Lecture for February 8, 2016

ECS 235A

UC Davis

Matt Bishop

Cryptographic Key Infrastructure

- Goal: bind identity to key
- Classical: not possible as all keys are shared
  - Use protocols to agree on a shared key (see earlier)
- 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

- 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
  - 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
  - Two approaches: Merkle's tree, signature chains

X.509 Chains

- 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

X.509 Certificate Validation

- Obtain issuer's public key
  - The one for the particular signature algorithm
- Decipher signature
  - Gives hash of certificate
- Recompute hash from certificate and compare
  - If they differ, there's a problem
- Check interval of validity
  - This confirms that certificate is current

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

Validation and Cross-Certifying

- Certificates:
  - Cathy<<Alice>>
  - Dan<<Bob>
  - Cathy<<Dan>>
  - Dan<<Cathy>>
- Alice validates Bob's certificate
  - Alice obtains Cathy<<Dan>>
  - Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
  - Alice uses Cathy<<Dan>> to validate Dan<<Bob>>

PGP Chains

- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash)
- Version 4 packet more complex

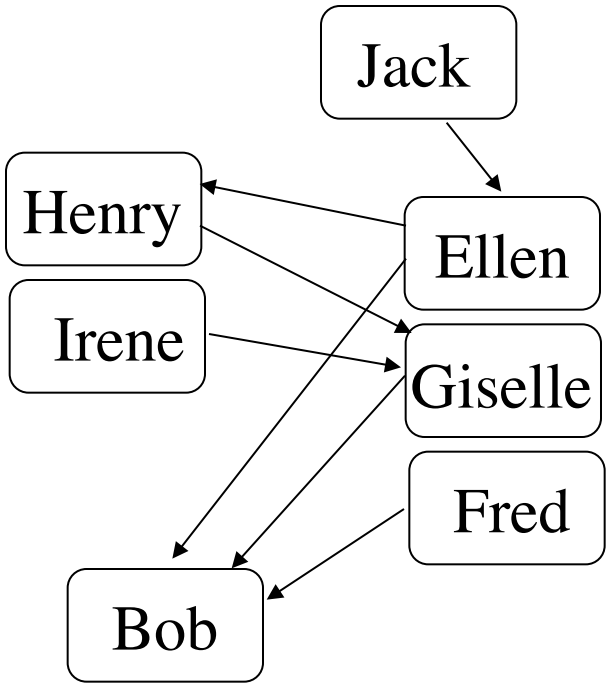
Signing

- Single certificate may have multiple signatures
- Notion of "trust" embedded in each signature
  - Range from "untrusted" to "ultimate trust"
  - Signer defines meaning of trust level (no standards!)
- All version 4 keys signed by subject
  - Called "self-signing"

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
    - Attacker can monitor keystrokes to decipher files
    - Key will be resident in memory that attacker may be able to read
  - Use physical devices like "smart card"
    - Key never enters system
    - Card can be stolen, so have 2 devices combine bits to make single key

Key Revocation

- Certificates 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

CRLs

- *Certificate revocation list* lists certificates that are revoked
- X.509: only certificate issuer can revoke certificate
  - Added to CRL
- PGP: signers can revoke signatures; owners can revoke certificates, or allow others to do so
  - Revocation message placed in PGP packet and signed
  - Flag marks it as revocation message

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

Common Error

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

This is a digital signature

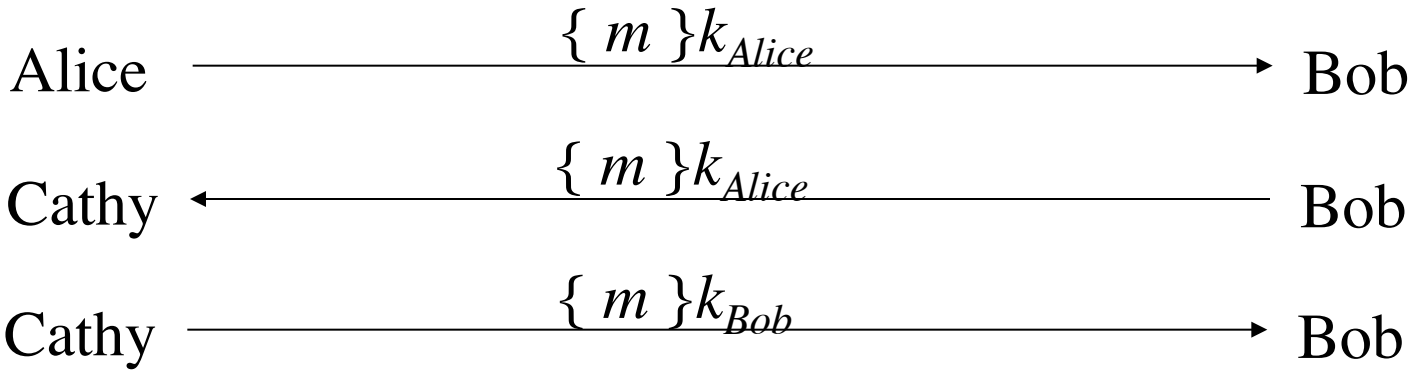
WRONG

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

- Alice's keys are d_{Alice}, e_{Alice}
- Alice sends Bob

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

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

Attack #1

- Example: Alice, Bob communicating
  - $n_A = 95, e_A = 59, d_A = 11$
  - $n_B = 77, e_B = 53, d_B = 17$
- 26 contracts, numbered 00 to 25
  - Alice has Bob sign 05 and 17:
    - $c = m^{d_B} \bmod n_B = 05^{17} \bmod 77 = 3$
    - $c = m^{d_B} \bmod n_B = 17^{17} \bmod 77 = 19$
  - Alice computes 05×17 mod 77 = 08; corresponding signature is 03×19 mod 77 = 57; claims Bob signed 08
  - Judge computes $c^{e_B} \bmod n_B = 57^{53} \bmod 77 = 08$
    - Signature validated; Bob is toast

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 $p = 29$, $g = 3$, $d = 6$

  $y = 3^6 \bmod 29 = 4$
- Alice wants to send Bob signed contract 23
  - Chooses $k = 5$ (relatively prime to 28)
  - This gives $a = g^k \bmod p = 3^5 \bmod 29 = 11$
  - Then solving $23 = (6 \times 11 + 5b) \bmod 28$ gives $b = 25$
  - Alice sends message 23 and signature $(11, 25)$
- Bob verifies signature: $g^m \bmod p = 3^{23} \bmod 29 = 8$ and $y^a a^b \bmod p = 4^{11} 11^{25} \bmod 29 = 8$
  - 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 $k = 5$

  $m = (da + kb) \bmod p–1 = (11d + 5\times25) \bmod 28$

  so Alice's private key is $d = 6$

Using Ciphers

- Problems
  - What can go wrong if you naively use ciphers
- Networks
  - Link vs end-to-end use
- Example
  - Secure Socket Layer (SSL)

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?
  - 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*

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
    - 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
  - $n_{Bob} = 77, e_{Bob} = 17, d_{Bob} = 53$
  - 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):

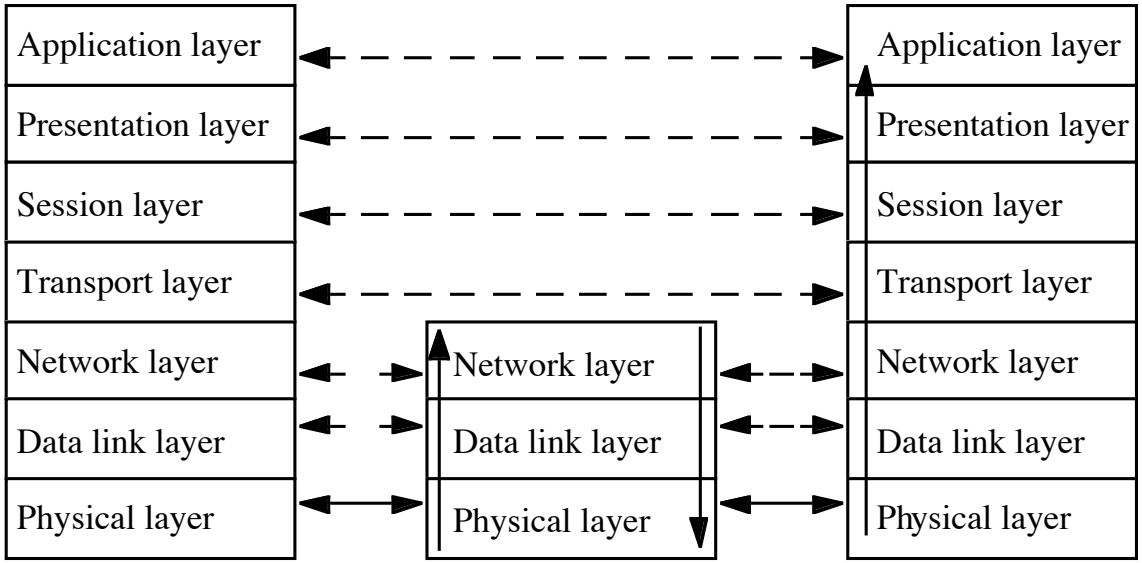
    `ef7c 4bb2 b4ce 6f3b ef7c 4bb2 b4ce 6f3b`
- Fix: cascade blocks together (chaining)
  - CBC mode does this, as do other modes

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)

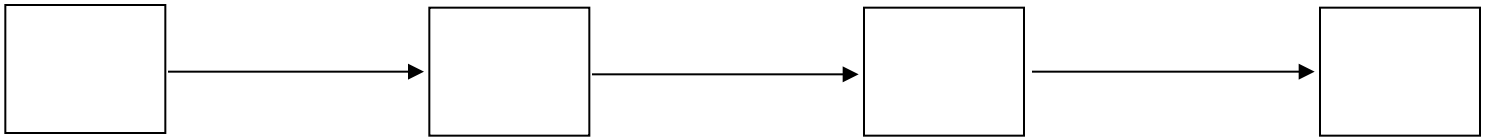
Networks and Cryptography

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

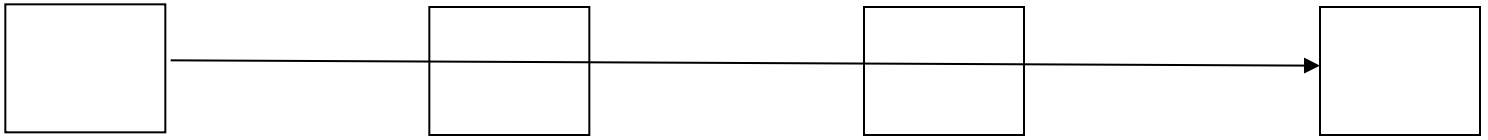


Link and End-to-End Protocols

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

- TELNET protocol
  - Messages between client, server 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

SSL

- Transport layer security
  - Provides confidentiality, integrity, authentication of endpoints
  - Developed by Netscape for WWW browsers and servers
- Internet protocol version: TLS
  - Compatible with SSL
  - Not yet formally adopted

SSL Session

- Association between two peers
  - May have many associated connections
  - Information for each association:
    - Unique session identifier
    - Peer's X.509v3 certificate, if needed
    - Compression method
    - Cipher spec for cipher and MAC
    - "Master secret" of 48 bits shared with peer

SSL Connection

- Describes how data exchanged with peer
- Information for each connection
  - Random data
  - Write keys (used to encipher data)
  - Write MAC key (used to compute MAC)
  - Initialization vectors for ciphers, if needed
  - Sequence numbers

Structure of SSL