Fun with Crypto – keys and protocols

some Bishop, some Jim, some RA

Keys and protocols

- Keys, notation, session keys
  - certs and digital signatures
  - Key infrastructure, storage
- protocols – how we use keys
  - Needham-Schroder/Kerberos
- stream/block ciphers
- crypto protocol examples, PEM (dead), IPSEC

can
you
export
this
t-shirt?

Basic 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)
- e – encipher, d - decipher

Cryptographic Key Infrastructure

- Goal: bind identity to key
- Classical: not possible as all keys are shared
  - Use protocols to agree on a shared key
- Public key: bind identity to public key
  - Crucial as people will use key to communicate with principal whose identity is bound to key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an **acceptable name**

Certificates – public key/name

- a cert is a signed public key
- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)

signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \,||\, \text{Alice} \,||\, T \}\, d_C$$

Use

- Bob gets Alice’s certificate SOMEHOW
  - If he knows Cathy’s public key, he can decipher the certificate
    - When was certificate issued?
    - Is the principal Alice?
  - Now Bob has Alice’s public key
- Problem: Bob needs Cathy’s public key to validate certificate
  - Problem pushed “up” a level
  - Problem is real though
  - Solution space: some distributed protocol tree to get CERTs OR a CERT (a message or file on a computer) has needed CERTS provided with it (a CERT chain)

Certificate Signature Chains

- Create certificate
  - Generate hash of certificate
  - sign hash with issuer's private key
- Validate signature
  - Obtain issuer's public key
  - Decipher enciphered hash
  - Recompute hash from certificate and compare
- Problem: getting issuer's public key

X.509 certificate format

- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key
  - Signature: enciphered hash

Issuers

- *Certification Authority (CA)*: entity that issues certificates
  - Multiple issuers pose validation problem
  - Alice's CA is Cathy; Bob's CA is Don; how can Alice validate Bob's certificate?
  - Have Cathy and Don cross-certify
    - Each issues certificate for the other
  - Have a hierarchical cert. authority
    - Cathy and Don have Eduard as a CA

CA tree

- Alice has CA1
- Bob has CA2
- CA1 and CA2 have CA3
- Alice gets CERT from Bob,
- must validate Bob with CA2 (no trust)
- then validate CA2 with CA3 (hierarchical trust relationship)

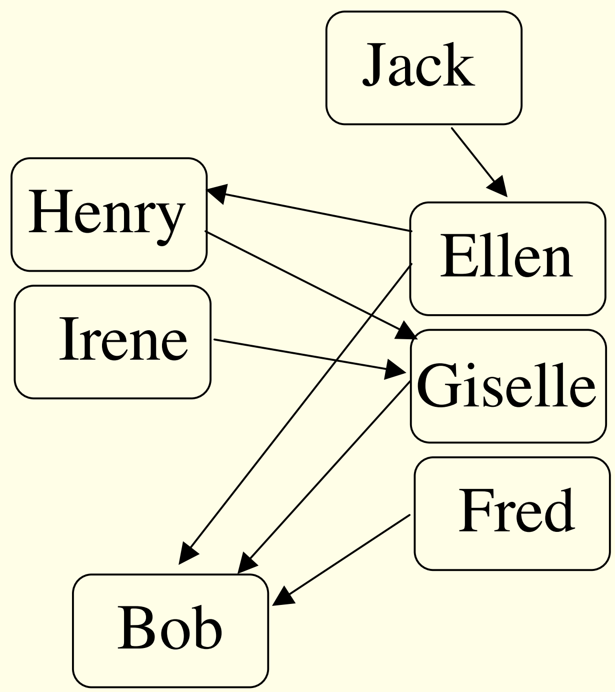
Signing with PGP

- Single certificate may have multiple signatures associated with it
- Notion of “trust” embedded in each signature
  - Range from “untrusted” to “ultimate trust”
  - Signer defines meaning of trust level (no standards!)
- with a hierarchy eventually you come to a CA that must trust itself …
  - Called “self-signing”
- PGP has notion of “web of trust”, no CA hierarchy

PGP Web of trust - Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Does not know Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Knows Henry slightly, but his signature is at "casual" level of trust
- Alice gets Ellen's cert
  - Knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's

Arrows show signatures
Self signatures not shown



Storing Keys

- Multi-user or networked systems: attackers may defeat access control mechanisms
  - Encipher file containing key – consider these problems
    - Attacker can monitor keystrokes to decipher files
    - Key will be resident in memory that attacker may be able to read (o.s. swap also possible)
  - Use physical devices like "smart card"
    - Key never enters system
    - Card can be stolen, so have 2 devices combine bits to make single key
    - attacks against smart keys exist

Key Revocation – timeout or CRL

- Certificates may be invalidated *before* expiration
  - Usually due to compromised key
  - May be due to change in circumstance (*e.g.,* someone leaving company)
- Problems
  - Entity revoking certificate authorized to do so
  - Revocation information circulates to everyone fast enough
    - Network delays, infrastructure problems may delay information
    - there is very little real experience with cert. revocation other than timestamp timeout

Digital Signature

- Construct that authenticated origin, contents of message in a manner provable to a disinterested third party ("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
    - Alice's box with cert was hacker by Malach, Malach made bank transactions …

Common Error

- Classical: Alice, Bob share key *k*
  - Alice sends *m* || { *m* } *k* to Bob

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

conventional wisdom with public key crypto

- we sign with our private key, they verify with their public key
- obviously they can't have our private key
- they encrypt with our public key, send us M,
- we decrypt with our private key
- RSA fits this model
- if they encrypted with our private key, and we decrypted with our public key
  - the world would be a tad cockeyed

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
    - Mathematical properties can be turned against signer
  - Sign message first, then encipher
    - Changing public keys causes forgery

session keys, and key exchange protocols (KMP)

- typically it is not a good idea to use the same key over and over again
- an adversary has better odds of cracking Ki with a greater number of messages
- therefore we may choose to generate “session-keys” based on previous shared secrets – and discard them at some point
- based on **too much time** or **too many messages**
- protocols exist for generating keys and setting them up between both sides (Alice and Bob)
- goal is typically generation of encryption or MD keys

simple session key – courtesy of public-key crypto

- 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

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 (DH)
  - Alice, Bob may trust third party (Kerberos)
  - All cryptosystems, protocols publicly known
    - secrets in keys
    - Anything transmitted is assumed available to attacker

Simple Symmetric-key exchange Protocol, Cathy is trusted 3rd party

Alice $\longrightarrow$ Cathy: { request for session key to Bob } k_A

Alice $\longleftarrow$ Cathy: $\{ k_s \} k_A \parallel \{ k_s \} k_B$

Alice $\longrightarrow$ Bob: $\{ k_s \} k_B$

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

Needham-Schroeder

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

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

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

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

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

Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party ("Cathy")
- Ticket
  - session-key with timestamp
- Authenticator (DNS like)
  - Identifies sender

Idea

- User *u* authenticates to Kerberos server
  - Obtains ticket $T_{u,TGS}$ for ticket granting service (TGS)
  - TGS is Kerberos form of single sign-on
- 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, note 3-way binding below
- 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:

  - **session key:** $k_{u,s}$ for user and service
  - **time:** is interval for which ticket valid
  - **identity:** *u*'s address may be IP address or something else

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

user → Cathy: $user \,||\, TGS$

user ← Cathy: $\{ k_{u,TGS} \} k_u \,||\, T_{u,TGS}$

user → TGS: $service \,||\, A_{u,TGS} \,||\, T_{u,TGS}$

user ← TGS: $user \,||\, \{ k_{u,s} \} k_{u,TGS} \,||\, T_{u,s}$

user → *service*: $A_{u,s} \,||\, T_{u,s}$

user ← *service*: $\{ t + 1 \} k_{u,s}$

Analysis

- First two steps get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with Cathy
- 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
- Bellovin poked homes in K4 in famous paper
  - so now we have K5
  - which uses ASN.1 (ouch ouch ouch)

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

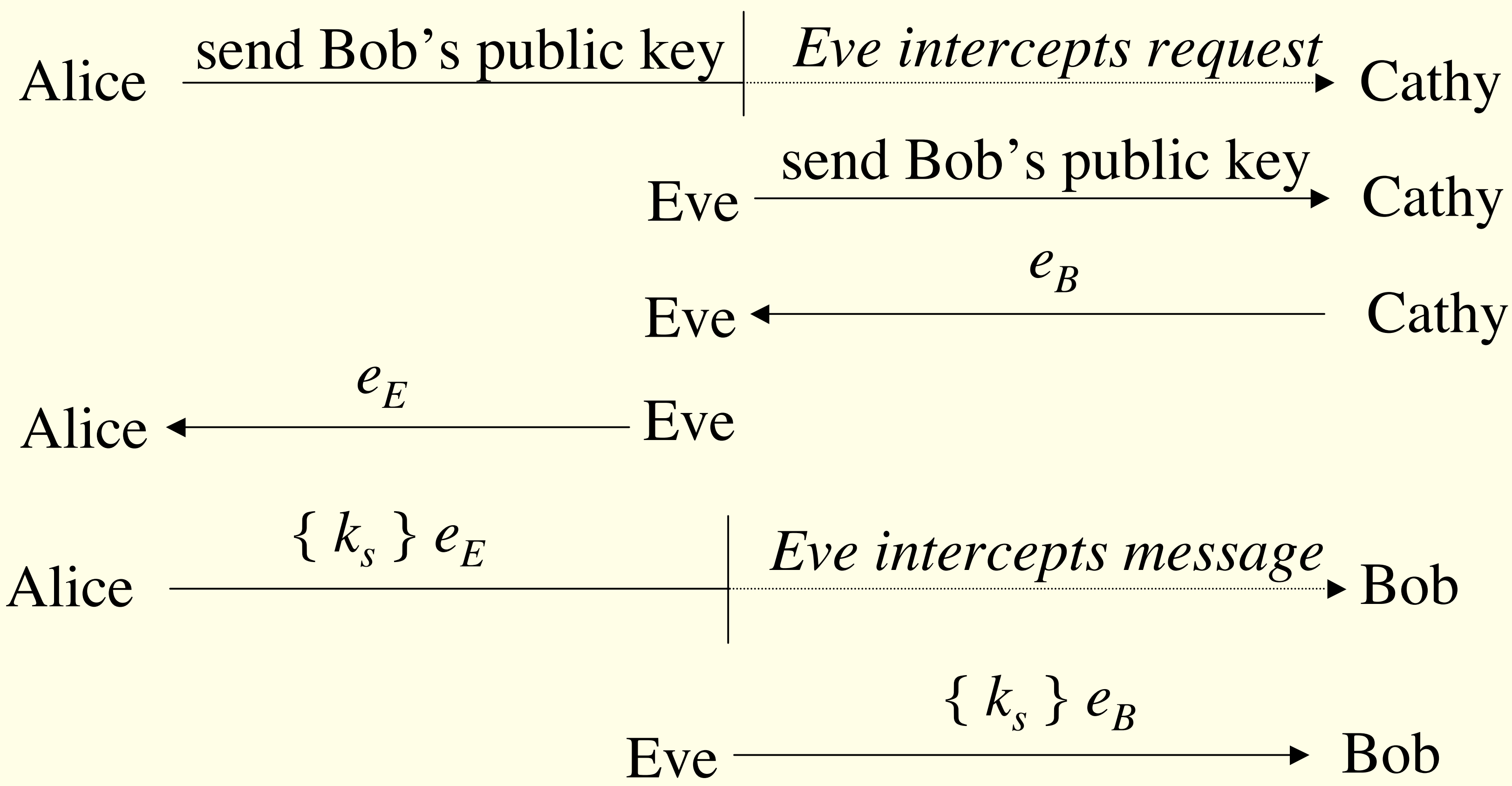
- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

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

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



Key Mgmt - Key Points

- Key management critical to effective use of cryptosystems
  - Different levels of keys (session *vs.* interchange)
- Keys need infrastructure to identify holders, allow revoking
  - Key escrowing complicates infrastructure
- Ultimately we still may need manual dissemination of something; e.g., root self-signed certificates
- Digital signatures provide integrity of origin and content

  Much easier with public key cryptosystems than with classical cryptosystems

common problems with ciphers

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

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*

message entropy space may be small

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

Misordered Blocks

- Alice sends Bob message
  - Message is LIVE (11 08 21 04)
  - Enciphered message is 44 57 21 16
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 16 21 57 44
- Bob gets enciphered message, deciphers it
  - He sees EVIL

Notes

- Digitally signing each block won't stop this attack
- Two approaches:
  - Cryptographically hash the *entire* message and sign it
  - Place sequence numbers in each block of message, so recipient can tell intended order
    - Then you sign each block

Statistical Regularities

- If plaintext repeats, ciphertext may too
- Example using DES:
  - input (in hex):

    3231 3433 3635 3837 3231 3433 3635 3837
  - corresponding output (in hex):

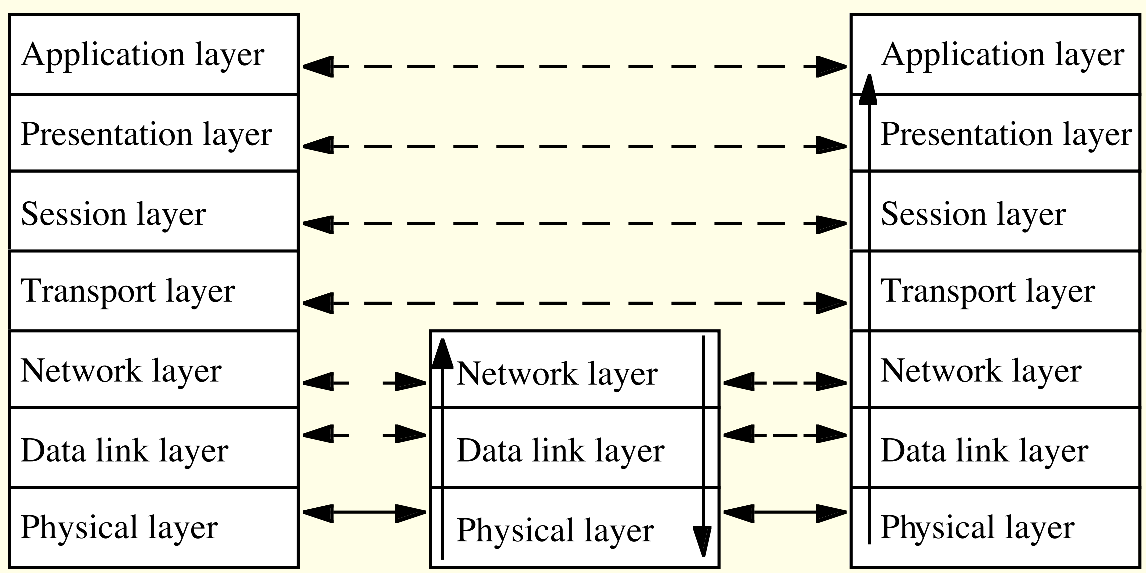
    ef7c 4bb2 b4ce 6f3b ef7c 4bb2 b4ce 6f3b
- Fix: cascade blocks together (chaining)
  - this is why DES-CBC is used

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)

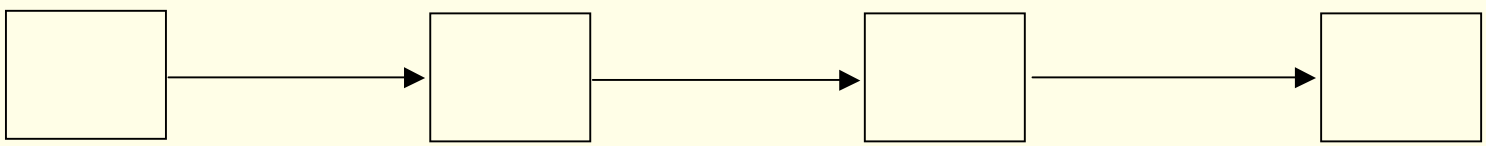
Networks and Cryptography

- ISO/OSI model
- Conceptually, each host has peer at each layer
  - Peers communicate with peers at same layer

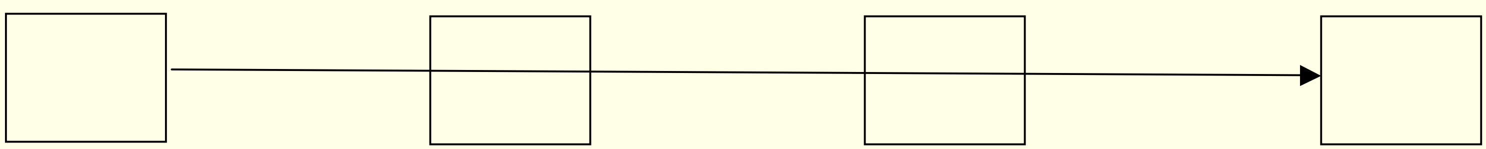


Link and End-to-End Protocols

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

- secure shell protocol
  - end to end, therefore good
  - password form does not send password in clear (unlike traditional telnet)
- 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 – not end to end

Cryptographic Considerations

- Link encryption
  - Each host shares key with neighbor
  - should be per host pair BUT
  - often per network (broadcast network in particular)
  - increasing tendency to have per host or per site certificate using SSL (yes public-key crypto)
- 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 IP packet headers
    - Intermediate nodes need to route packet
  - Attacker can read source, destination
  - Can’t hide L3 on Internet (can’t route without it)
  - if application encryption, not hiding L4 TCP/UDP port numbers either

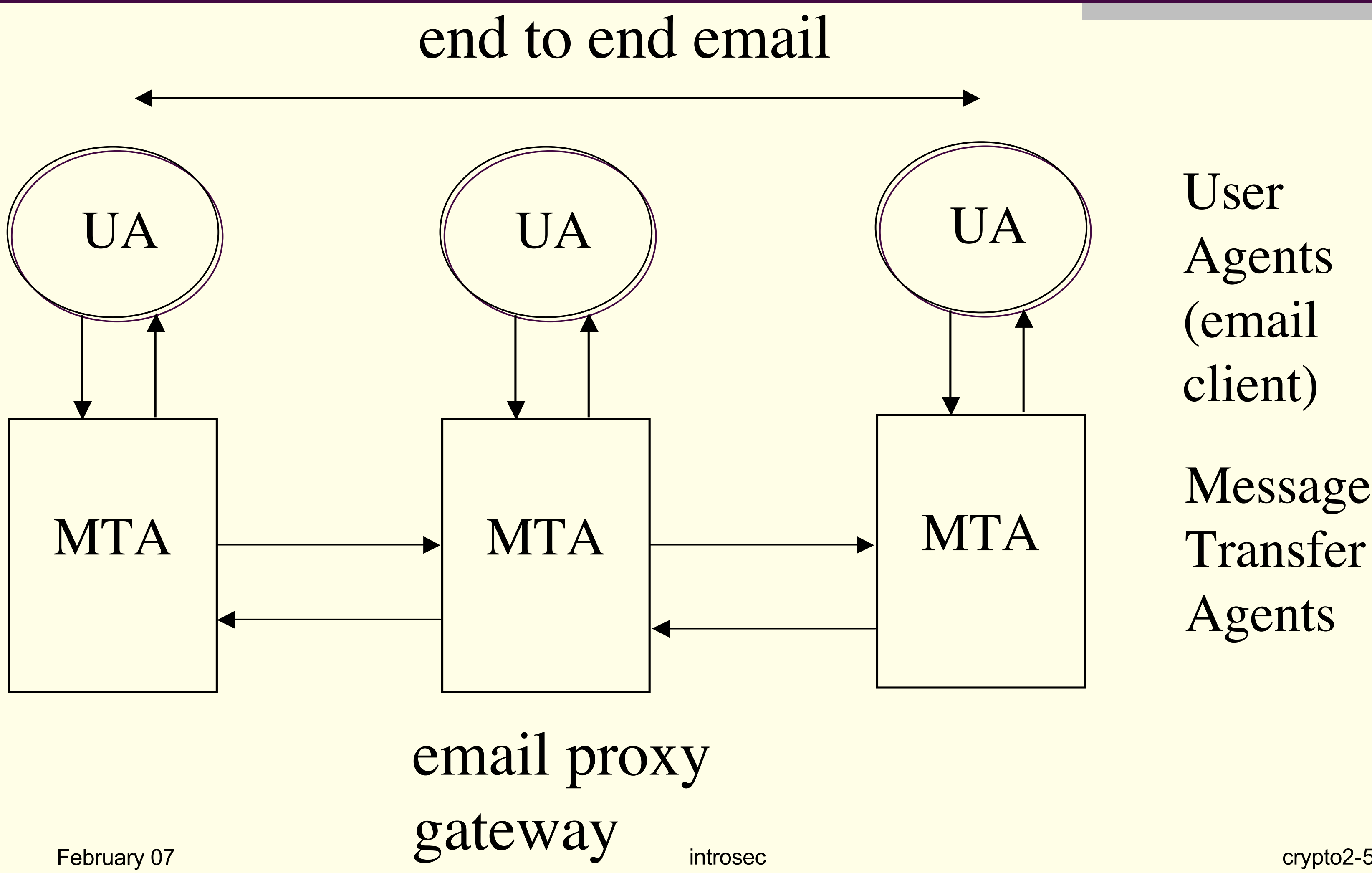
Example Protocols

- Privacy-Enhanced Electronic Mail (PEM)
  - Applications layer protocol
  - PEM is not used in real world
  - was breakthru of sorts in IETF/crypto history
  - typically might use PGP/SSL at this point
    - email is often tunneled in some sense
- IP Security (IPSEC)
  - Network layer protocol

Goals of PEM

1. Confidentiality
   - Only sender and recipient(s) can read message
2. Origin authentication
   - Identify the sender precisely
3. Data integrity
   - Any changes in message are easy to detect
4. Non-repudiation of origin
   - Whenever possible …

Message Handling System



Design Principles

- Do not change related existing protocols
  - Cannot alter SMTP
- Do not change existing software
  - Need compatibility with existing software
- Make use of PEM optional
  - Available if desired, but email still works without them
  - Some recipients may use it, others not
- Enable communication without prearrangement
  - Out-of-band authentication, key exchange problematic

Basic Design: Keys

- Two keys
  - *Interchange keys* tied to sender, recipients and is static (for some set of messages)
    - Like a public/private key pair
    - Must be available *before* messages sent
  - *Data exchange keys* generated for each message
    - a session key, session being the message

Basic Design: Sending

Confidentiality

- m message
- k_s data exchange key
- k_B Bob's interchange key

$$\{ m \} k_s \,||\, \{ k_s \} k_B$$

Alice ⟶ Bob

Basic Design: Integrity

Integrity and authentication:

- m message
- $h(m)$ hash of message m —Message Integrity Check (MIC)
- k_A Alice's interchange key

$$\text{Alice} \xrightarrow{m\ \{\ h(m)\ \}\ k_A} \text{Bob}$$

Non-repudiation: if k_A is Alice's private key, this establishes that Alice's private key was used to sign the message

Basic Design: Everything

Confidentiality, integrity, authentication:

- Notations as in previous slides
- If k_A is private key, get non-repudiation too

$$\{ m \} k_s \,||\, \{ h(m) \} k_A \,||\, \{ k_s \} k_B$$

Alice ⟶ Bob

Practical Considerations

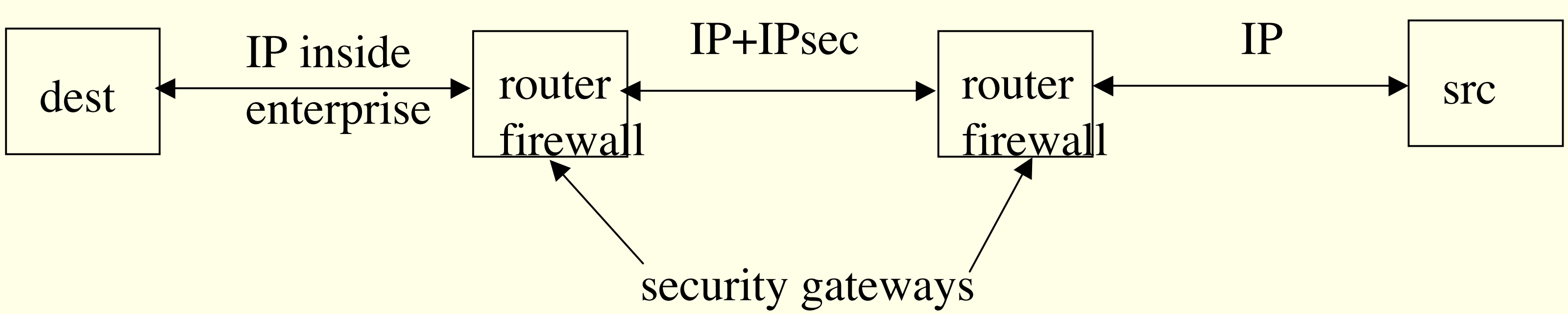
- Limits of SMTP
  - Only ASCII characters, limited length lines
- Use encoding procedure
  1. Map local char representation into canonical format
     - Format meets SMTP requirements
  2. Compute and encipher MIC over the canonical format; encipher message if needed
  3. Map each 6 bits of result into a character; insert newline after every 64th character
  4. Add delimiters around this ASCII message

PEM vs. PGP

- Use different ciphers
  - PGP originally used IDEA cipher
  - PEM used DES in CBC mode
- Use different certificate models
  - PGP uses general “web of trust”
  - PEM uses hierarchical certification structure
    - fatal flaw … **no such beastie Inet-wide**
- Handle end of line differently
  - PGP remaps end of line if message tagged “text”, but leaves them alone if message tagged “binary”
  - PEM always remaps end of line

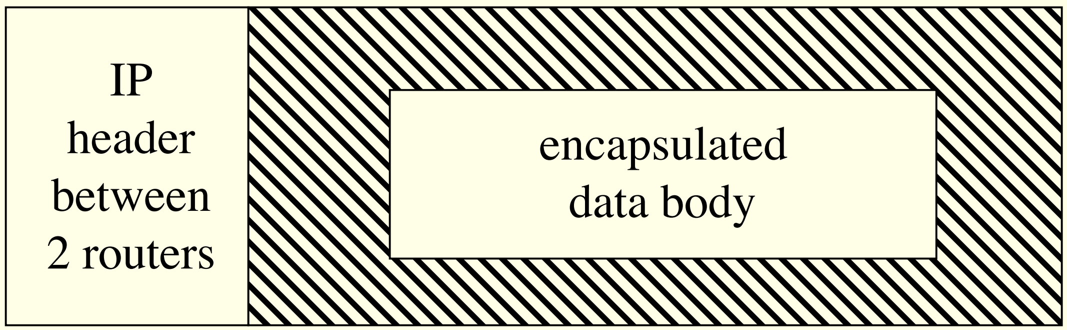
IPsec

- Network layer security
  - Provides confidentiality, integrity, authentication of endpoints, replay detection
- Protects all messages sent along a path



IPsec Tunnel Mode

IP ESP {previous IP packet}



- Encapsulate IP packet (IP header *and* IP data)
- Use IP to send IPsec-wrapped packet
- Note: inner IP header protected
- typically end to router, or router to router

IPsec Protocols

- Authentication Header (AH)
  - integrity, authentication
  - weak anti-replay
- Encapsulating Security Payload (ESP)
  - Confidentiality + anti-replay
  - in current version hash is also available
- one either uses AH or ESP, but not both
- IKE = Oakley (DH more or less) + ISAKMP
  - ISAKMP is a metaprotocol for KMP design

IPsec Architecture

- Security Policy Database (SPD)
  - Says how to handle messages (discard them, add security services, forward message unchanged)
  - SPD associated with network interface
  - SPD determines appropriate entry from packet attributes
    - Including source, destination, transport protocol

Example

- Goals
  - Discard SMTP packets from host 192.168.2.9
  - Forward packets from 192.168.19.7 without change
- SPD entries

```
src 192.168.2.9, dest 10.1.2.3 to 10.1.2.103, port 25, discard
src 192.168.19.7, dest 10.1.2.3 to 10.1.2.103, port 25, bypass
dest 10.1.2.3 to 10.1.2.103, port 25, apply IPsec
```

- Note: entries scanned in order
  - If no match for packet, it is discarded

IPsec Architecture

- Security Association (SA)
  - Association between peers for security services
    - Identified uniquely by dest address, security protocol (AH or ESP), unique 32-bit number (security parameter index, or SPI)
  - Unidirectional (routing is 2 one-way problems)
    - Can apply different services in either direction
  - SA uses either ESP or AH; if both required, 2 SAs needed

SA Database (SAD)

- Entry describes SA; some fields for all packets:
  - AH algorithm identifier, keys
    - When SA uses AH
  - ESP encipherment algorithm identifier, keys
    - When SA uses confidentiality from ESP
  - ESP authentication algorithm identifier, keys
    - When SA uses authentication, integrity from ESP
  - SA lifetime (time for deletion or max byte count)
  - IPsec mode (tunnel, transport, either)

SAD Fields

- Antireplay (inbound only)
  - When SA uses antireplay feature
- Sequence number counter (outbound only)
  - Generates AH or ESP sequence number
- Sequence counter overflow field
  - Stops traffic over this SA if sequence counter overflows
- Aging variables
  - Used to detect time-outs

Which to Use: Gnu PGP, IPSEC?

- What do the security services apply to?
  - If applicable to one application *and* application layer mechanisms available, use that
    - PGP/SSL for electronic mail
    - IPSEC is VPN, can cover ALL applications, but maybe not end to end

    might be

    host to IPSEC server inside enterprise

    router to router between enterprises

study questions

- what session-key algorithms did we talk about?
  - miss any major ones?
- is crypto the problem with network protocols using it (or the packaging)?
- people have a hard time with keys, why?
  - public-key crypto
  - shared secrets (in symmetric or MD algorithms)
- what does single sign-on mean?
  - and do you think it will ever happen?