Cryptographic Checksums

- Mathematical function to generate a set of k bits from a set of n bits (where $k \leq n$).
  - k is smaller then n except in unusual circumstances
- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is “parity”
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits

Example Use

- Bob receives "10111101" as bits.
  - Sender is using even parity; 6 1 bits, so character was received correctly
    - Note: could be garbled, but 2 bits would need to have been changed to preserve parity
  - Sender is using odd parity; even number of 1 bits, so character was not received correctly

Definition

- Cryptographic checksum $h: A \rightarrow B$:
  1. For any $x \in A$, $h(x)$ is easy to compute
  2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
  3. It is computationally infeasible to find two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
     – Alternate form (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Collisions

- If $x \neq x'$ and $h(x) = h(x')$, x and x' are a *collision*
  - Pigeonhole principle: if there are n containers for $n+1$ objects, then at least one container will have 2 objects in it.
  - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files

Keys

- Keyed cryptographic checksum: requires cryptographic key
  - DES in chaining mode: encipher message, use last *n* bits. Requires a key to encipher, so it is a keyed cryptographic checksum.
- Keyless cryptographic checksum: requires no cryptographic key
  - MD5 and SHA-1 are best known; others include MD4, HAVAL, and Snefru

HMAC

- Make keyed cryptographic checksums from keyless cryptographic checksums
- h keyless cryptographic checksum function that takes data in blocks of b bytes and outputs blocks of l bytes. k' is cryptographic key of length b bytes
  - If short, pad with 0 bytes; if long, hash to length b
- *ipad* is 00110110 repeated b times
- *opad* is 01011100 repeated b times
- HMAC-$h(k, m) = h(k' \oplus opad \,\|\, h(k' \oplus ipad \,\|\, m))$
  - $\oplus$ exclusive or, $\|$ concatenation

Key Points

- Two main types of cryptosystems: classical and public key
- Classical cryptosystems encipher and decipher using the same key
  - Or one key is easily derived from the other
- Public key cryptosystems encipher and decipher using different keys
  - Computationally infeasible to derive one from the other
- Cryptographic checksums provide a check on integrity

Overview

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

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)

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

Needham-Schroeder

Alice → Cathy: $\text{Alice} \,\|\, \text{Bob} \,\|\, r_1$

Alice ← Cathy: $\{ \text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_s \,\|\, \{ \text{Alice} \,\|\, k_s \}\, k_B \}\, k_A$

Alice → Bob: $\{ \text{Alice} \,\|\, k_s \}\, k_B$

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 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

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

Eve ← Bob: $\{ r_2 \}\, k_s$

Eve → Bob: $\{ r_2 - 1 \}\, k_s$

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 $\longrightarrow$ Cathy: Alice || Bob || r_1

Alice $\longleftarrow$ Cathy: { Alice || Bob || r_1 || k_s || { Alice || T || k_s } k_B } k_A

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

Alice $\longleftarrow$ Bob: { r_2 } k_s

Alice $\longrightarrow$ Bob: { $r_2 - 1$ } 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 $\xrightarrow{n \,||\, \text{Alice} \,||\, \text{Bob} \,||\, \{\, r_1 \,||\, n \,||\, \text{Alice} \,||\, \text{Bob} \,\}\, k_A}$ Bob

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

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
  - $n \| \{ r_1 \| k_s \} k_A \| \{ r_2 \| k_s \} k_B$
- 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, *and* Eve sent the relevant part *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

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

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

$$\textit{user} \xrightarrow{\textit{user} \parallel \textit{TGS}} \text{Cathy}$$

$$\text{Cathy} \xleftarrow{\{ k_{u,TGS} \} k_u \parallel T_{u,TGS}} \textit{user}$$

$$\textit{user} \xrightarrow{\textit{service} \parallel A_{u,TGS} \parallel T_{u,TGS}} \text{TGS}$$

$$\textit{user} \xleftarrow{\textit{user} \parallel \{ k_{u,s} \} k_{u,TGS} \parallel T_{u,s}} \text{TGS}$$

$$\textit{user} \xrightarrow{A_{u,s} \parallel T_{u,s}} \textit{service}$$

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

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

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

Problem and Solution

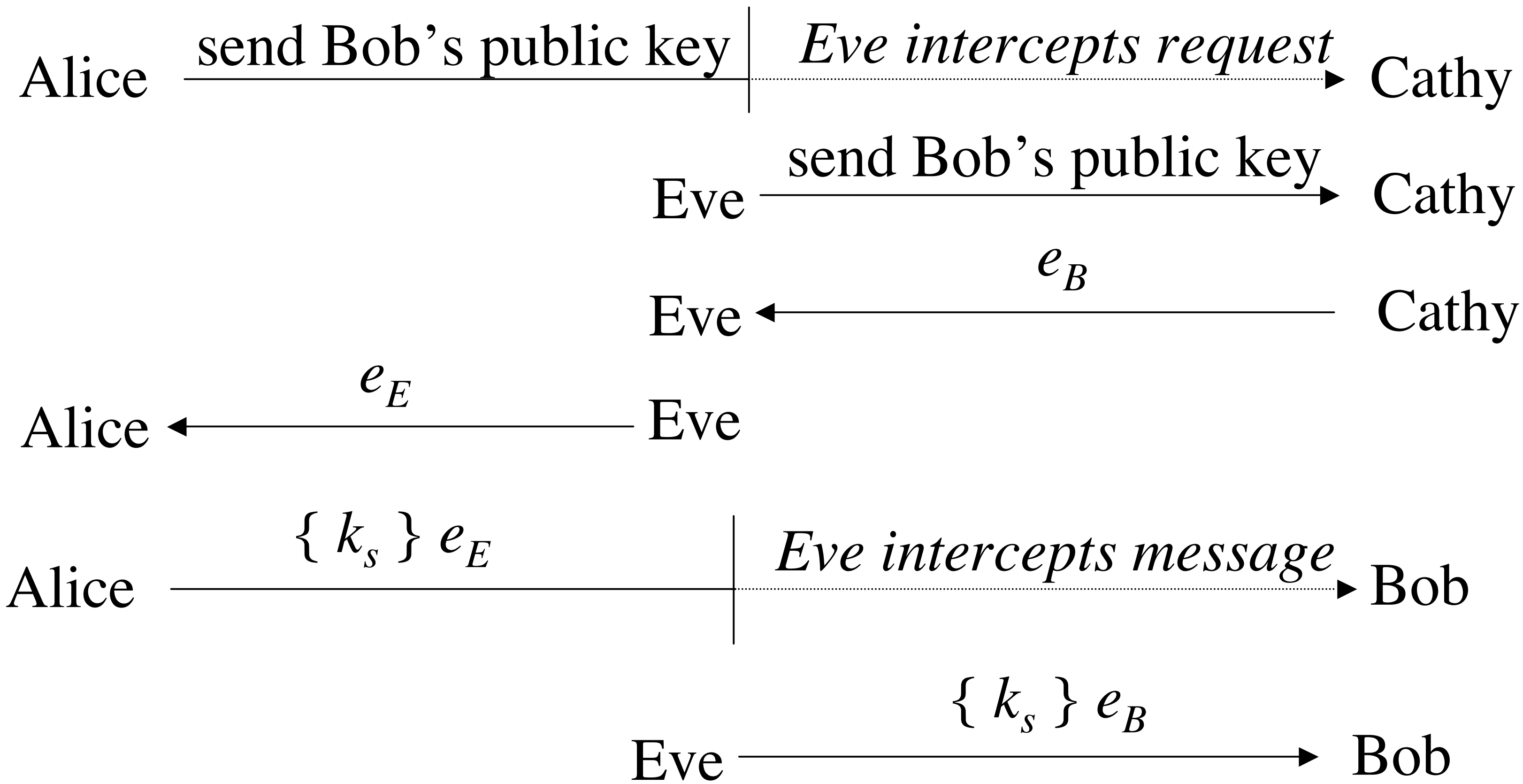
- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - k_s is desired session key

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

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



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)

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

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