Overview

- Key exchange
  - Session vs. interchange keys
  - Classical, public key methods
  - Key generation
- Cryptographic key infrastructure
  - Certificates
- Key storage
  - Key escrow
  - Key revocation
- Digital signatures

Notation

- $X \rightarrow Y : \{ Z \parallel W \} k_{X,Y}$
  - $X$ sends $Y$ the message produced by concatenating $Z$ and $W$
    enciphered by key $k_{X,Y}$, which is shared by users $X$ and $Y$
- $A \rightarrow T : \{ Z \} k_A \parallel \{ W \} k_{A,T}$
  - $A$ 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)
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
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

Classical 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 $\{ \text{request for session key to Bob} \} k_A \rightarrow$ Cathy

Alice $\{ k_s \} k_A \mathbin{||} \{ k_s \} k_B \leftarrow$ Cathy

Alice $\{ k_s \} k_B \rightarrow$ 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
Needham-Schroeder

Alice || Bob || \( r_1 \)

Alice || Bob || \{ Alice || Bob \} \( k_A \)

Alice || Bob || \{ Alice || k_s \} \( k_B \)

Alice || Bob || \{ r_2 \} \( k_s \)

Alice || Bob || \{ r_2 - 1 \} \( k_s \)

Argument: Alice talking to Bob

- Second message
  - Enciphered using key only she, Cathy know
    - 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 $k_s$
    - In what follows, Eve knows $k_s$
      - { Alice $\| k_s$ } $k_B$
      - Bob
      - $\{ r_2 \} k_s$
      - Bob
      - $\{ r_2 - 1 \} k_s$
      - Bob
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 $\rightarrow$ Cathy
  - $\{ \text{Alice} \parallel \text{Bob} \parallel r_1 \} \parallel k_s \parallel \{ \text{Alice} \parallel T \parallel k_s \} \parallel k_B$ \parallel $k_A$
- Bob $\rightarrow$ Alice
  - $\{ r_2 \} \parallel k_s$
- Cathy $\rightarrow$ Alice
  - $\{ \text{Alice} \parallel T \parallel k_s \} \parallel k_B$
- Bob $\rightarrow$ Alice
  - $\{ r_2 - 1 \} \parallel k_s$
Otway-Rees Protocol

• Corrects problem
  – That is, Eve replaying the third message in the protocol
• Does not use timestamps
  – Not vulnerable to the problems that Denning-Sacco modification has
• Uses integer $n$ to associate all messages with particular exchange

The Protocol

Alice $\langle n \parallel Alice \parallel Bob \parallel \{ r_1 \parallel n \parallel Alice \parallel Bob \} k_A \rangle$ $\rightarrow$ Bob

Cathy $\langle n \parallel Alice \parallel Bob \parallel \{ r_1 \parallel n \parallel Alice \parallel Bob \} k_A \parallel \{ r_2 \parallel n \parallel Alice \parallel Bob \} k_B \rangle$ $\rightarrow$ Bob

Cathy $\langle n \parallel \{ r_1 \parallel k_A \} k_A \parallel \{ r_2 \parallel k_A \} k_B \rangle$ $\rightarrow$ Bob

Alice $\langle n \parallel \{ r_1 \parallel k_A \} k_A \rangle$ $\rightarrow$ Bob
Argument: Alice talking to Bob

• Fourth message
  – If $n$ matches first message, Alice knows it is part of this protocol exchange
  – Cathy generated $k_s$ because only she, Alice know $k_A$
  – Enciphered part belongs to exchange as $r_1$ matches $r_1$ in encrypted part of first message

Argument: Bob talking to Alice

• Third message
  – If $n$ matches second message, Bob knows it is part of this protocol exchange
  – Cathy generated $k_s$ because only she, Bob know $k_B$
  – Enciphered part belongs to exchange as $r_2$ matches $r_2$ in encrypted part of second message
Replay Attack

- Eve acquires old $k_s$, message in third step
  - $n \parallel \{ r_1 \parallel k_s \} k_A \parallel \{ r_2 \parallel k_s \} k_B$
- Eve forwards appropriate part to Alice
  - Alice has no ongoing key exchange with Bob: $n$ matches nothing, so is rejected
  - Alice has ongoing key exchange with Bob: $n$ does not match, so is again rejected
    - If replay is for the current key exchange, \textit{and} Eve sent the relevant part \textit{before} Bob did, Eve could simply listen to traffic; no replay involved

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 $s$
  $$T_{u,s} = s || \{ u || u’s \text{ address} || \text{valid time} || k_{u,s} \} k_s$$
  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 \[\text{user} \| \text{TGS}\] \rightarrow \text{Cathy}
\text{Cathy} \leftarrow \{ k_{u,TGS} \} k_u \| T_{u,TGS} \ \text{user}
user \[\text{service} \| A_{u,TGS} \| T_{u,TGS}\] \rightarrow \text{TGS}
user \leftarrow \{ k_{u,s} \} k_{u,TGS} \| T_{u,s} \ \text{TGS}
user \[A_{u,s} \| T_{u,s}\] \rightarrow \text{service}
user \leftarrow \{ t + 1 \} k_{u,s} \ \text{service}
```
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
• 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

\[
\text{Alice} \quad \{ k_s \} e_B \quad \text{Bob}
\]

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

\[
\text{Alice} \quad \{ \{ k_s \} d_A \} e_B \quad \text{Bob}
\]
Notes

- Can include message enciphered with $k_i$
- 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 $\xrightarrow{\text{send Bob’s public key}}$ Eve $\xleftarrow{e_E}$ Alice $\leftarrow$ Eve $\xrightarrow{e_B}$ Cathy $\xleftarrow{\text{Eve intercepts request}}$ Cathy $\xrightarrow{Eve intercepts message}$ Bob

Alice $\xleftarrow{\{k_s\}e_E}$ Eve $\xrightarrow{Eve intercepts message}$ Bob

Eve $\xleftarrow{\{k_s\}e_B}$ Bob
Key Generation

- Goal: generate difficult to guess keys
- Problem statement: given a set of $K$ potential keys, choose one randomly
  - Equivalent to selecting a random number between 0 and $K-1$ inclusive
- Why is this hard: generating random numbers
  - Actually, numbers are usually pseudo-random, that is, generated by an algorithm

What is “Random”?

- Sequence of cryptographically random numbers: a sequence of numbers $n_1, n_2, \ldots$ such that, for any integer $k > 0$, an observer cannot predict $n_k$ even if all of $n_1, \ldots, n_{k-1}$ are known
  - Best: physical source of randomness
    - Random pulses
    - Electromagnetic phenomena
    - Characteristics of computing environment such as disk latency
    - Ambient background noise
What is “Pseudorandom”?  

- *Sequence of cryptographically pseudorandom numbers*:  
  sequence of numbers intended to simulate a sequence of cryptographically random numbers but generated by an algorithm  
  - Very difficult to do this well  
  - Linear congruential generators \[ n_k = (an_{k-1} + b) \mod n \] broken  
  - Polynomial congruential generators \[ n_k = (a_n n_{k-1} + \ldots + a_1 n_{k-1} a_0) \mod n \] broken too  
  - Here, “broken” means next number in sequence can be determined

Best Pseudorandom Numbers

- *Strong mixing function*: function of 2 or more inputs with each bit of output depending on some nonlinear function of all input bits  
  - Examples: DES, MD5, SHA-1  
  - Use on UNIX-based systems:  
    \[(date;\ ps\ gaux)\ |\ md5\]  
    where “ps gaux” lists all information about all processes on system
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)

\[ C_A = \{ e_A \parallel Alice \parallel T \} d_C \]
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

Merkle’s Tree Scheme

- Keep certificates in a file
  - Changing any certificate changes the file
  - Use crypto hash functions to detect this
- Define hashes recursively
  - \( h \) is hash function
  - \( C_i \) is certificate \( i \)
- Hash of file (\( h(1,4) \) in example) known to all

\[
\begin{align*}
  h(1,4) & \quad h(1,2) \quad h(3,4) \\
  h(1,1) & \quad h(2,2) \quad h(3,3) \quad h(4,4) \\
  C_1 & \quad C_2 \quad C_3 \quad C_4
\end{align*}
\]
Validation

- To validate $C_i$:
  - Compute $h(1, 1)$
  - Obtain $h(2, 2)$
  - Compute $h(1, 2)$
  - Obtain $h(3, 4)$
  - Compute $h(1, 4)$
  - Compare to known $h(1, 4)$

- Need to know hashes of children of nodes on path that are not computed

Details

- $f: D \times D \rightarrow D$ maps bit strings to bit strings
- $h: N \times N \rightarrow D$ maps integers to bit strings
  - if $i \geq j$, $h(i, j) = f(C_i, C_j)$
  - if $i < j$,
    $$h(i, j) = f(h(i, \lfloor (i+j)/2 \rfloor), h(\lceil (i+j)/2 \rceil + 1, j))$$
Problem

- File must be available for validation
  - Otherwise, can’t recompute hash at root of tree
  - Intermediate hashes would do
- Not practical in most circumstances
  - Too many certificates and users
  - Users and certificates distributed over widely separated systems

Certificate Signature Chains

- Create certificate
  - Generate hash of certificate
  - Encipher hash with issuer’s private key
- Validate
  - Obtain issuer’s public key
  - Decipher enciphered hash
  - Recompute hash from certificate and compare
- Problem: getting issuer’s public key
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 Escrow

- *Key escrow system* allows authorized third party to recover key
  - Useful when keys belong to roles, such as system operator, rather than individuals
  - Business: recovery of backup keys
  - Law enforcement: recovery of keys that authorized parties require access to
- Goal: provide this without weakening cryptosystem
- Very controversial
Desirable Properties

- Escrow system should not depend on encipherment algorithm
- Privacy protection mechanisms must work from end to end and be part of user interface
- Requirements must map to key exchange protocol
- System supporting key escrow must require all parties to authenticate themselves
- If message to be observable for limited time, key escrow system must ensure keys valid for that period of time only

Components

- User security component
  - Does the encipherment, decipherment
  - Supports the key escrow component
- Key escrow component
  - Manages storage, use of data recovery keys
- Data recovery component
  - Does key recovery
Example: EES, Clipper Chip

- Escrow Encryption Standard
  - Set of interlocking components
  - Designed to balance need for law enforcement access to enciphered traffic with citizens’ right to privacy
- Clipper chip prepares per-message escrow information
  - Each chip numbered uniquely by UID
  - Special facility programs chip
- Key Escrow Decrypt Processor (KEDP)
  - Available to agencies authorized to read messages

User Security Component

- Unique device key $k_{unique}$
- Nonunique family key $k_{family}$
- Cipher is Skipjack
  - Classical cipher: 80 bit key, 64 bit input, output blocks
- Generates Law Enforcement Access Field (LEAF) of 128 bits:
  - $\{ \text{UID} \parallel \{ k_{session} \} k_{unique} \parallel \text{hash} \} k_{family}$
  - $\text{hash}$: 16 bit authenticator from session key and initialization vector