Last time

- P2P
- Security
  - Intro
  - Principles of cryptography

This time

- Message integrity
- Authentication
- Key distribution and certification

Chapter 8 roadmap

Digital Signatures

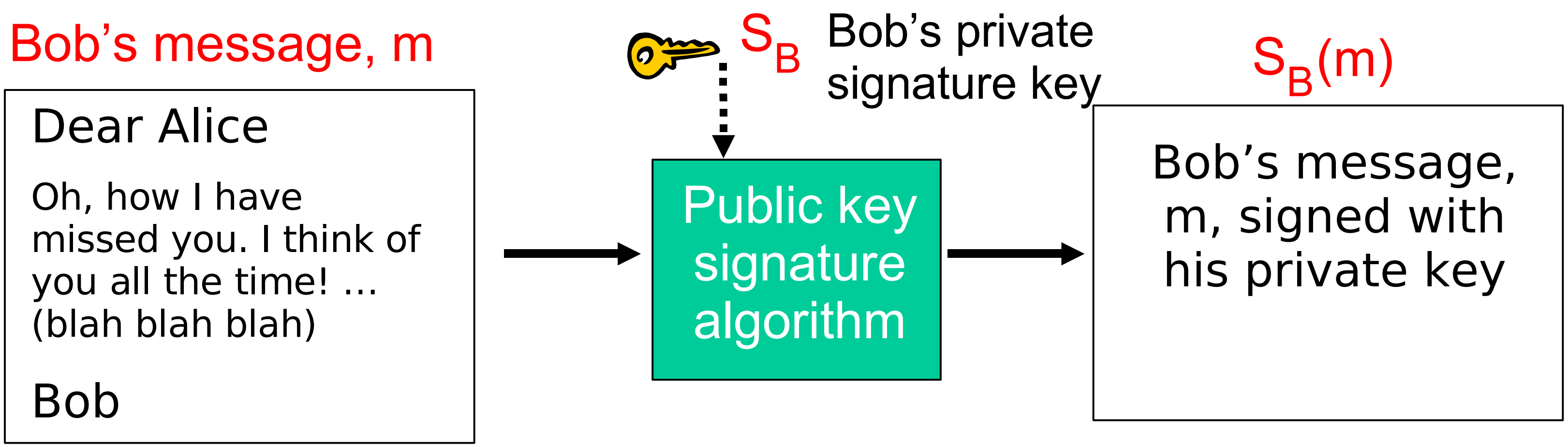
Cryptographic technique analogous to hand-written signatures.

- 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 signature key S_B, creating "signed" message, $S_B(m)$



- Bob also has a public verification key V_B such that $V_B(S_B(m)) = m$.

Digital Signatures (more)

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

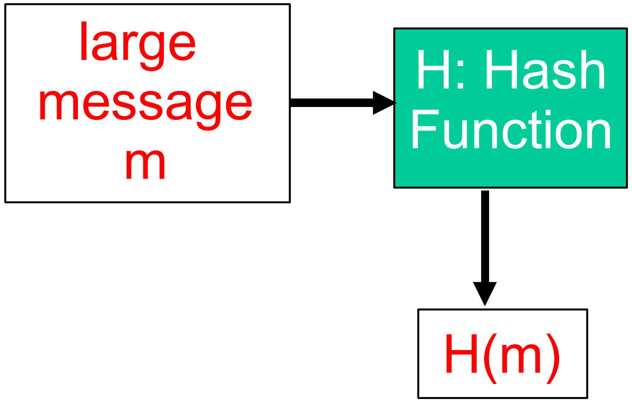
Alice thus verifies that:

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

Non-repudiation:

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

Message Digests



Computationally expensive to public-key sign 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), or two messages m1, m2 with H(m1)=H(m2)

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:

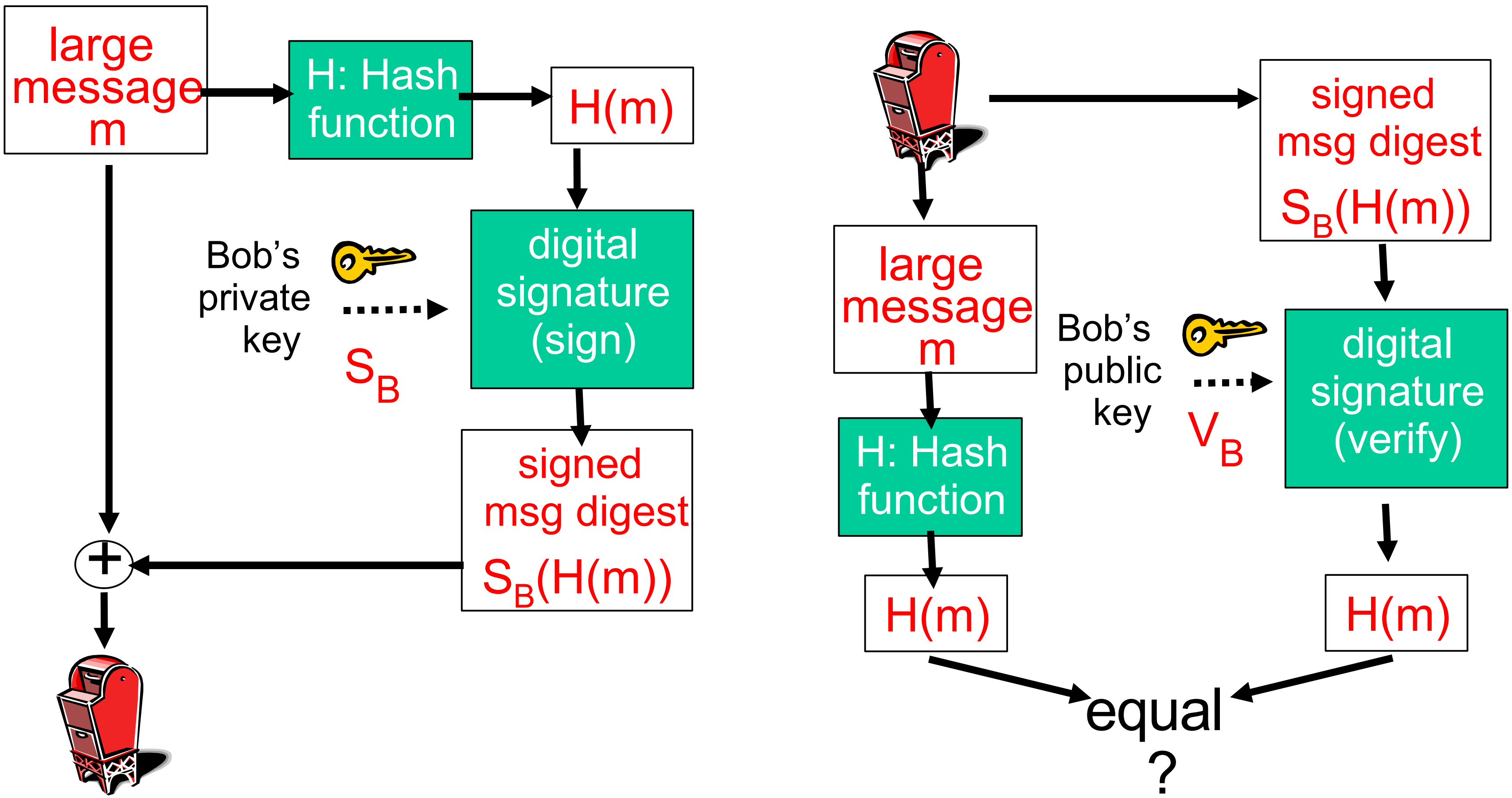
message	ASCII format	message	ASCII format
I O U 1	49 4F 55 31	I O U 9	49 4F 55 39
0 0 . 9	30 30 2E 39	0 0 . 1	30 30 2E 31
9 B O B	39 42 4F 42	9 B O B	39 42 4F 42
	B2 C1 D2 AC		B2 C1 D2 AC

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

- Traditionally: MD5 hash function (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.
  - it's been figured out how to make collisions!
- Newer: SHA-1
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
  - many people think collisions are imminent!
- Starting to switch to SHA-256
  - Newer US standard [NIST, FIPS PUB 180-2]
  - 256-bit message digest

Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Authentication
8.4 Integrity
8.5 Key Distribution and certification
8.6 Access control: firewalls
8.7 Attacks and counter measures
8.8 Security in many layers

Authentication

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

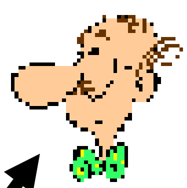
Protocol ap1.0: Alice says “I am Alice”

Failure scenario??

Authentication

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

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



“I am Alice”

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

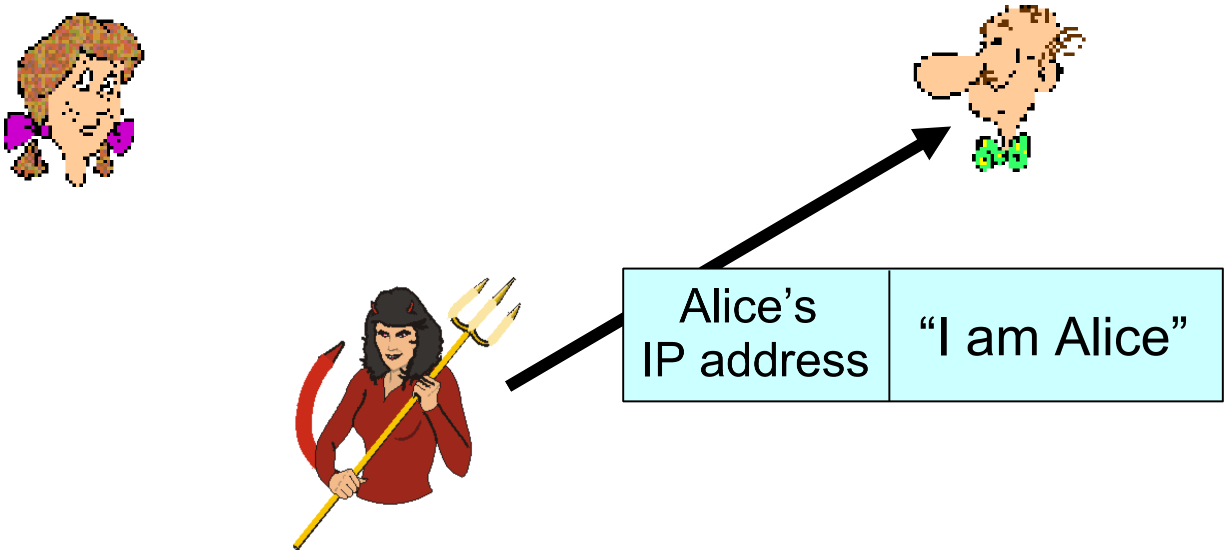
Authentication: another try

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

Failure scenario??

Authentication: another try

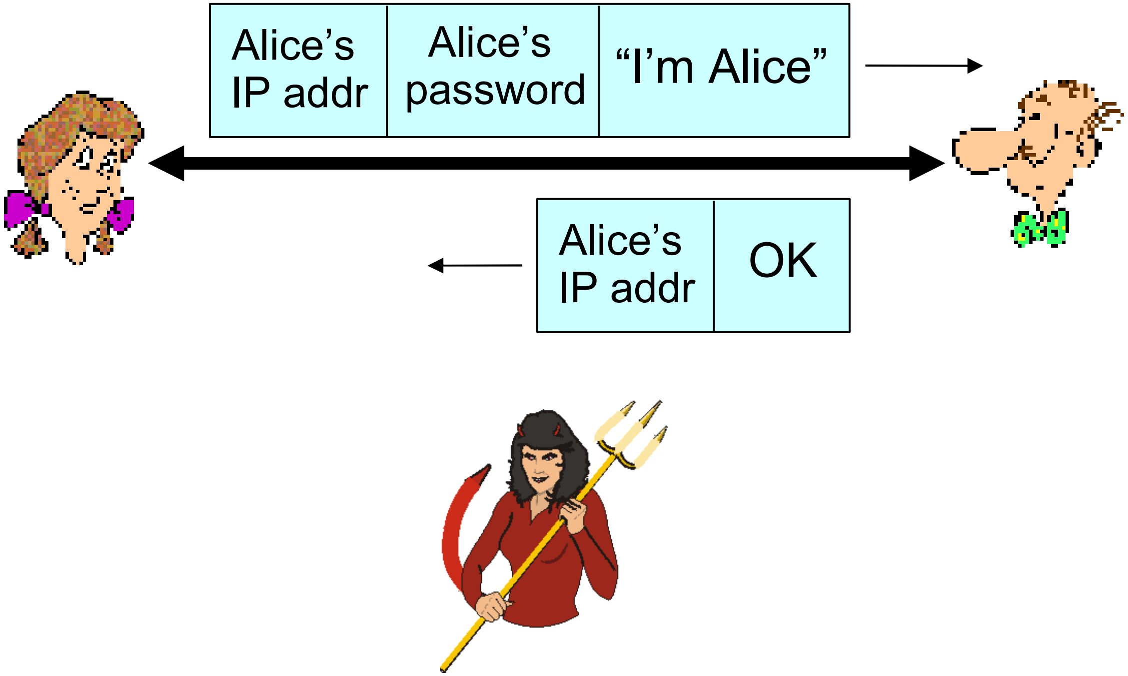
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



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

Authentication: another try

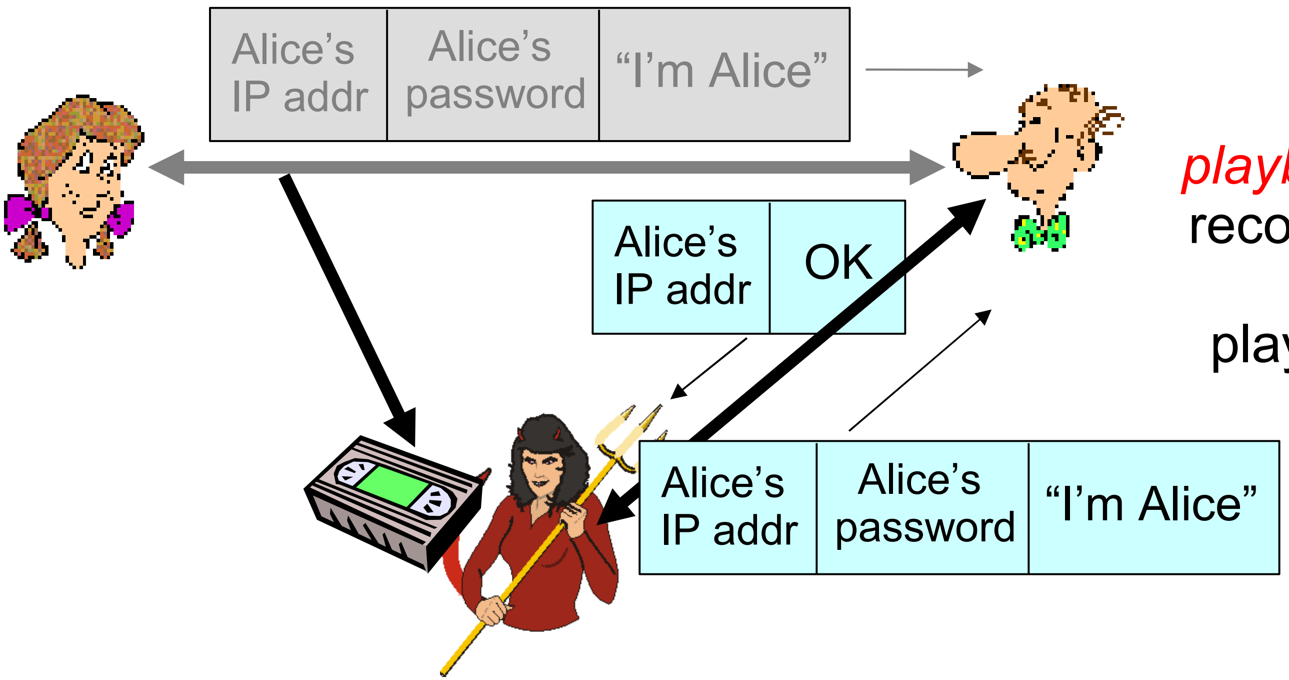
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

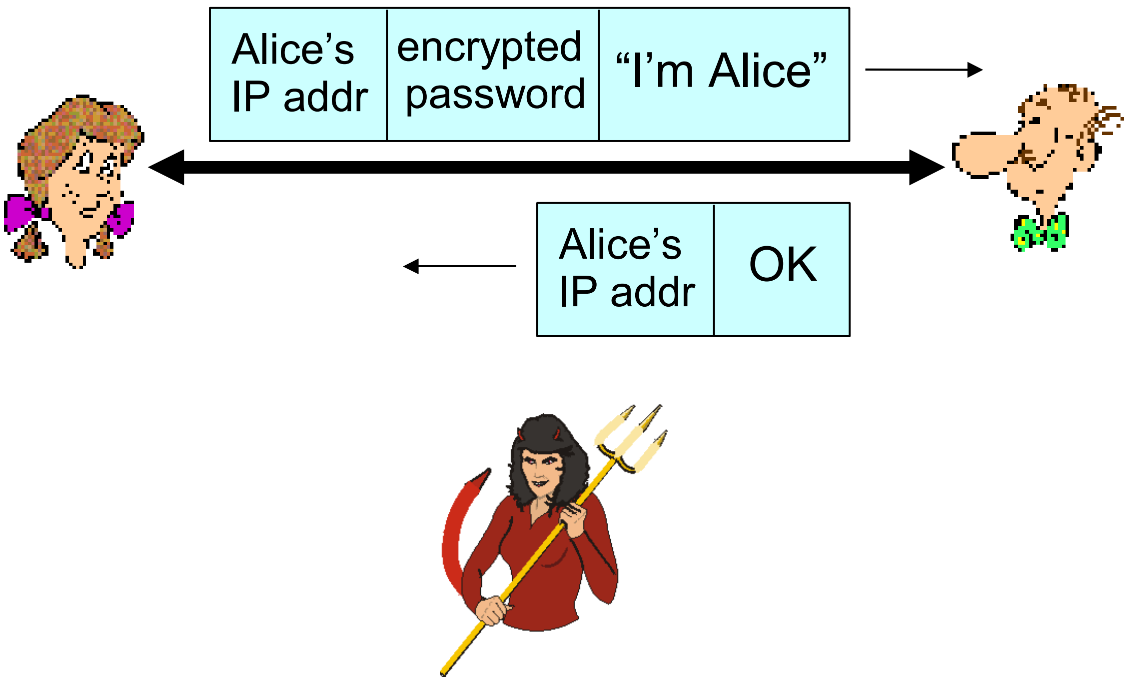
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

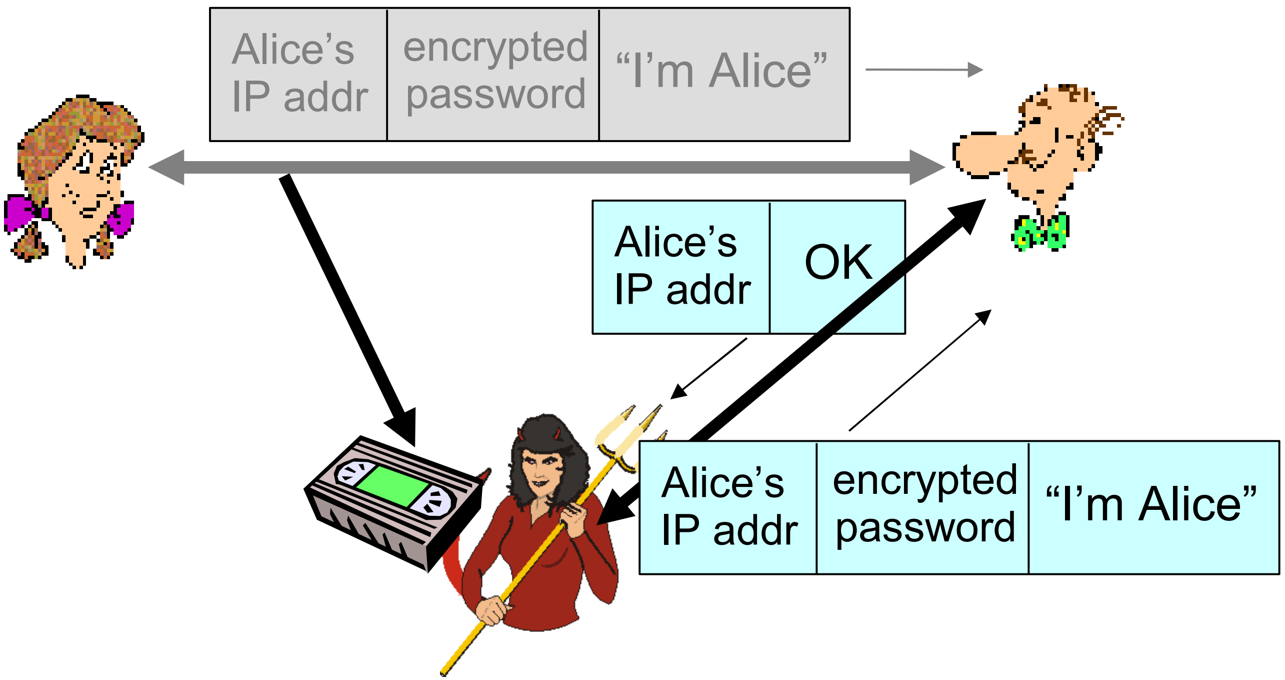
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: another try

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



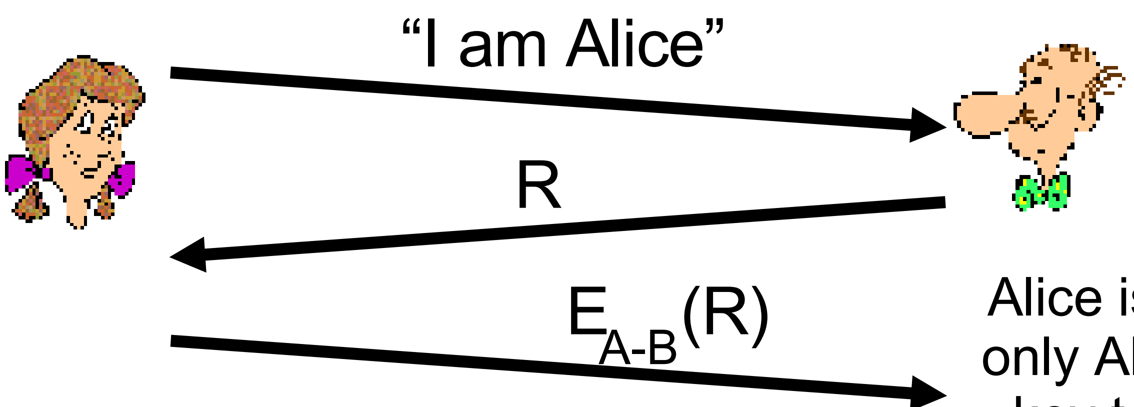
record and playback still works!

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!

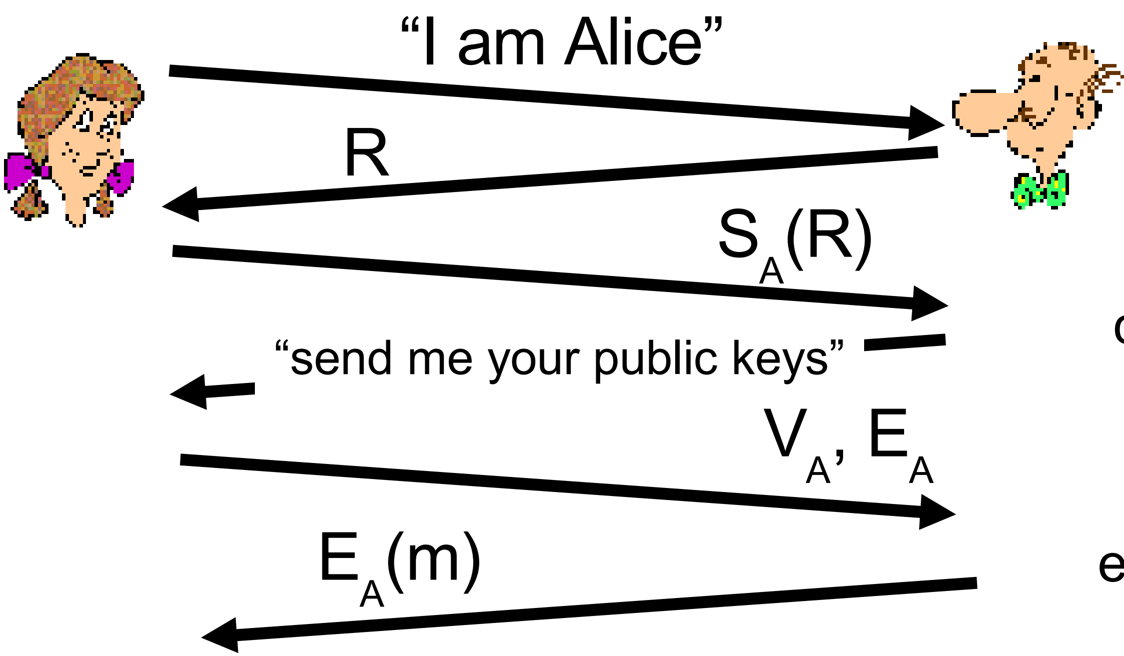
Failures, drawbacks?

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

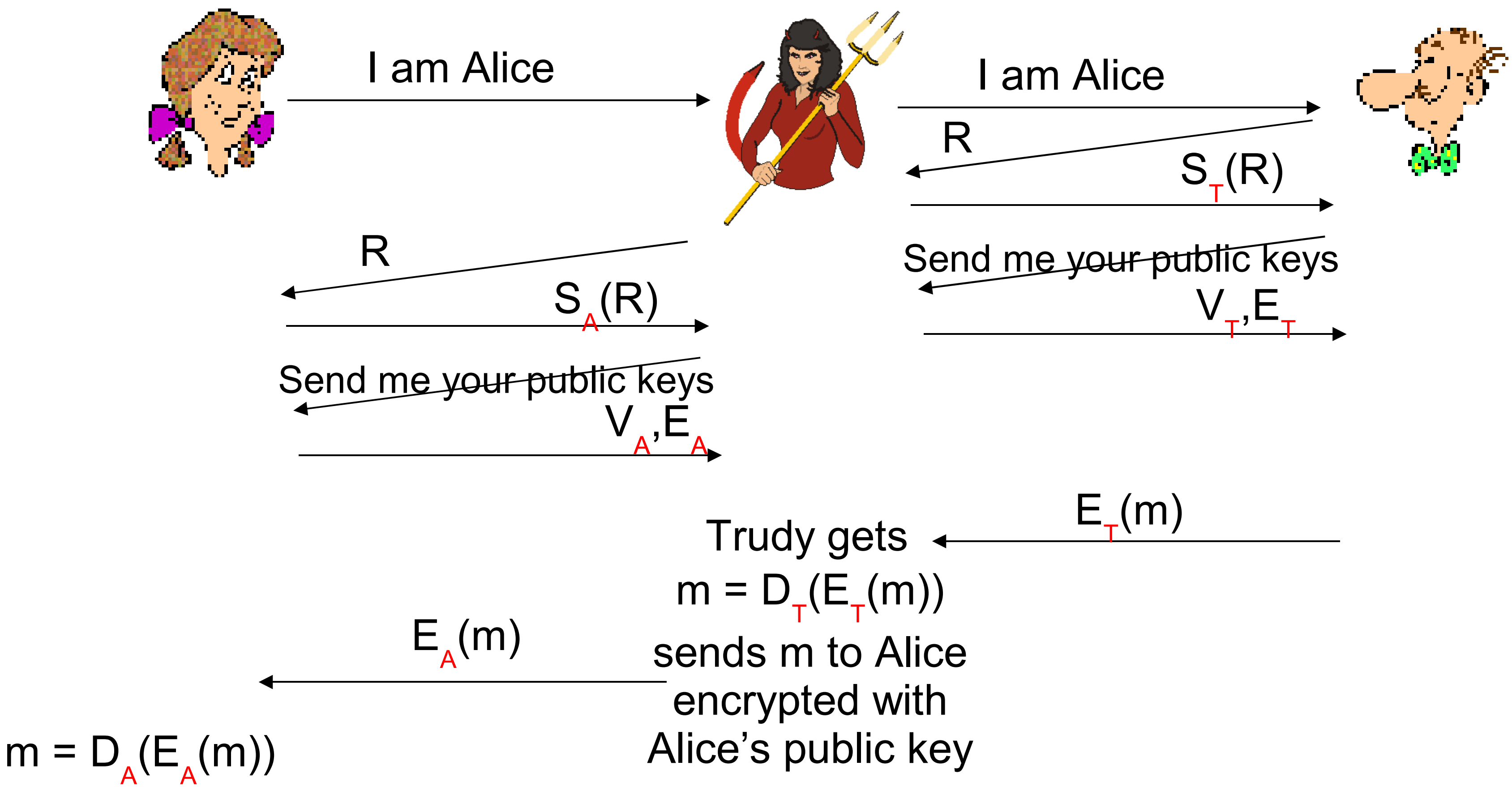
$$V_A(S_A(R)) = R$$

and knows only Alice could have the private key that signed R

Bob then sends encrypted messages to Alice

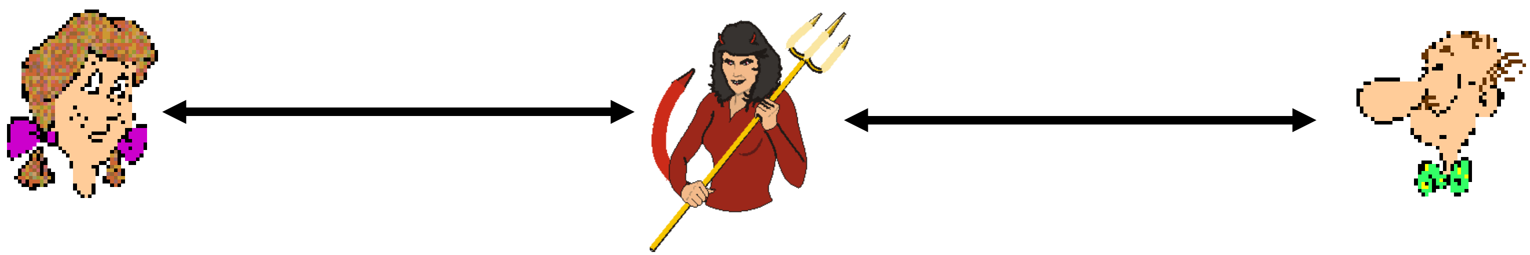
ap5.0: security hole

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)
- The problem is that Trudy receives all messages as well!

Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Authentication
8.4 Integrity
8.5 Key distribution and certification
8.6 Access control: firewalls
8.7 Attacks and counter measures
8.8 Security in many layers

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:

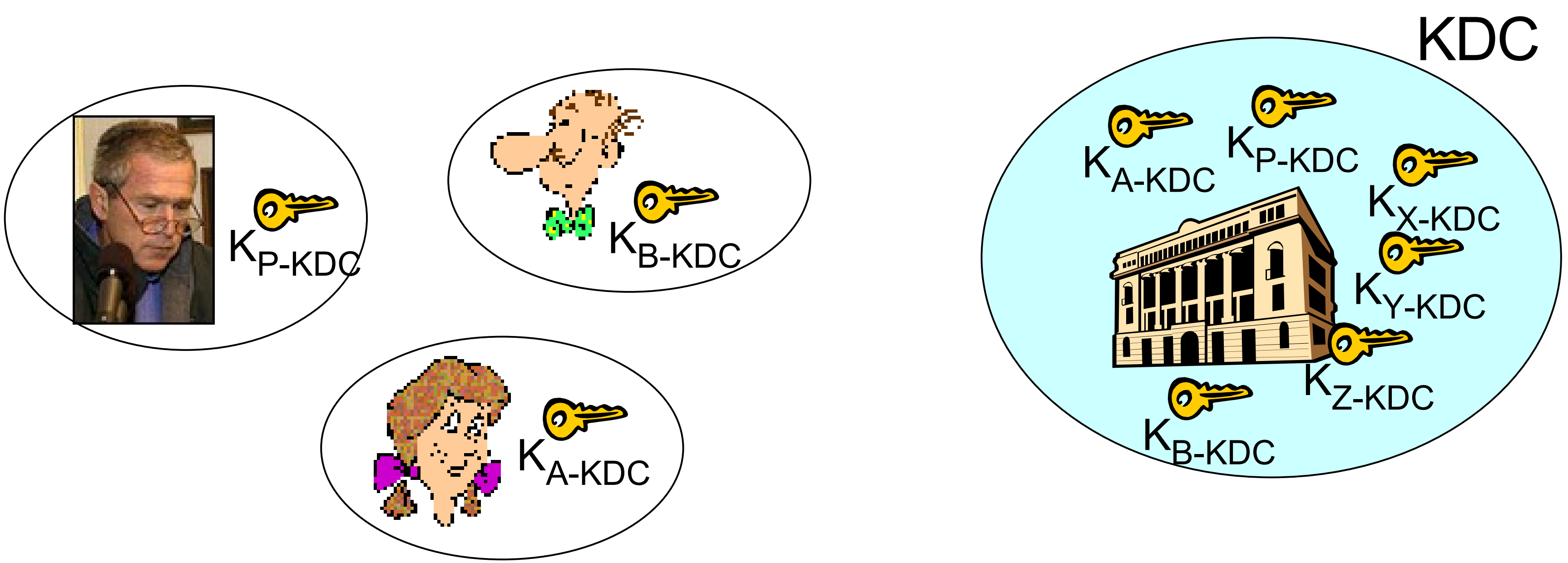
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Solution:

- trusted certification authority (CA)

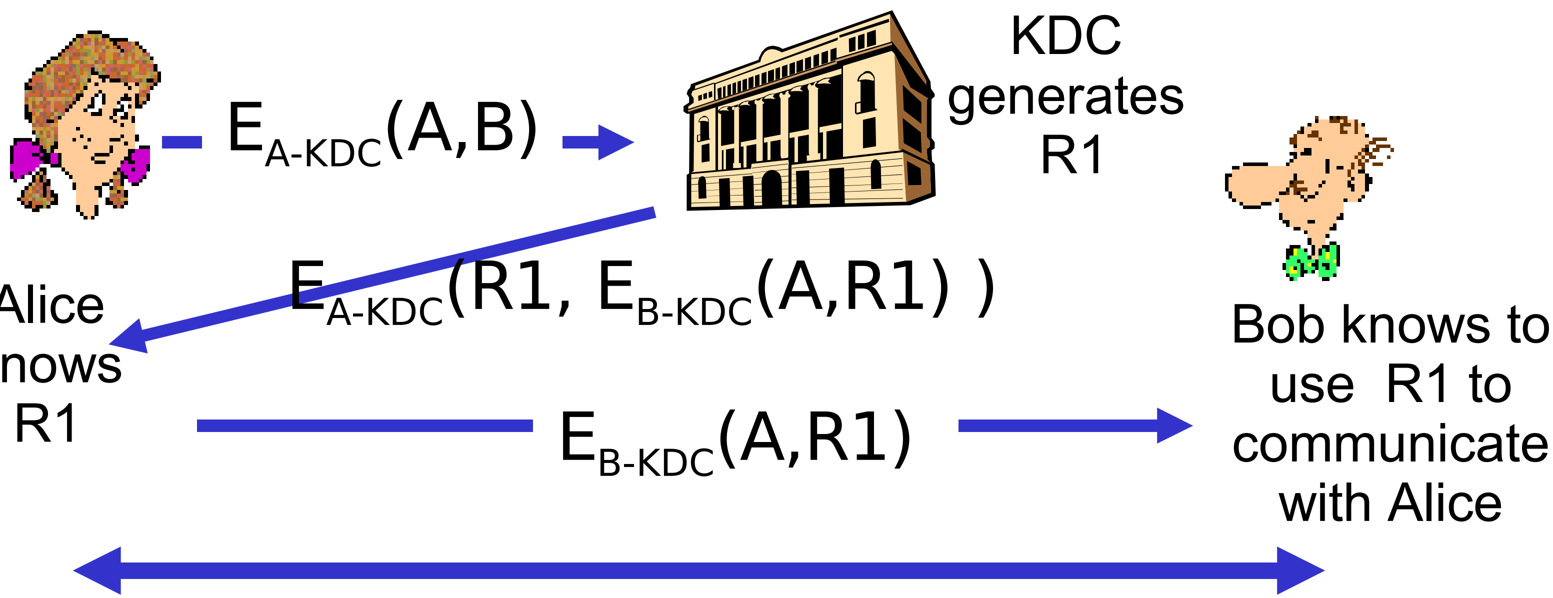
Key Distribution Center (KDC)

- 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)

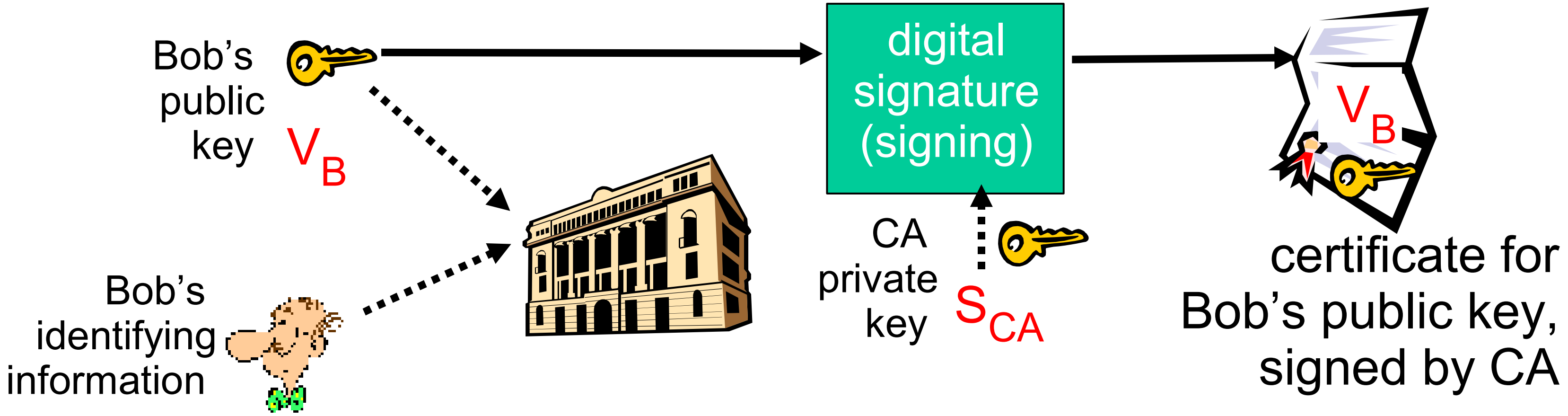
Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



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

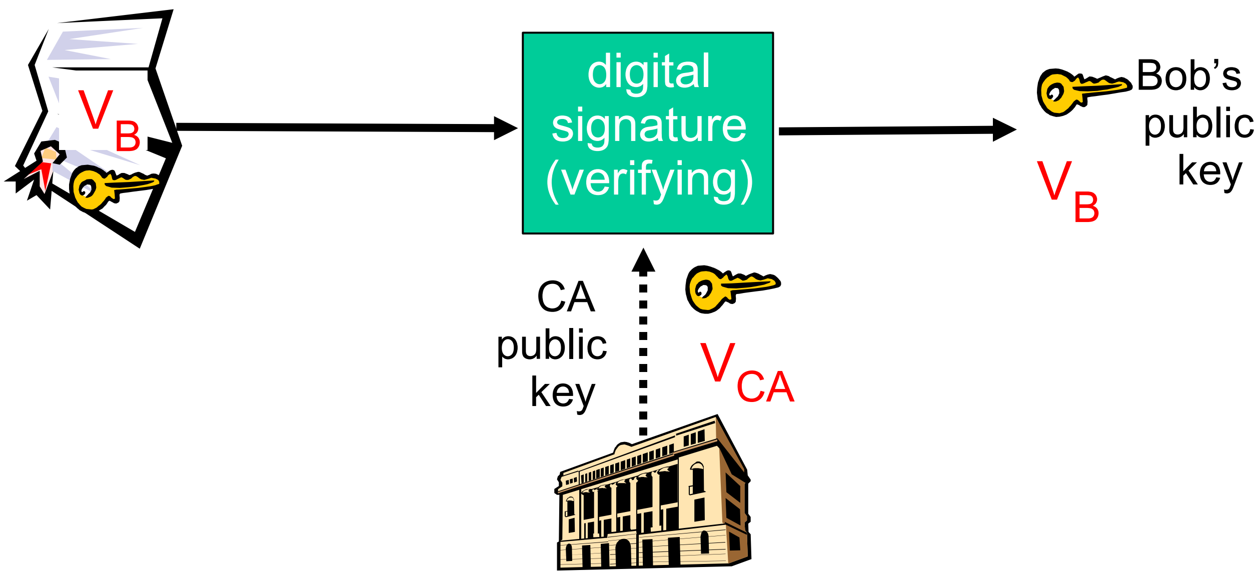
Certification Authorities

- **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”



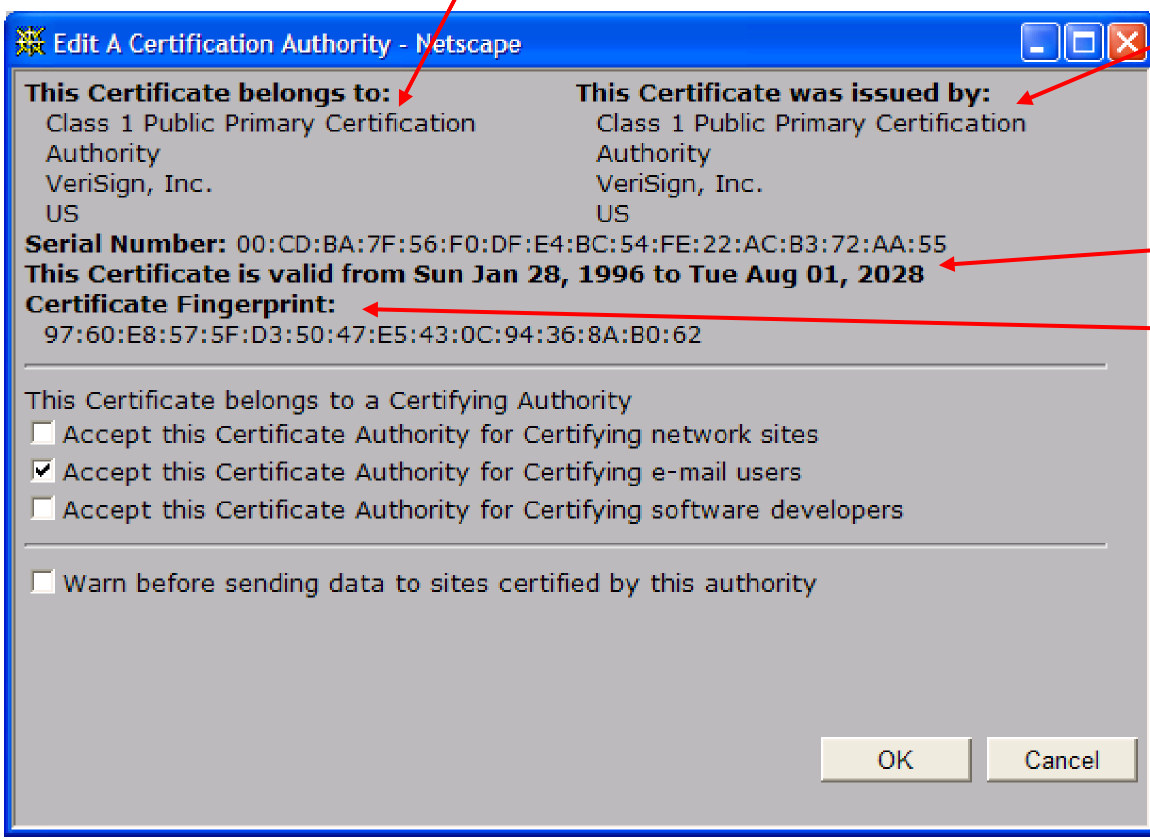
Certification Authorities

- 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



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

Recap

- Message Integrity
- Authentication
- Key distribution and certification

Next time

- Firewalls
- Attacks and countermeasures
- Security in many layers