Security: Integrity, Authentication, Non-repudiation

CS 352, Lecture 20

http://www.cs.rutgers.edu/~sn624/352-S19

Srinivas Narayana

(heavily adapted from slides by Prof. Badri Nath and the textbook authors)

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Today

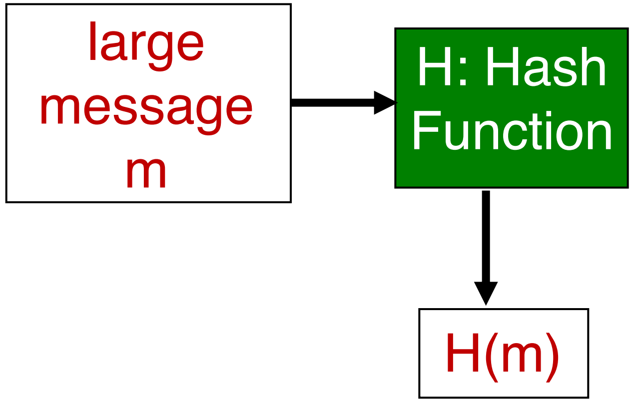
- Last two lectures: cryptography for confidentiality
- Today: Message digests (integrity)
- Digital signatures (non-repudiation, integrity)
- Certificate authorities (authentication)
- Using these techniques to secure a specific application (email)

Message Digests

Integrity: Did my message get across without tampering?

Message digests

Can we ensure that a receiver can detect message tampering?



Idea: fixed-length, easy- to-compute digital "fingerprint" of a message

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

Cryptographic hash function properties:

- Easy to calculate
- Produces fixed-size msg digest (fingerprint)
- Hard to reverse: given msg digest x,
  - computationally infeasible to find m such that x = H(m)
  - Or another m' such that H(m) = H(m')

Internet checksum: a poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- Is easy to compute

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!

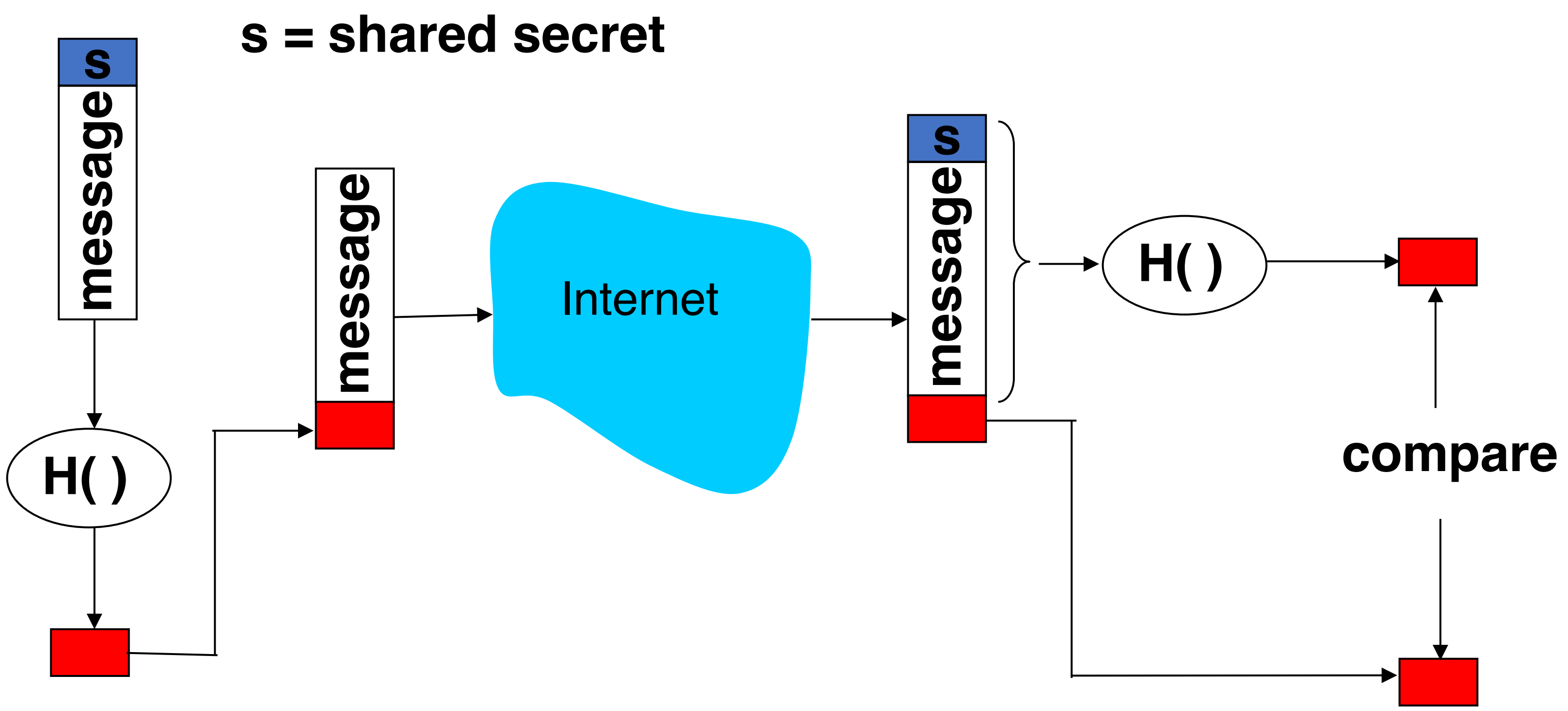
Hash function algorithms

- 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

Basic idea of crypto hash function

- Use a message as key and transform a constant string of length N repeatedly into another string of length N which is the digest
- Simple example: XOR the constant string with the message bytes
- In practice, use a set of Boolean operations

Message Authentication Code (MAC)



- ***Authenticates sender***
- ***Verifies message integrity***
- No encryption!
- Also called “keyed hash”
- Notation: MD_m = H(s ll m) ; send m ll MD_m

HMAC

- popular MAC standard
- addresses some subtle security flaws
- operation:
  - concatenates secret to front of message.
  - hashes concatenated message
  - concatenates secret to front of digest
  - hashes combination again

Digital Signatures

Non-repudiation and integrity

Digital signatures

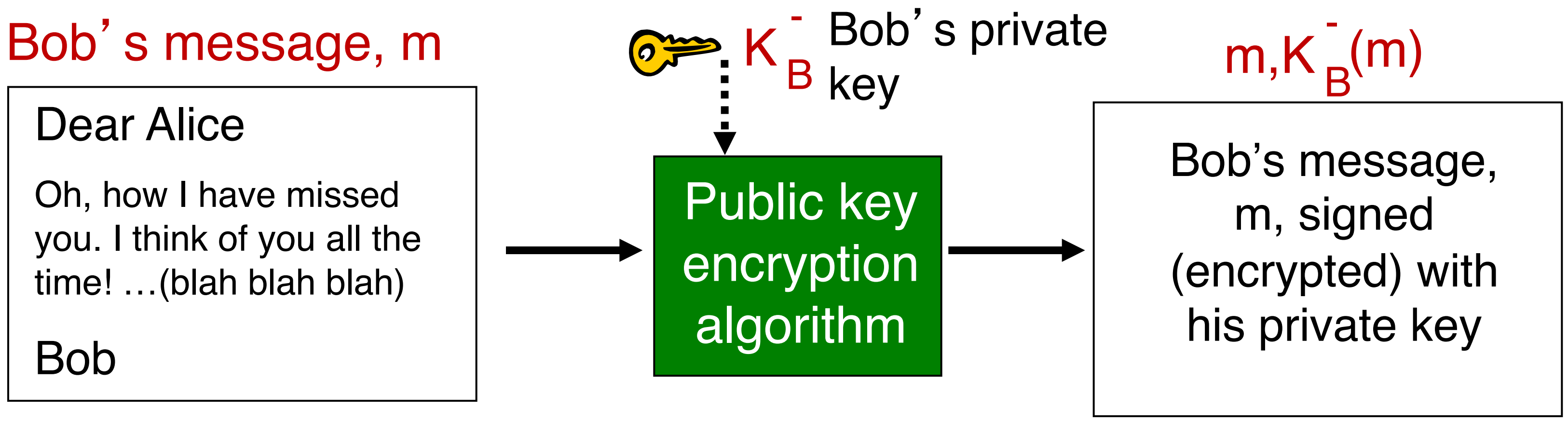
cryptographic technique analogous to hand-written signatures:

- 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

- 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:

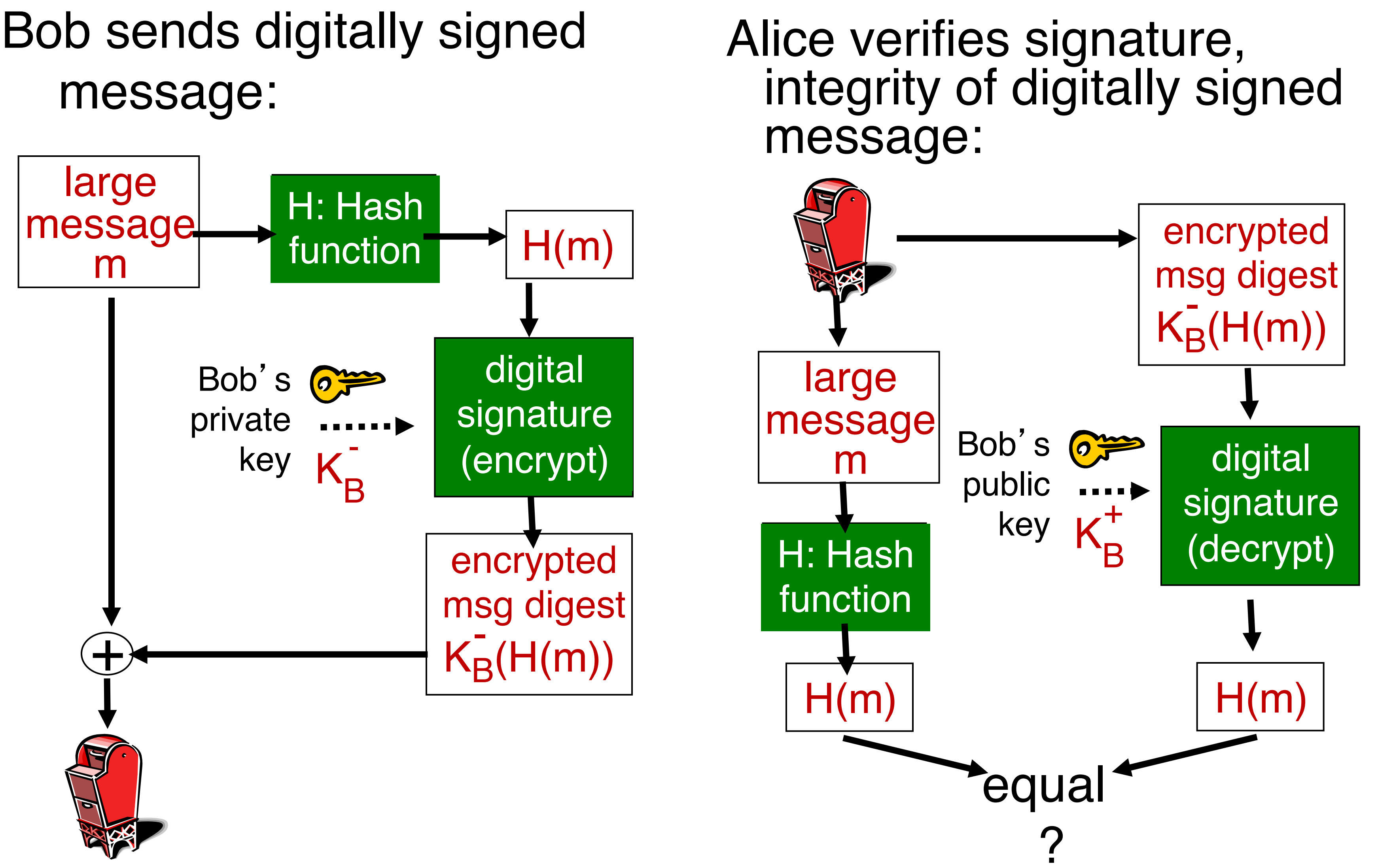
- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

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

One problem: we need to encrypt (large) messages using public key crypto!

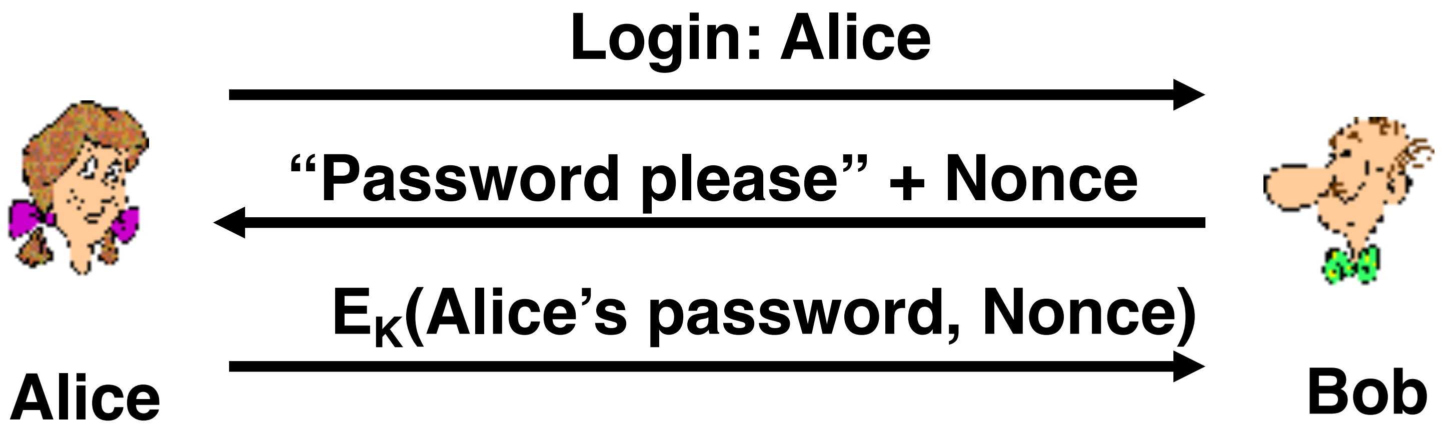
Digital signature = encrypted message digest



Authentication & Key Certification

Hello… is it me you're looking for?

Recall: Implement authentication using crypto



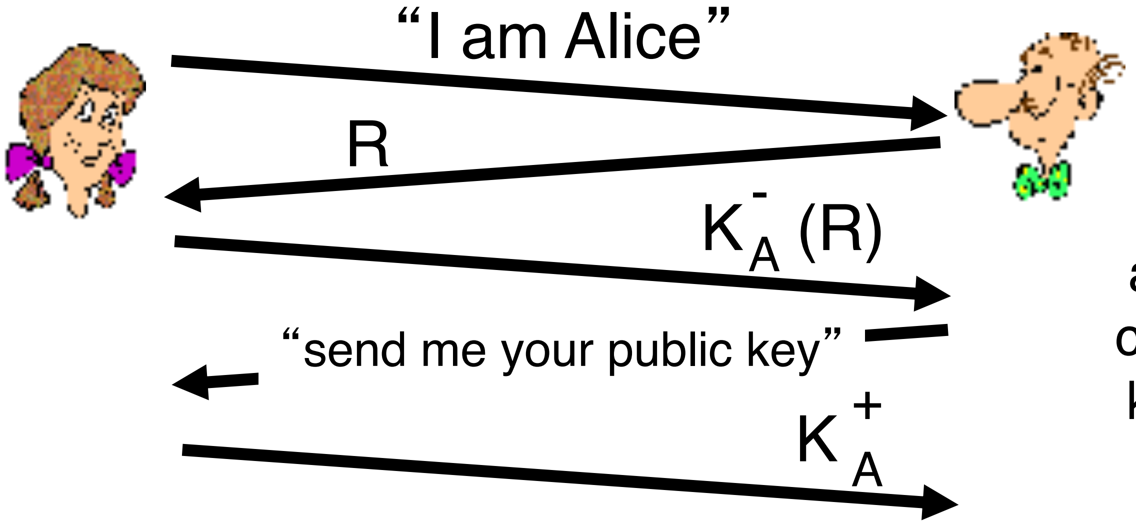
- Use a nonce to prevent replay attacks
- Communicate using a shared secret K

Authentication

Previous proposal requires shared symmetric key

- Can we authenticate using public key techniques?

Sure! use nonce and public key cryptography



Bob computes

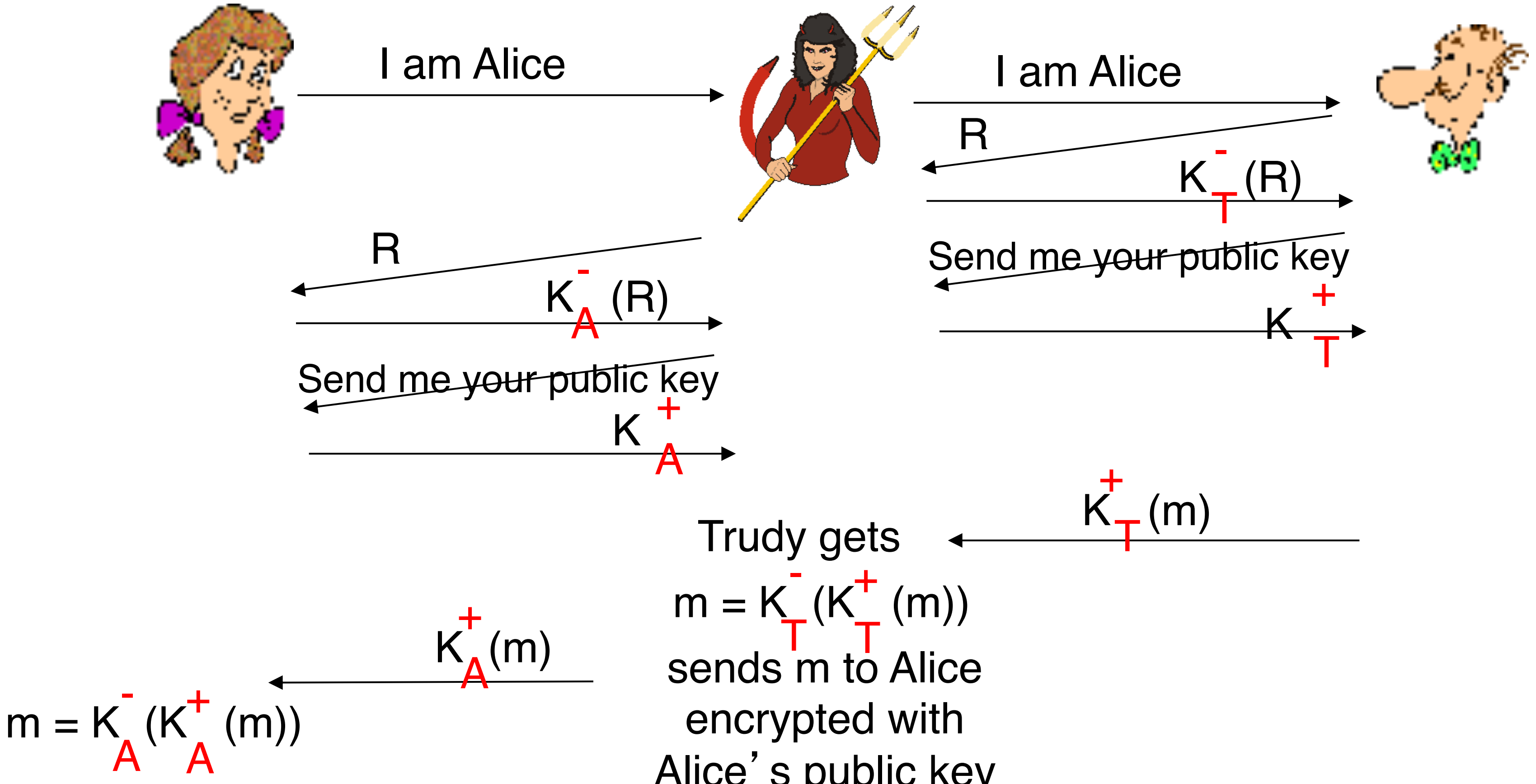
$$K_A^+(K_A^-(R)) = R$$

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

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

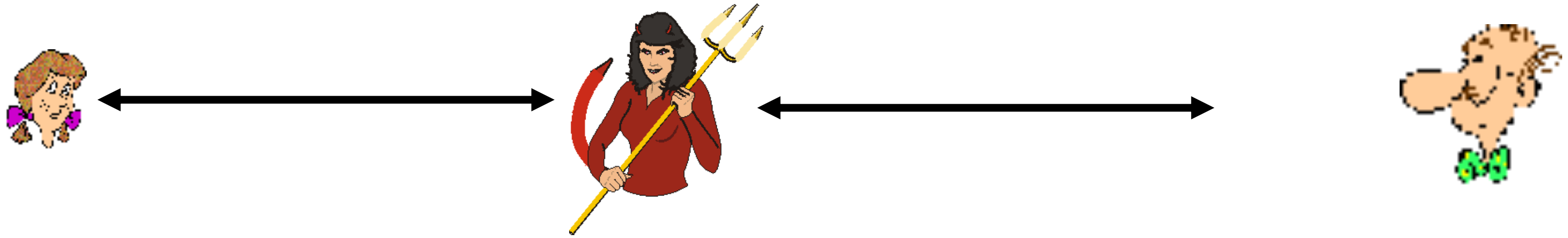
Security hole: if you ask for public keys!

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



Security hole: if you ask for public keys!

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 (plaintext) messages as well!

Key certification: Motivation

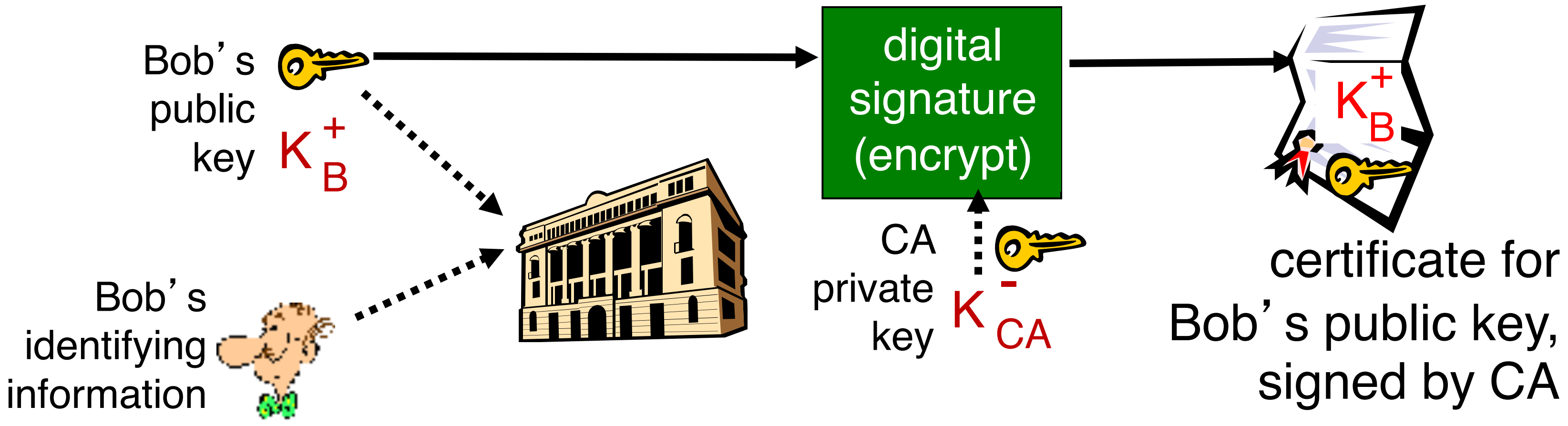
- Is there a way to ensure we can reliably know the public key of a communicating entity?

- Trust *someone else* (namely: a centralized authority) to check this for us

- On the Internet, trust is transitive:
  - We trust X (Ex: Alice trusts a certification authority)
  - X trusts Y (Ex: CA attests to Bob’s public key)
  - Hence, we can trust Y (Ex: Alice can trust Bob’s public key)

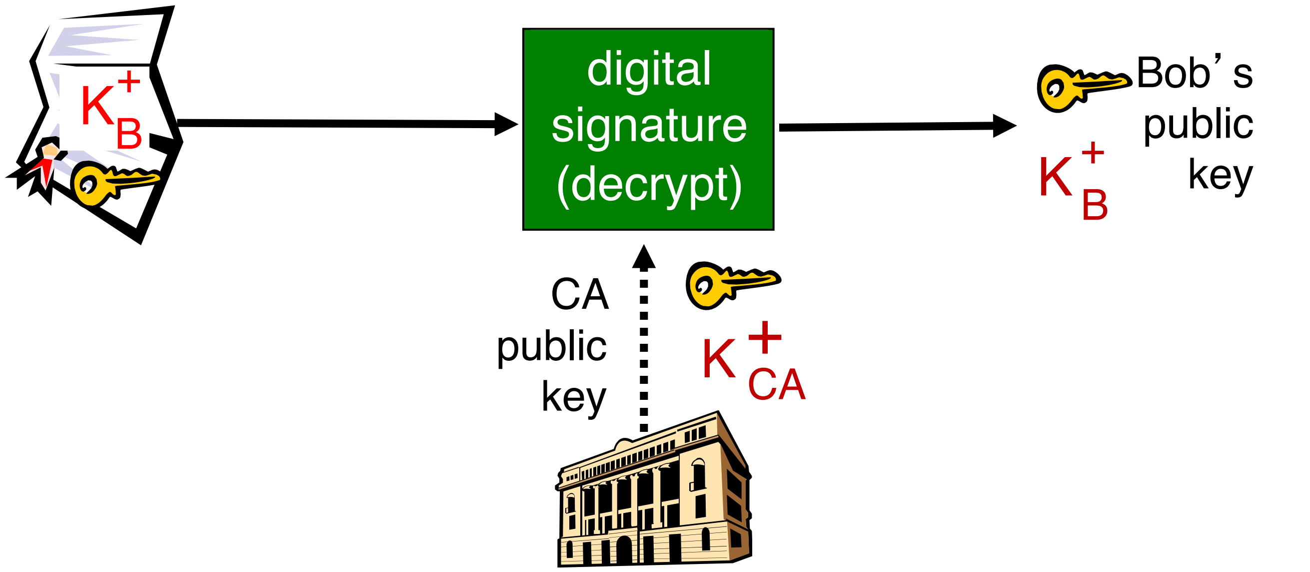
Certification authorities

- *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 (from Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key

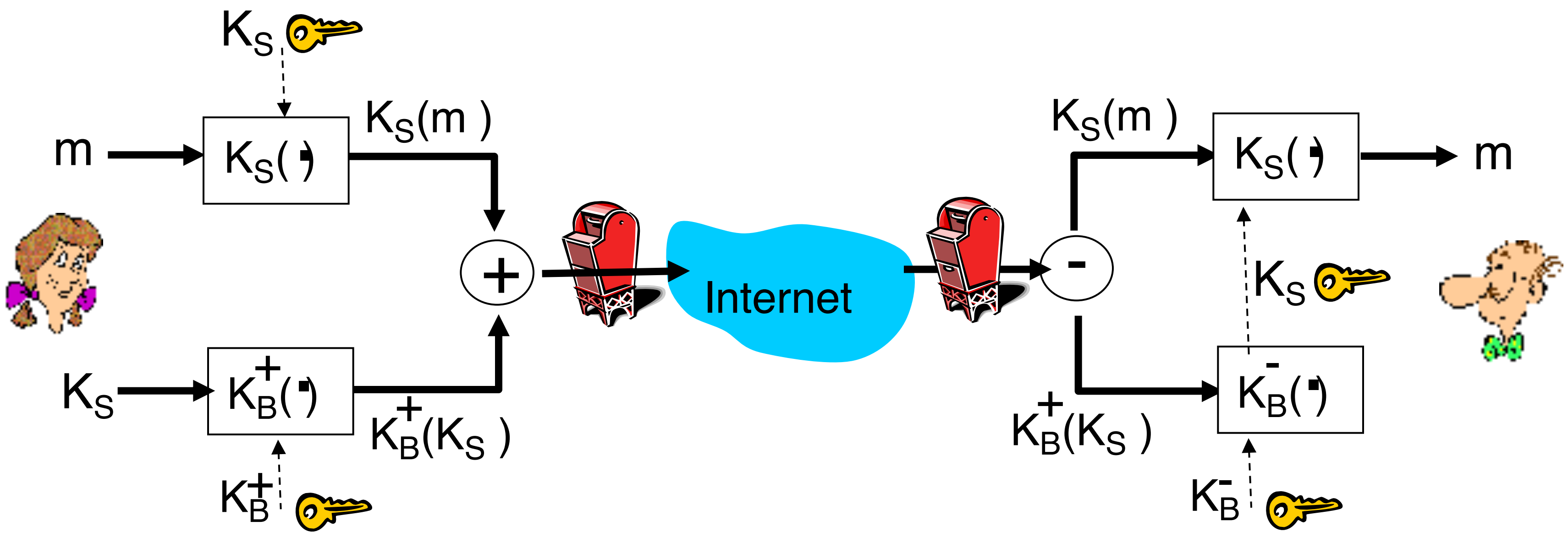


PGP: E-mail Security

An application of security principles to application-layer security

Secure e-mail

Alice wants to send confidential e-mail, m, to Bob.

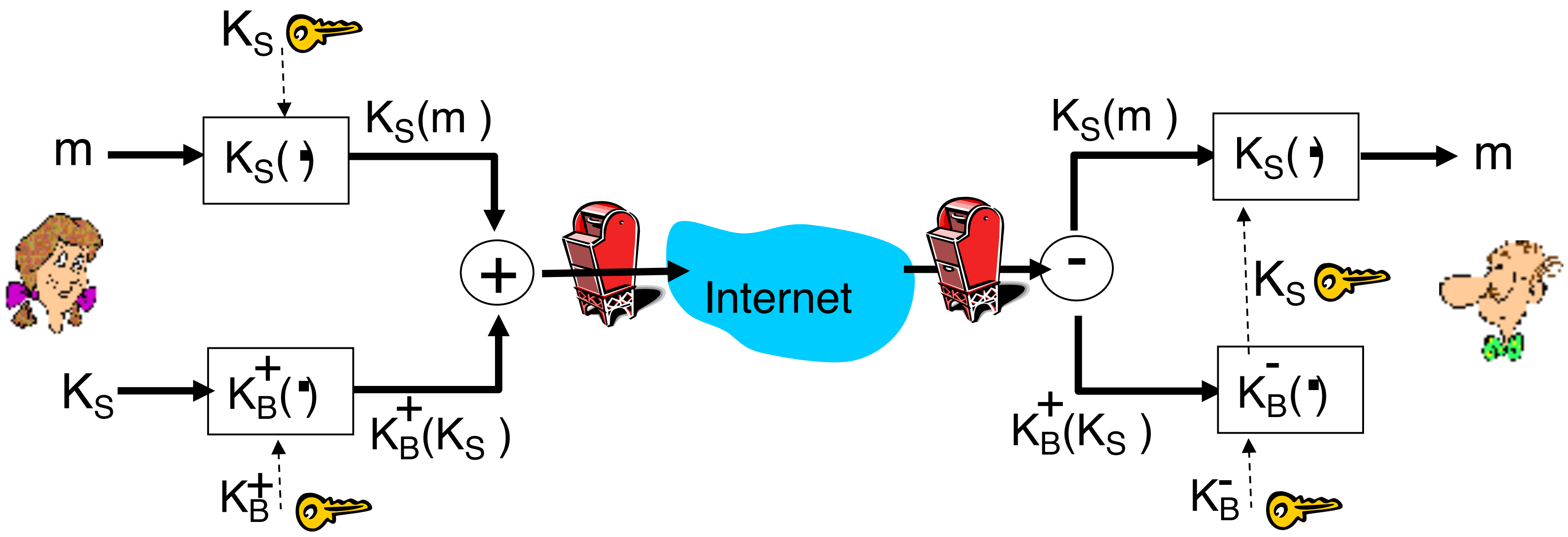


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob

Secure e-mail

Alice wants to send confidential e-mail, m, to Bob.

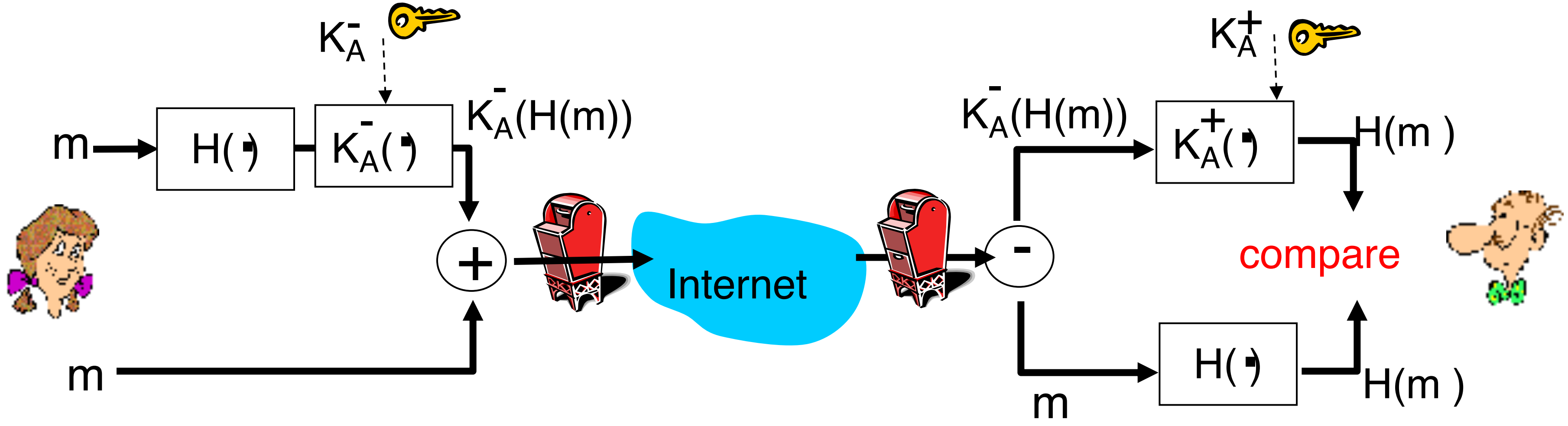


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (continued)

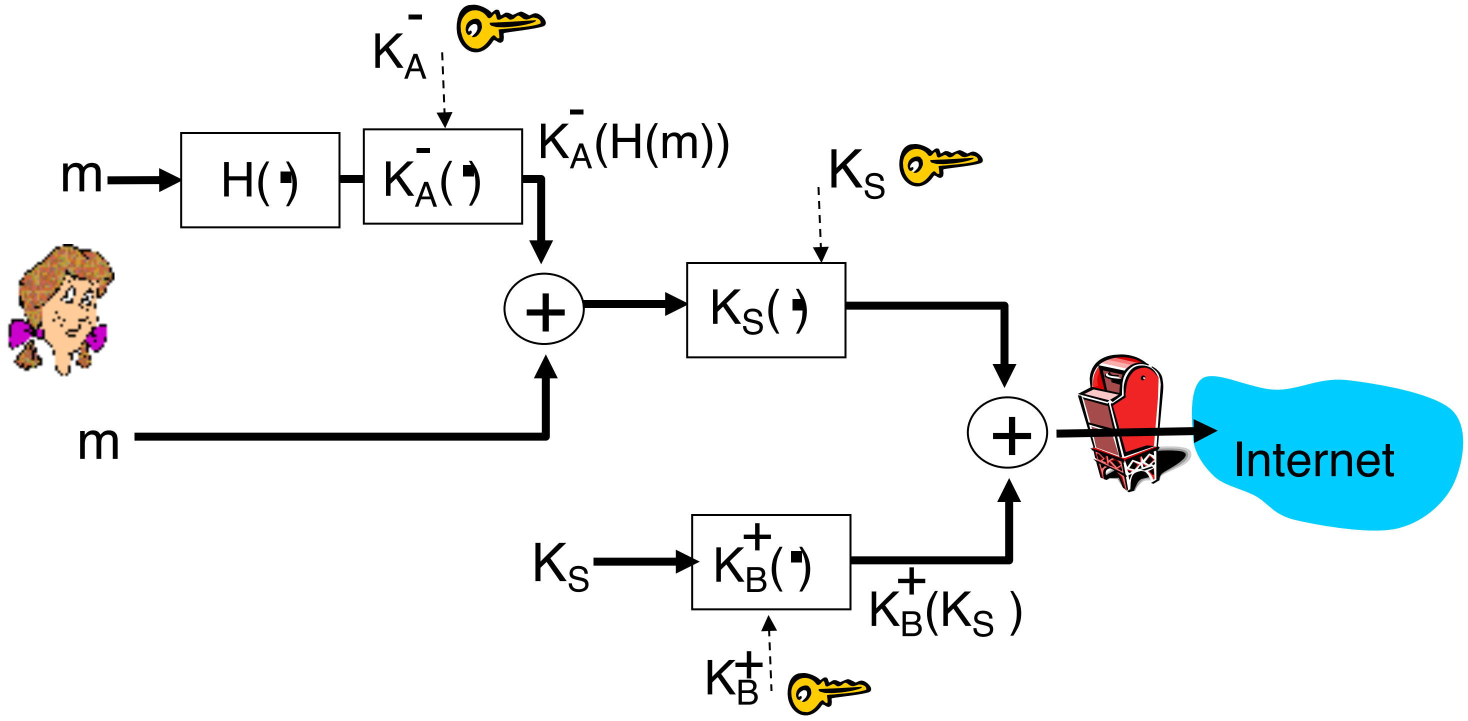
Alice wants to provide sender authentication and message integrity



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

Secure e-mail (continued)

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



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

PGP: Pretty Good Privacy

- Security implemented at the application level
  - Allows all of the communication modes described earlier

- Uses a “web of trust” for key exchange

- Key signing: any party X can “sign” that they trust the public key of Y using their private keys

- Propagate trust: If Z trusts X, Z can now trust Y