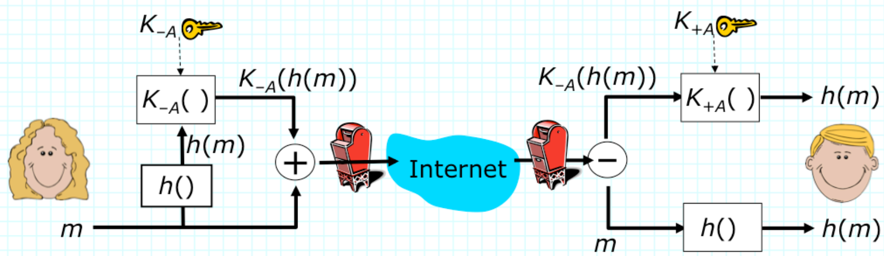
18. Application, Network and Link Layer Security

- Secure email
- Secure Socket Layer (SSL)
- Securing VPNs with IPSec
- Securing Wireless LANs

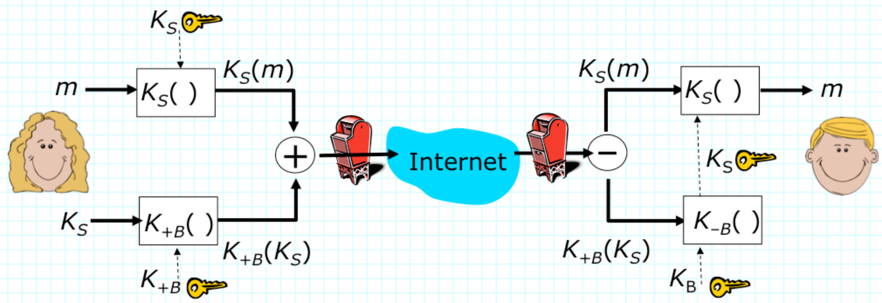
Jon Turner – based on slides from Kurose & Ross

Sending a Signed Email



- Alice wants to send signed e-mail m, to Bob
  - » computes message digest (hash of message) and encrypts it with private key
  - » sends both m and $K_{-A}(h(m))$ to Bob
- Bob checks message
  - » uses Alice's public key to obtain original message digest
  - » computes message digest directly and compares

Sending Confidential Email



- Alice wants to send confidential e-mail, *m*, to Bob
  - » 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 private key to recover symmetric key created by Alice, then decrypts message

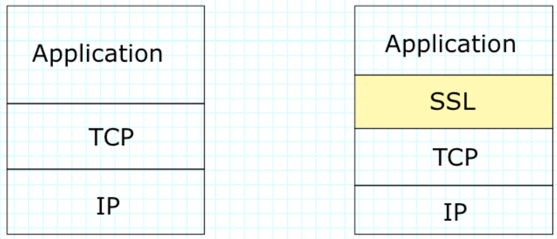
Washington University in St. Louis — Engineering

Issues with Secure Email

- Many mail clients support encrypted email (Outlook, Thunderbird, Apple)
  - » straightforward to use, in principle
- Key distribution problem inhibits widespread use
  - » need correspondent's public key in order to encrypt messages
  - » but how do you get their key in reliable way
- Original PGP system used so-called "web-of-trust"
  - » individuals to certify keys of other individuals they know
  - » appealing idea, but has not been broadly successful
- Alternate approach uses certificates obtained from certificate authority
  - » less effective than for web-site authentication
  - » certificate cost barrier to users, little benefit until universal

4

Secure Sockets Layer



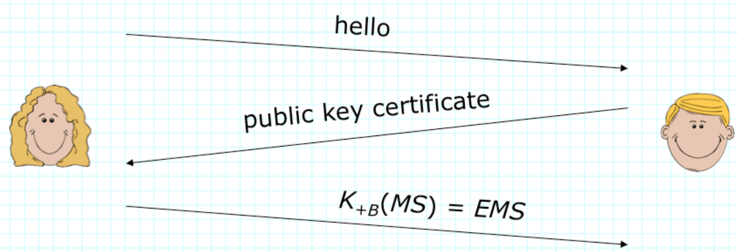
normal application *application with SSL*

- Supports secure TCP connections
  - » provides privacy, authentication, data integrity
  - » used by almost all web browsers (https) for e-commerce
  - » application libraries available for C/C++, python, Java,...
- Note: TCP header is *not* protected
- TLS (RFC 2246) is IETF standard version

Main Phases in SSL

- *Handshake:* Alice and Bob exchange and verify certificates, agree on shared secret
  - » usually, only server provides certificate
  - » most data sent in clear at this point
- *Key derivation:* Alice and Bob use shared secret to derive set of keys
  - » different keys for different purposes
- *Data transfer:* data to be transferred is broken up into series of records
  - » data integrity checked for each record
- *Connection closure:* special messages to securely close connection
  - » prevents premature termination by an attacker

Basic Handshake (simplified)



MS: master secret

EMS: encrypted master secret

Washington University in St. Louis — Engineering

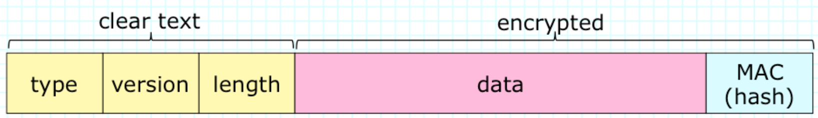
Key Derivation

- 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

8

Data Records

- Why not encrypt data in constant stream?
  - » so we can check data integrity as we process stream
    - e.g., with instant messaging, we want to check integrity of each line, not just the whole session
  - » but if stream is encrypted, why do we need to check integrity?
    - attacker can still corrupt data in way that might disrupt application
- So, break stream in series of records
  - » to support variable length records, add length field
  - » each record carries a hash based on secret MAC key, *M*
    - hash computed over entire record
  - » receiver can check and act on each record as it arrives

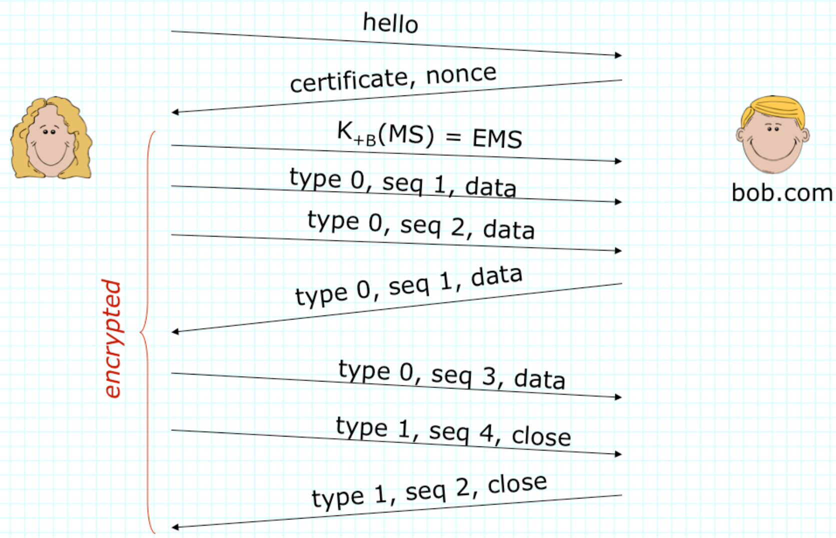


Washington University in St. Louis — Engineering

Replay Attacks, Premature Closure

- Problem: attacker can capture and replay records or re-order records
  - » solution: include sequence number when performing MAC hash
    - h_M(seq#+record)
  - » note: no explicit sequence number field
    - sender/receiver simply count records and use appropriate seq#
- Problem: attacker could replay entire session
  - » solution: define random nonce at start of session and use it to generate keys
- Problem: attacker could close connection early using FIN
  - » solution: type field has special value for last record of session

Basic SSL: Summary



More Details

- SSL supports several cipher suites
  - » symmetric encryption algorithm
    - options include DES, 3DES, AES, RC2, RC4
  - » public-key algorithm - RSA
  - » MAC algorithm - MD5, SHA
- Cipher suite negotiated during handshake
  - » client offers choice
  - » server picks one

Washington University in St. Louis — Engineering

SSL Handshake Details

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/Server exchange hash of all the handshake messages (these are encrypted)
   » to detect tampering of handshakes (such as removing stronger encryption methods from list of options)

13

Implementing SSL/TLS Apps in Java

- To implement SSL/TLS apps, need key pair
- Java apps obtain keys and certificates from a *keystore*
  - » a keystore is a password-protected binary file containing multiple entries
    - *key entry* holds private key and certificate containing public key
    - *certificate entry* contains a certificate with public key of some "trusted peer"
    - a *truststore* is a keystore with only certificate entries
    - each entry is identified by a string called an "alias"
- *Keytool* is a utility for creating/managing keystores
  - » keytool -genkey -alias mykey -keystore kstore
  - » keytool –list –keystore kstore
  - » keytool -export -keystore kstore -alias myKey -file certs.cer
  - » keytool -import -keystore tstore -alias myCert -file certs.cer

Washington University in St.Louis

Engineering

Secure Echo Server

```
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import java.io.*;
public class EchoServer {
  public static void main(String[] args) throws Exception
    System.console().writer().print("password:");
    System.console().writer().flush();
    char[] password = System.console().readPassword();

    System.setProperty("javax.net.ssl.keyStore",args[0]);
    System.setProperty("javax.net.ssl.keyStorePassword",
          new String(password));

    SSLServerSocket listenSock = (SSLServerSocket)
       SSLServerSocketFactory.getDefault().createServerSocket(30123);
    SSLSocket connSock = (SSLSocket) listenSock.accept();

    BufferedReader fromClient = ...;
    BufferedWriter toClient = ...;
    String string = null;
    while ((string = fromCllient.readLine()) != null) {
      toClient.write(string); toClient.newLine(); toClient.flush();
} } }
```

Get password for keystore

Set system properties identifying keystore file and its password

from connSock

Create secure listening socket, then accept incoming

15

Secure Echo Client

Set system properties identifying truststore file and its password

```
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.io.*;
public class EchoClient {
  public static void main(String[] args) throws Exception {
    System.setProperty("javax.net.ssl.trustStore",args[1]);
    System.setProperty("javax.net.ssl.trustStorePassword",
            "echoECHO");

    SSLSocket sock = (SSLSocket)
      SSLSocketFactory.getDefault().createSocket(args[0], 30123);

    BufferedReader sysin = ...;
    BufferedReader fromServer ...;
    BufferedWriter toServer ...;

    String string = null;
    while ((string = sysin.readLine()) != null) {
      toServer.write(string); toServer.newLine();
      toServer.flush();
      System.out.println(fromServer.readLine());
} } }
```

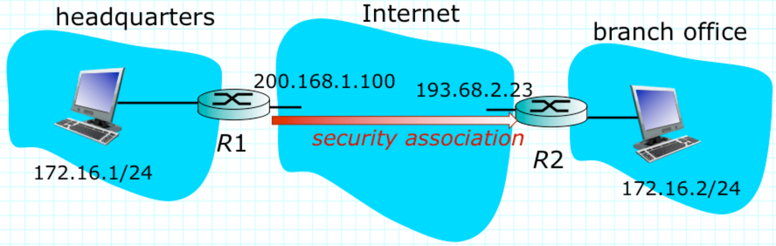
Create secure socket to remote server

Network Layer Security – IPsec

- Protects *all* data sent between two network layer components
  - » sending component encrypts datagram payload
    - could be TCP or UDP segment, ICMP message, OSPF message
- Used mainly for *Virtual Private Networks* (VPN)
  - » allows remote host to communicate securely with corporate network across public internet using encrypted tunnel
  - » two main protocols
    - *Authentication Protocol* (AP) – authentication, message integrity
    - *Encapsulation Security Protocol* (ESP) – also, confidentiality
- IPsec operates between pairs of endpoints
  - » requires some shared state, which is called a *Security Association* (SA)
    - an SA supports one-way communication, so typically used in pairs

Washington University in St. Louis — Engineering

Example SA from R1 to R2



R1 stores for SA:

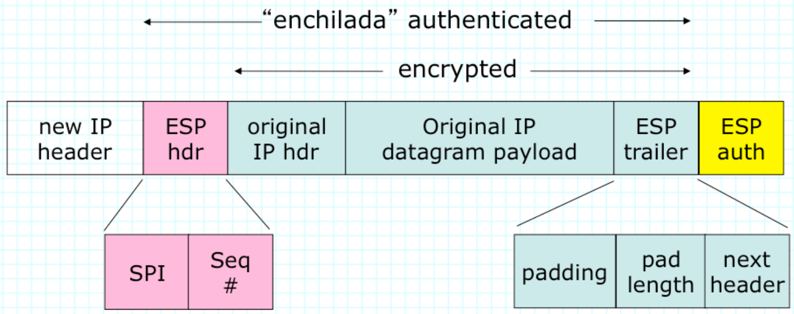
- 32-bit SA identifier: *Security Parameter Index (SPI)*
- origin SA interface (200.168.1.100)
- destination SA interface (193.68.2.23)
- type of encryption used (e.g., 3DES with CBC)
  - » and encryption key
- type of integrity check used (e.g., HMAC with MD5)
  - » and authentication key

18

SAD and SPD

- SA Database
  - holds state information for all SAs
  - when sending IPsec datagram,sender accesses SAD to determine how to process datagram
  - when IPsec datagram is received, SPI in IPsec header used to select entry from receiver's SAD
- Security Policy Database
  - used by gateway router to decide if IPsec should be used when forwarding an outgoing packet
    - not all packets require IPsec
  - looks for entry in Security Policy Database, based on protocol and source and destination IP addresses
  - entry specifies which SA to use

IPsec Datagram – tunnel mode



- ESP trailer: padding for block ciphers
- ESP header:
  - » SPI, so receiving entity knows what to do
  - » sequence number, to thwart replay attacks (no wrap-around)
- MAC in ESP authentication field is created with shared secret key

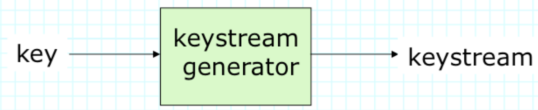
Creating Security Associations

- Can be done manually, but that's usually not practical
  - » SAs can be created automatically using Internet Key Exchange protocol (IKE)
- IKE operates in two phases
  - » first phase creates a secure channel used in second phase
    - includes authentication to verify identities of endpoints
  - » second phase used to create one or more SAs for use between the two entities

Securing Wireless LANs

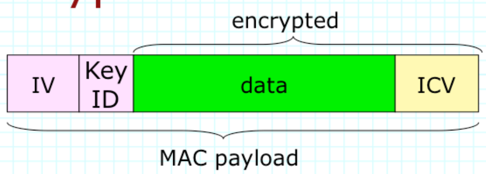
- Wired Equivalent Privacy (WEP) was original security protocol for 802.11
  - » not very secure, but still makes useful case study
  - » uses symmetric key cryptography to provide confidentiality, end-host authorization and data integrity
    - keys are exchanged "out-of-band"
  - » self-synchronizing: each packet separately encrypted
  - » designed for efficiency – implementable in hardware or software
- 802.11i standard includes much stronger security mechanisms
  - » choice of encryption methods
  - » separate authentication server
    - typically uses public key encryption for authentication and key distribution

Symmetric Stream Ciphers



- Combine each byte of keystream with byte of plaintext to get ciphertext:
  - » $m(i)$ = i-th unit of message
  - » $ks(i)$ = i-th unit of keystream
  - » $c(i)$ = i-th unit of ciphertext
  - » $c(i) = ks(i) \oplus m(i)$ ($\oplus$ = exclusive or)
  - » $m(i) = ks(i) \oplus c(i)$
- WEP uses RC4
  - » key combines a shared secret (40 or 104 bits) with a 24 bit *Initialization Vector* (IV) to generate key stream
    - separate IV per frame, sent as clear text

WEP Encryption



- Integrity Check Value computed over data
  - » four-byte hash/CRC for data integrity
- Initialization Vector (IV) created for each packet
  - » sent with packet
  - » sender and receiver combine IV with shared key to initialize keystream generator
- Key ID is an 8-bit identifier
- Data in frame + ICV are encrypted with RC4
  - » bytes of keystream are XORed with bytes of data & ICV
  - » IV & keyID are appended to encrypted data to create payload
  - » payload inserted into 802.11 frame

Breaking 802.11 WEP encryption

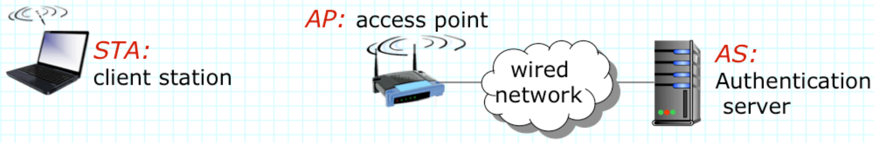
Security hole:

- 24-bit IV, one IV per frame, -> IV's eventually reused
- IV transmitted in plaintext, so IV reuse easily detected

Attack:

- » Trudy induces Alice to encrypt known plaintext $d_1\ d_2\ d_3\ d_4$ …
- » Trudy sees: $c_i = d_i$ XOR k_i^{IV}
- » Trudy knows $c_i\ d_i$, so can compute k_i^{IV}
- » Trudy knows encrypting key sequence $k_1^{IV}\ k_2^{IV}\ k_3^{IV}$ …
- » next time IV is used, Trudy can decrypt!

802.11i: four phases of operation



1. Discovery of security capabilities

2. STA and AS mutually authenticate, together generate Master Key (MK). *AP serves as "pass through"*

3. STA derives Pairwise Master Key (PMK)

3. AS derives same PMK, sends to AP

4. STA, AP use PMK to derive Temporal Key (TK) used for message encryption, integrity

EAP: Extensible Authentication Protocol

- EAP: end-end client (mobile) to authentication server protocol
- EAP sent over separate “links”
  - » mobile-to-AP (EAP over LAN)
  - » AP to authentication server (RADIUS over UDP)