ELEC5471M

Data Communications and Network Security

Lectures on Network Security
Handout 3: Email Security & SSL

Mohsen Razavi

Institute of Integrated Information Systems
School of Electronic and Electrical Engineering
University of Leeds

What would network security entail?

Consider the email example. As a user, what would you expect from this service?

- We may like that nobody, but the recipient, learns about the content → confidentiality
- We expect the message arrives safely with no changes in the content → message integrity
- The sender needs to make sure that the intended recipient picks up the email; similarly, the recipient needs to make sure that the message is genuinely sent by the declared sender → authentication

Confidentiality is provided by encryption techniques:

- Symmetric key cryptosystems
- Public key crypto systems

- Public key systems are slow, but they can be used to provide the initial seed for faster symmetric key cryptosystems such as AES

Summary of ideas used in authentication

- So, authentication is possible via the following techniques:
  - Public-key signature: the private key of each user (K^-) can be used to digitally sign the message
  - Certification authority (CA): We can verify the public key of any user registered at the CA. Each user has its own certificates.
  - Nonce and sequence number: To certify that the session is live, and it is not a playback attack
  - Message digest: to apply a Hash function to the message to get a fixed-size output.
  - Message authentication code (MAC): A message digest for a combination of a message and a secret key. It provides message integrity and authentication for two (trusted) users.
  - Sharing a secret key using public-key cryptography

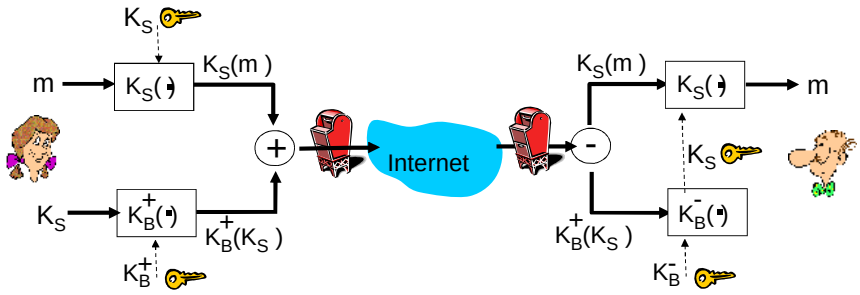
Let's apply all these in practice. Next, we look at how security is achieved in emails and at the transport layer (SSL).

Secure email

- Phil Zimmerman, in 1991, devised a pretty good privacy (PGP) protocol that enabled confidential and authenticated transmission of an email.
- Here, we look at PGP in three steps:
  - First we see how PGP provides confidentiality;
  - Then, we look at how it enables sender's authentication;
  - And, finally, we combine both above techniques to provide both confidentiality and authentication for emails
- Zimmerman case: criminal investigation of Zimmermann, for allegedly violating the Arms Export Control Act; the case was dropped in 1996.
- PGP initially relied on a web network of trust for its authentication; the more recent versions use CAs as well

Confidential email

- Suppose Alice wants to send a confidential email, m, to Bob. Then,



Alice:

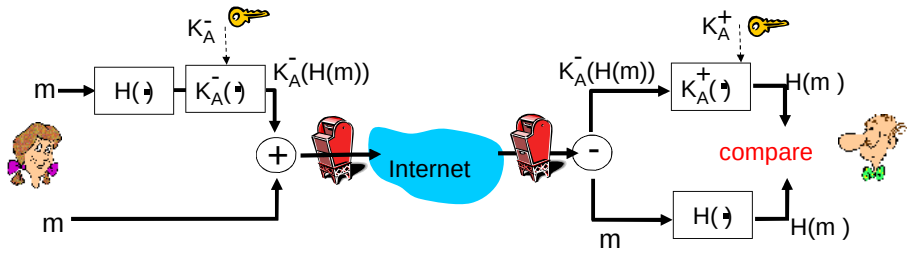
- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob

Bob:

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

Authenticated email

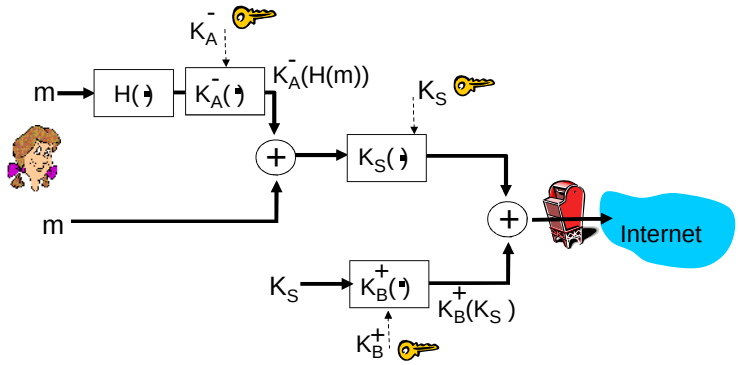
- Suppose Alice wants to send an authenticated email, m, to Bob. Then,



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

Secure email

- Now, let's achieve both confidentiality and authentication:



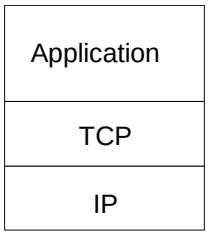
- *Alice uses three keys:* her private key, Bob's public key, and newly created symmetric key

Securing TCP connections: SSL

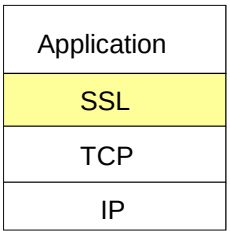
SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation -TLS: transport layer security, RFC 2246
- provides
  - *confidentiality*
  - *integrity*
  - *authentication*
- original goals:
  - Web e-commerce transactions
  - 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



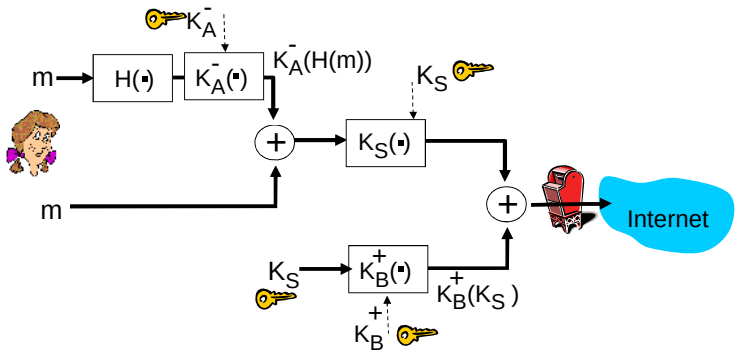
• *normal application*



• *application with SSL*

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

Why not using something like PGP?

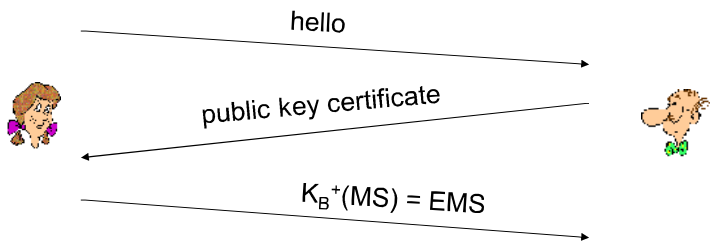


- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

Toy SSL: a simple secure channel

- *handshake:* Alice and Bob use their certificates, 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 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.
  - e.g., 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

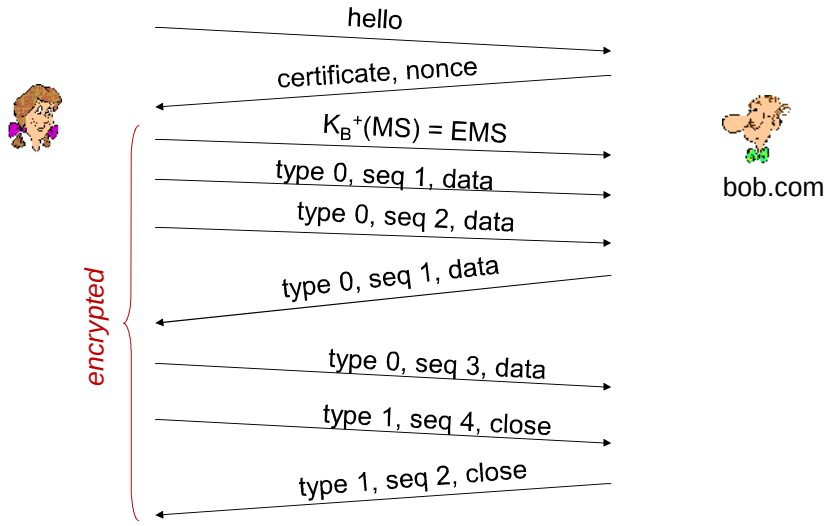
- *problem:* attacker can capture and replay record or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field

- *problem:* attacker could replay all records
- *solution:* use nonce

Toy: control information

- *problem:* 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)

Toy SSL: summary



Toy SSL isn't complete

Here is more detail for you to do your wireshark project

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer

SSL cipher suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
  - client offers choice
  - server picks one

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

SSL Public key encryption
- RSA

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)

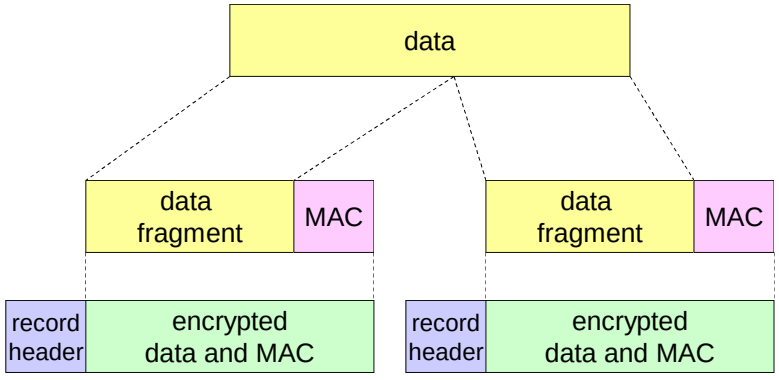
last 2 steps protect handshake from tampering

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

Real SSL: handshaking (4)

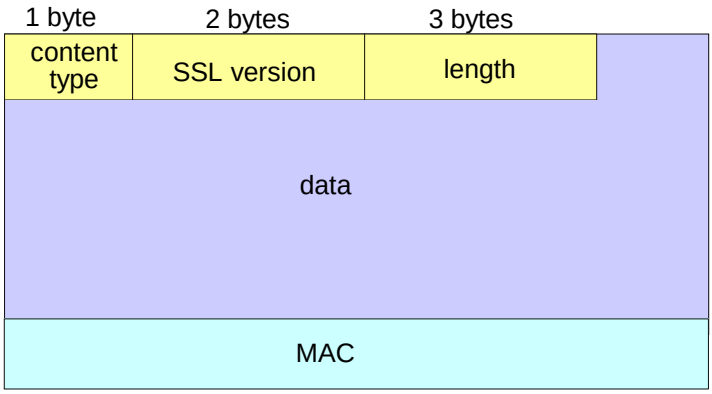
- why two random nonces?
- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection.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

SSL record protocol



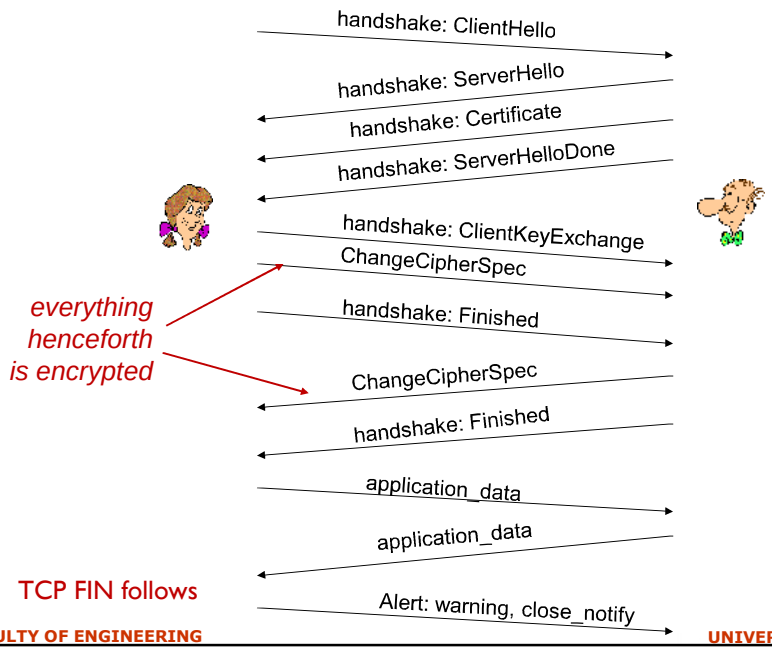
- *record header:* content type; version; length
- *MAC:* includes sequence number, MAC key M_x
- *fragment:* each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL record format



- data and MAC encrypted (symmetric algorithm)

Real SSL Connection



Key derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - produces master secret
- master secret and new nonces input 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)

Summary of learning outcomes

- We learned about three key problems, among others, that network security is dealing with.
- We looked at encryption techniques, which provide us with privacy and confidentiality.
- We saw symmetric-key techniques (one-time pad, DES, AES), and public-key systems (RSA)
- We saw how we can provide message integrity and authentication using either schemes
- We saw how one can use RSA to share secret keys among users. Such a secret can then be used for other applications.
- We looked at how our email communication is made secure.
- We finally got an idea of how e-commerce can be made safe using SSL for securing TCP connections.