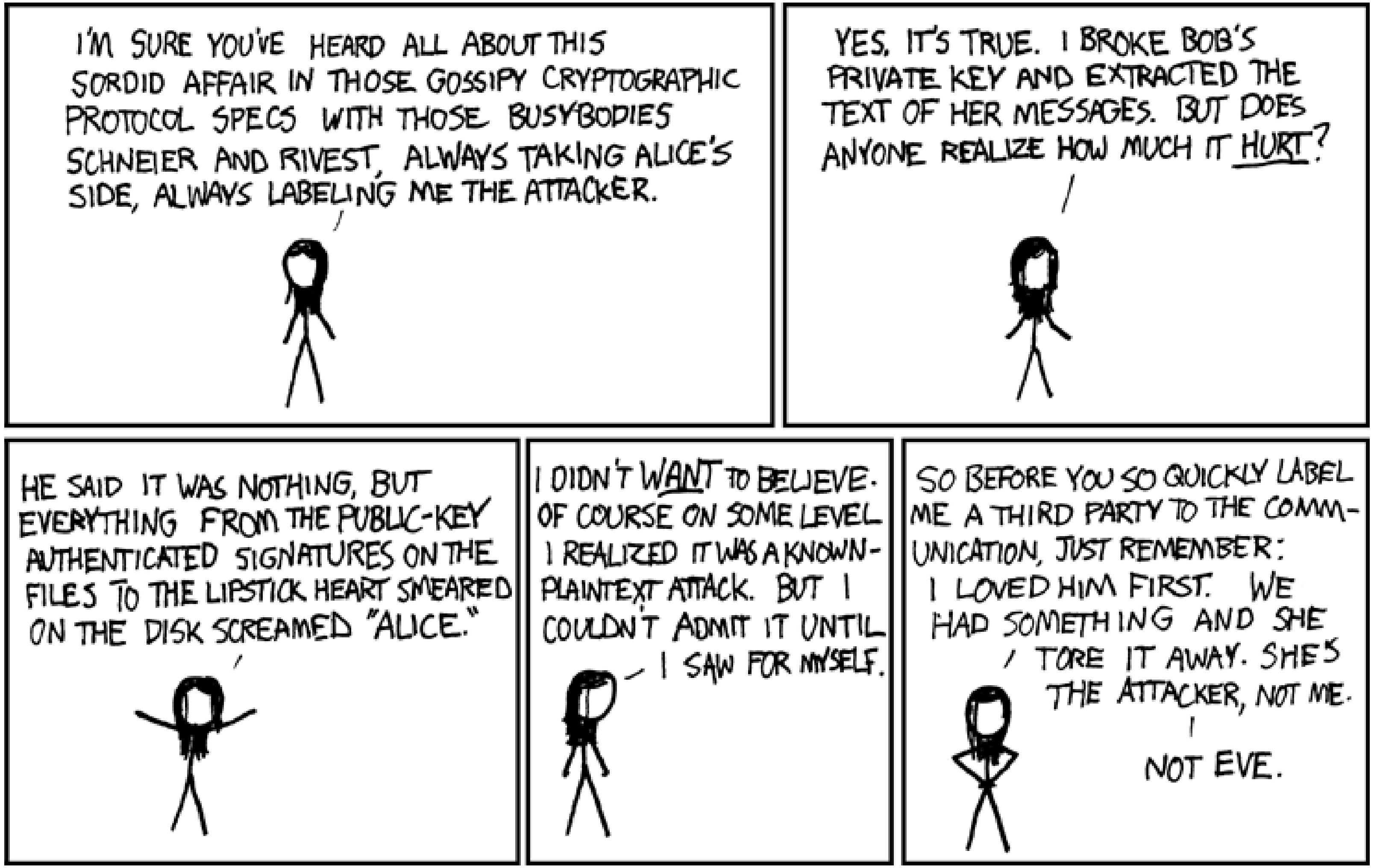
Lecture 11
October 17, 2025

ECS 235A, Computer and Information Security

Administrative Stuff

- Office hour change: Thursday's office hours are now 12:10pm–1:00pm in 2203 Watershed Sciences
  - They were 11:00am–11:50am
- The final project and video are due on Tuesday, December 9
  - This is the day of the final exam
  - The Canvas **Term Project** web page had it wrong; the PDF had it right

xkcd: Alice, Bob, and Eve … A Love Triangle



https://xkcd.com/177

Fast Exponentiation

- Idea: compute 2^5 mod 9
- Initially, *base* = 2 and *value* = 1
- 5 = 101 in binary so look at the rightmost bit of 101 …
  - 1 bit gives *value* = *value* × *base* mod 9 = 1 × 2 mod 9 = 2 mod 9 = 2
  - Compute *base* = $base^2$ mod 9 = 2^2 mod 9 = 4
- Shift 101 right by 1 bit so look at the rightmost bit of 10 …
  - The current bit is 0, meaning don't multiply the *base* by *value*
  - Compute *base* = $base^2$ mod 9 = 4^2 mod 9 = 7
- Shift 10 right by 1 bit so look at the rightmost bit of 1 …
  - 1 bit gives *value* = *value* × *base* mod 9 = 2 × 7 mod 9 = 14 mod 9 = 5
  - Compute *base* = $base^2$ mod 9 = 7^2 mod 9 = 49 mod 9 = 4 (note this is *ignored*)
- 1 shifted right 1 bit is 0, so done; result is 5

Another Example

- Compute 12^{64} mod 93
- In bits, $64 = 1_7 0_6 0_5 0_4 0_3 0_2 0_1$; we start with *base* = 12, *value* = 1
  1. Rightmost bit is 0, so *base* = 12^2 mod 93 = 51; *value* = 1
  2. Shift right, rightmost bit is 0, so *base* = 51^2 mod 93 = 90; *value* = 1
  3. Shift right, rightmost bit is 0, so *base* = 90^2 mod 93 = 9; *value* = 1
  4. Shift right, rightmost bit is 0, so *base* = 9^2 mod 93 = 81; *value* = 1
  5. Shift right, rightmost bit is 0, so *base* = 81^2 mod 93 = 51; *value* = 1
  6. Shift right, rightmost bit is 0, so *base* = 51^2 mod 93 = 90; *value* = 1
  7. Shift right, rightmost bit is 1, so:
     - *value* = *value* × *base* mod 93 = 1 × 90 mod 93 = 90 mod 93 = 90
     - *base* = 90^2 mod 93 = 9 (note this is *ignored*)
- So 12^{64} mod 93 = 90

Algorithm (in Python)

```
# compute g^k mod n
def fastexp(g, n, k):
    value = 1
    base = g
    while k != 0:
        r = k % 2
        if r == 1:
            value = (value * base) % n
        k = k // 2
        base = (base * base) % n
    return value
```

Algorithm (in C)

```
/* compute g^k mod n */
long fastexp(int g, int n, int k)
{
    long value = 1;
    long base = g
    do{
        if (k&01)
            value = (value * base) % n;
        k >>= 1;
        base = (base * base) % n;
    }while (k);
    return value;
}
```

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

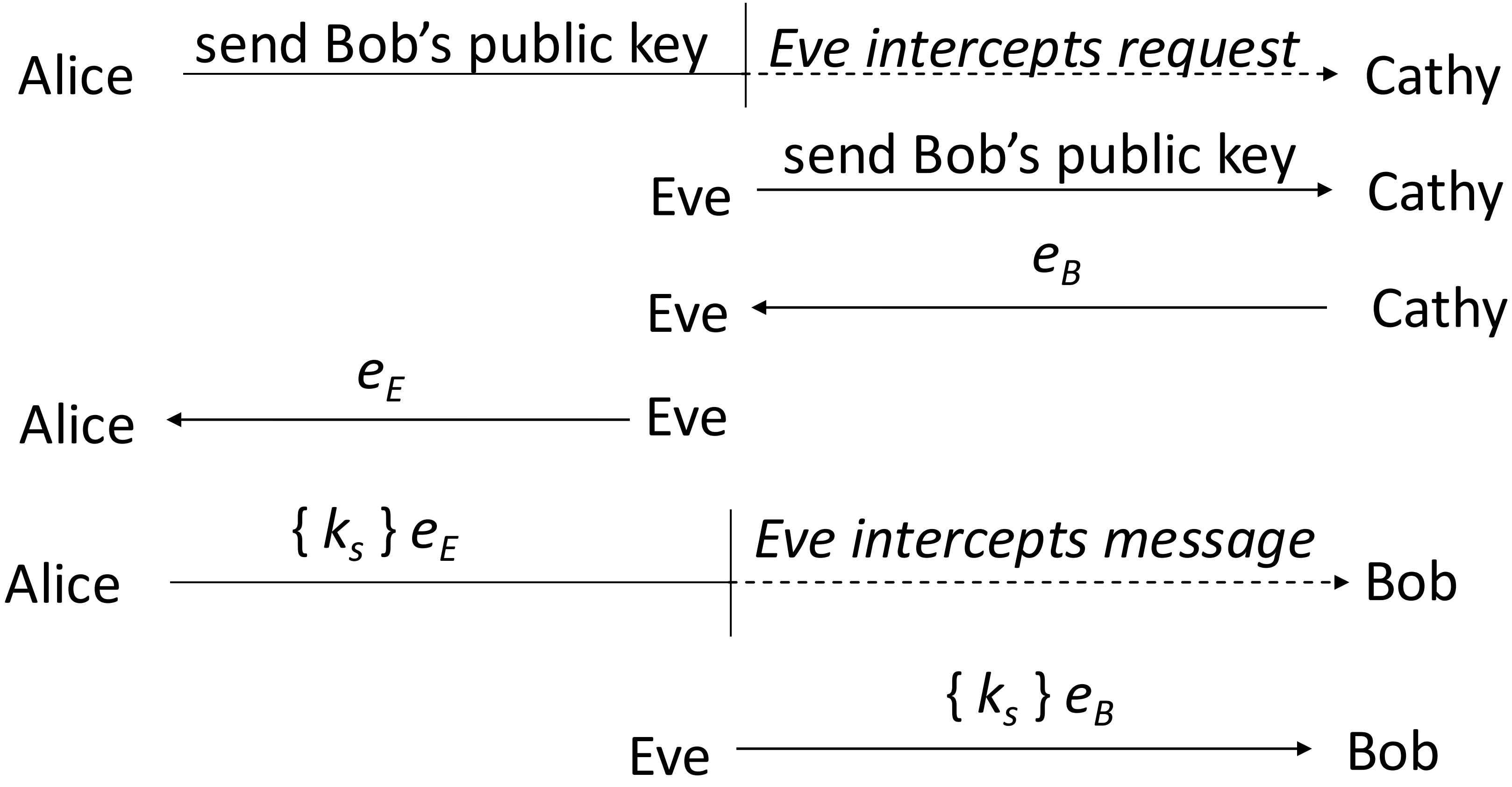
- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

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

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



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 $k_{\text{Alice}} = 26784$
  - Then $K_{\text{Alice}} = 6981^{26784} \bmod 121001 = 100025$
- Bob chooses $k_{\text{Bob}} = 5596$
  - Then $K_{\text{Bob}} = 6981^{5596} \bmod 121001 = 112706$
- Shared key:
  - $K_{\text{Bob}}{}^{k_{\text{Alice}}} \bmod p = 112706^{26784} \bmod 121001 = 15970$
  - $K_{\text{Alice}}{}^{k_{Bob}} \bmod p = 100025^{5596} \bmod 121001 = 15970$

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

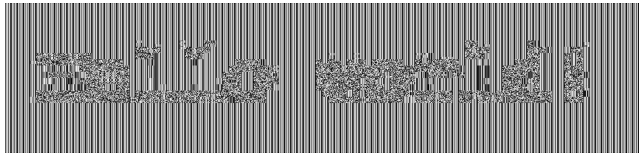
Misordered Blocks

- 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:

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

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

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

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

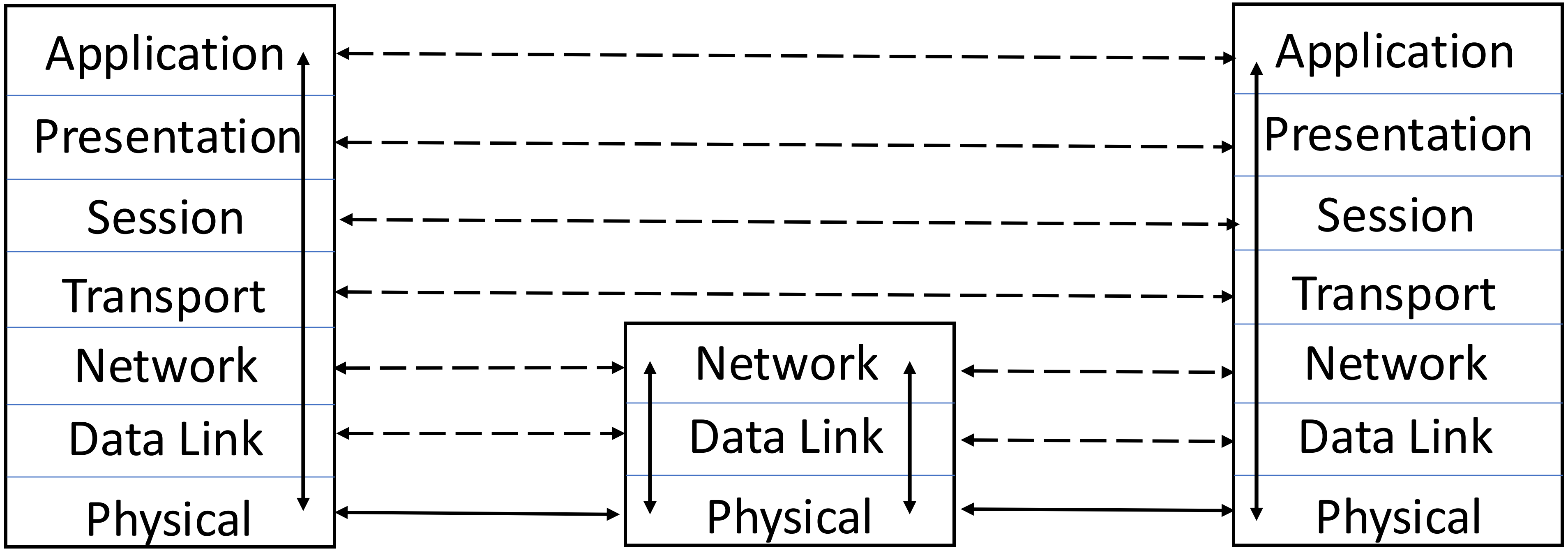
- Tag components of cryptographic messages with information about what the component is
    - But the tags themselves may be confused with data …

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)

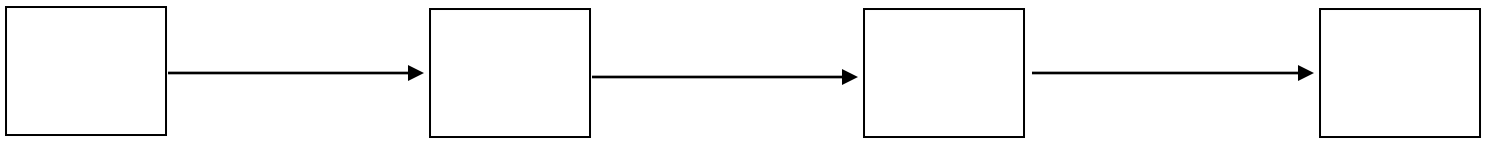
Networks and Cryptography

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

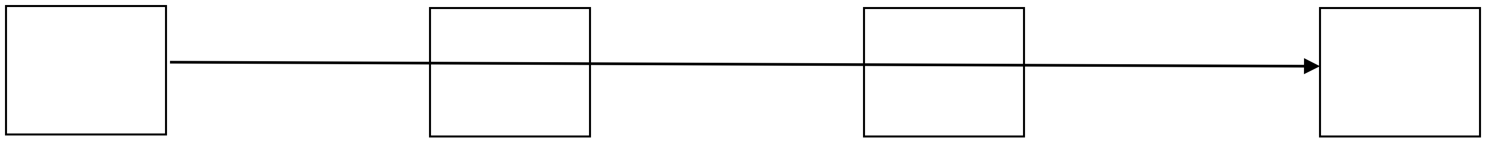


Link and End-to-End Protocols

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