am Asha I want to talk to Bharat
What Prem know and does
What Bharat know and does

Step 2
Bharat's Ticket

Part	Abr.	Explanation
Asha		The initial ticket requester
Bharat		The end recipient of ticket
Time Stamp	TS	The time that KDC developed the ticket
Time Duration	TD	Duration of validity of ticket
Session Key	K_{AB}	Session Key

Step 2
Asha's Ticket

Part	Abr.	Explanation
Asha		The initial ticket requester
Bharat		The end recipient of ticket
Time Stamp	TS	The time that KDC developed the ticket
Time Duration	TD	Duration of validity of ticket
Session Key	K_{AB}	Session Key
Recipient's Key	{///}	End Recipient's Ticket

- Step 2: Asha receives two tickets
  – Asha’s Ticket (decrypted using Asha’s password)
  – Bharat’s Ticket (decrypted using Bharat’s password)

- Step 3: Asha sends a message to Bharat
  – Asha sends a message (containing current time) + Bharat’s ticket to Bharat

- Step 5: Bharat composes a reply to Asha

- Step 4: Bharat decrypts Asha's message
  - Bharat receives the message and Processes it. Therefore he gets current session's key.

- Step 5: Bharat composes a reply to Asha

- Step 6
  - Asha Receives and decrypts Bharat’s reply
- Step 7
  - Asha and Bharat communicate in secure session