Lecture 9
October 16, 2023

Digital Signature

- Construct that authenticates origin, contents of message in a manner provable to a disinterested third party (a “judge”)
- Sender cannot deny having sent message (service is “nonrepudiation”)
  - Limited to *technical* proofs
    - Inability to deny one’s cryptographic key was used to sign
  - One could claim the cryptographic key was stolen or compromised
    - Legal proofs, *etc.,* probably required; not dealt with here

Common Error

- Symmetric: Alice, Bob share key k
  - Alice sends m || { m } k to Bob
  - { m } k means m enciphered with key k, || means concatenation

  Claim: This is a digital signature

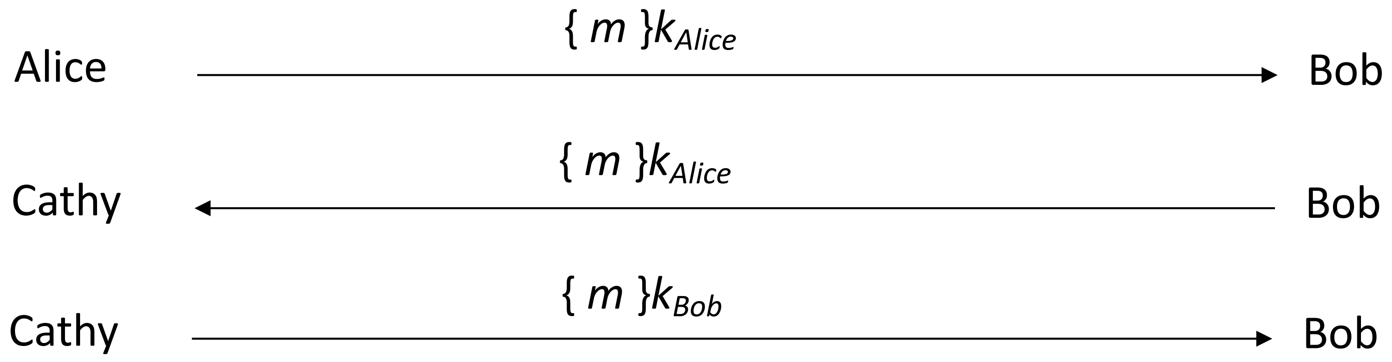
<u>***WRONG***</u>

This is not a digital signature

- Why? Third party cannot determine whether Alice or Bob generated message

Classical Digital Signatures

- Require trusted third party
  - Alice, Bob each share keys with trusted party Cathy
- To resolve dispute, judge gets $\{ m \} k_{Alice}$, $\{ m \} k_{Bob}$, and has Cathy decipher them; if messages matched, contract was signed



Public Key Digital Signatures

- Basically, Alice enciphers the message, or its cryptographic hash, with her private key
- In case of dispute or question of origin or whether changes have been made, a judge can use Alice's public key to verify the message came from Alice and has not been changed since being signed

RSA Digital Signatures

- Alice's keys are (e_{Alice},n_{Alice}) (public key), d_{Alice} (private key)
  - In what follows, we use e_{Alice} to represent the public key
- Alice sends Bob

$$m \mid\mid \{ m \}\, d_{Alice}$$

- In case of dispute, judge computes

$$\{ \{ m \}\, d_{Alice} \}\, e_{Alice}$$

- and if it is *m*, Alice signed message
  - She's the only one who knows d_{Alice}!

RSA Digital Signatures

- Use private key to encipher message
  - Protocol for use is *critical*
- Key points:
  - Never sign random documents, and when signing, always sign hash and never document
  - Don’t just encipher message and then sign, or vice versa
    - Changing public key and private key can cause problems
    - Messages can be forwarded, so third party cannot tell if original sender sent it to her

Attack #1

- Example: Alice, Bob communicating
  - $n_A = 262631$, $e_A = 154993$, $d_A = 95857$
  - $n_B = 288329$, $e_B = 22579$, $d_B = 138091$
- Alice asks Bob to sign 225536 so she can verify she has the right public key:
  - $c = m^{d_B} \bmod n_B = 225536^{138091} \bmod 288329 = 271316$
- Now she asks Bob to sign the statement AYE (002404):
  - $c = m^{d_B} \bmod n_B = 002404^{138091} \bmod 288329 = 182665$

Attack #1

- Alice computes:
  - new message NAY (130024) by (002404)(225536) mod 288329 = 130024
  - corresponding signature (271316)(182665) mod 288329 = 218646
- Alice now claims Bob signed NAY (130024), and as proof supplies signature 218646
- Judge computes $c^{e_B} \bmod n_B = 218646^{22579} \bmod 288329 = 130024$
  - Signature validated; Bob is toast

Preventing Attack #1

- Do not sign random messages
  - This would prevent Alice from getting the first message
- When signing, always sign the cryptographic hash of a message, not the message itself

Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract LUR (112017)
  - But Bob really wants her to sign contract EWM (042212), but knows she won't
- Alice enciphers, then signs:
  - $(m^{e_B} \bmod n_A)^{d_A} \bmod n_A = (112017^{22579} \bmod 288329)^{95857} \bmod 262631 = 42390$
- Bob now changes his public key
  - Computes r such that $042212^r \bmod 288329 = 112017$; one such $r = 9175$
  - Computes $re_B \bmod \phi(n_B) = (9175)(22579) \bmod 287184 = 102661$
  - Replace public key with (102661,288329), private key with 161245
- Bob claims contract was EWM
- Judge computes:
  - $(42390^{154993} \bmod 262631)^{161245} \bmod 288329 = 042212$, which is EWM
  - Verified; now Alice is toast

Preventing Attack #2

- Obvious thought: instead of encrypting message and then signing it, sign the message and then encrypt it
  - May not work due to surreptitious forwarding attack
  - Idea: Alice sends Cathy an encrypted signed message; Cathy deciphers it, re-enciphers it with Bob's public key, and then sends message and signature to Bob – now Bob thinks the message came from Alice (right) and was intended for him (wrong)
- Several ways to solve this:
  - Put sender and recipient in the message; changing recipient invalidates signature
  - Sign message, encrypt it, then sign the result

El Gamal Digital Signature

- Relies on discrete log problem
  - Choose p prime, g, $d < p$; compute $y = g^d \bmod p$
- Public key: (y, g, p); private key: d
- To sign contract m:
  - Choose k relatively prime to p–1, and not yet used
  - Compute $a = g^k \bmod p$
  - Find b such that $m = (da + kb) \bmod p-1$
  - Signature is (a, b)
- To validate, check that
  - $y^a a^b \bmod p = g^m \bmod p$

Example

- Alice chooses p = 262643, g = 9563, d = 3632, giving y = 274598
- Alice wants to send Bob signed contract PUP (152015)
  - Chooses k = 601 (relatively prime to 262642)
  - This gives $a = g^k \bmod p = 9563^{601} \bmod 29 = 202897$
  - Then solving $152015 = (3632 \times 202897 + 601b) \bmod 262642$ gives $b = 225835$
  - Alice sends Bob message m = 152015 and signature (a,b) = (202897, 225835)
- Bob verifies signature: $g^m \bmod p = 9563^{152015} \bmod 262643 = 157499$ and $y^a a^b \bmod p = 27459^{202897} 202897^{225835} \bmod 262643 = 157499$
  - They match, so Alice signed

Attack

- Eve learns k, corresponding message m, and signature (a, b)
  - Extended Euclidean Algorithm gives d, the private key
- Example from above: Eve learned Alice signed last message with $k = 5$

$$m = (da + kb) \bmod p{-}1 \Rightarrow 152015 = (202897d + 601 \times 225835) \bmod 262642$$

giving Alice's private key $d = 3632$

Notation

- $X \rightarrow Y : \{ Z \,||\, W \}\, k_{X,Y}$
    - X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y
- $A \rightarrow T : \{ Z \}\, k_A \,||\, \{ W \}\, k_{A,T}$
    - A sends T a message consisting of the concatenation of Z enciphered using k_A, A's key, and W enciphered using $k_{A,T}$, the key shared by A and T
- r_1, r_2 nonces (nonrepeating random numbers)

Key Exchange Algorithms

- Goal: Alice, Bob get shared key
  - Key cannot be sent in clear
    - Attacker can listen in
    - Key can be sent enciphered, or derived from exchanged data plus data not known to an eavesdropper
  - Alice, Bob may trust third party
  - All cryptosystems, protocols publicly known
    - Only secret data is the keys, ancillary information known only to Alice and Bob needed to derive keys
    - Anything transmitted is assumed known to attacker

Symmetric Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
  - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Protocol

Alice $\xrightarrow{\{\text{ request for session key to Bob }\}\, k_A}$ Cathy

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

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

Problems

- How does Bob know he is talking to Alice?
  - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't
  - Session key reuse: Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defense against replay

Session, Interchange Keys

- Alice wants to send a message *m* to Bob
  - Assume public key encryption
  - Alice generates a random cryptographic key k_s and uses it to encipher *m*
    - To be used for this message *only*
    - Called a *session key*
  - She enciphers k_s with Bob's public key k_B
    - k_B enciphers all session keys Alice uses to communicate with Bob
    - Called an interchange *key*
  - Alice sends $\{ m \} k_s \{ k_s \} k_B$

Benefits

- Limits amount of traffic enciphered with single key
  - Standard practice, to decrease the amount of traffic an attacker can obtain
- Prevents some attacks
  - Example: Alice will send Bob message that is either “BUY” or “SELL”. Eve computes possible ciphertexts { “BUY” } k_B and { “SELL” } k_B. Eve intercepts enciphered message, compares, and gets plaintext at once

Needham-Schroeder

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

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

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

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

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

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

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

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

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

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

Alice $\xrightarrow{\{\, \text{Alice} \,||\, T \,||\, k_s \,\}\, k_B}$ 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*

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

  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*

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

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

Protocol

$$\mathit{user} \xrightarrow{\quad \mathit{user} \;||\; \mathit{TGS} \quad} \text{AS}$$

$$\text{AS} \xleftarrow{\quad \{\, k_{u,TGS} \,\}\, k_u \;||\; T_{u,TGS} \quad} \mathit{user}$$

$$\mathit{user} \xrightarrow{\quad \mathit{service} \;||\; A_{u,TGS} \;||\; T_{u,TGS} \quad} \text{TGS}$$

$$\mathit{user} \xleftarrow{\quad \mathit{user} \;||\; \{\, k_{u,s} \,\}\, k_{u,TGS} \;||\; T_{u,s} \quad} \text{TGS}$$

$$\mathit{user} \xrightarrow{\quad A_{u,s} \;||\; T_{u,s} \quad} \mathit{service}$$

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

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

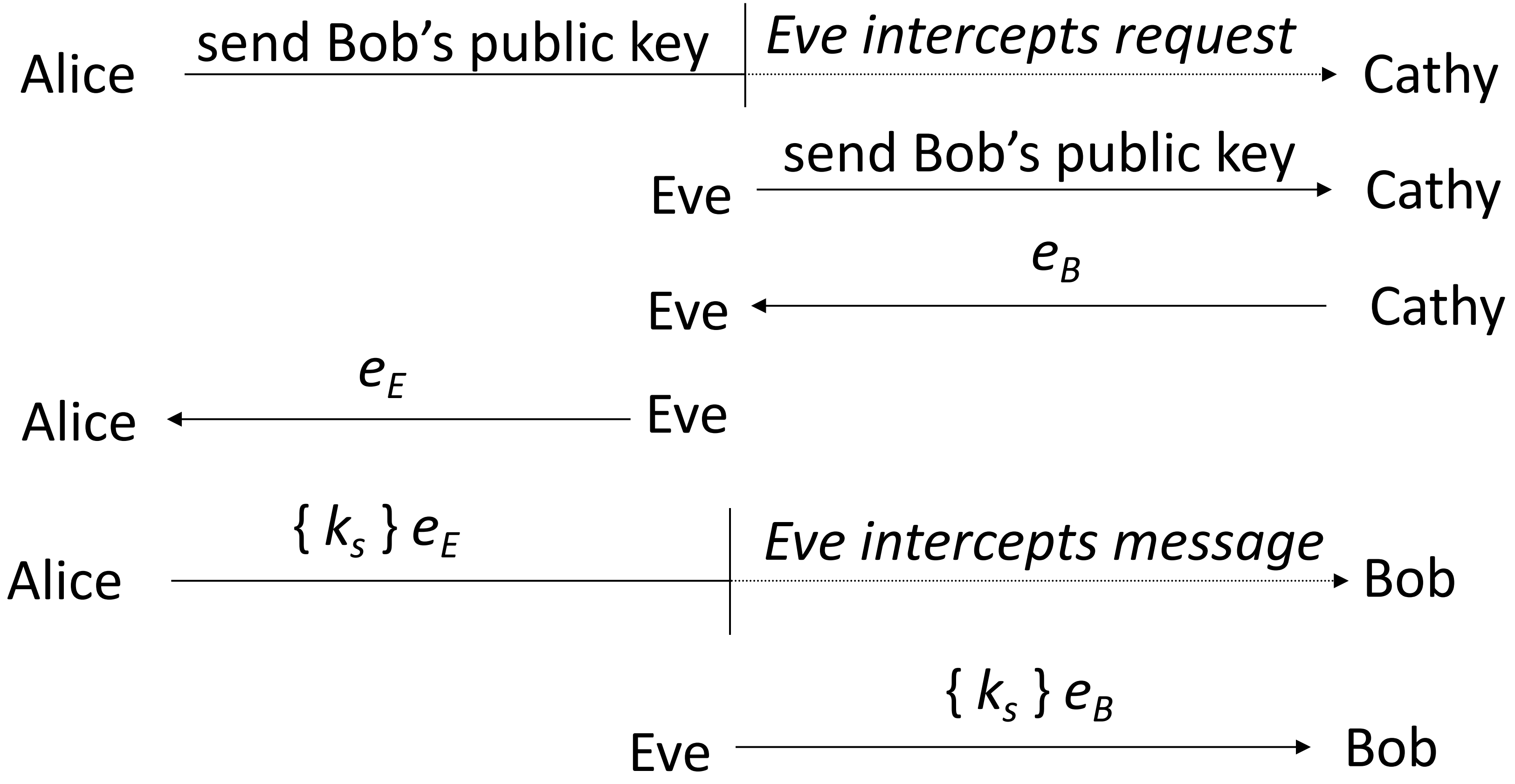
- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

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

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



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

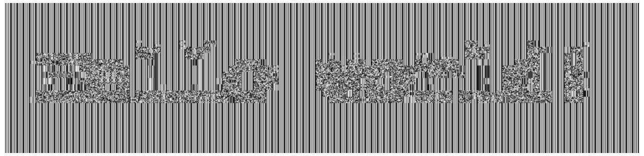
Misordered Blocks

- 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:

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

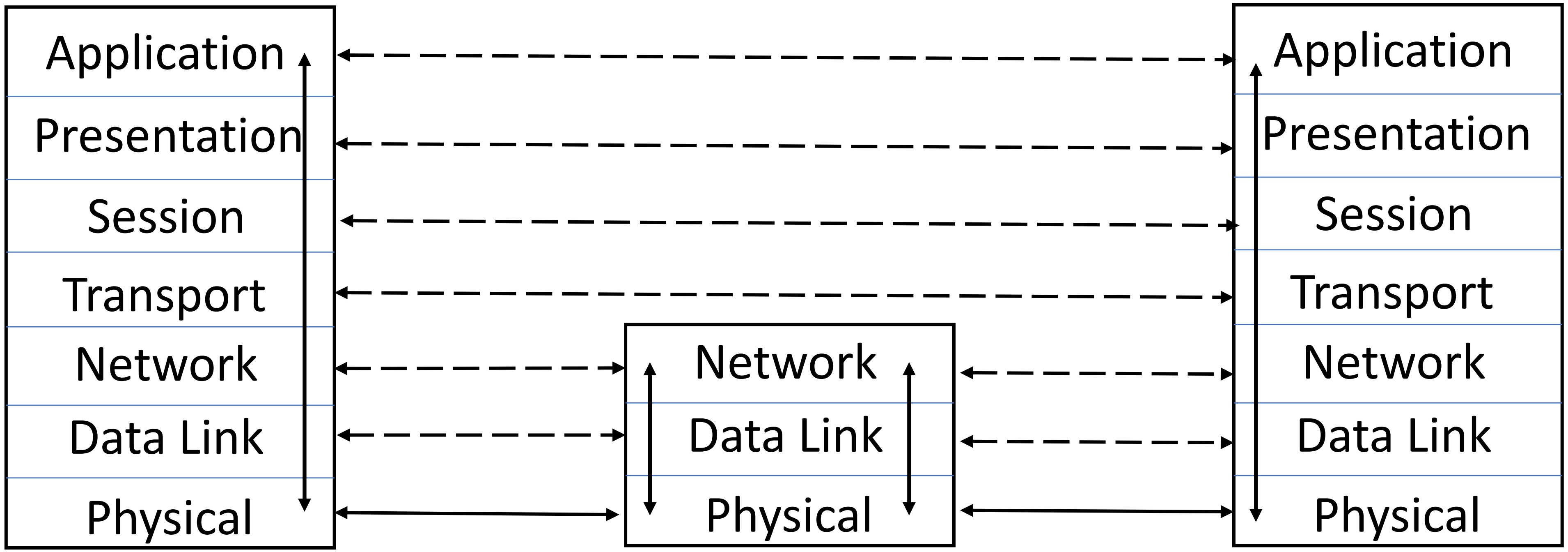
- Tag components of cryptographic messages with information about what the component is
  - But the tags themselves may be confused with data …

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)

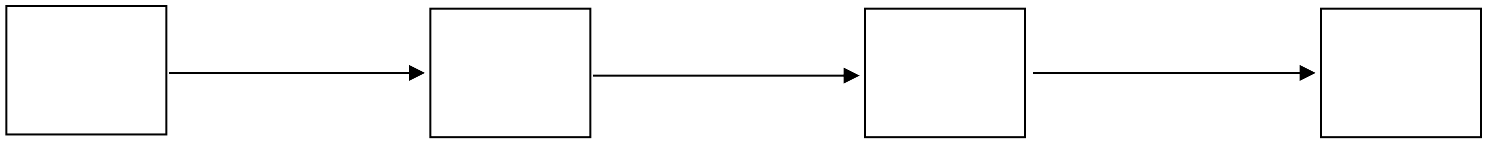
Networks and Cryptography

- 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