Lecture 8
October 14, 2024

Needham-Schroeder

$$\text{Alice} \xrightarrow{\text{Alice} \,||\, \text{Bob} \,||\, r_1} \text{Cathy}$$

$$\text{Alice} \xleftarrow{\{\, \text{Alice} \,||\, \text{Bob} \,||\, r_1 \,||\, k_s \,||\, \{\, \text{Alice} \,||\, k_s \,\}\, k_B \,\}\, k_A} \text{Cathy}$$

$$\text{Alice} \xrightarrow{\{\, \text{Alice} \,||\, k_s \,\}\, k_B} \text{Bob}$$

$$\text{Alice} \xleftarrow{\{\, r_2 \,\}\, k_s} \text{Bob}$$

$$\text{Alice} \xrightarrow{\{\, r_2 - 1 \,\}\, k_s} \text{Bob}$$

Argument: Alice talking to Bob

- Second message
  - Enciphered using key only she, Cathy knows
    - So Cathy enciphered it
  - Response to first message
    - As r_1 in it matches r_1 in first message
- Third message
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Any messages enciphered with that key are from Bob

Argument: Bob talking to Alice

- Third message
  - Enciphered using key only he, Cathy know
    - So Cathy enciphered it
  - Names Alice, session key
    - Cathy provided session key, says Alice is other party
- Fourth message
  - Uses session key to determine if it is replay from Eve
    - If not, Alice will respond correctly in fifth message
    - If so, Eve can’t decipher r_2 and so can’t respond, or responds incorrectly

Denning-Sacco Modification

- Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
  - In what follows, Eve knows k_s

$$\text{Eve} \xrightarrow{\{ \text{Alice} \mid\mid k_s \}\, k_B} \text{Bob}$$

$$\text{Eve} \xleftarrow{\{ r_2 \}\, k_s} \text{Bob}$$

$$\text{Eve} \xrightarrow{\{ r_2 - 1 \}\, k_s} \text{Bob}$$

Problem and Solution

- In protocol above, Eve impersonates Alice
- Problem: replay in third step
  - First in previous slide
- Solution: use time stamp *T* to detect replay
- Weakness: if clocks not synchronized, may either reject valid messages or accept replays
  - Parties with either slow or fast clocks vulnerable to replay
  - Resetting clock does *not* eliminate vulnerability

Needham-Schroeder with Denning-Sacco Modification

Alice $\xrightarrow{\text{Alice} \mid\mid \text{Bob} \mid\mid r_1}$ Cathy

Alice $\xleftarrow{\{\, \text{Alice} \mid\mid \text{Bob} \mid\mid r_1 \mid\mid k_s \mid\mid \{\, \text{Alice} \mid\mid T \mid\mid k_s \,\}\, k_B \,\}\, k_A}$ Cathy

Alice $\xrightarrow{\{\, \text{Alice} \mid\mid T \mid\mid k_s \,\}\, k_B}$ Bob

Alice $\xleftarrow{\{\, r_2 \,\}\, k_s}$ Bob

Alice $\xrightarrow{\{\, r_2 - 1 \,\}\, k_s}$ Bob

Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party (“Cathy”)
- Ticket
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender

Idea

- User u authenticates to Kerberos server
  - Obtains ticket $T_{u,TGS}$ for ticket granting service (TGS)
- User u wants to use service s:
  - User sends authenticator A_u, ticket $T_{u,TGS}$ to TGS asking for ticket for service
  - TGS sends ticket $T_{u,s}$ to user
  - User sends A_u, $T_{u,s}$ to server as request to use s
- Details follow

Ticket

- Credential saying issuer has identified ticket requester
- Example ticket issued to user *u* for service *s*

$$T_{u,s} = s \,||\, \{\, u \,||\, u\text{'s address} \,||\, \text{valid time} \,||\, k_{u,s} \,\}\, k_s$$

where:

- $k_{u,s}$ is session key for user and service
- Valid time is interval for which ticket valid
- *u*'s address may be IP address or something else
  - Note: more fields, but not relevant here

Authenticator

- Credential containing identity of sender of ticket
  - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user *u* generates for service *s*

$$A_{u,s} = \{ u \,||\, \text{generation time} \,||\, k_t \}\, k_{u,s}$$

  where:
  - k_t is alternate session key
  - Generation time is when authenticator generated
    - Note: more fields, not relevant here

Protocol

$user \xrightarrow{\quad user \mid\mid TGS \quad} \text{AS}$

$\text{AS} \xleftarrow{\quad \{ k_{u,TGS} \} k_u \mid\mid T_{u,TGS} \quad} user$

$user \xrightarrow{\quad service \mid\mid A_{u,TGS} \mid\mid T_{u,TGS} \quad} \text{TGS}$

$user \xleftarrow{\quad user \mid\mid \{ k_{u,s} \} k_{u,TGS} \mid\mid T_{u,s} \quad} \text{TGS}$

$user \xrightarrow{\quad A_{u,s} \mid\mid T_{u,s} \quad} service$

$user \xleftarrow{\quad \{ t + 1 \} k_{u,s} \quad} service$

Analysis

- First two steps get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with AS
- Next four steps show how *u* gets and uses ticket for service *s*
  - Service *s* validates request by checking sender (using $A_{u,s}$) is same as entity ticket issued to
  - Step 6 optional; used when *u* requests confirmation

Problems

- Relies on synchronized clocks
  - If not synchronized and old tickets, authenticators not cached, replay is possible
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Kerberos 4 session keys weak (had much less than 56 bits of randomness); researchers at Purdue found them from tickets in minutes

Public Key Key Exchange

- Here interchange keys known
  - e_A, e_B Alice and Bob’s public keys known to all
  - d_A, d_B Alice and Bob’s private keys known only to owner
- Simple protocol
  - k_s is desired session key

$$\text{Alice} \xrightarrow{\{ k_s \}\, e_B} \text{Bob}$$

Problem and Solution

- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

$$\text{Alice} \xrightarrow{\{ \{ k_s \} d_A \} e_B} \text{Bob}$$

Notes

- Can include message enciphered with k_s
- Assumes Bob has Alice's public key, and *vice versa*
  - If not, each must get it from public server
  - If keys not bound to identity of owner, attacker Eve can launch a *man-in-the-middle* attack (next slide; Cathy is public server providing public keys)
    - Solution to this (binding identity to keys) discussed later as public key infrastructure (PKI)

Man-in-the-Middle Attack

Alice $\xrightarrow{\text{send Bob's public key}}$ | $\xrightarrow{\textit{Eve intercepts request}}$ Cathy

Eve $\xrightarrow{\text{send Bob's public key}}$ Cathy

Eve $\xleftarrow{e_B}$ Cathy

Alice $\xleftarrow{e_E}$ Eve

Alice $\xrightarrow{\{ k_s \}\, e_E}$ | $\xrightarrow{\textit{Eve intercepts message}}$ Bob

Eve $\xrightarrow{\{ k_s \}\, e_B}$ Bob

Diffie-Hellman

- Compute a common, shared key
  - Called a *symmetric key exchange protocol*
- Based on discrete logarithm problem
  - Given integers n, g and prime number p, compute k such that $n = g^k \bmod p$
  - Solutions known for small p
  - Solutions computationally infeasible as p grows large

Algorithm

- Constants: prime p, integer $g \neq 0, 1, p-1$
  - Known to all participants
- Alice chooses private key k_{Alice}, computes public key $K_{\text{Alice}} = g^{k_{\text{Alice}}} \bmod p$
- Bob chooses private key k_{Bob}, computes public key $K_{\text{Bob}} = g^{k_{\text{Bob}}} \bmod p$
- To communicate with Bob, Alice computes $K_{\text{Alice,Bob}} = K_{\text{Bob}}^{\ k_{\text{Alice}}} \bmod p$
- To communicate with Alice, Bob computes $K_{\text{Bob,Alice}} = K_{\text{Alice}}^{\ k_{\text{Bob}}} \bmod p$
- It can be shown $K_{\text{Alice,Bob}} = K_{\text{Bob,Alice}}$

Example

- Assume p = 121001 and g = 6981
- Alice chooses k_{Alice} = 526784
  - Then $K_{\text{Alice}} = 6981^{26874} \bmod 121001 = 22258$
- Bob chooses k_{Bob} = 5596
  - Then $K_{\text{Bob}} = 6981^{5596} \bmod 121001 = 112706$
- Shared key:
  - ${K_{\text{Bob}}}^{k_{\text{Alice}}} \bmod p = 112706^{26874} \bmod 121001 = 78618$
  - ${K_{\text{Alice}}}^{k_{Bob}} \bmod p = 22258^{5596} \bmod 121001 = 78618$

Problems

- Using cipher requires knowledge of environment, and threats in the environment, in which cipher will be used
  - Is the set of possible messages small?
  - Can an active wiretapper rearrange or change parts of the message?
  - Do the messages exhibit regularities that remain after encipherment?
  - Can the components of the message be misinterpreted?

Attack #1: Precomputation

- Set of possible messages *M* small
- Public key cipher *f* used
- Idea: precompute set of possible ciphertexts *f*(*M*), build table (*m*, *f*(*m*))
- When ciphertext *f*(*m*) appears, use table to find *m*
- Also called *forward searches*

Example

- Cathy knows Alice will send Bob one of two messages: enciphered BUY, or enciphered SELL
- Using public key e_{Bob}, Cathy precomputes

$$m_1 = \{ \text{BUY} \}\, e_{Bob},\ m_2 = \{ \text{SELL} \}\, e_{Bob}$$

- Cathy sees Alice send Bob m_2
- Cathy knows Alice sent SELL

May Not Be Obvious

- Digitized sound
  - Seems like far too many possible plaintexts, as initial calculations suggest 2^{32} such plaintexts
  - Analysis of redundancy in human speech reduced this to about 100,000 ($\approx 2^{17}$), small enough for precomputation attacks

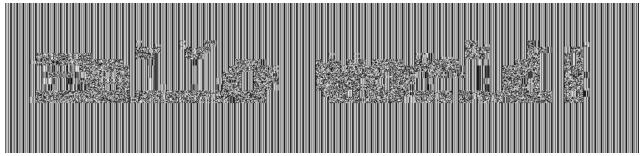
Misordered Blocks

- Alice sends Bob message
  - n_{Bob} = 262631, e_{Bob} = 45539, d_{Bob} = 235457
- Message is TOMNOTANN (191412 131419 001313)
- Enciphered message is 193459 029062 081227
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 081227 029062 193459
- Bob gets enciphered message, deciphers it
  - He sees ANNNOTTOM, opposite of what Alice sent

Statistical Regularities

- If plaintext repeats, ciphertext may too
- Example using AES-128:
  - Input image: 

  - corresponding output image:



  - Note you can still make out the words
- Fix: cascade blocks together (chaining); more details later

Type Flaw Attacks

- Assume components of messages in protocol have particular meaning
- Example: Otway-Rees:

$$\text{Alice} \xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \}\, k_A} \text{Bob}$$

$$\text{Cathy} \xleftarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{ r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \}\, k_A \,||\, \{ r_2 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \}\, k_B} \text{Bob}$$

$$\text{Cathy} \xrightarrow{n \,||\, \{ r_1 \,||\, k_s \}\, k_A \,||\, \{ r_2 \,||\, k_s \}\, k_B} \text{Bob}$$

$$\text{Alice} \xleftarrow{n \,||\, \{ r_1 \,||\, k_s \}\, k_A} \text{Bob}$$

The Attack

- Ichabod intercepts message from Bob to Cathy in step 2
- Ichabod *replays* this message, sending it to Bob
  - Slight modification: he deletes the cleartext names
- Bob *expects* $n \mid\mid \{ r_1 \mid\mid k_s \} k_A \mid\mid \{ r_2 \mid\mid k_s \} k_B$
- Bob *gets* $n \mid\mid \{ r_1 \mid\mid n \mid\mid \text{Alice} \mid\mid \text{Bob} \} k_A \mid\mid \{ r_2 \mid\mid n \mid\mid \text{Alice} \mid\mid \text{Bob} \} k_B$
- So Bob sees $n \mid\mid \text{Alice} \mid\mid \text{Bob}$ as the session key — and Ichabod knows this
- When Alice gets her part, she makes the same assumption
- Now Ichabod can read their encrypted traffic

Solution

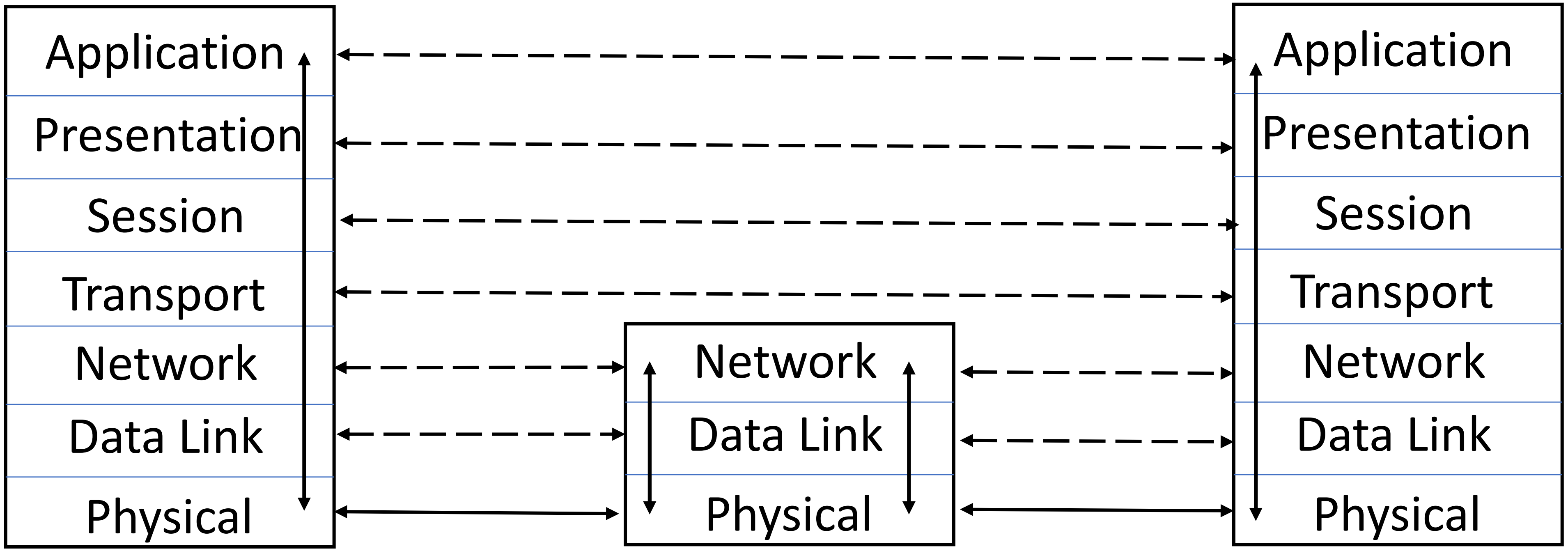
- Tag components of cryptographic messages with information about what the component is
  - But the tags themselves may be confused with data …

What These Mean

- Use of strong cryptosystems, well-chosen (or random) keys not enough to be secure
- Other factors:
  - Protocols directing use of cryptosystems
  - Ancillary information added by protocols
  - Implementation (not discussed here)
  - Maintenance and operation (not discussed here)

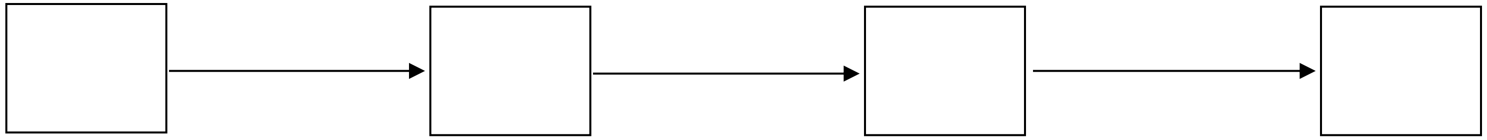
Networks and Cryptography

- ISO/OSI model
- Conceptually, each host communicates with peer at each layer

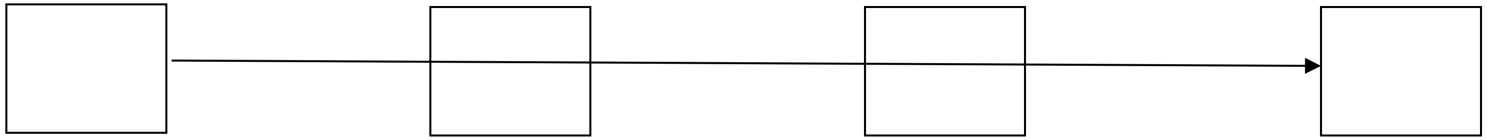


Link and End-to-End Protocols

Link Protocol



End-to-End (or E2E) Protocol



Encryption

- Link encryption
  - Each host enciphers message so host at “next hop” can read it
  - Message can be read at intermediate hosts
- End-to-end encryption
  - Host enciphers message so host at other end of communication can read it
  - Message cannot be read at intermediate hosts

Examples

- SSH protocol
  - Messages between client, server are enciphered, and encipherment, decipherment occur only at these hosts
  - End-to-end protocol
- PPP Encryption Control Protocol
  - Host gets message, deciphers it
    - Figures out where to forward it
    - Enciphers it in appropriate key and forwards it
  - Link protocol

Cryptographic Considerations

- Link encryption
  - Each host shares key with neighbor
  - Can be set on per-host or per-host-pair basis
    - Windsor, stripe, seaview each have own keys
    - One key for (windsor, stripe); one for (stripe, seaview); one for (windsor, seaview)
- End-to-end
  - Each host shares key with destination
  - Can be set on per-host or per-host-pair basis
  - Message cannot be read at intermediate nodes

Traffic Analysis

- Link encryption
  - Can protect headers of packets
  - Possible to hide source and destination
    - Note: may be able to deduce this from traffic flows
- End-to-end encryption
  - Cannot hide packet headers
    - Intermediate nodes need to route packet
  - Attacker can read source, destination