Security

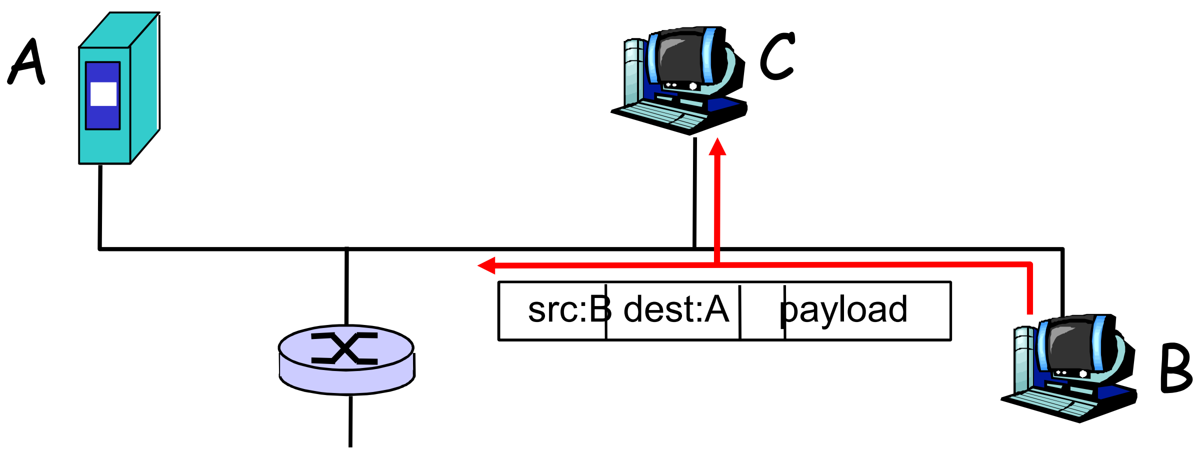
What can Trudy do?

- *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)

http://www.kb.cert.org/vuls/

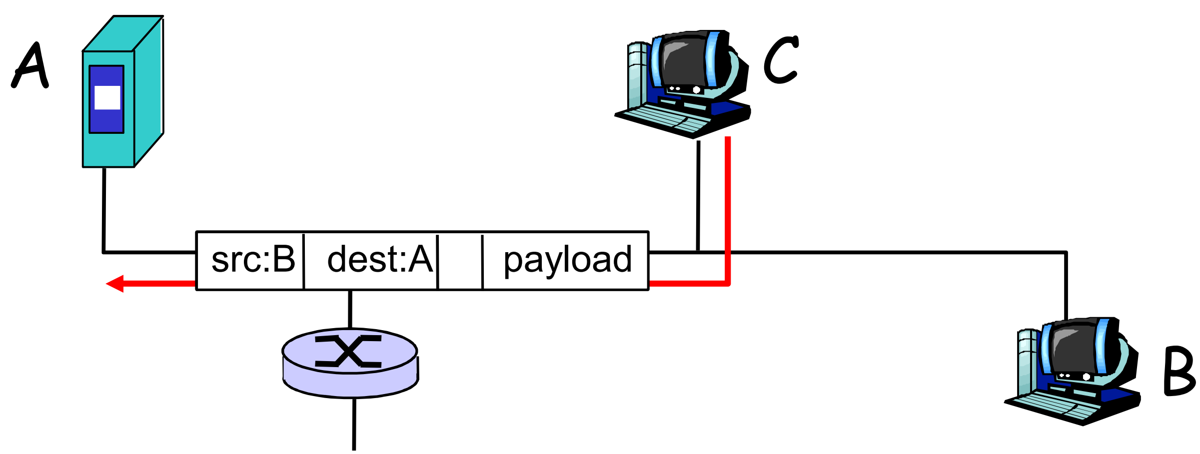
Internet security threats: eavesdrop

- <u>Packet sniffing:</u>
  - broadcast media (but can be done in switched fabric too!)
  - promiscuous NIC reads all packets passing by
  - can read all unencrypted data (e.g. passwords)
  - e.g.: C sniffs B’s packets
  - Wireshark on the LAN or WiFi

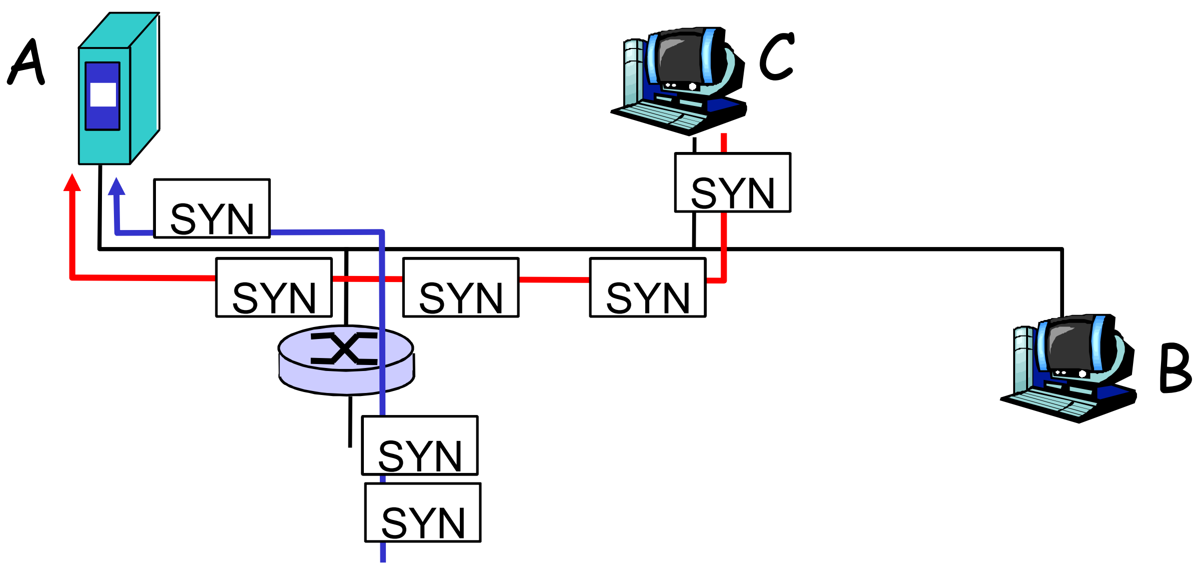


Internet Security Threats: Impersonation

- IP Spoofing:
  - can generate “raw” IP packets directly from application, putting any value into IP source address field
  - receiver can’t tell if source is spoofed
  - e.g.: C pretends to be B



- Denial of service (DOS):
  - flood of maliciously generated packets “swamp” receiver
  - Distributed DOS (DDOS): multiple coordinated sources swamp receiver
  - For example: Exploit protocol specific features and OS implementation decisions
    - Exploit TCP connection’s three way handshake (SYN, SYNACK, ACK).
      - host C and remote host SYN-attack host A
    - Remember: IP Fragmentation? ...

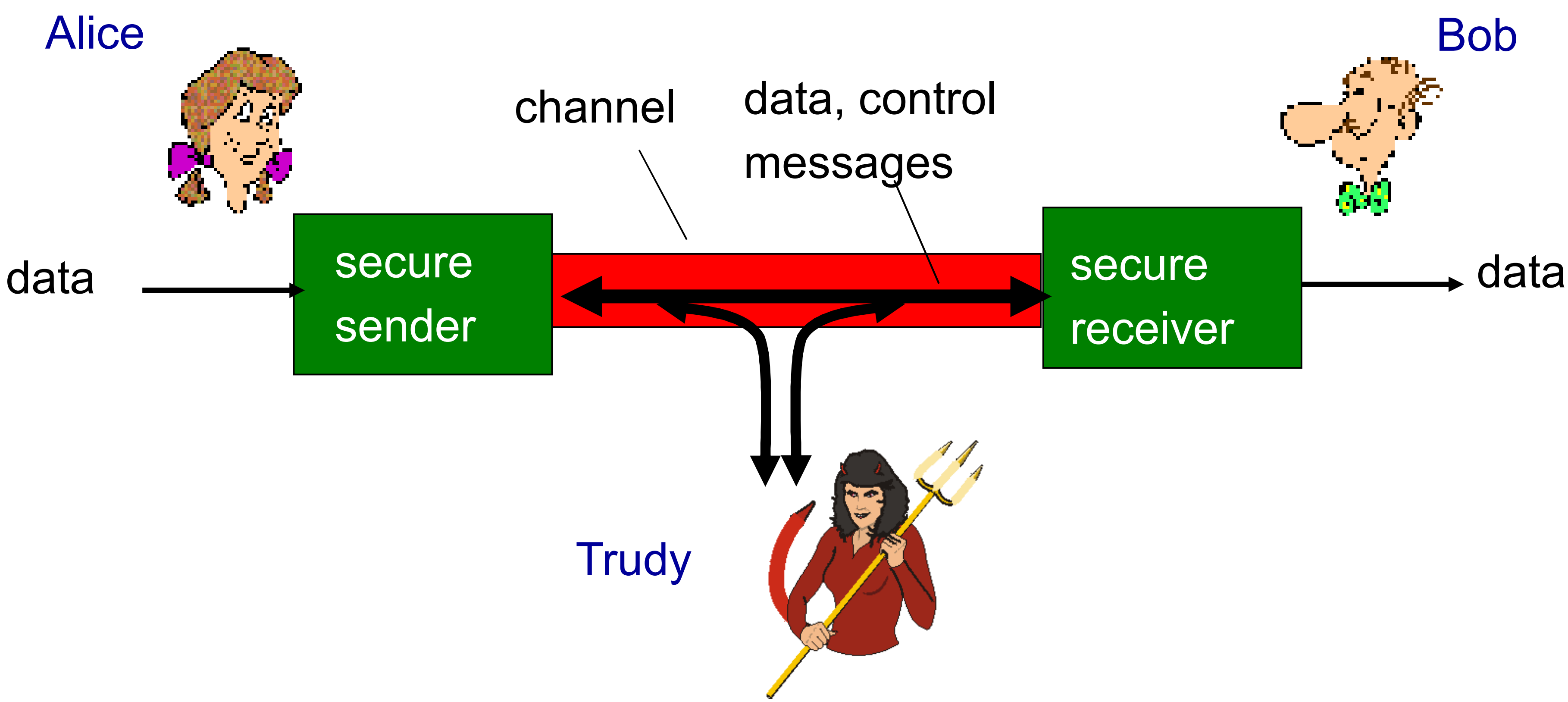


Overview

- understand principles of network security:
  - cryptography and its *many* uses
- security in practice:
  - application layer: secure e-mail
  - transport layer: Internet commerce, SSL
  - network layer: IP security (not examined)

Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages, alter messages



What is network security?

confidentiality: only sender, intended receiver should "understand" message contents

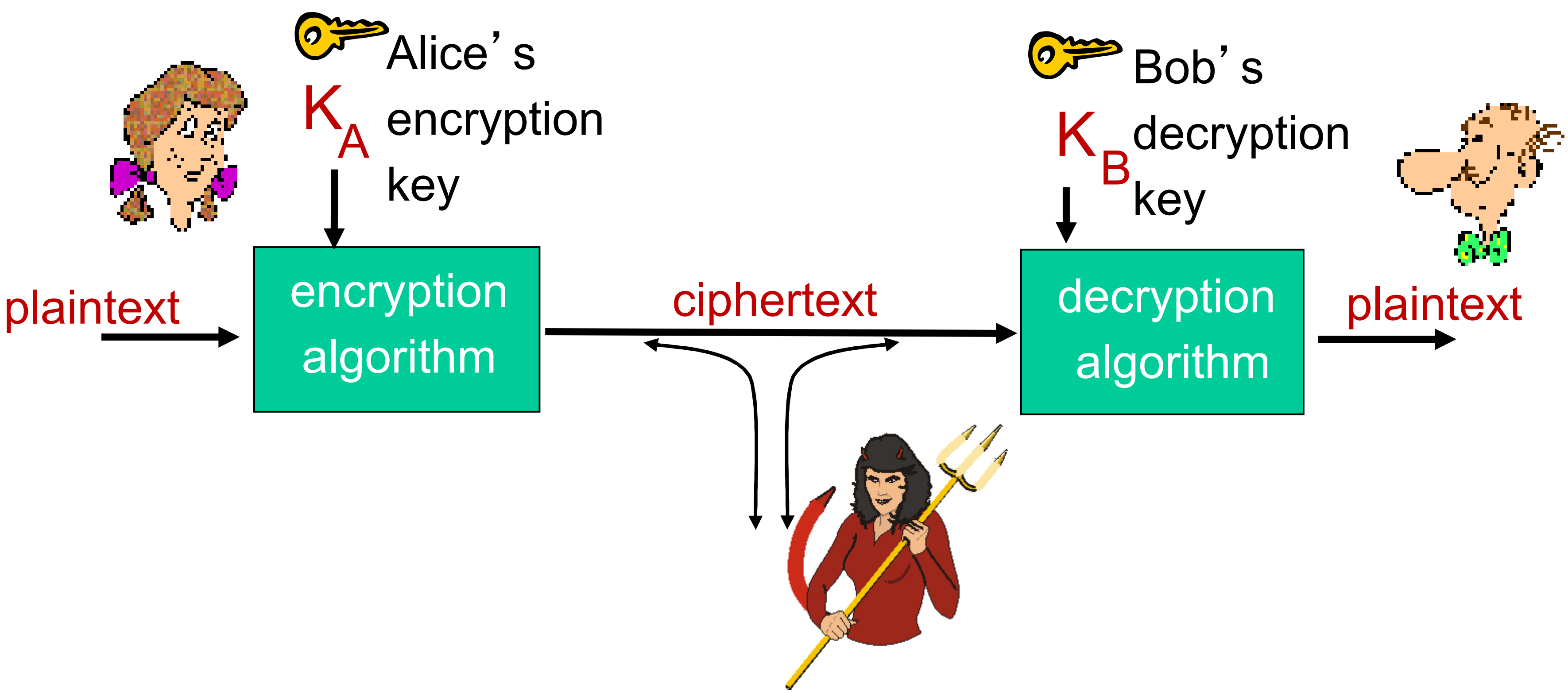
- sender encrypts message
- receiver decrypts message

authentication: sender, receiver want to confirm identity of each other

message integrity: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

access and availability: services must be accessible and available to users

The language of cryptography

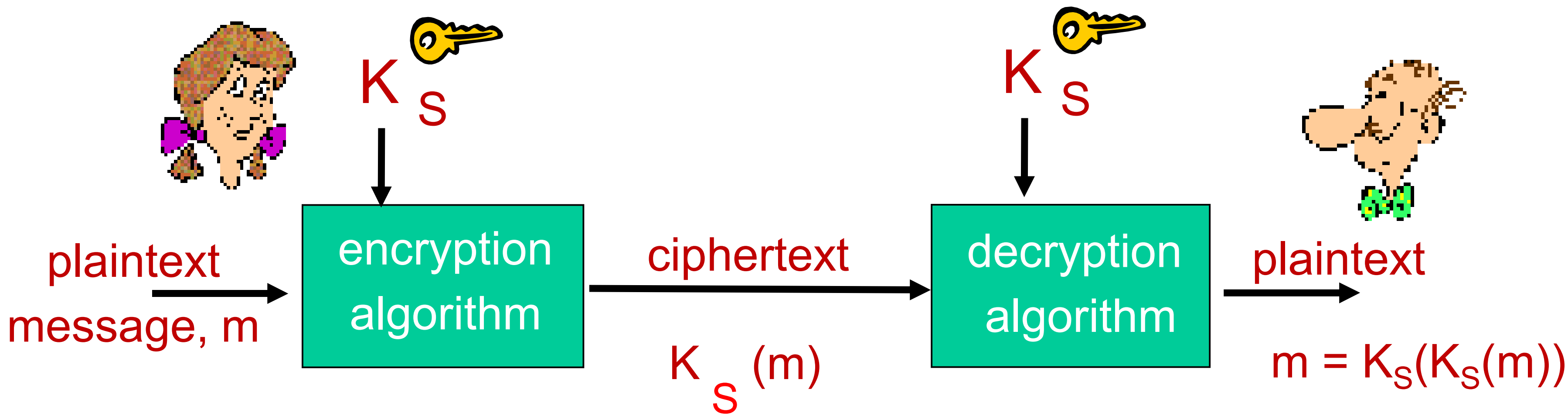


m plaintext message

$K_A(m)$ ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

Substitution cipher: substituting one thing for another

- *mono*alphabetic cipher: substitute *one* letter for another

```
plaintext:   abcdefghijklmnopqrstuvwxyz

Ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Symmetric key cryptography

Substitution cipher: substituting one thing for another

- *mono*alphabetic cipher: substitute *one* letter for another

```
 plaintext:  abcdefghijklmnopqrstuvwxyz

Ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
  Plaintext: bob. i love you. alice
 ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key?

Symmetric key cryptography

Substitution cipher: substituting one thing for another

- *mono*alphabetic cipher: substitute *one* letter for another

```
 plaintext:  abcdefghijklmnopqrstuvwxyz

Ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key: mapping from set of 26 letters to set of 26 letters

Symmetric key cryptography

Substitution cipher: substituting one thing for another

– *mono*alphabetic cipher: substitute *one* letter for another

```
plaintext:  abcdefghijklmnopqrstuvwxyz

Ciphertext: mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key: mapping from set of 26 letters to set of 26 letters

Q: How hard to break this simple cipher?:

- brute force (how hard?)
- other?

Substitution cipher: substituting one thing for another

– *mono*alphabetic cipher: substitute *one* letter for another

```
 plaintext:   abcdefghijklmnopqrstuvwxyz

Ciphertext:   mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

```
 Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Q: How hard to break this simple cipher?:

- brute force (how hard?)
- other?

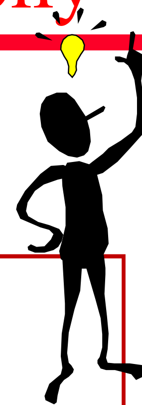
Modern examples: AES (128, 192, 256), 3DES, RC4

Public Key Cryptography

symmetric key crypto

- requires sender, receiver know shared secret key
- Q: *how to agree on key in first place (particularly if never "met")?*

Public Key Cryptography



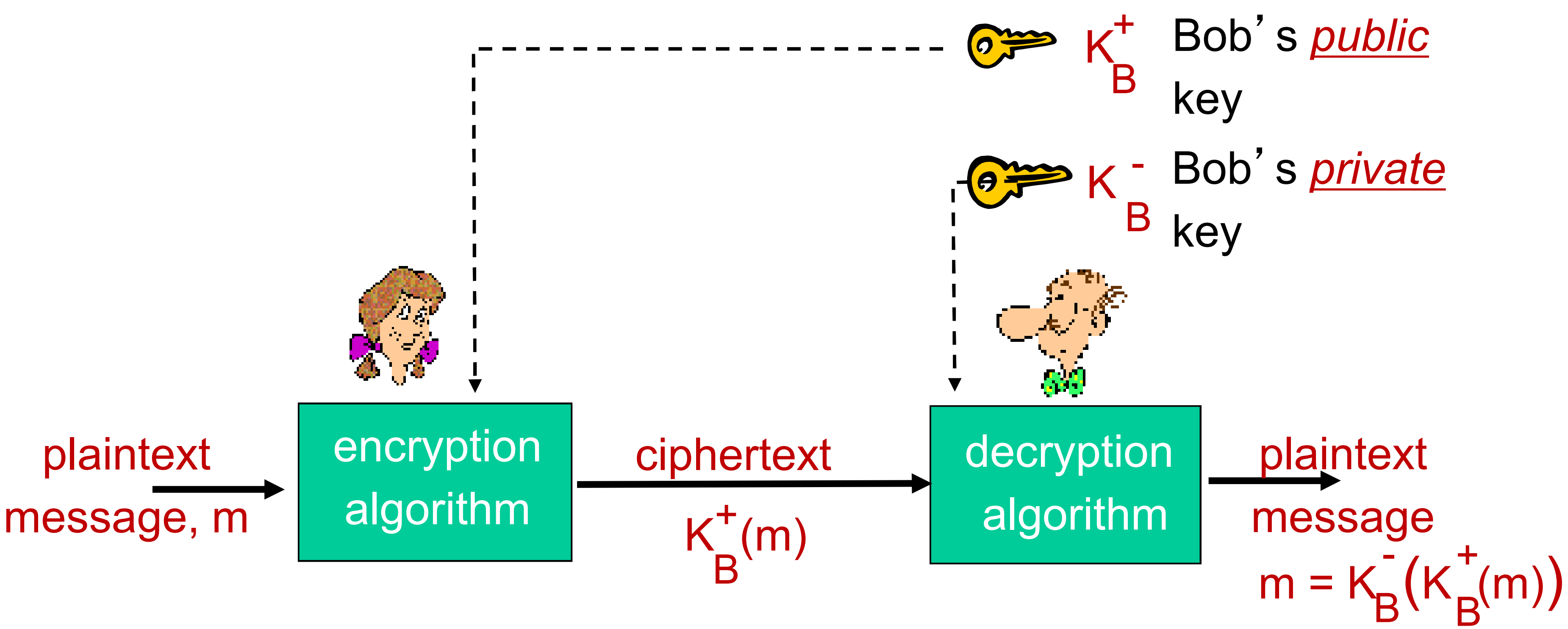
symmetric key crypto

- requires sender, receiver know shared secret key
- Q: *how to agree on key in first place (particularly if never "met")?*

public key crypto

- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Public key cryptography



Public key encryption algorithms

requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

Modern examples: *RSA:* Rivest, Shamir, Adelson algorithm (1024), Elliptic Curve Cryptography

RSA: another important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, followed by public key}}$$

result is the same!

RSA in practice: session keys

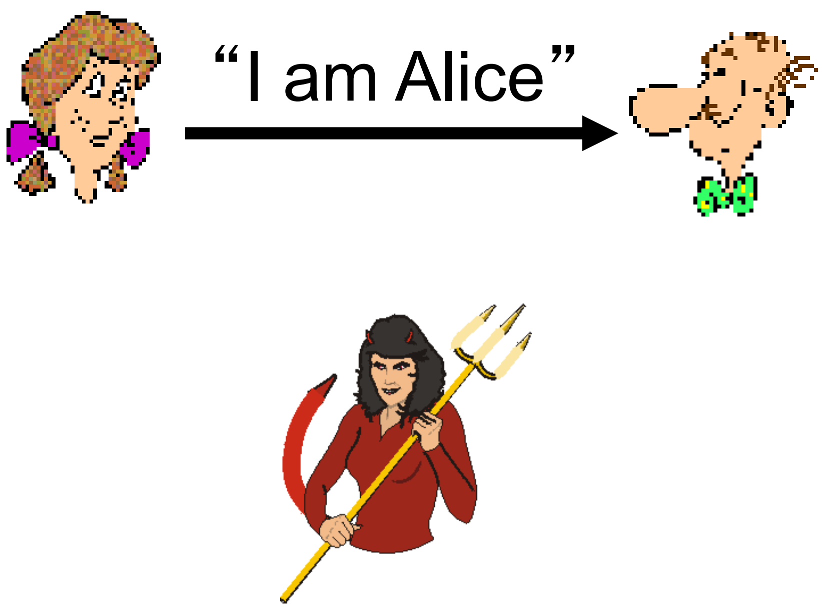
session key, K_S

- Bob and Alice use RSA to exchange a symmetric key K_S
- once both have K_S, they use symmetric key cryptography

Security Services: Authentication

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

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

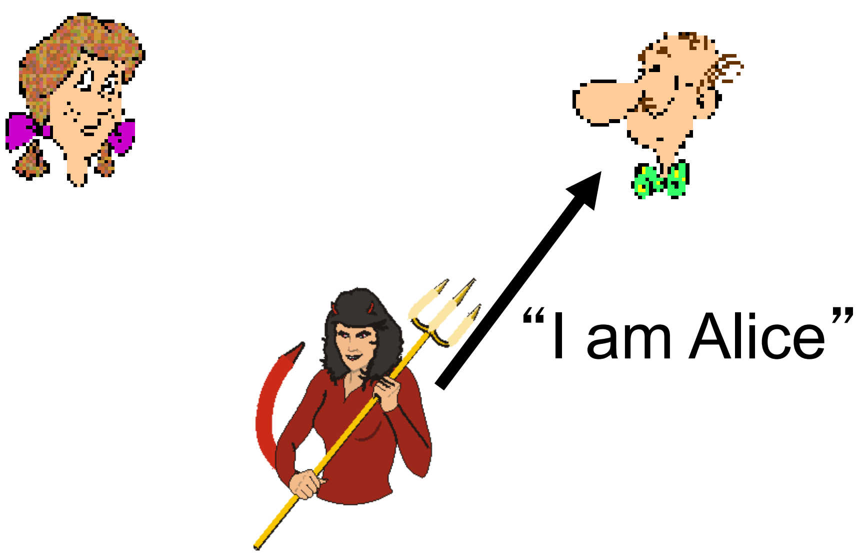


Failure scenario??

Security Services: 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

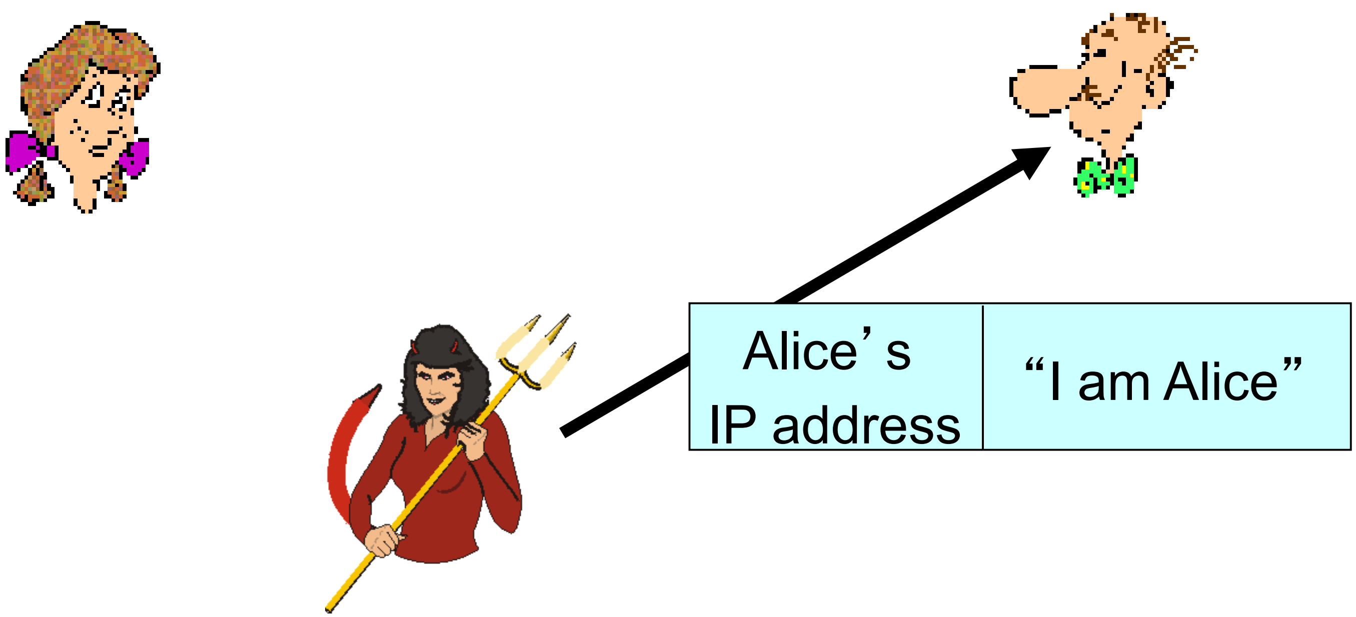
Authentication: another try

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

Failure scenario??

Authentication: another try

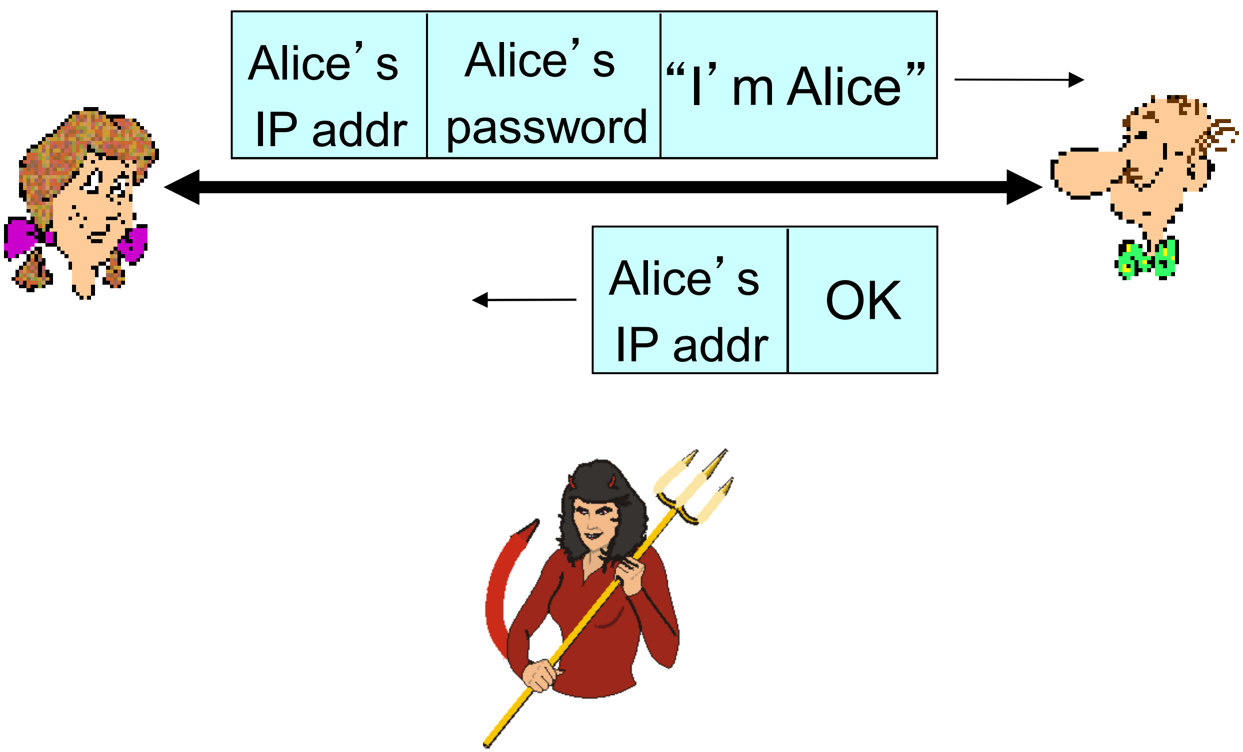
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



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

Authentication: another try

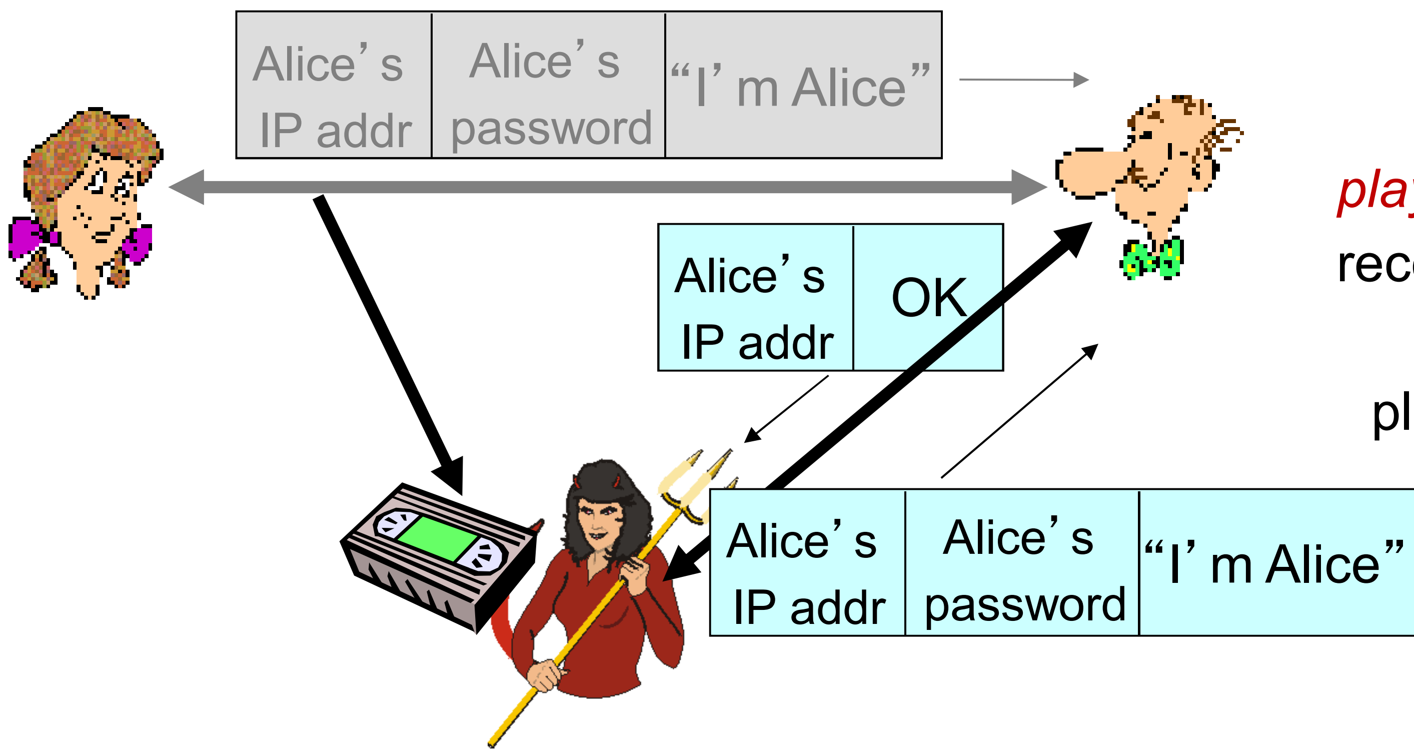
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



Failure scenario??

Authentication: another try

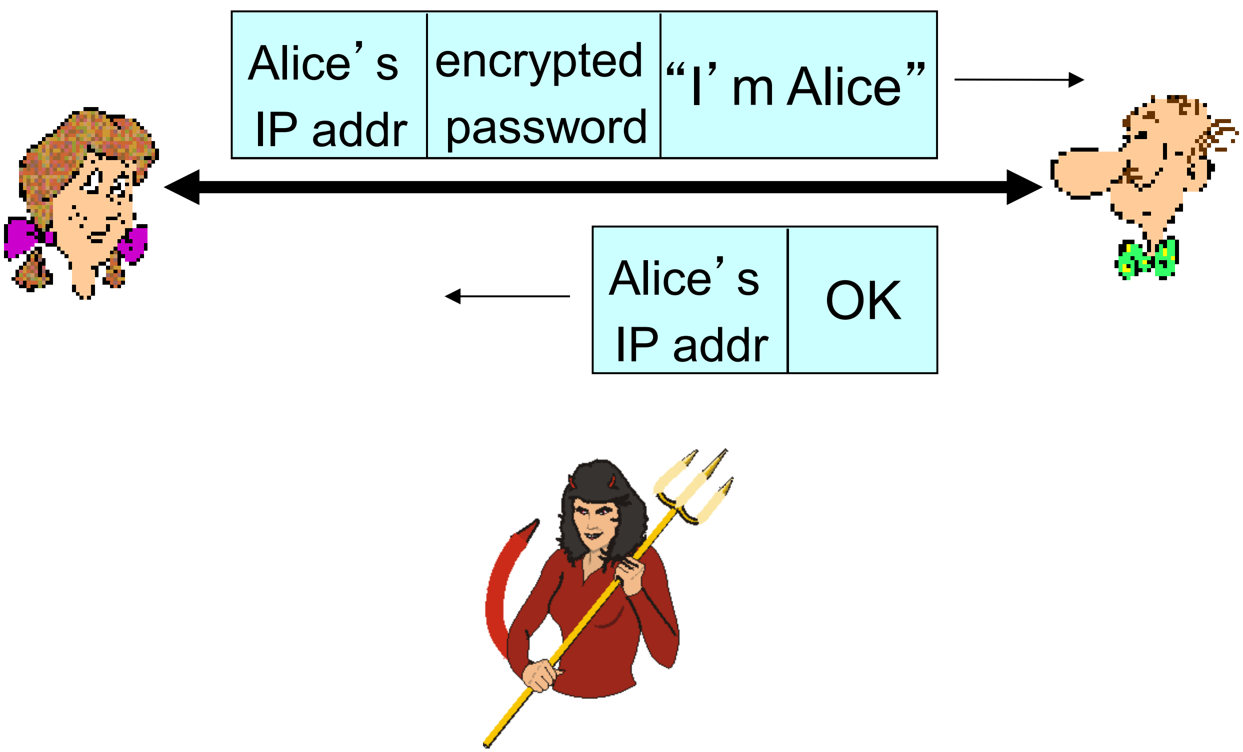
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



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

Authentication: yet another try

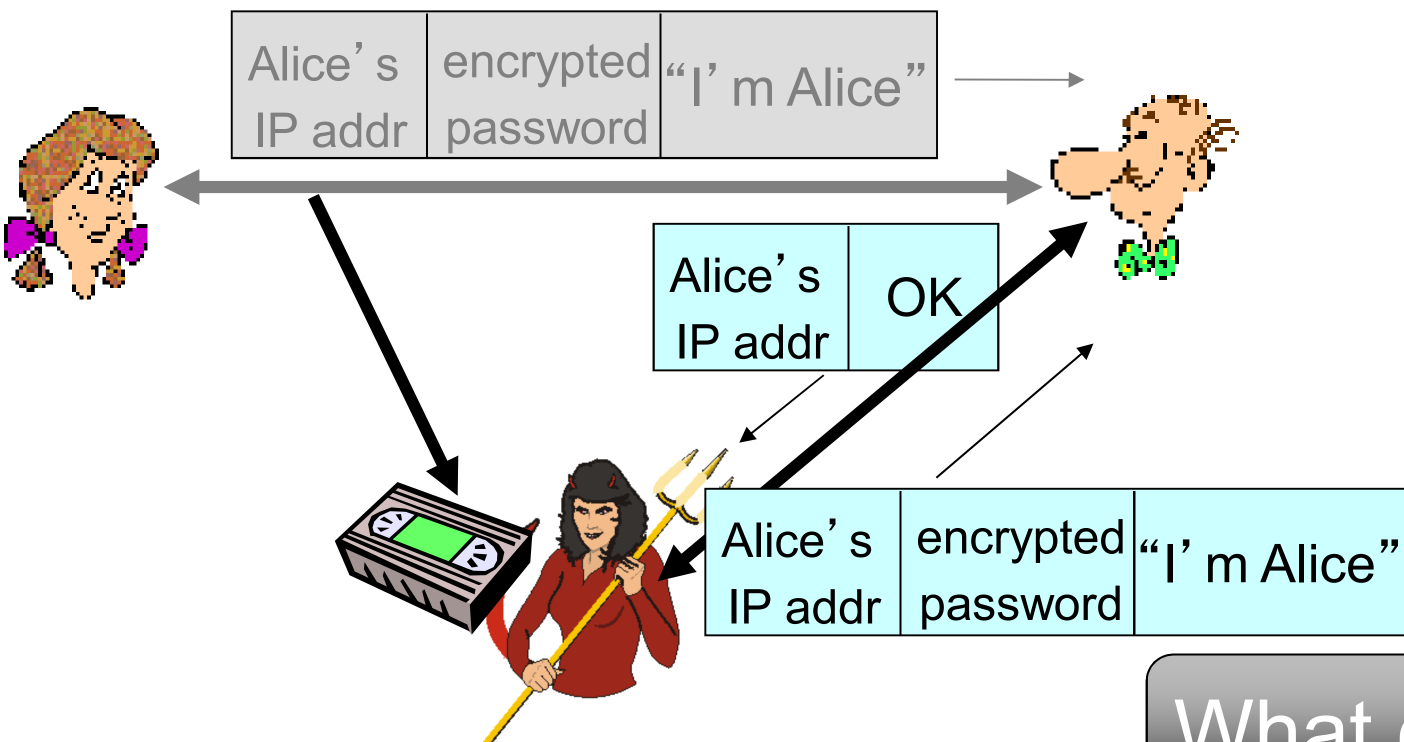
Protocol ap3.1: Alice says “I am Alice” and sends her *encrypted* secret password to “prove” it.



Failure scenario??

Authentication: yet another try

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



record
and
playback
still works!

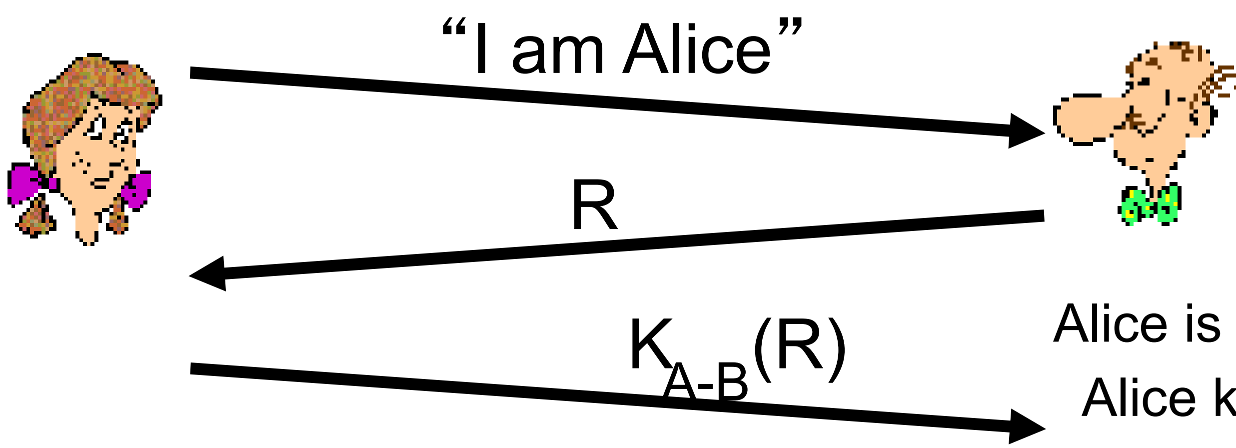
What can we do?

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!

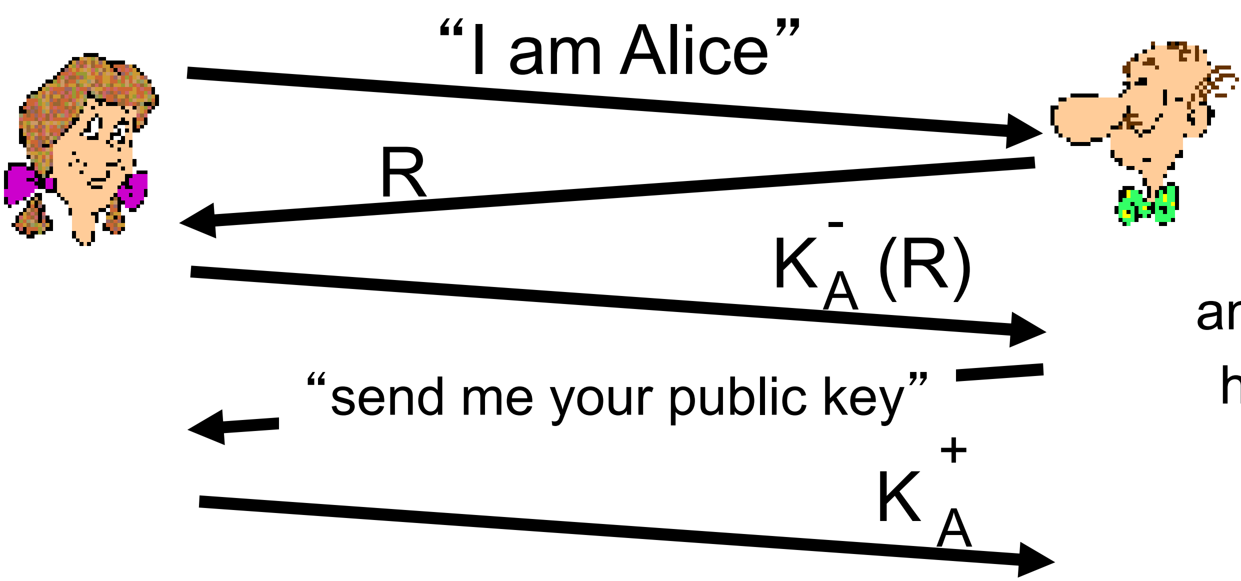
can we authenticate using public key techniques?

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

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

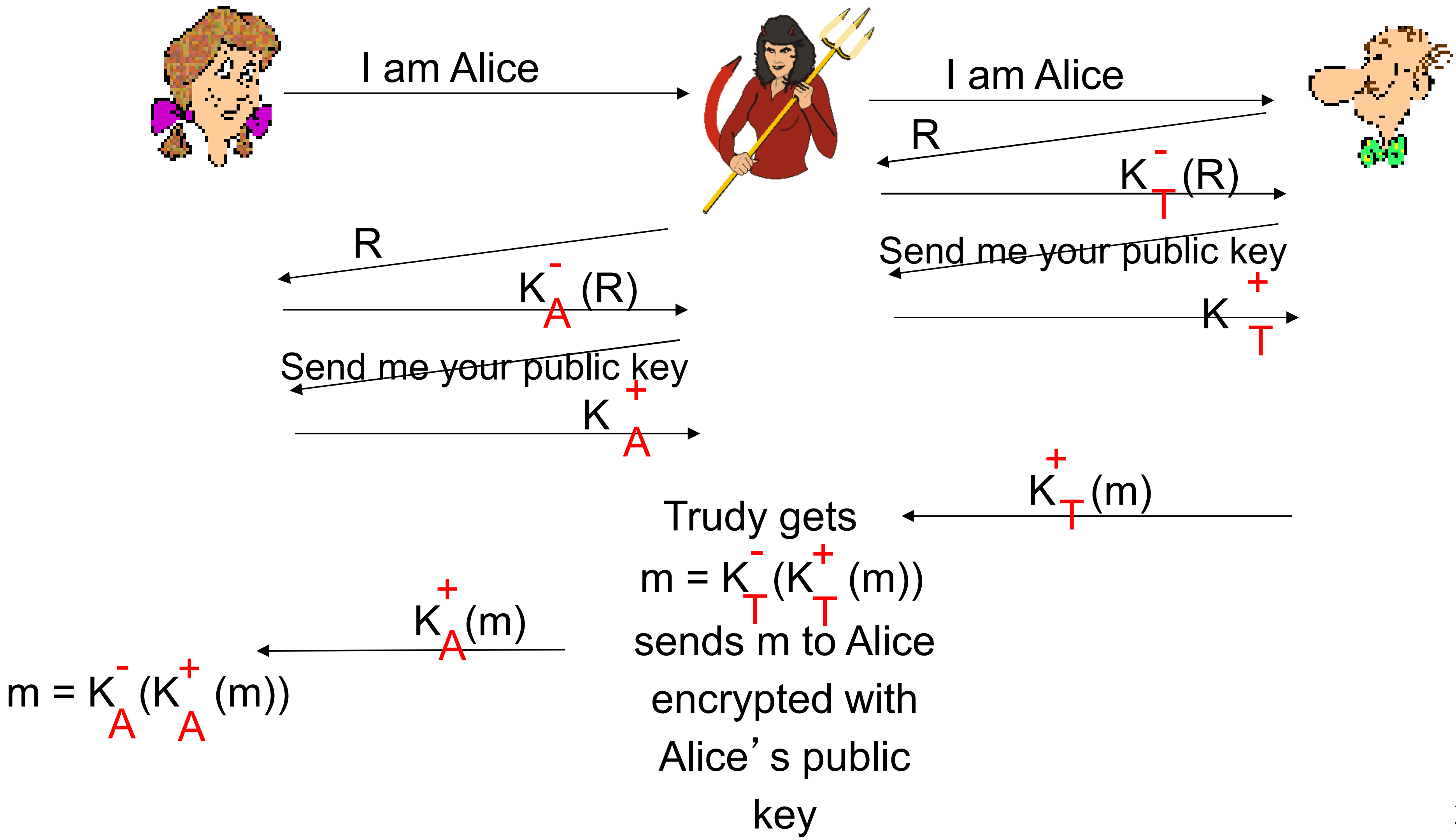
and knows only Alice could have the private key, that encrypted R such that

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

Failures, drawbacks?

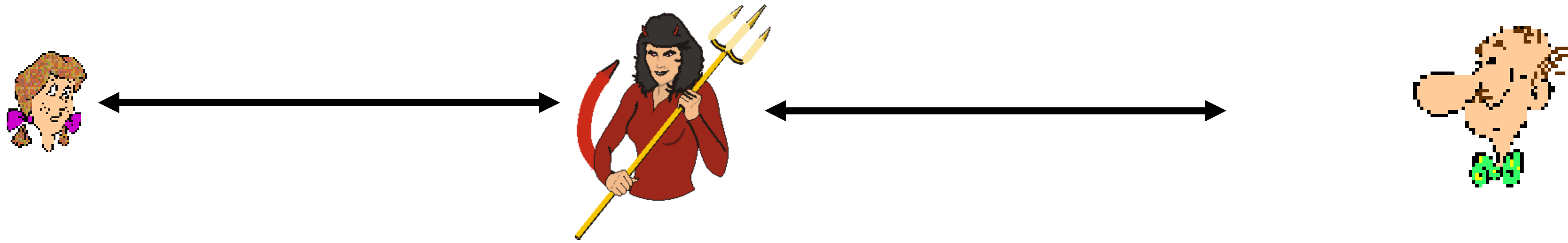
ap5.0: security hole

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



ap5.0: security hole

man (or 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!

Need "certified" public keys

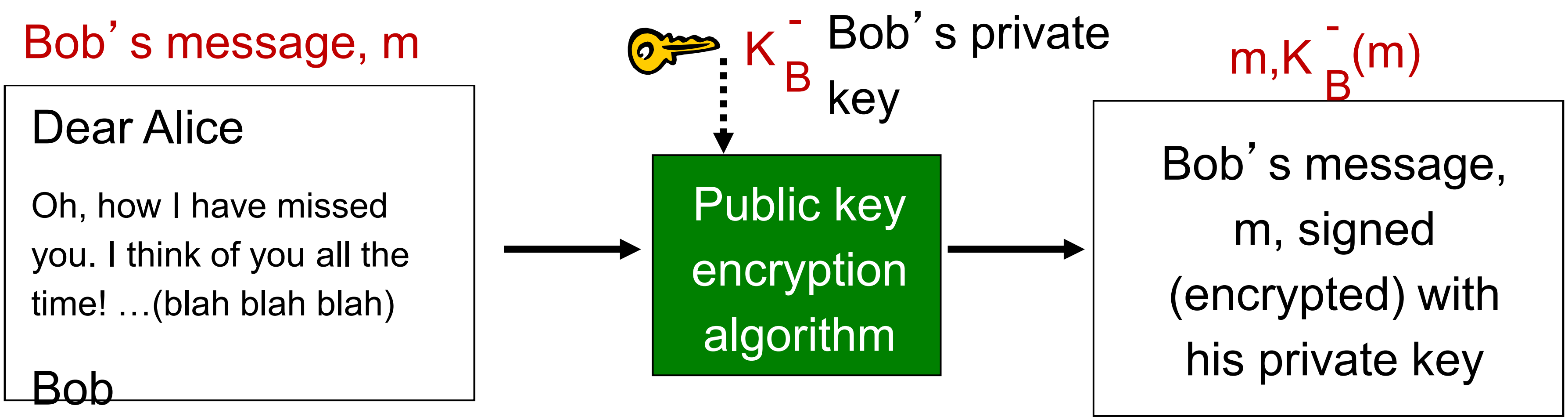
Digital signatures

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

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

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- 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'

Message is verifiable and nonforgeable

Thus allows non-repudiation :

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

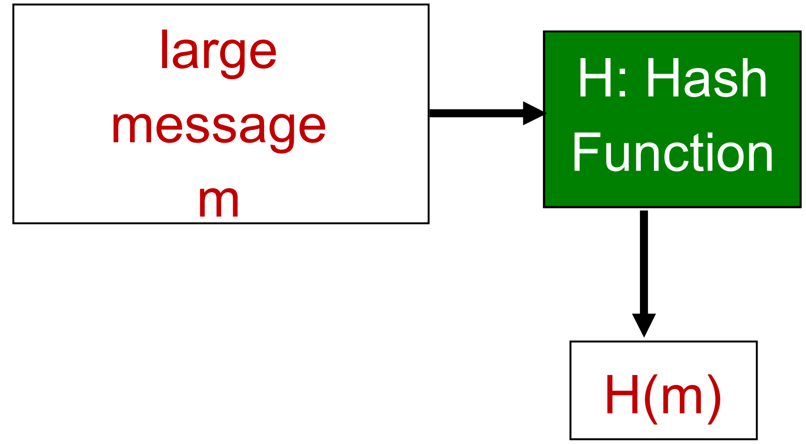
Message digests

Problem?

computationally expensive to public-key-encrypt long messages

need: fixed-length, easy- to-compute digital “fingerprint”

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



Hash function properties:

- produces fixed-size msg digest (fingerprint)

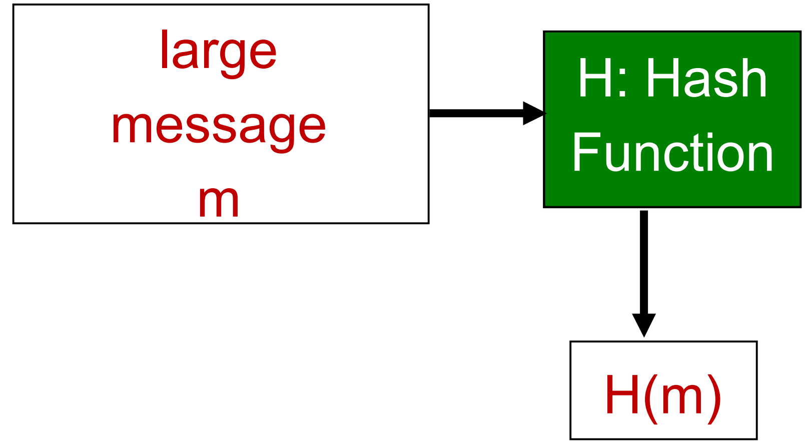
Message digests

Problem?

computationally expensive to public-key-encrypt long messages

need: fixed-length, easy- to-compute digital “fingerprint”

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



Hash function properties:

- 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

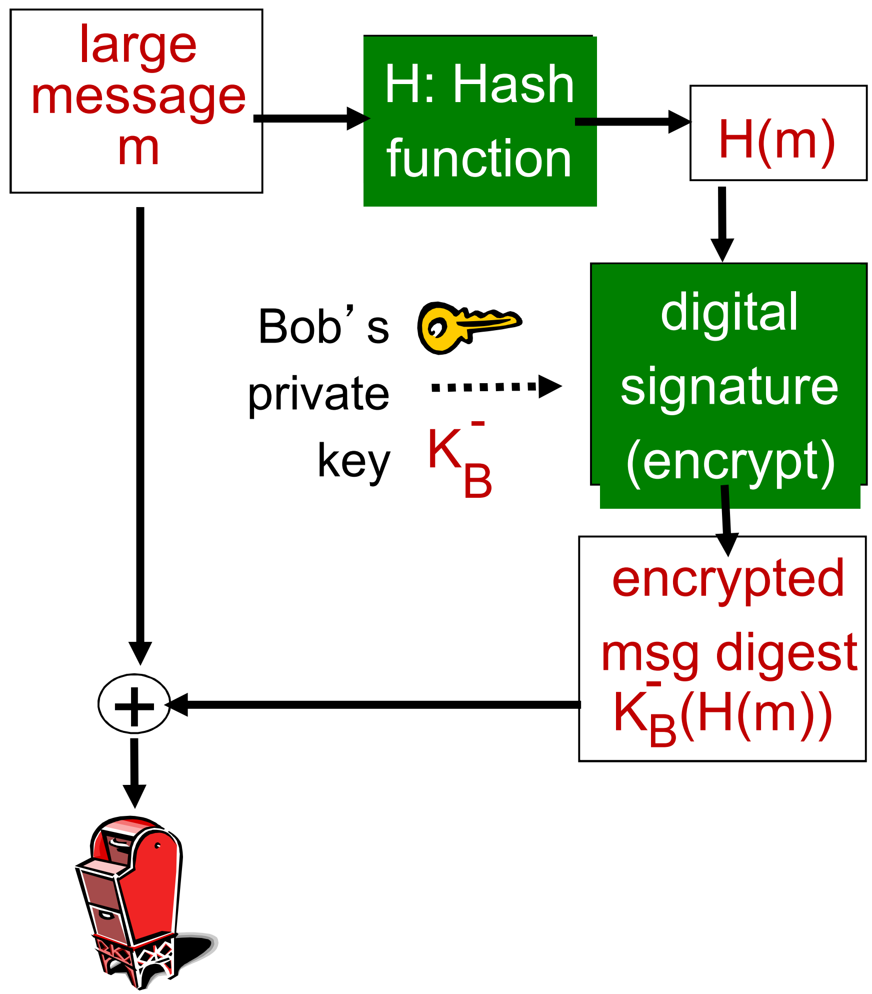
message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

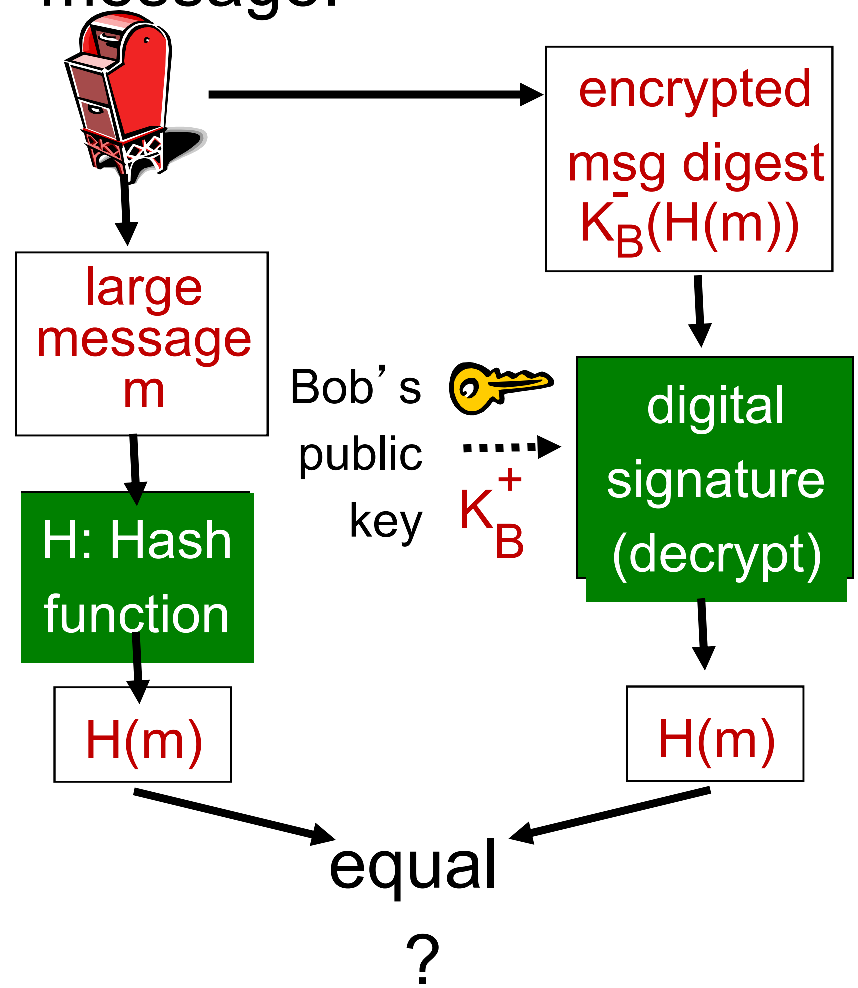
Need to be vigilant

- In Aug 2004
  - MD-5 (computes a 128 bit hash) found to be vulnerable!
    - Can find m' such that MD5(m) = MD5(m')
  - NOT a threat to its use, however as m' is *weird compared to m (bears no useful relationship)*
- *SHA-2 (US federal standard) is considered as more secure*
  - *However it might have a similar vulnerability to SHA-1 (2005 attack)*
  - *We need to keep our eyes open!*

Digital signature = signed message digest

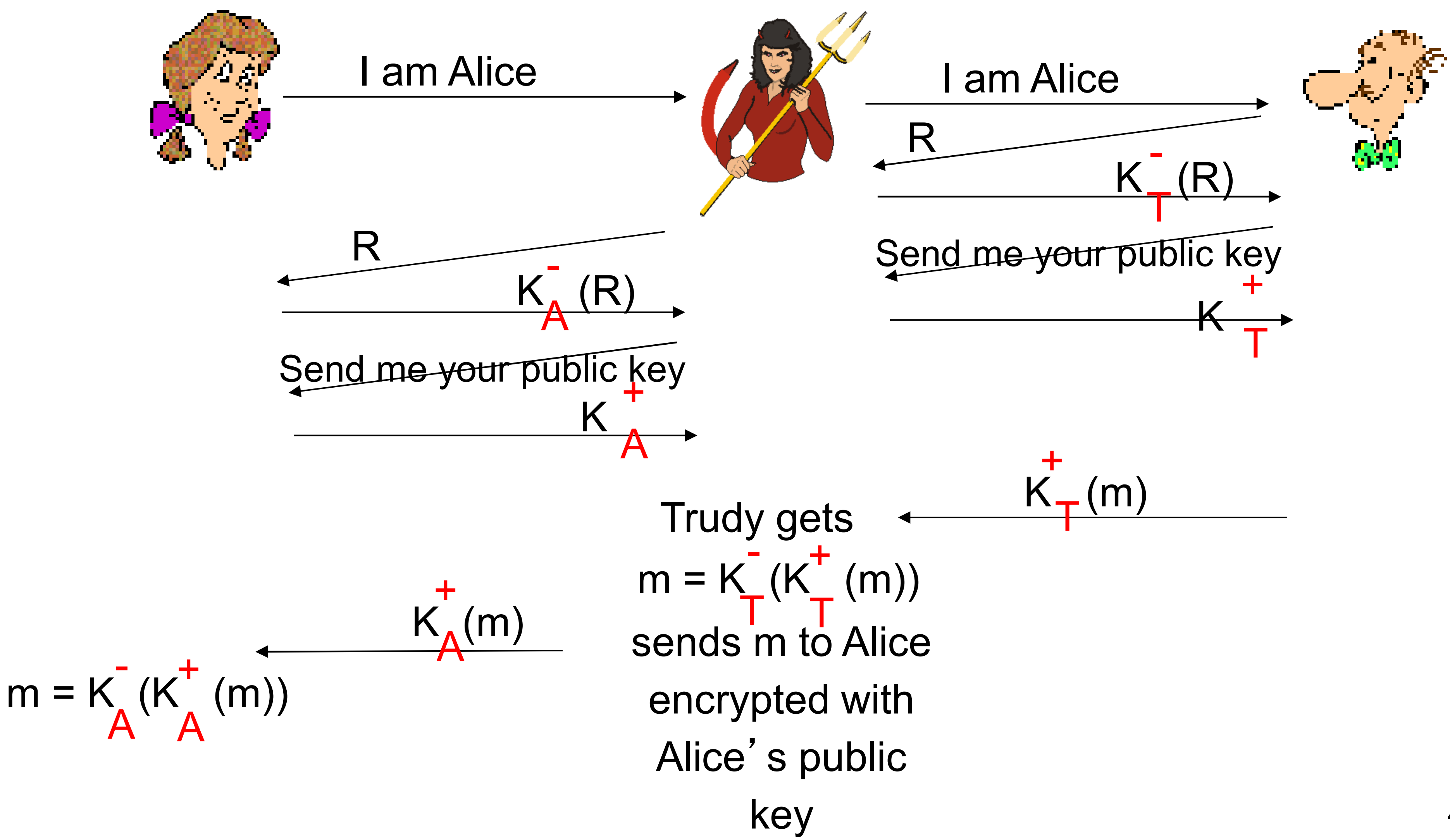


Alice verifies signature, integrity of digitally signed message:



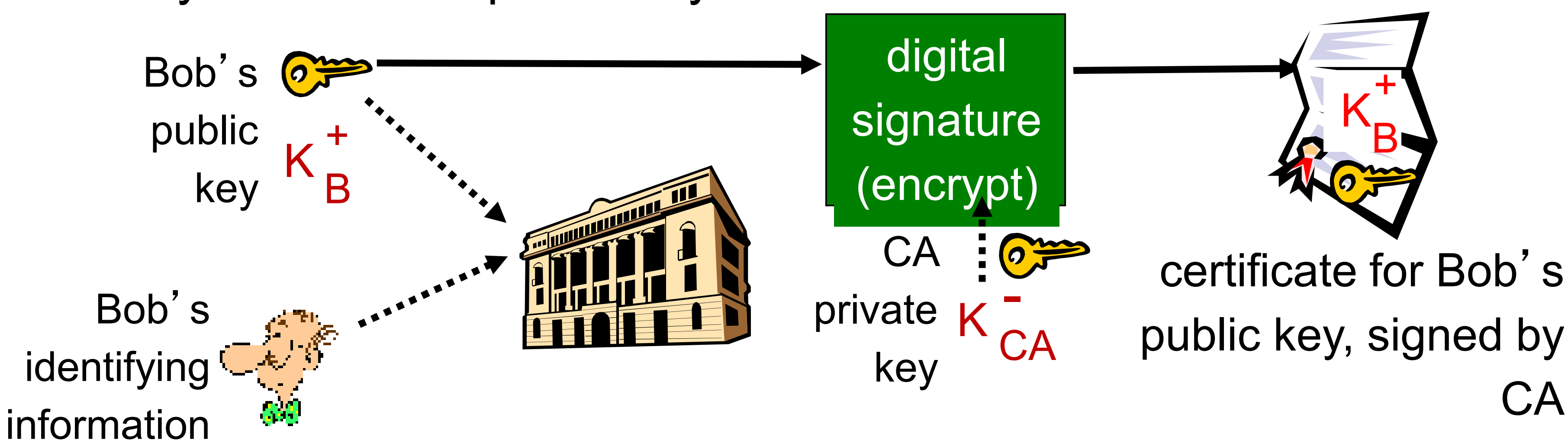
Recall: ap5.0 security hole

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



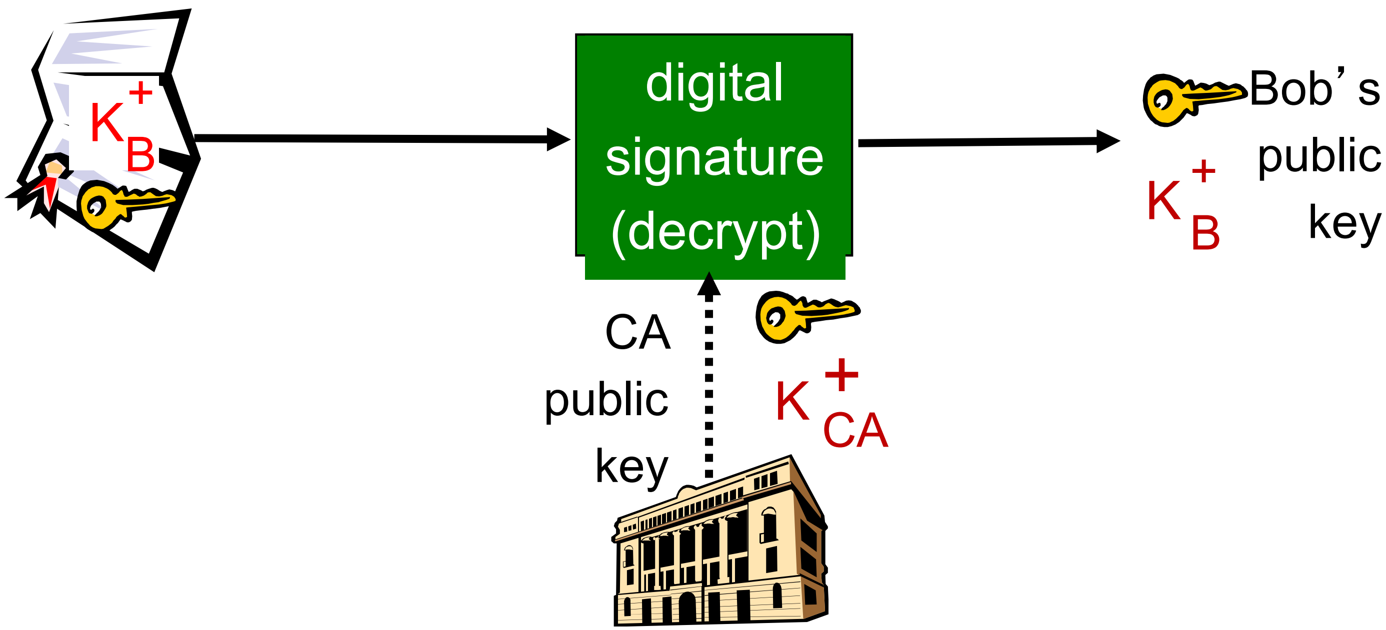
Certification authorities

- *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



Secure email

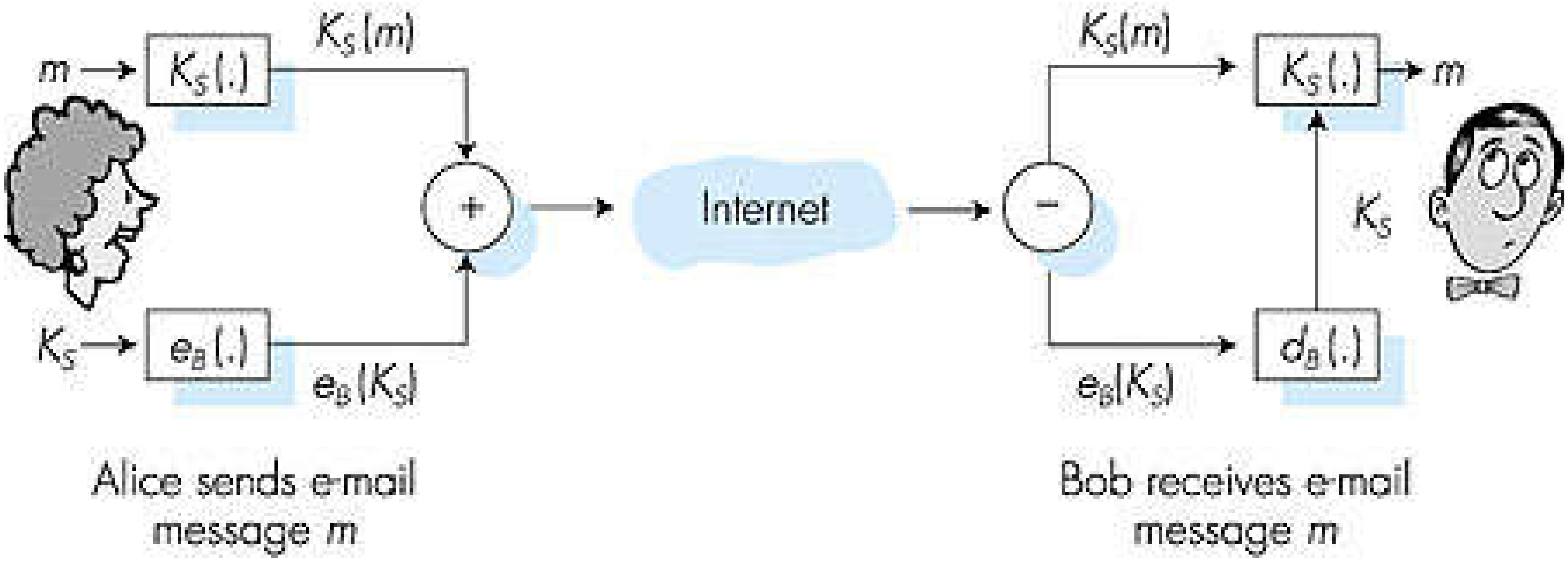
Alice wants to send secret e-mail message, m, to Bob.

We want confidentiality: encrypt messages (symm. Or public ?)

Secure email

Alice wants to send secret e-mail message, m, to Bob.

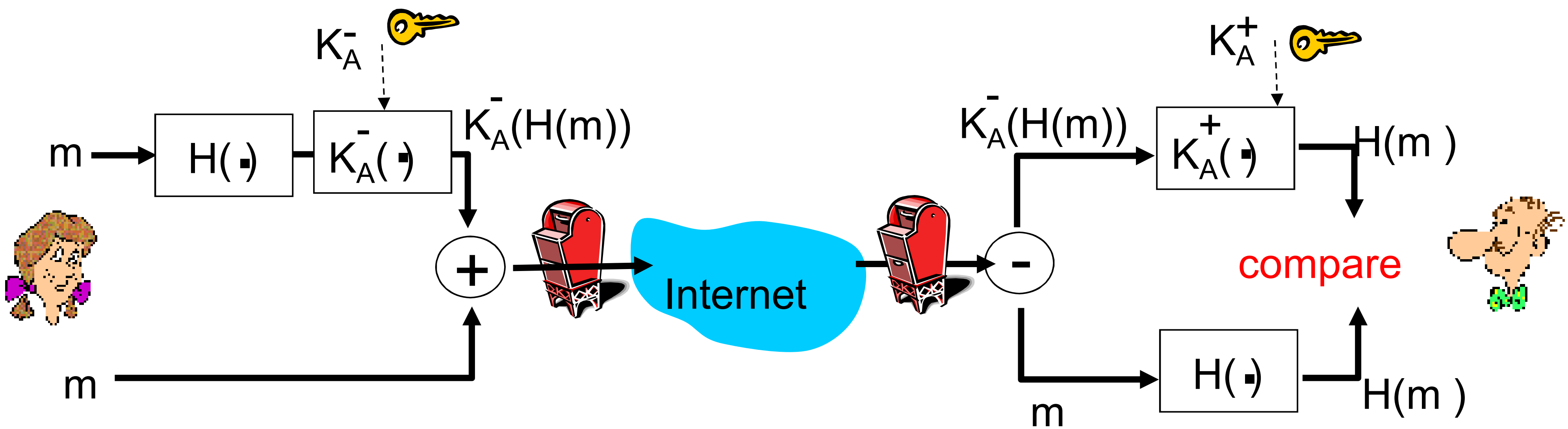
We want confidentiality: encrypt messages (symm. Or public ?)



- generates random symmetric private key, K_S. (**session key**)
- encrypts message with K_S
- ***Problem?***
- **Key Distribution Problem!**

Secure e-mail (continued)

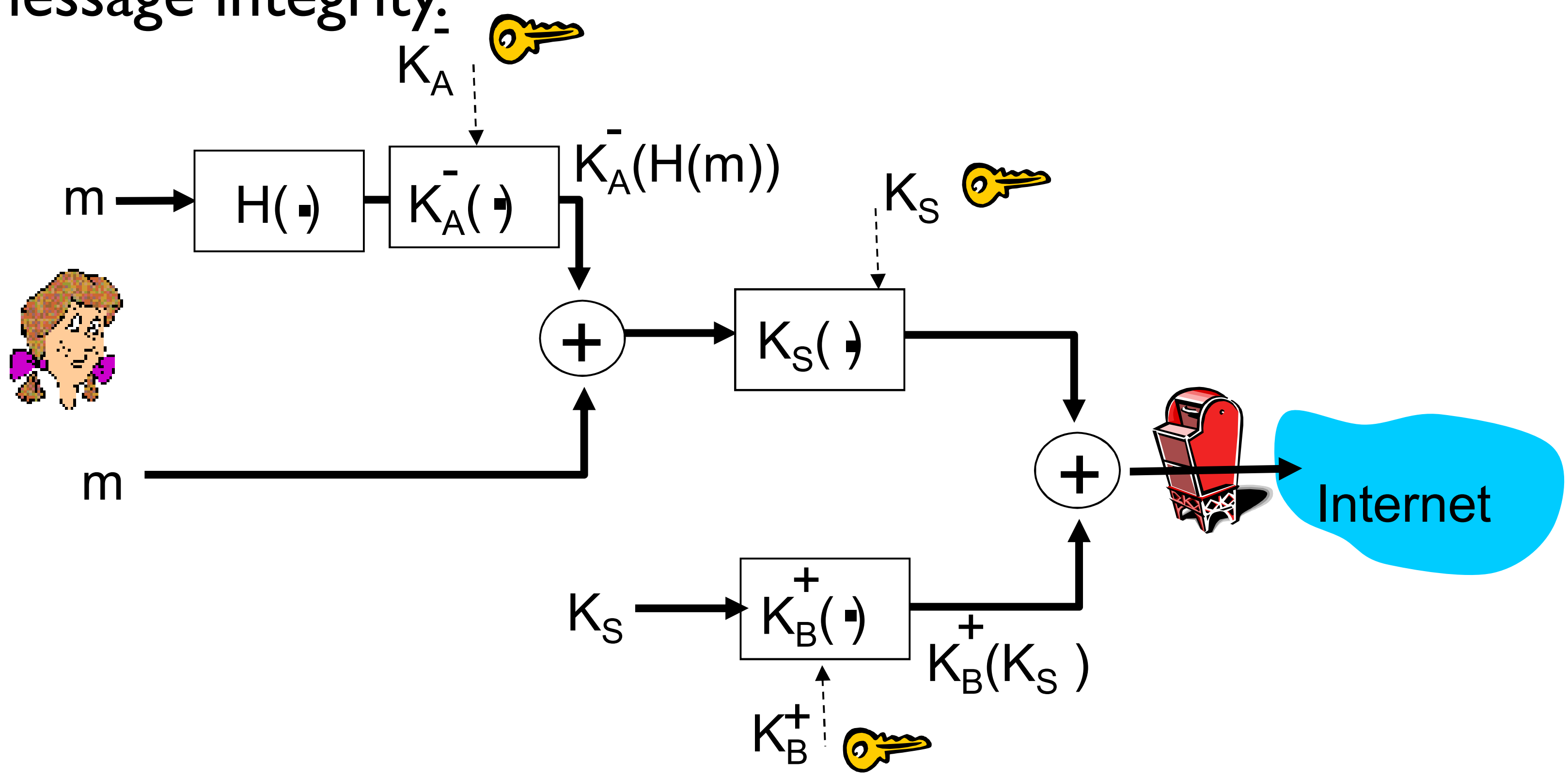
- Alice wants to provide sender authentication & message integrity



- Alice digitally signs message
- sends both message (in the clear) and digital signature

Secure e-mail (continued)

- Alice wants to provide confidentiality, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key