Sécurité

Computer Networking: A Top Down Approach ,
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley,

roadmap

1 What is network security?

2 Principles of cryptography

3 Authentication Message integrity

4 Securing e-mail

5 Securing TCP connections: SSL

Authentication

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

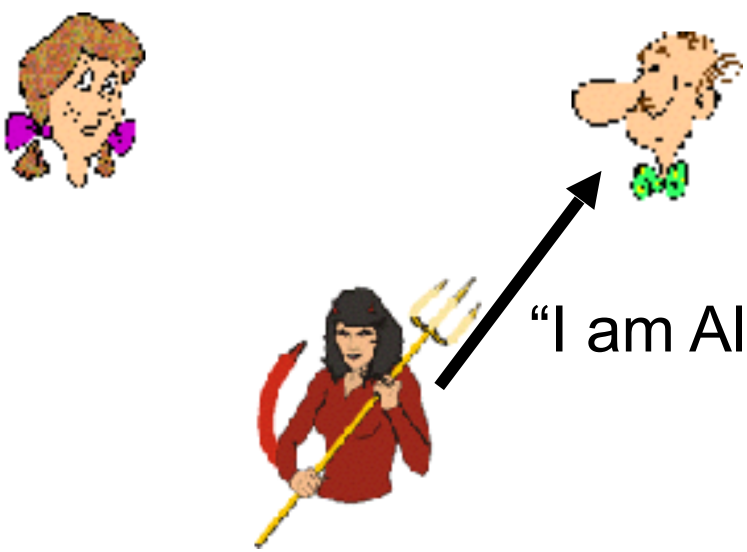
Protocol ap1.0: Alice says “I am Alice”

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

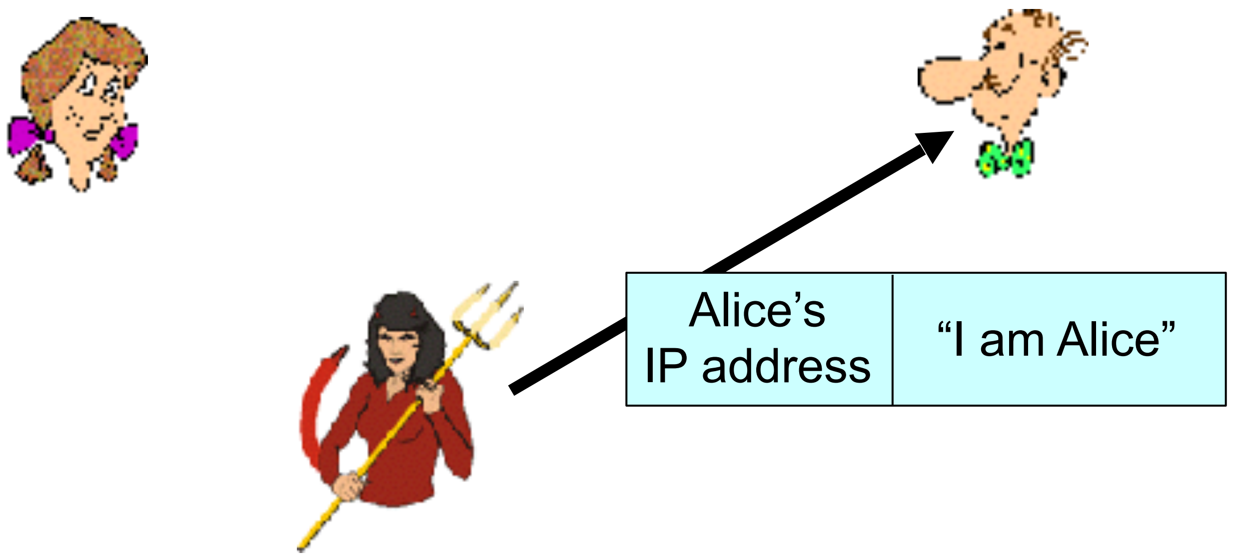
Authentication: another try

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

Failure scenario??

Authentication: another try

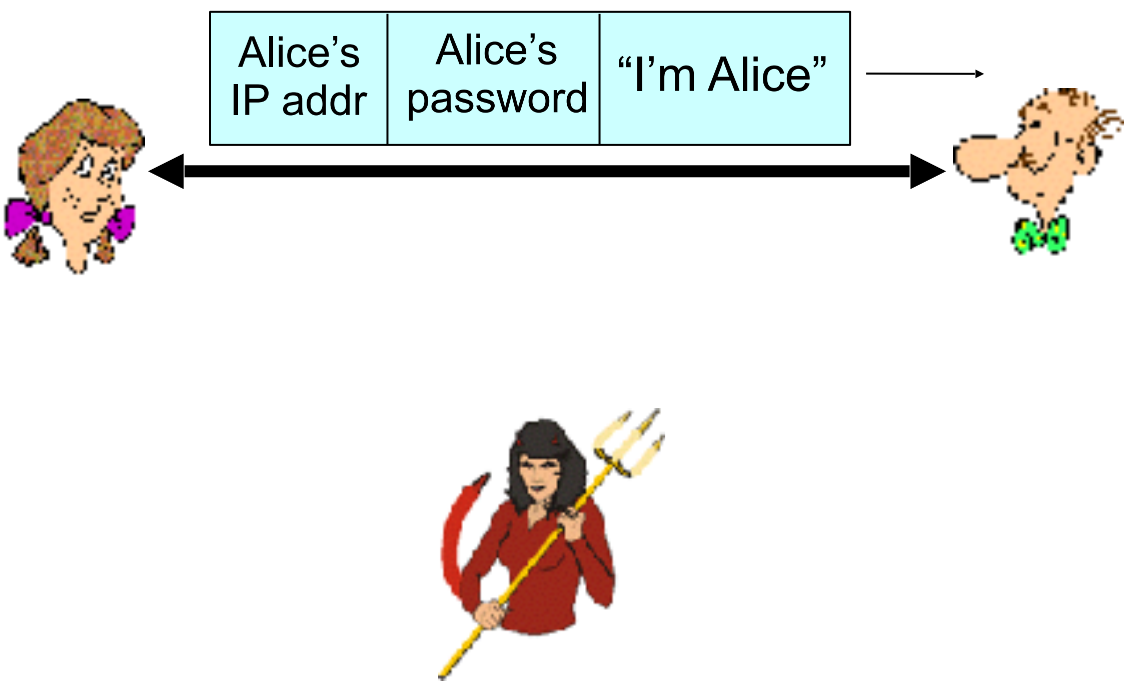
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address



Trudy can create a packet “spoofing” Alice’s address

Authentication: another try

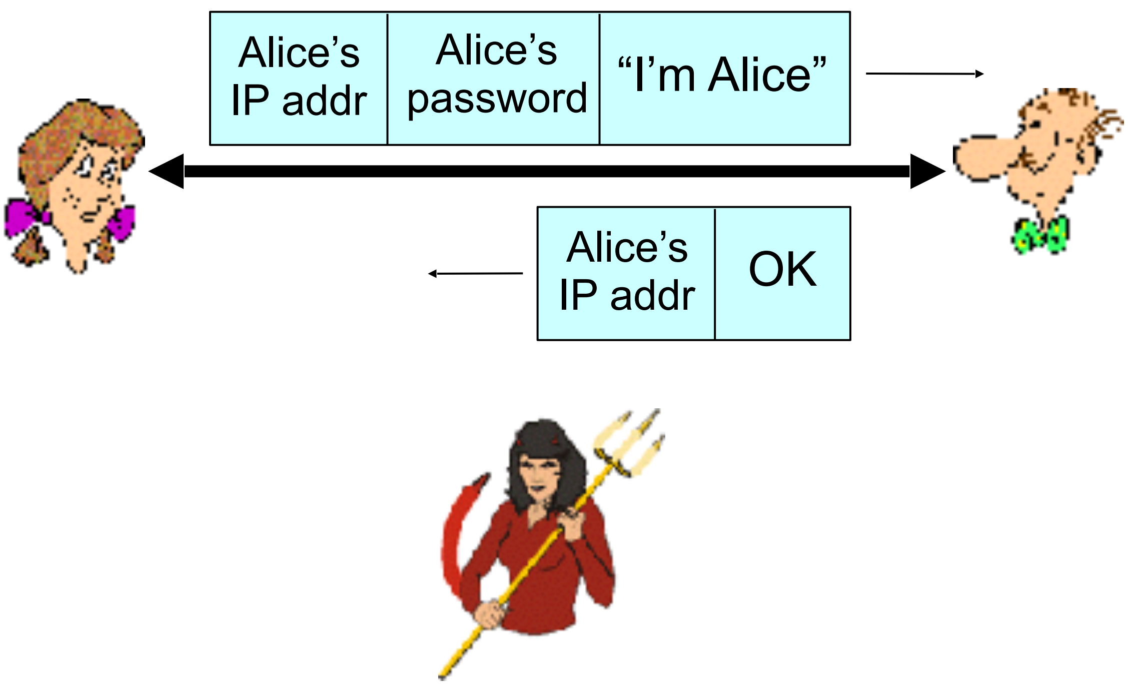
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



Failure scenario??

Authentication: another try

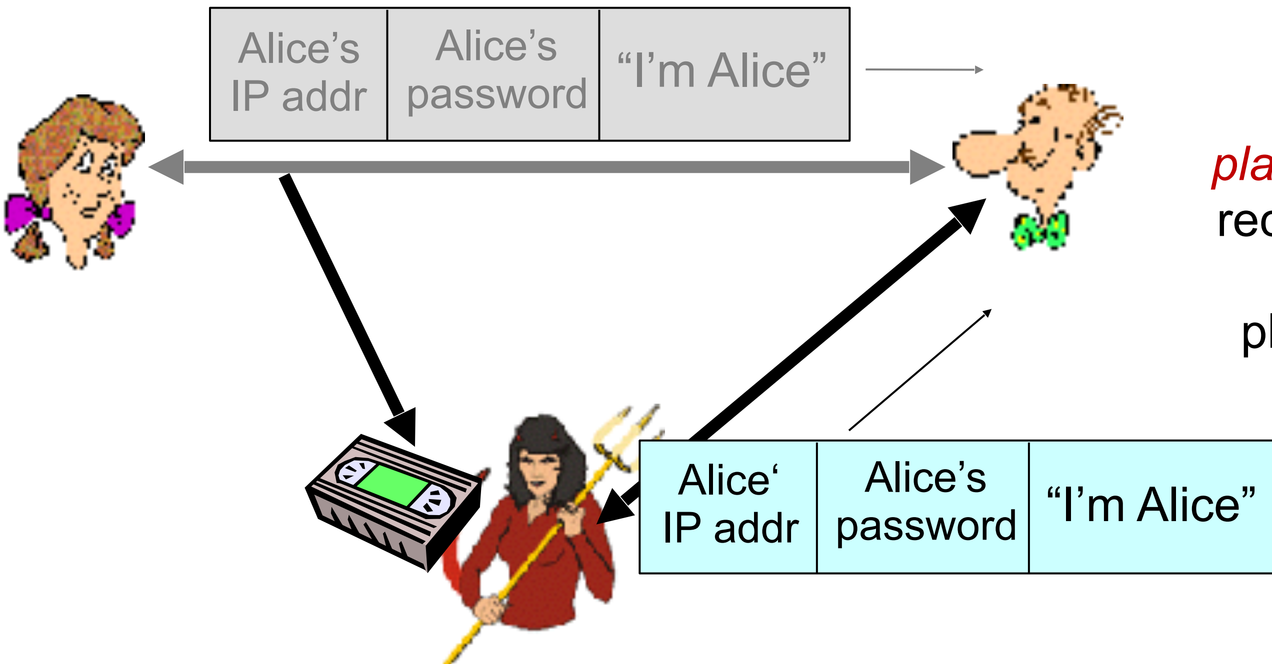
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



Failure scenario??

Authentication: another try

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



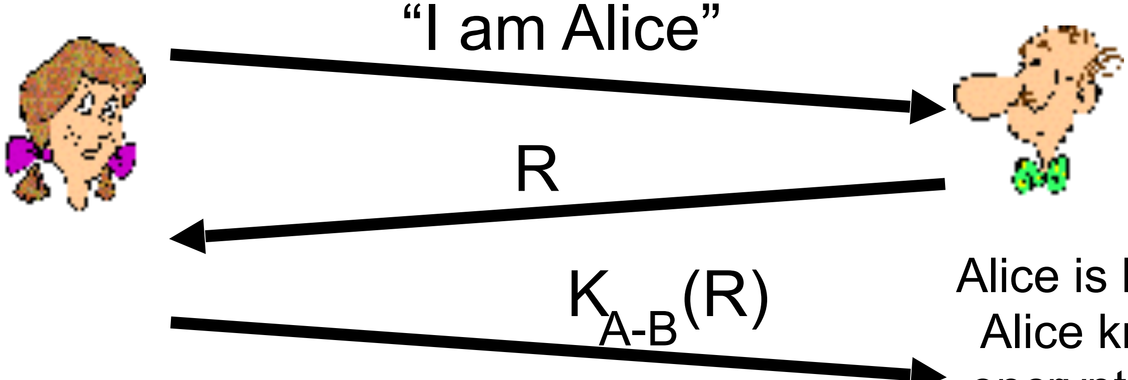
playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication: yet another try

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



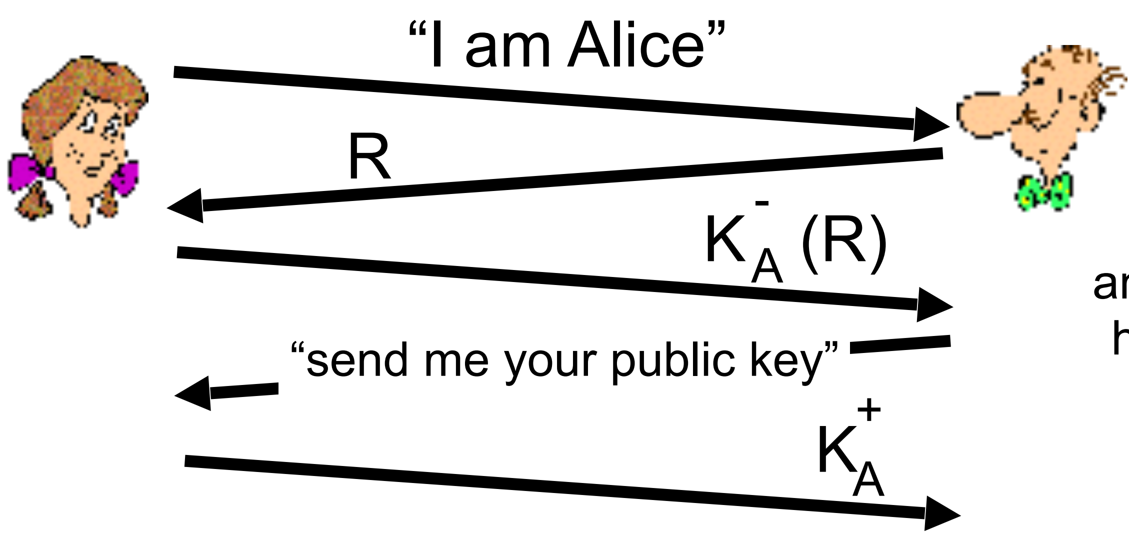
Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

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

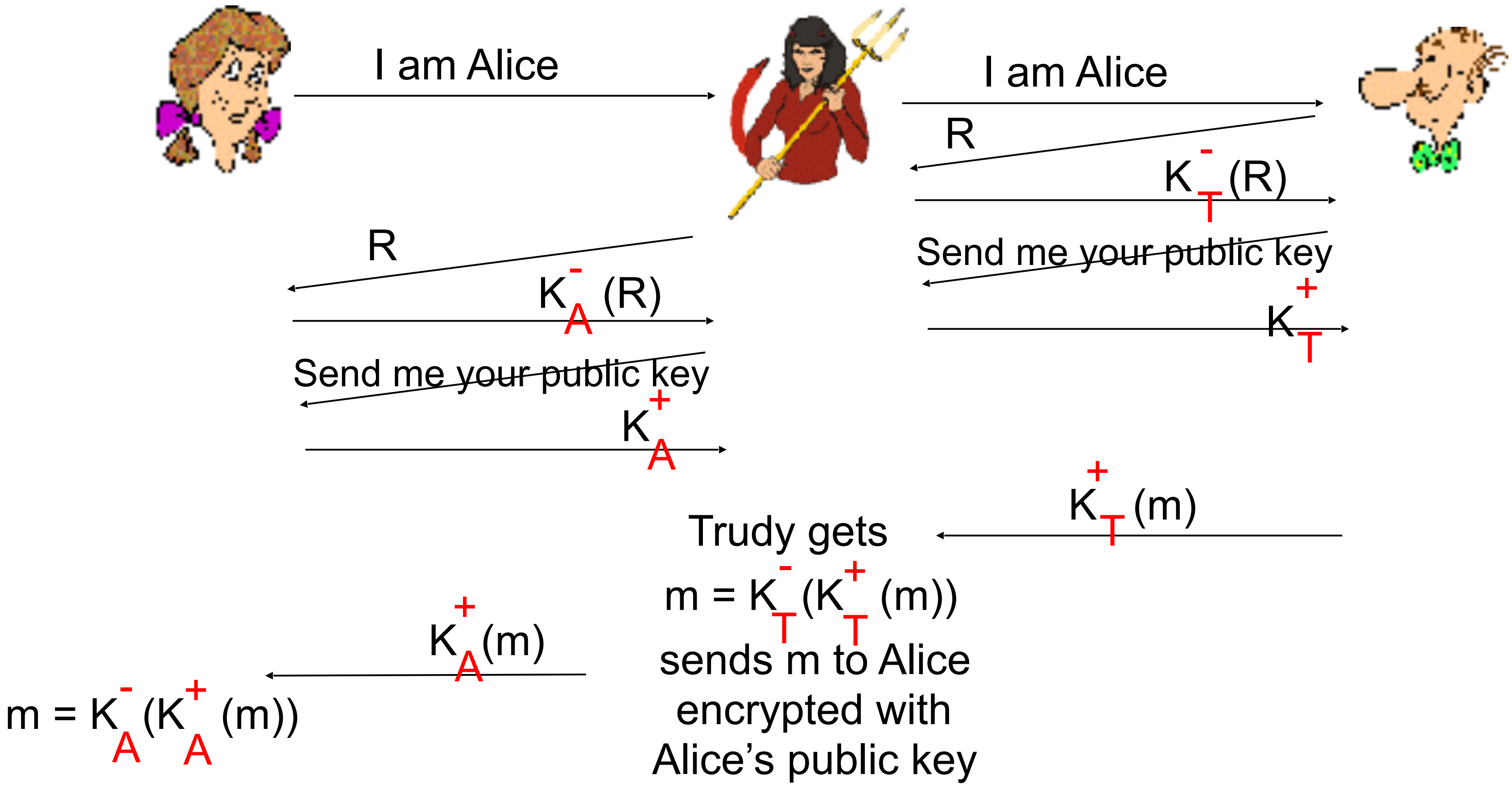
$$K_A^+(K_A^-(R)) = R$$

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 (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)

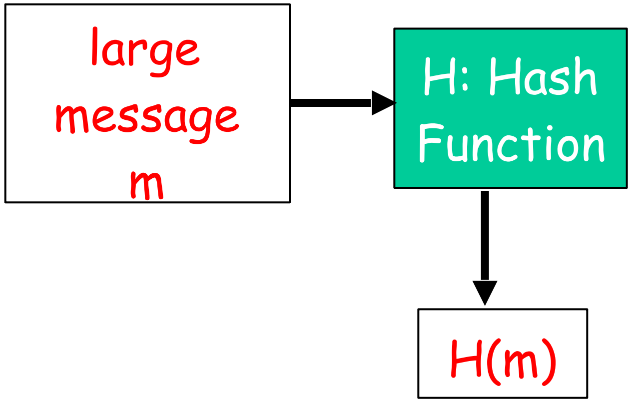
difficult to detect:

- 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!

Message Integrity

- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained
- Let's first talk about message digests

Message Digests



- Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: "message signature"
- Note that H( ) is a many-to-1 function
- H( ) is often called a "hash function"

- Desirable properties:
  - Easy to calculate
  - Irreversibility: Can't determine m from H(m)
  - Collision resistance: Computationally difficult to produce m and m' such that H(m) = H(m')
  - Seemingly random output

Internet checksum: poor message digest

Internet checksum has some properties of hash function:

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

- But given message with given hash value, it is easy to find another message with same hash value.
- Example: Simplified checksum: add 4-byte chunks at a time:

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

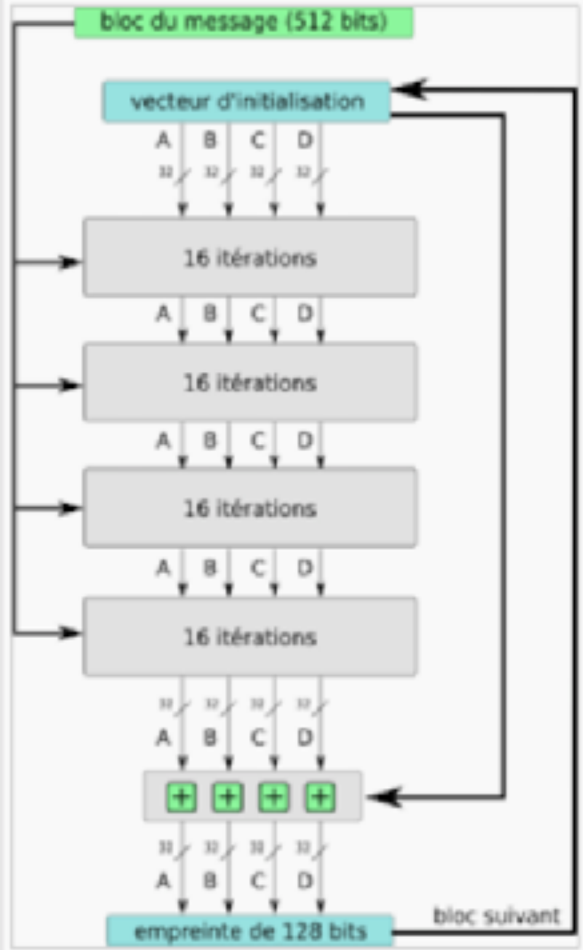
different messages but identical checksums!

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
- SHA-1 is also used.
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

MD5 (Message Digest 5)

- Rivest 91
- Produit un condencat de 128 bits
- Entrée décomposée en bloc de 512 bits (avec remplissage)
- 4 étapes:
- ajout de 1 puis remplissage par des 0 (telle sorte que la longueur de l'entrée soit un multiple de 512), ajout de la taille du message (un entier sur 64 bits)

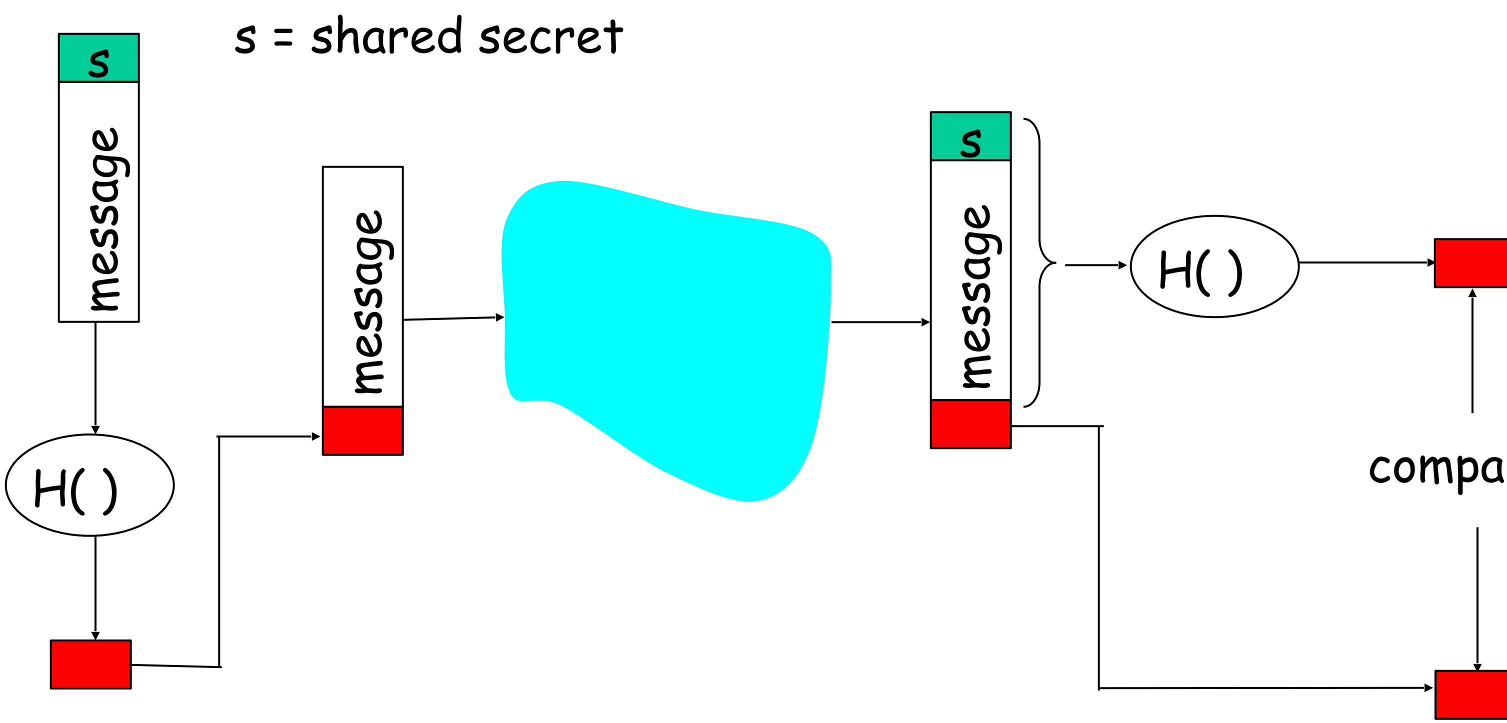


Vue générale de MD5.

SHA-1 (Secure Hash Algorithm)

- Conçu par la NSA (US federal standard)
- Condensat de 160 bits
- Entrée au plus 2^{64}
- Similaire à MD4, prédécesseur de MD5
- Pour chaque bloc d'entrée 80 tours

Message Authentication Code (MAC)



- **Authenticates sender**
- **Verifies message integrity**
- No encryption !
- Also called "keyed hash"
- Notation: $MD_m = H(s||m)$; send $m||MD_m$
- MAC de m : MD_m

HMAC

- Popular MAC standard
- Addresses some subtle security flaws

1. Concatenates secret to front of message.
2. Hashes concatenated message
3. Concatenates the secret to front of digest
4. Hashes the combination again.

Example: OSPF

- Recall that OSPF is an intra-AS routing protocol
- Each router creates map of entire AS (or area) and runs shortest path algorithm over map.
- Router receives link-state advertisements (LSAs) from all other routers in AS.

Attacks:

- Message insertion
- Message deletion
- Message modification

- How do we know if an OSPF message is authentic?

OSPF Authentication

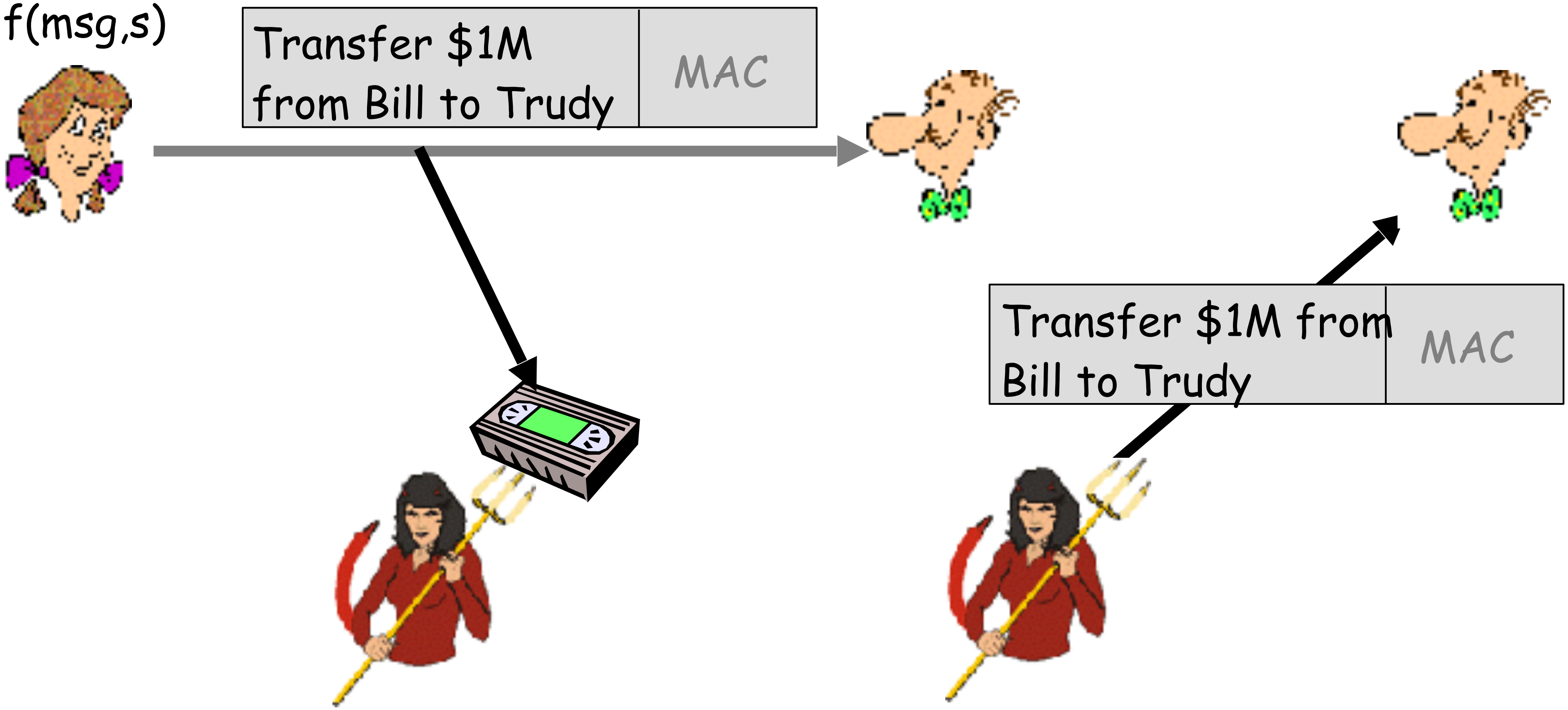
- Within an Autonomous System, routers send OSPF messages to each other.
- OSPF provides authentication choices
  - No authentication
  - Shared password: inserted in clear in 64-bit authentication field in OSPF packet
  - Cryptographic hash
- Cryptographic hash with MD5
  - 64-bit authentication field includes 32-bit sequence number
  - MD5 is run over a concatenation of the OSPF packet and shared secret key
  - MD5 hash then appended to OSPF packet; encapsulated in IP datagram

End-point authentication

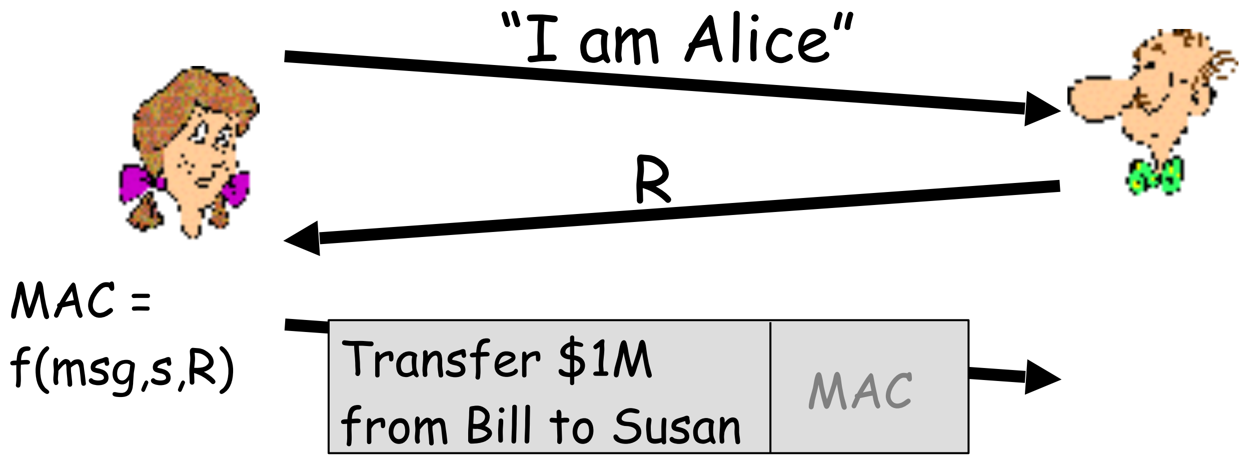
- Want to be sure of the originator of the message - end-point authentication.
- Assuming Alice and Bob have a shared secret, will MAC provide end-point authentication.
  - We do know that Alice created the message.
  - But did she send it?

Playback attack

MAC = f(msg,s)



Defending against playback attack: nonce



Digital Signatures

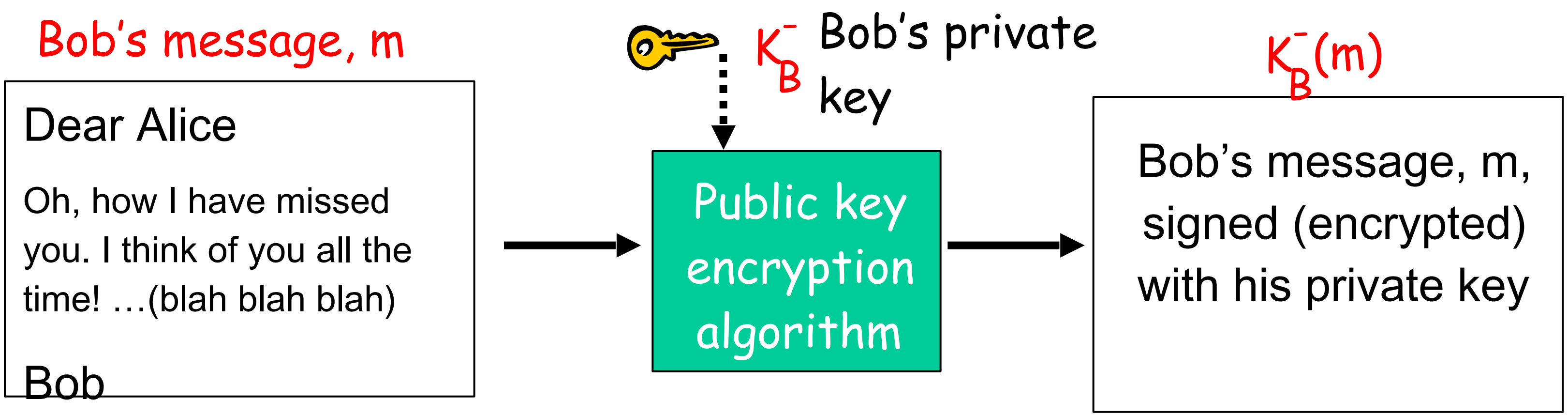
Cryptographic technique analogous to hand-written signatures.

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- Goal is similar to that of a MAC, except now use public-key cryptography
- 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:

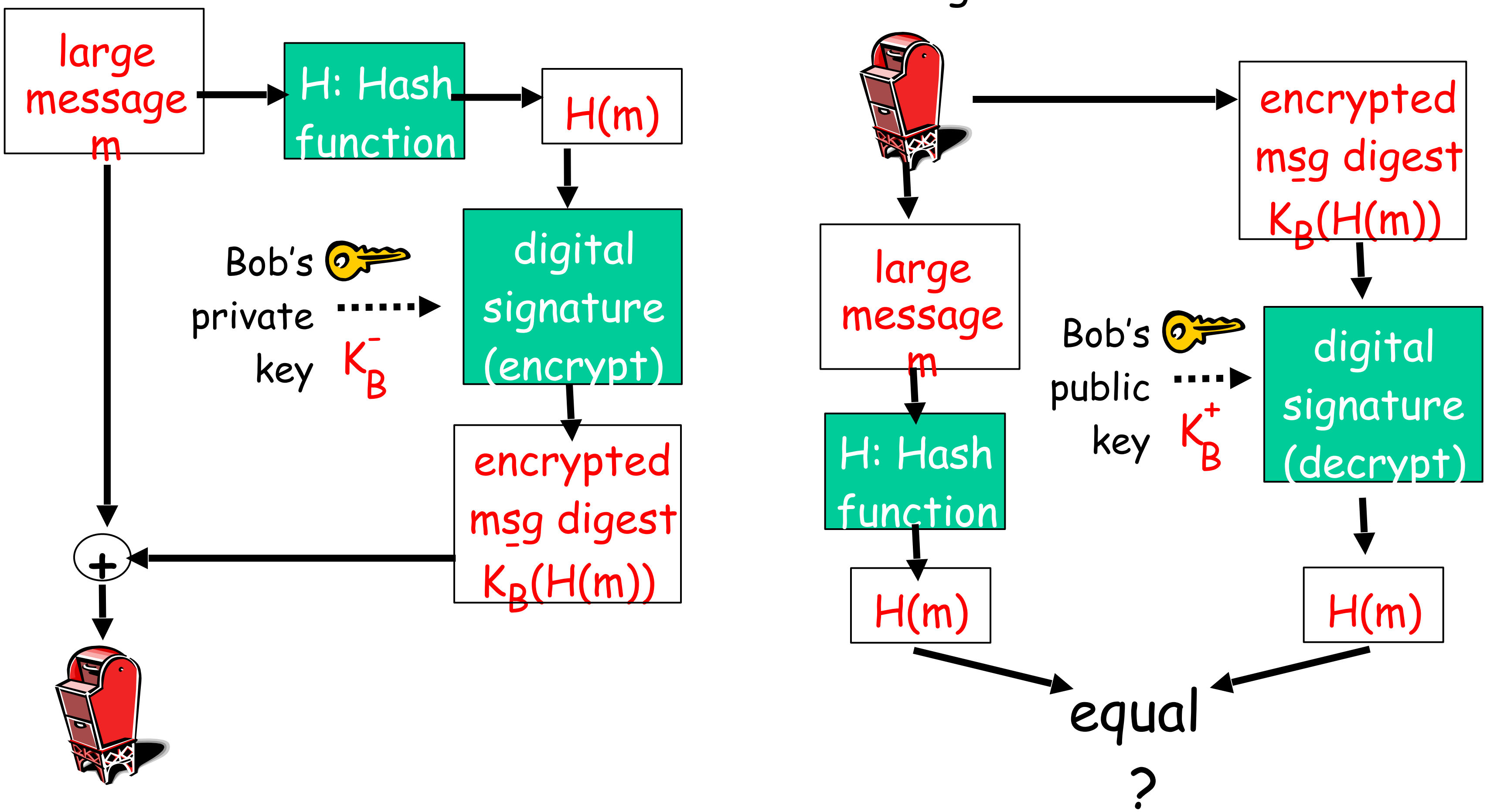
- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$



Digital signature = signed message digest

Bob sends digitally signed message:

Alice verifies signature and integrity of digitally signed message:



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:

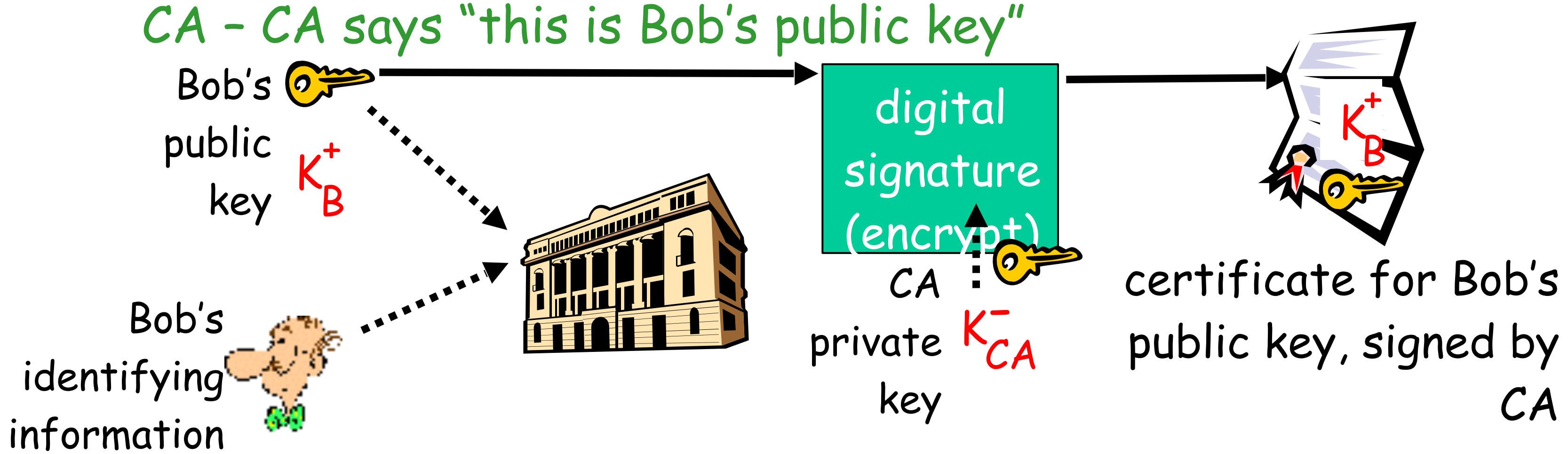
- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m.

Public-key certification

- Motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key.
  - Pizza Store verifies signature; then delivers four pizzas to Bob.
  - Bob doesn't even like Pepperoni

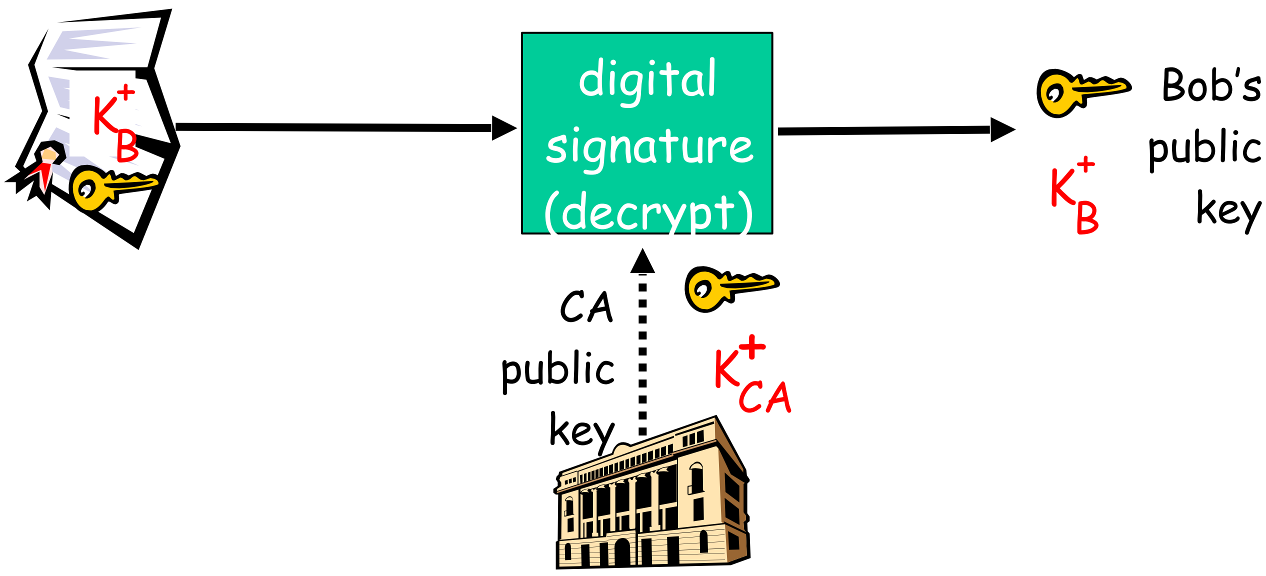
Certification Authorities

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



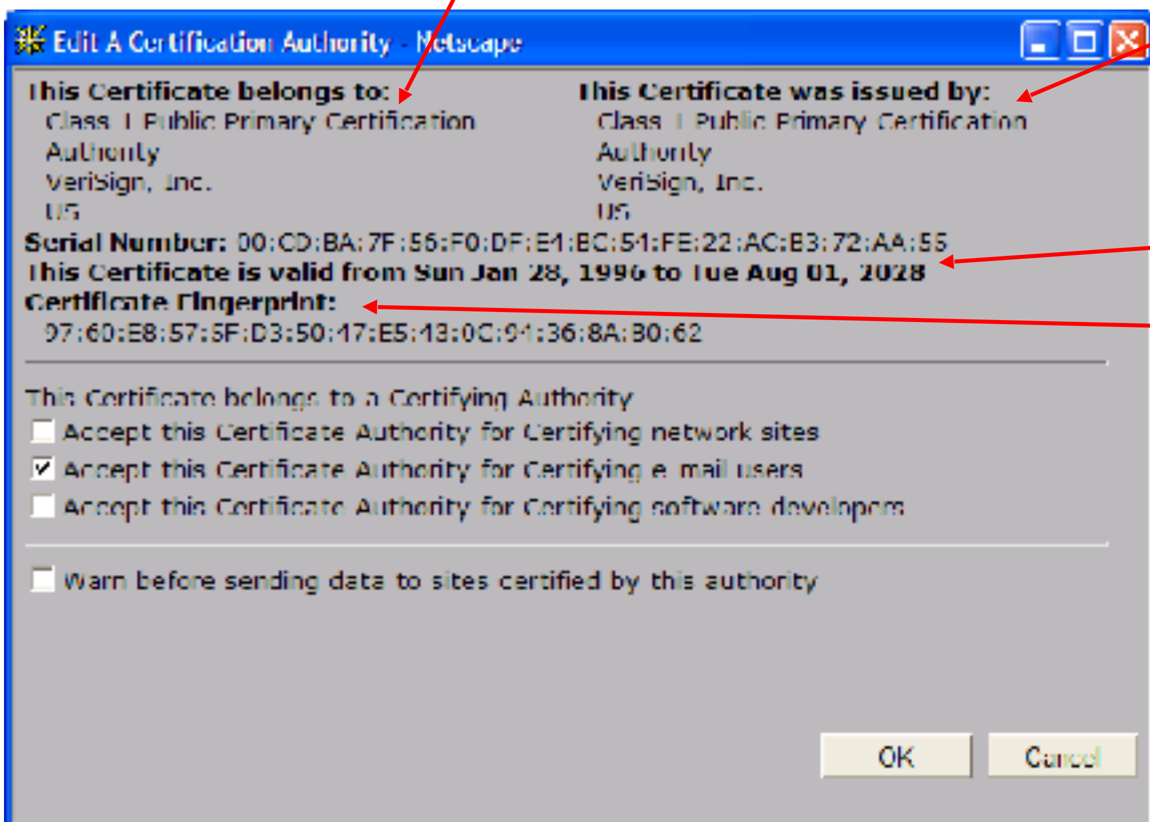
Certification Authorities

- 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

Certificates: summary

- Primary standard X.509 (RFC 2459)
- Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered "heavy"

roadmap

1 What is network security?

2 Principles of cryptography

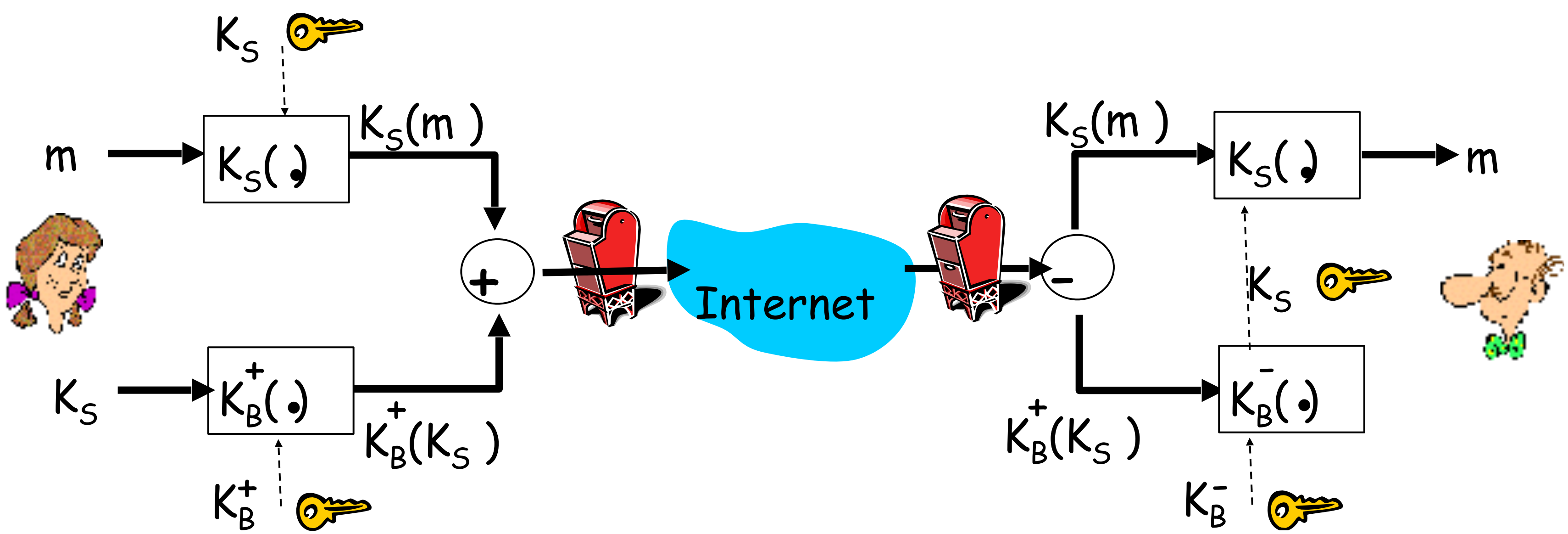
3 Message integrity

4 Securing e-mail

5 Securing TCP connections: SSL

Secure e-mail

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

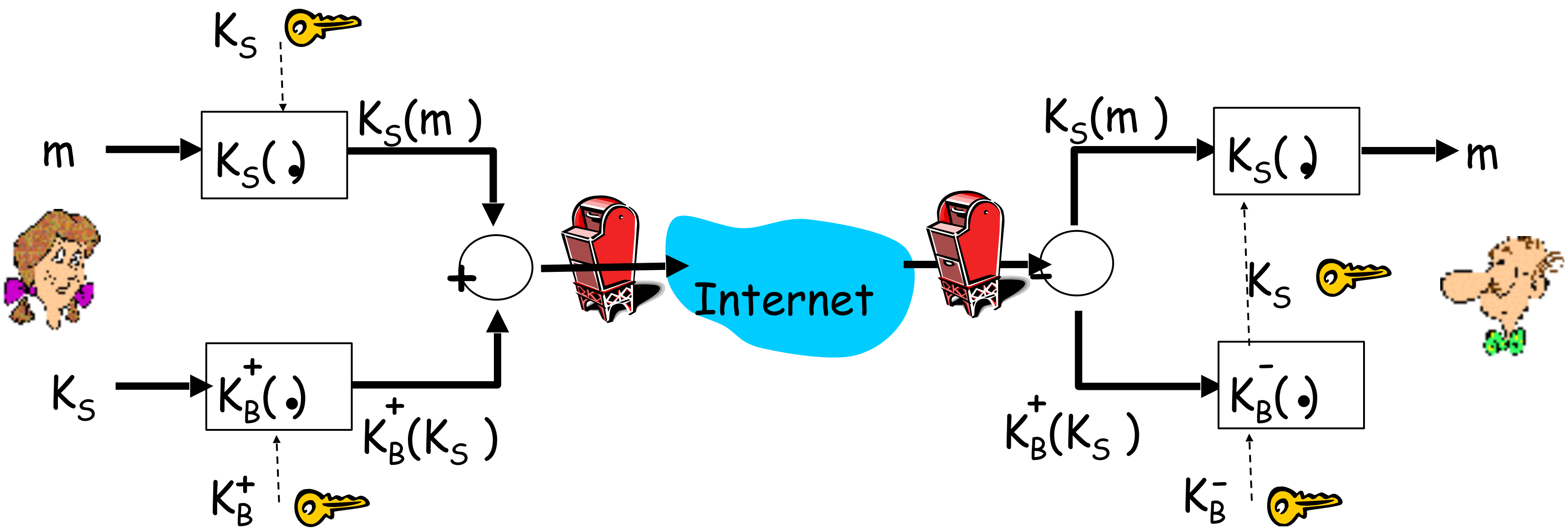


Alice:

- 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.

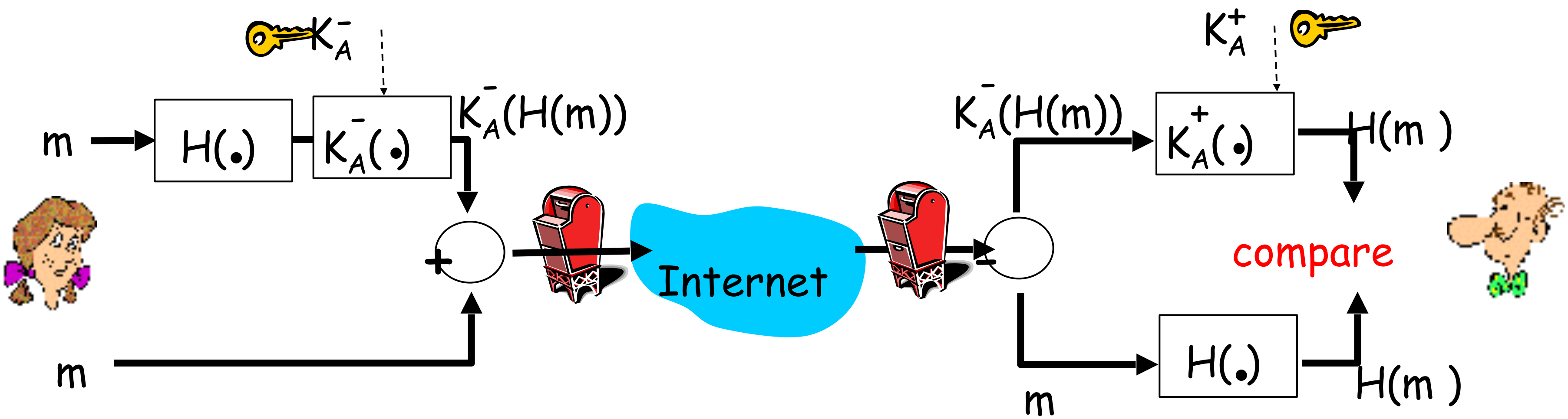


Bob:

- 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)

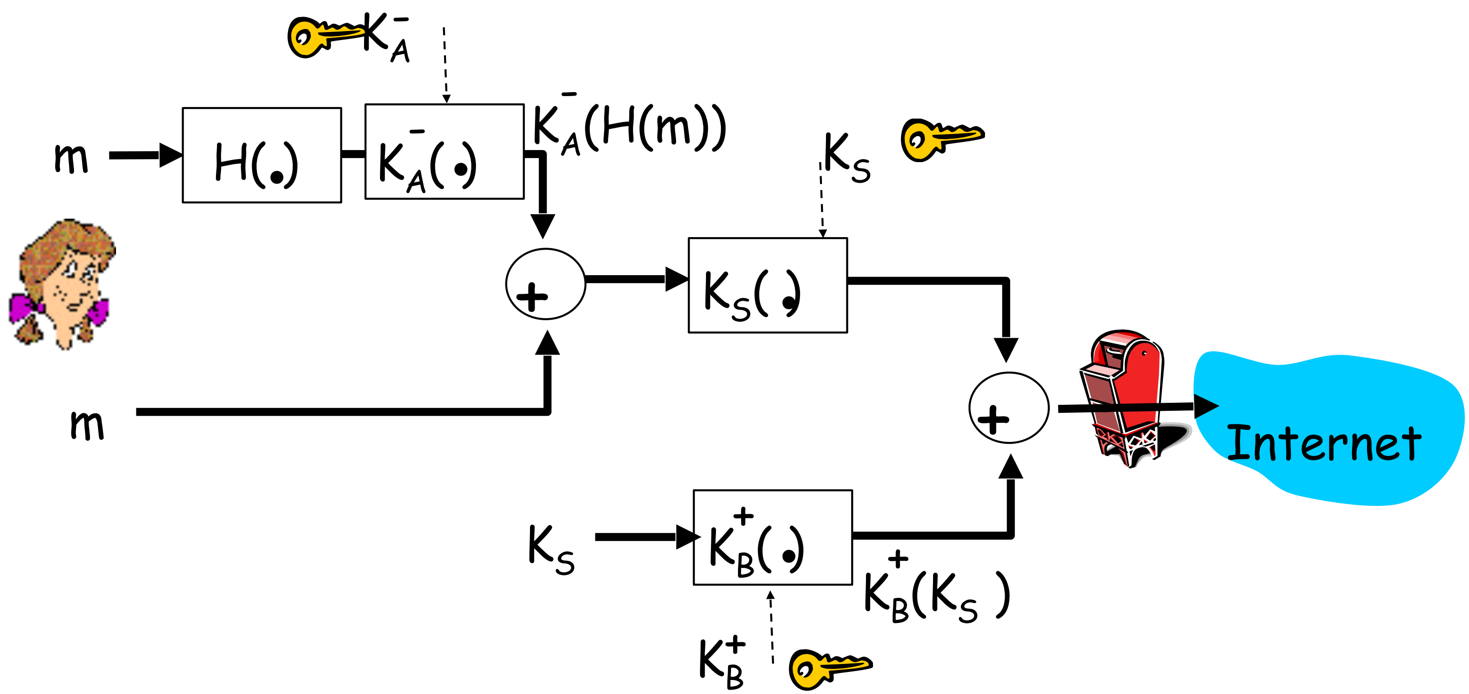
- Alice wants to provide sender authentication message integrity.



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

Secure e-mail (continued)

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



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

PGP

- Zimmermann en 91
- Standard de fait
- Utilise:
  - MD5 ou SHA pour le condensat (message digest)
  - CAST, triple-DES, IDEA pour la clef symétrique
  - RSA pour les clefs asymétriques

- A l'installation du software, création de la clef public→ publication par l'utilisateur (web par ex)
- Clef privé protégé par un mot de passe
- En option: signature digitale, chiffrage, les 2.
- Mécanisme de certification : clef public/ propriétaire PGP certifié par un « web de confiance »

roadmap

1 What is network security?

2 Principles of cryptography

3 Message integrity

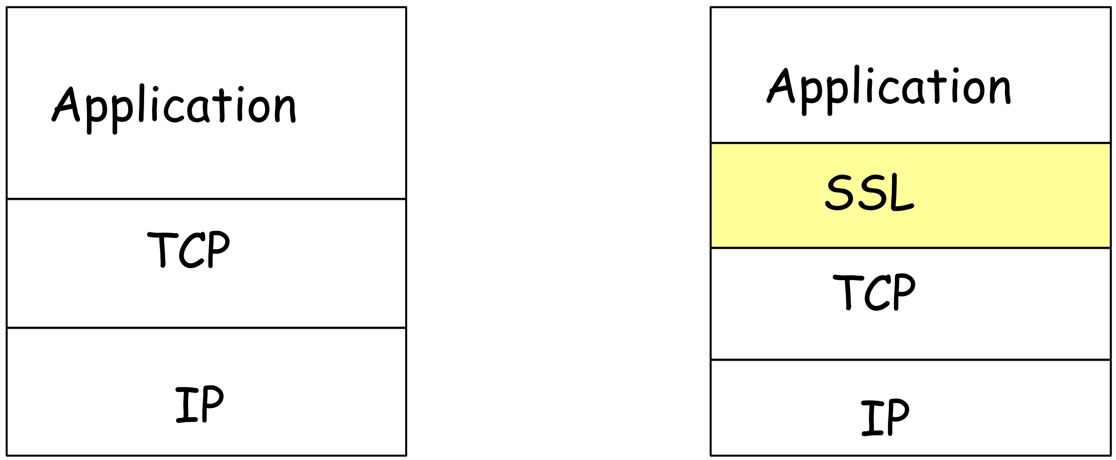
4 Securing e-mail

5 Securing TCP connections: SSL

SSL: Secure Sockets Layer

- Widely deployed security protocol
  - Supported by almost all browsers and web servers
  - https
  - Tens of billions $ spent per year over SSL
- Originally designed by Netscape in 1993
- Number of variations:
  - TLS: transport layer security, RFC 2246
- Provides
  - Confidentiality
  - Integrity
  - Authentication
- Original goals:
  - Had Web e-commerce transactions in mind
  - Encryption (especially credit-card numbers)
  - Web-server authentication
  - Optional client authentication
  - Minimum hassle in doing business with new merchant
- Available to all TCP applications
  - Secure socket interface

SSL and TCP/IP

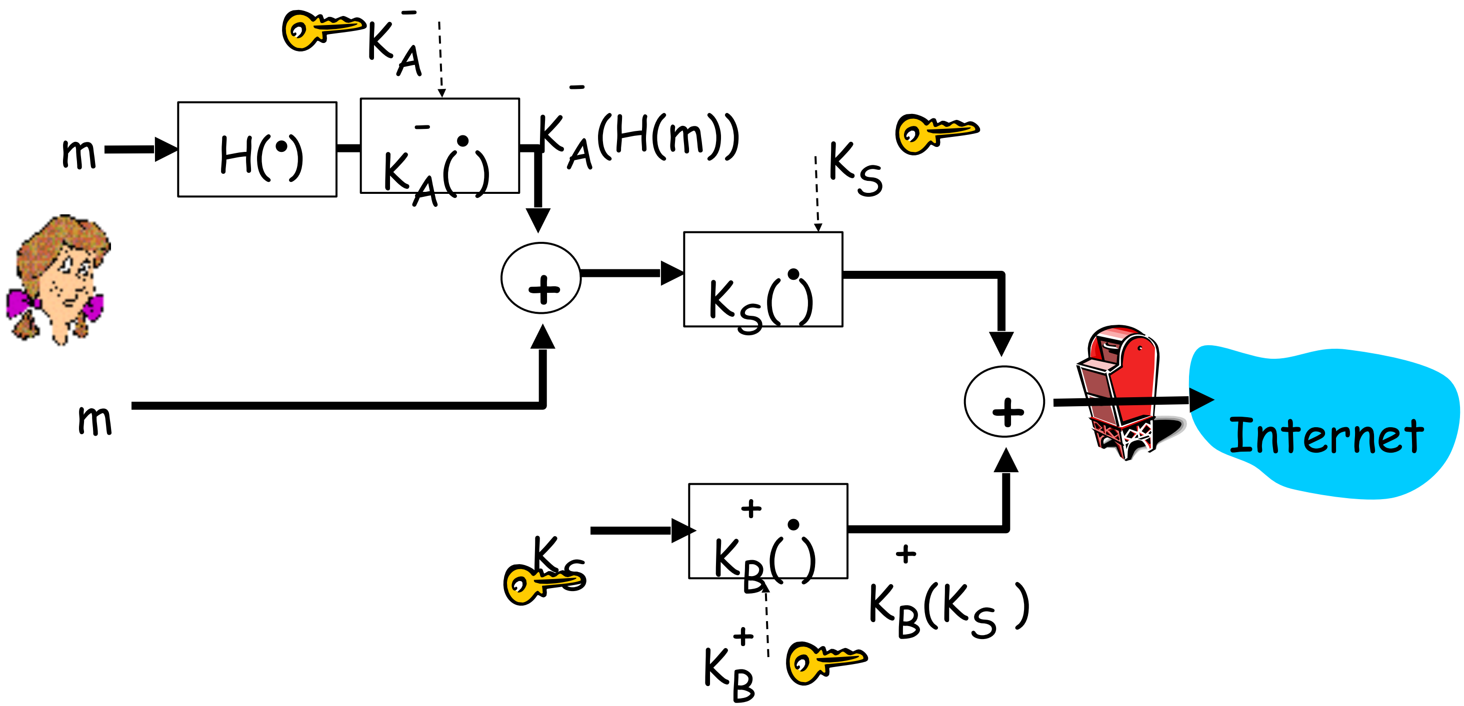


Normal Application

Application with SSL

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

Could do something like PGP:

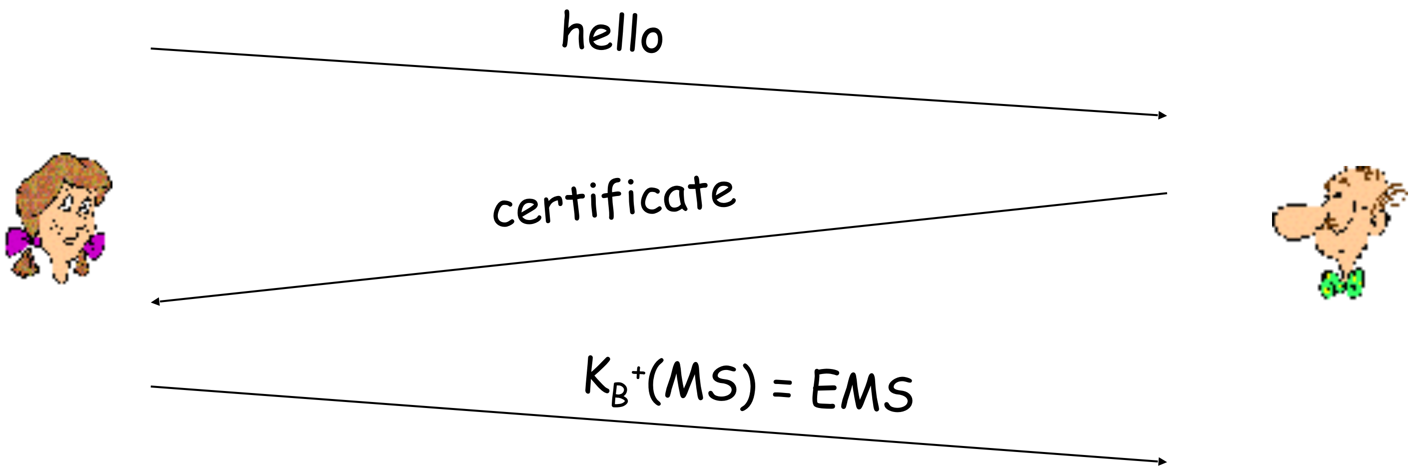


- But want to send byte streams & interactive data
- Want a set of secret keys for the entire connection
- Want certificate exchange part of protocol: handshake phase

Toy SSL: a simple secure channel

- Handshake: Alice and Bob use their certificates and private keys to authenticate each other and exchange shared secret
- Key Derivation: Alice and Bob use shared secret to derive set of keys
- Data Transfer: Data to be transferred is broken up into a series of records
- Connection Closure: Special messages to securely close connection

Toy: A simple handshake



- MS = master secret
- EMS = encrypted master secret

Toy: Key derivation

- Considered bad to use same key for more than one cryptographic operation
  - Use different keys for message authentication code (MAC) and encryption
- Four keys:
  - K_c = encryption key for data sent from client to server
  - M_c = MAC key for data sent from client to server
  - K_s = encryption key for data sent from server to client
  - M_s = MAC key for data sent from server to client
- Keys derived from key derivation function (KDF)
  - Takes master secret and (possibly) some additional random data and creates the keys

Toy: Data Records

- Why not encrypt data in constant stream as we write it to TCP?
  - Where would we put the MAC? If at end, no message integrity until all data processed.
  - For example, with instant messaging, how can we do integrity check over all bytes sent before displaying?
- Instead, break stream in series of records
  - Each record carries a MAC
  - Receiver can act on each record as it arrives
- Issue: in record, receiver needs to distinguish MAC from data
  - Want to use variable-length records

Toy: Sequence Numbers

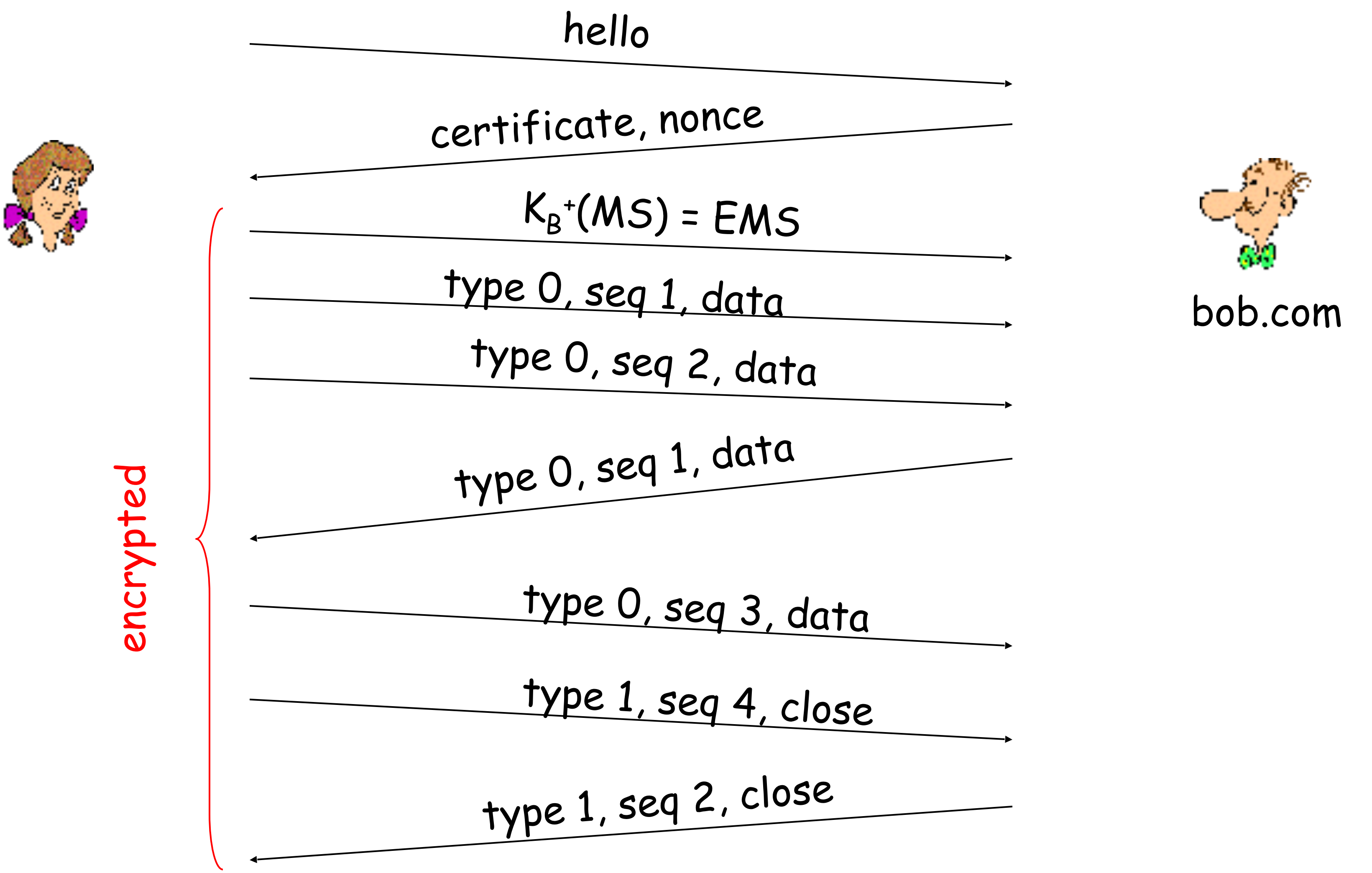
- Attacker can capture and replay record or re-order records
- Solution: put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - Note: no sequence number field
- Attacker could still replay all of the records
  - Use random nonce

Toy: Control information

- Truncation attack:
  - attacker forges TCP connection close segment
  - One or both sides thinks there is less data than there actually is.
- Solution: record types, with one type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, sequence||type||data)

length	type	data	MAC

Toy SSL: summary



Toy SSL isn't complete

- How long are the fields?
- What encryption protocols?
- No negotiation
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer

Most common symmetric ciphers in SSL

- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

Public key encryption

- RSA

SSL Cipher Suite

- Cipher Suite
  - Public-key algorithm
  - Symmetric encryption algorithm
  - MAC algorithm
- SSL supports a variety of cipher suites
- Negotiation: client and server must agree on cipher suite
- Client offers choice; server picks one

Real SSL: Handshake (1)

Purpose

1. Server authentication
2. Negotiation: agree on crypto algorithms
3. Establish keys
4. Client authentication (optional)

Real SSL: Handshake (2)

1. Client sends list of algorithms it supports, along with client nonce
2. Server chooses algorithms from list; sends back: choice + certificate + server nonce
3. Client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. Client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. Client sends a MAC of all the handshake messages
6. Server sends a MAC of all the handshake messages

Real SSL: Handshaking (3)

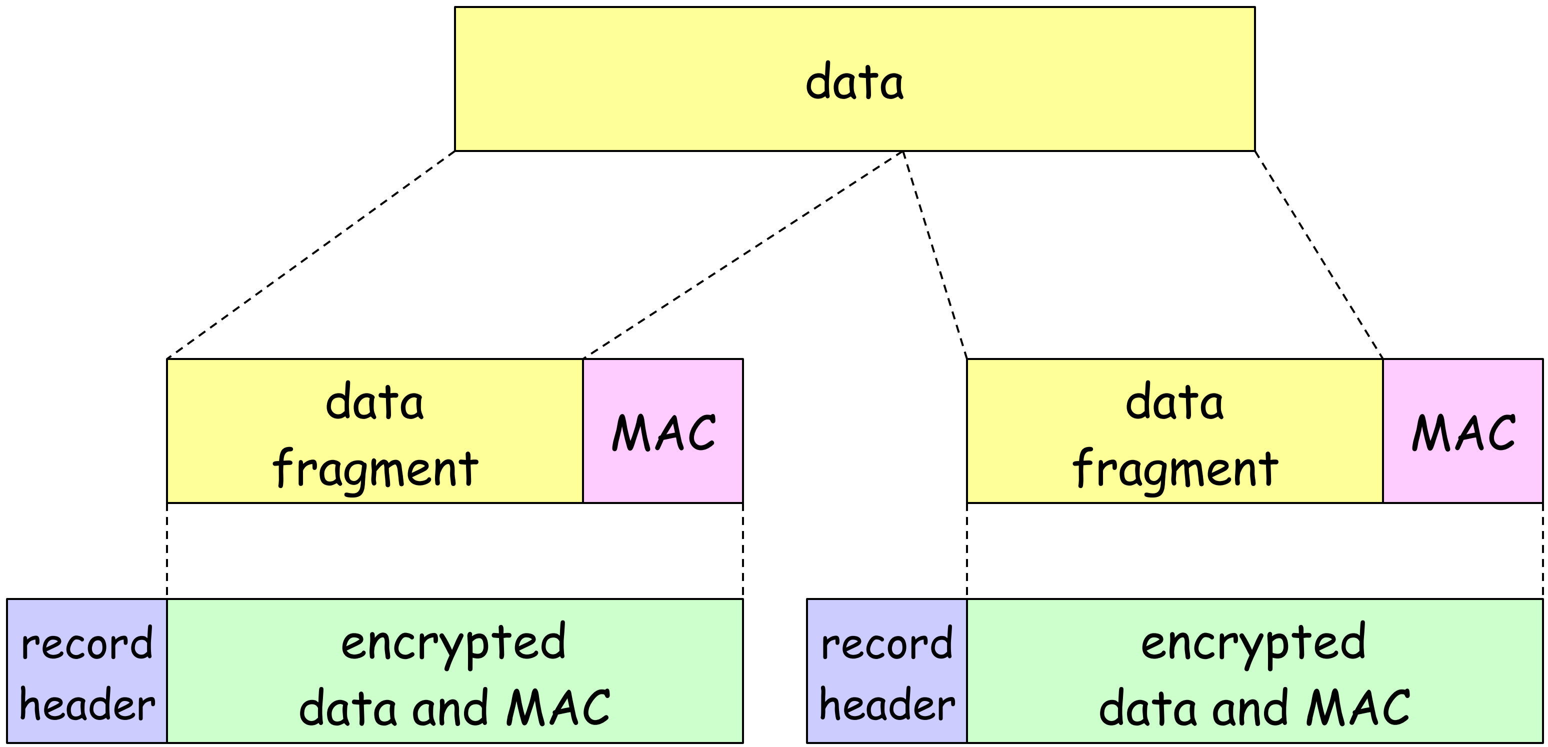
Last 2 steps protect handshake from tampering

- Client typically offers range of algorithms, some strong, some weak
- Man-in-the middle could delete the stronger algorithms from list
- Last 2 steps prevent this
  - Last two messages are encrypted

Real SSL: Handshaking (4)

- Why the two random nonces?
- Suppose Trudy sniffs all messages between Alice & Bob.
- Next day, Trudy sets up TCP connection with Bob, sends the exact same sequence of records.
  - Bob (Amazon) thinks Alice made two separate orders for the same thing.
  - Solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days.
  - Trudy's messages will fail Bob's integrity check.

SSL Record Protocol

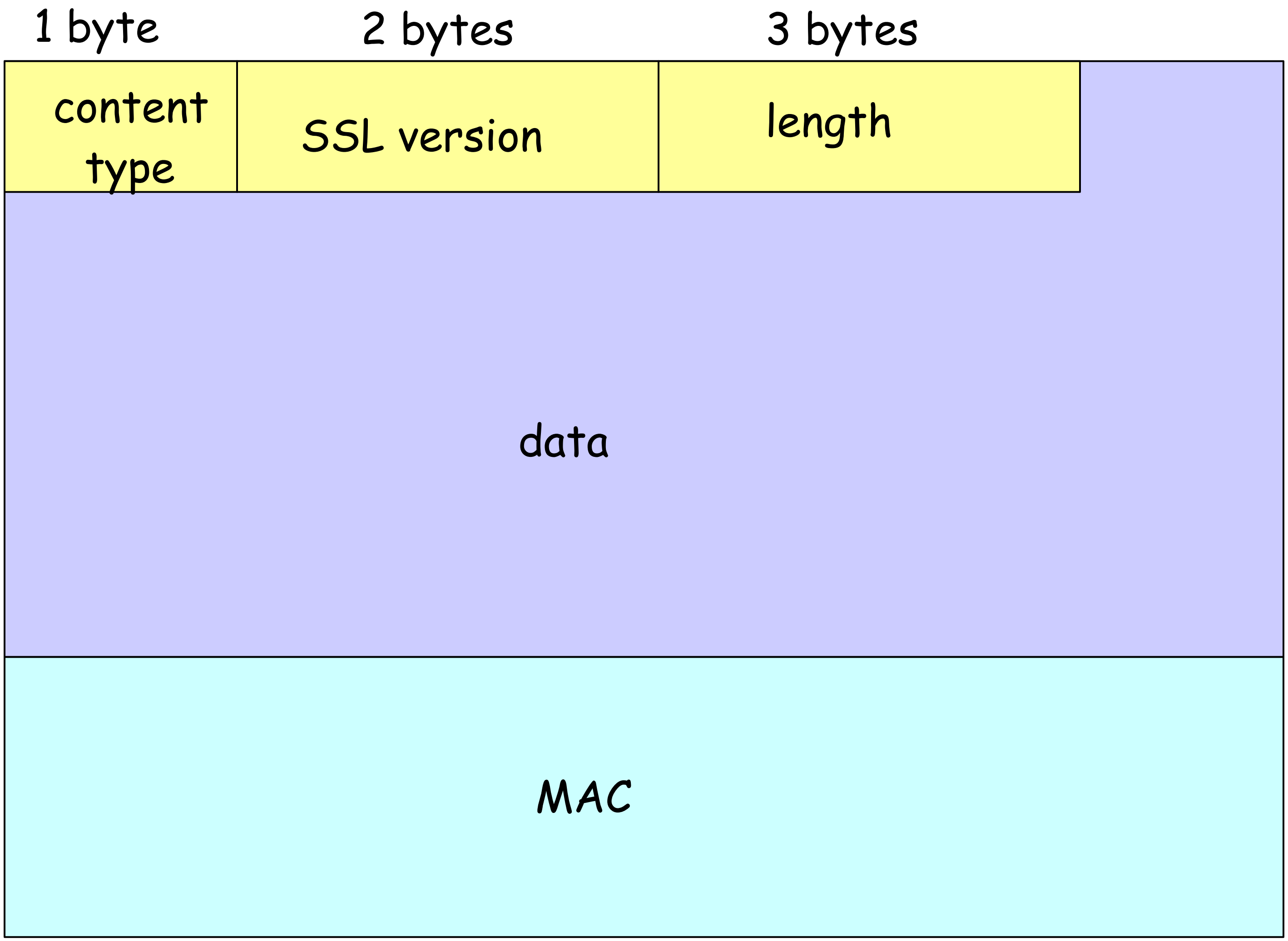


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

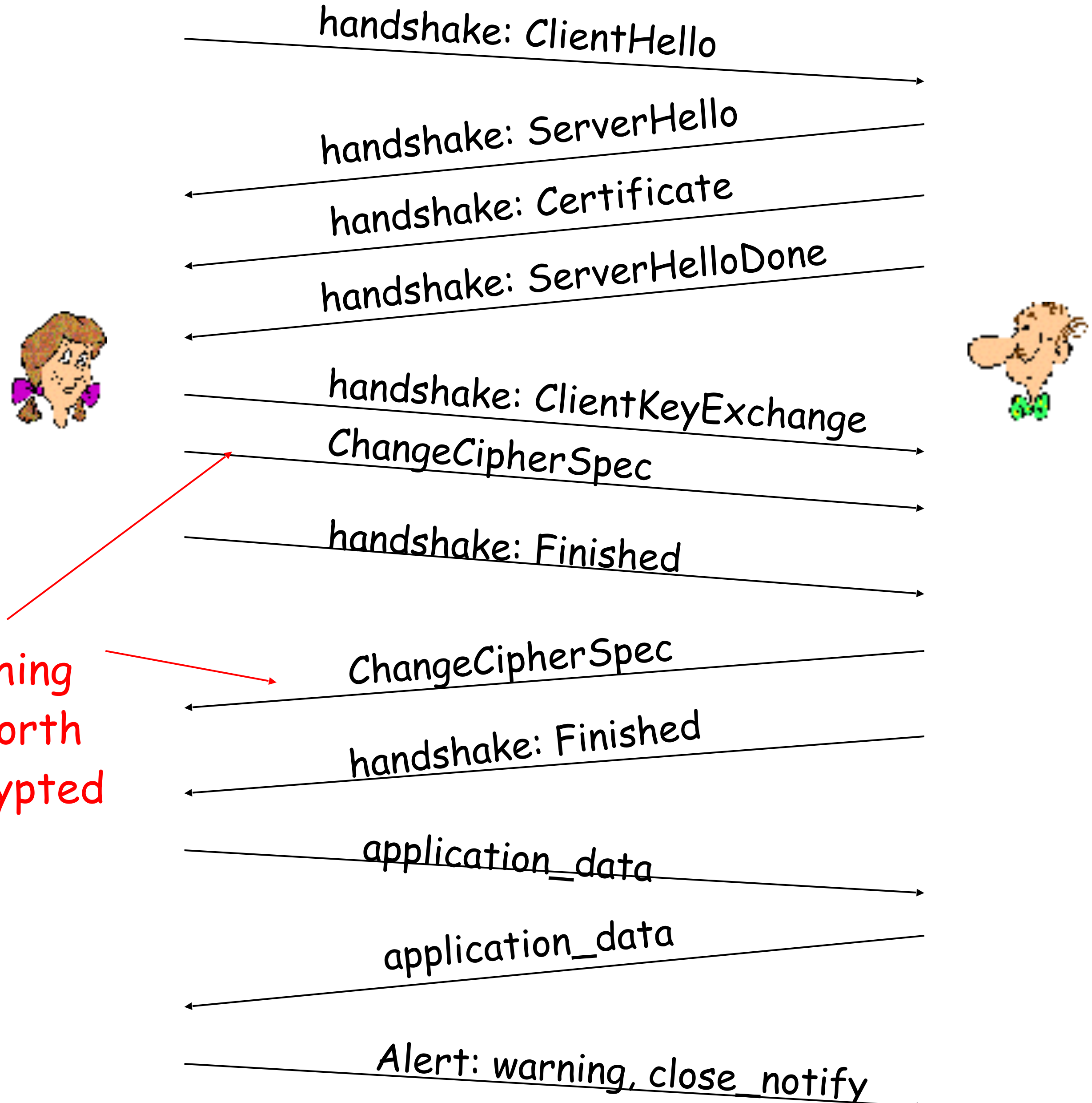
Fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL Record Format



Data and MAC encrypted (symmetric algo)

Real Connection



Everything henceforth is encrypted

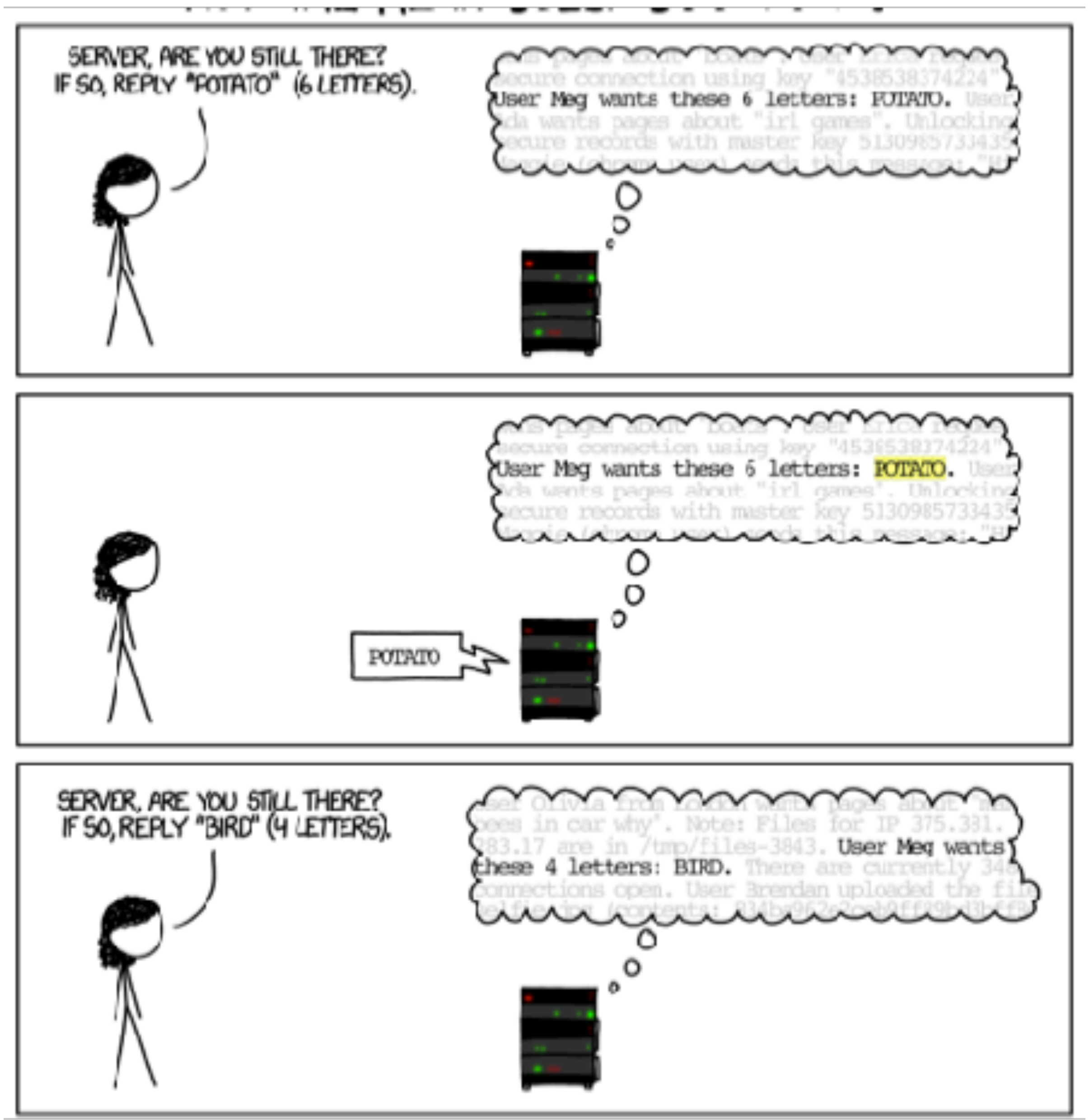
TCP Fin follow

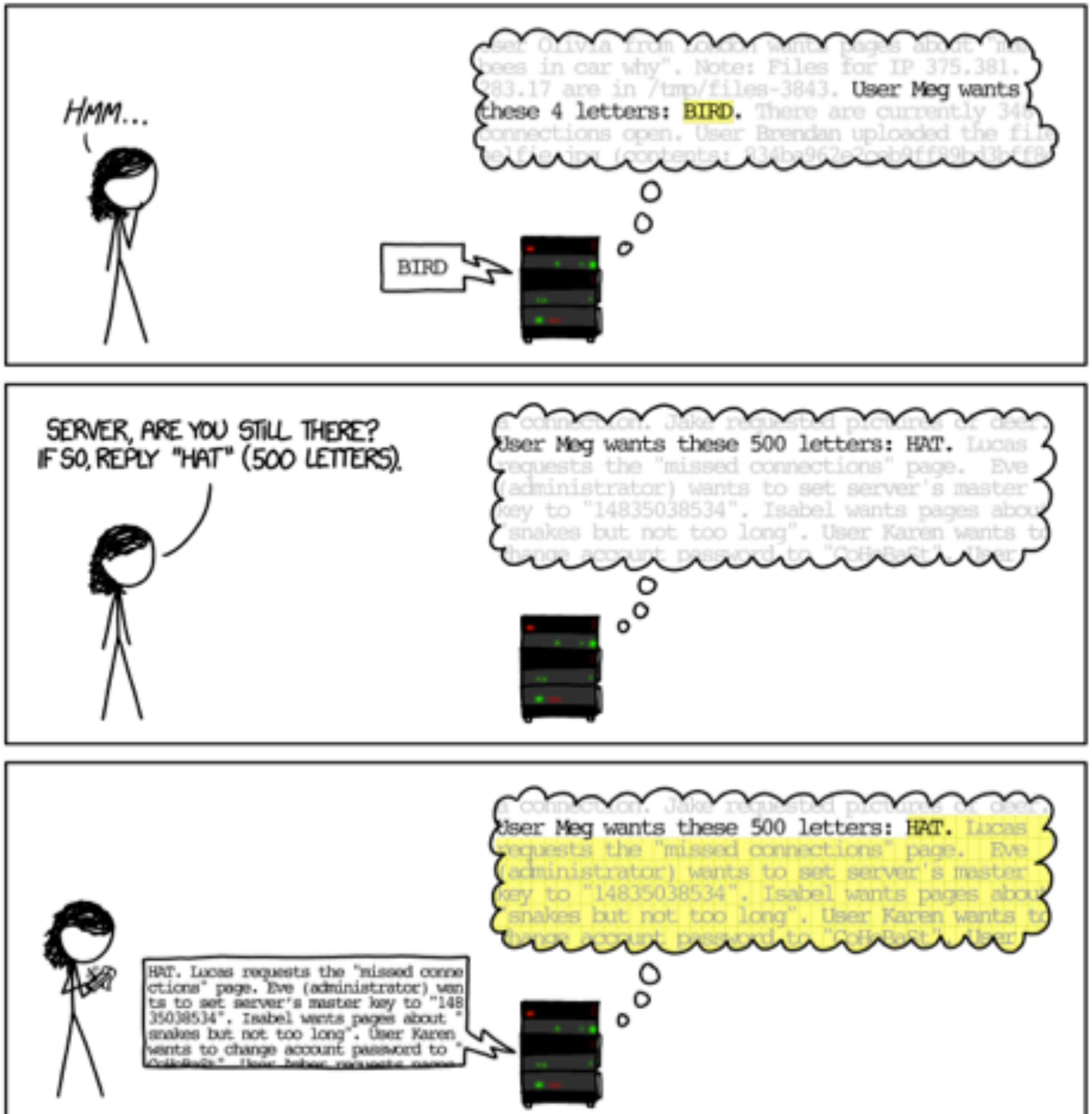
Key derivation

- Client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - Produces master secret
- Master secret and new nonces inputed into another random-number generator: “key block”
  - Because of resumption: TBD
- Key block sliced and diced:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)

HeartBleed bug www.xkde.com

- Introduite par erreur dans OpenSSL
- Sites affectées: Akamai Technologies, Amazon, SourceForge, GitHub,.....


SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
User Meg wants these 6 letters: POTATO.
User Meg wants these 6 letters: POTATO.
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Meg wants these 4 letters: BIRD.


HMM...
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 34
connections open. User Brendan uploaded the fi
BIRD
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT. Lucas
requests the "missed connections" page. Eve
(administrator) wants to set server's master
key to "14835038534". Isabel wants pages abou
"snakes but not too long". User Karen wants t
User Meg wants these 500 letters: HAT. Lucas
requests the "missed connections" page. Eve
(administrator) wants to set server's master
key to "14835038534". Isabel wants pages abou
"snakes but not too long". User Karen wants t
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

2014: l'année où toutes les piles TLS majeures ont fait l'objet de vulnérabilités critiques

- février : goto fail Apple
- février : goto fail GnuTLS
- avril : Heartbleed dans OpenSSL
- juin : Early CCS dans OpenSSL
- septembre : Universal signature forgery dans NSS (Mozilla)
- septembre : Universal signature forgery dans CyaSSL
- septembre : Universal signature forgery dans PolarSSL
- novembre : exécution de code arbitraire dans SChannel (MS)

- CVE-2014-0244

- CVE.mitre.org

Common Vulnerabilities and Exposures
The Standard for Information Security Vulnerability Names