IS 2150 / TEL 2810
Introduction to Security

James Joshi
Assistant Professor, SIS

Lecture 10
Nov 8, 2007

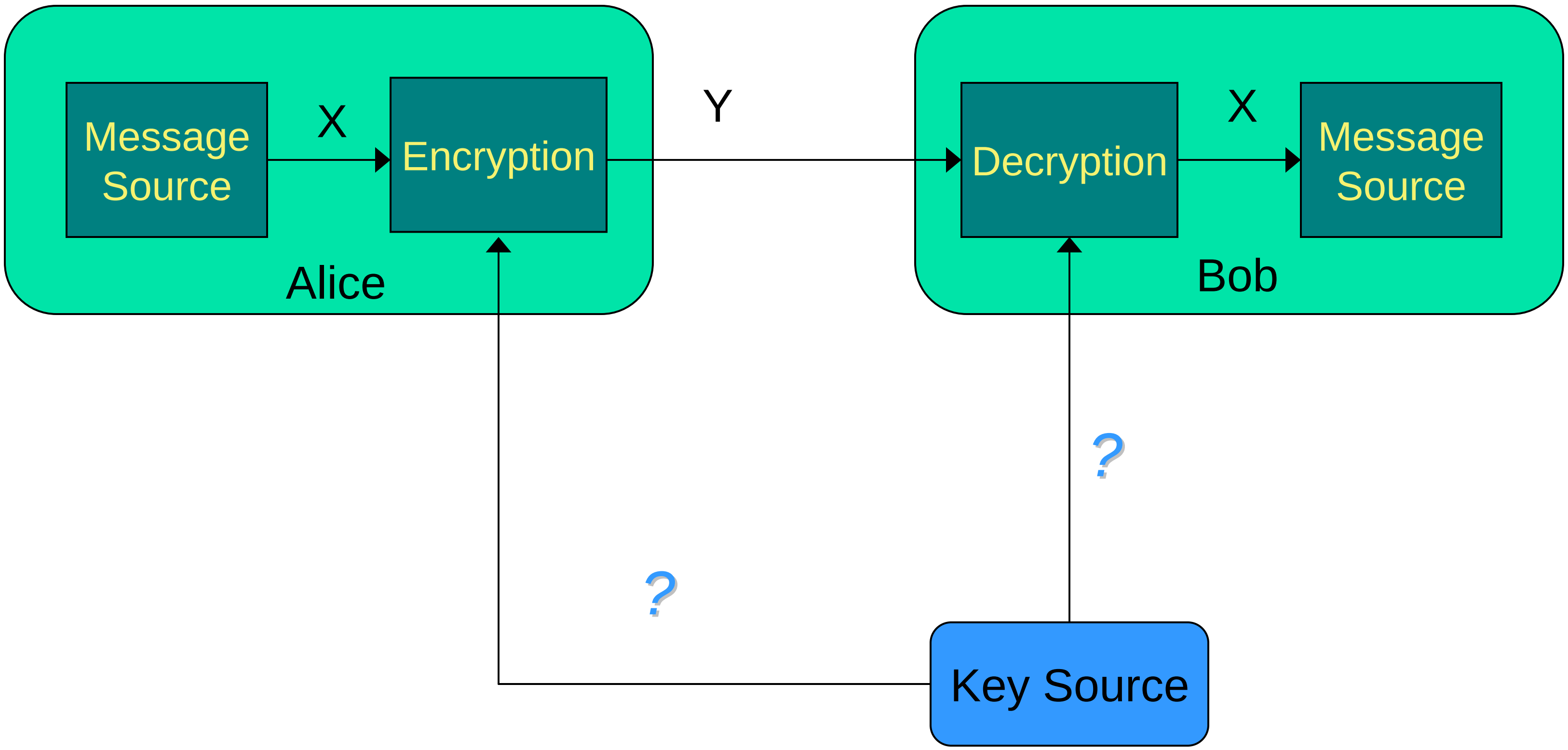
Hash Functions
Key Management

Objectives

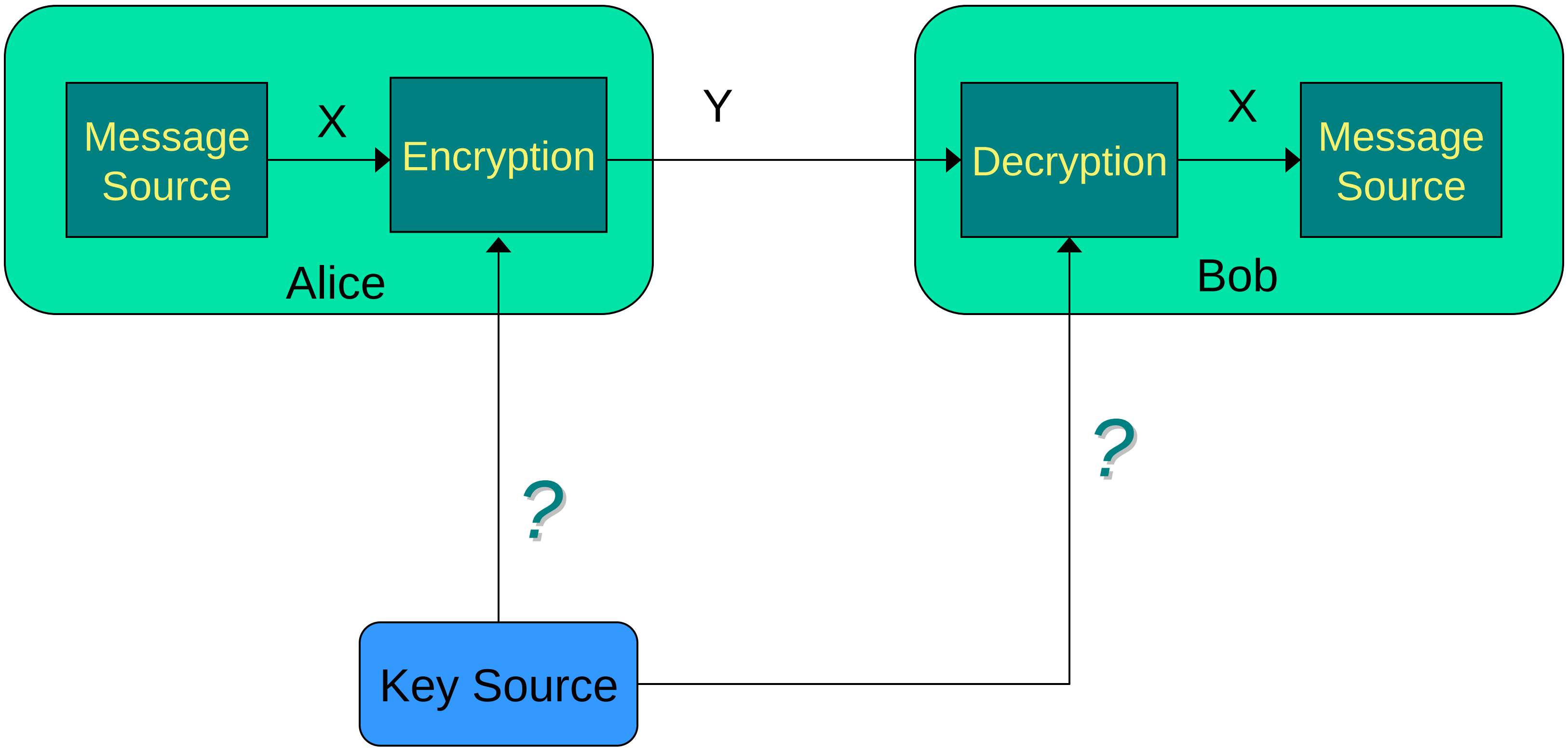
- Understand/explain the issues related to, and utilize the techniques
  - Hash functions
  - Key management
    - Authentication and distribution of keys
      - Session key
      - Key exchange protocols
      - Kerberos
    - Mechanisms to bind an identity to a key
    - Generation, maintenance and revoking of keys

Quick ReCap

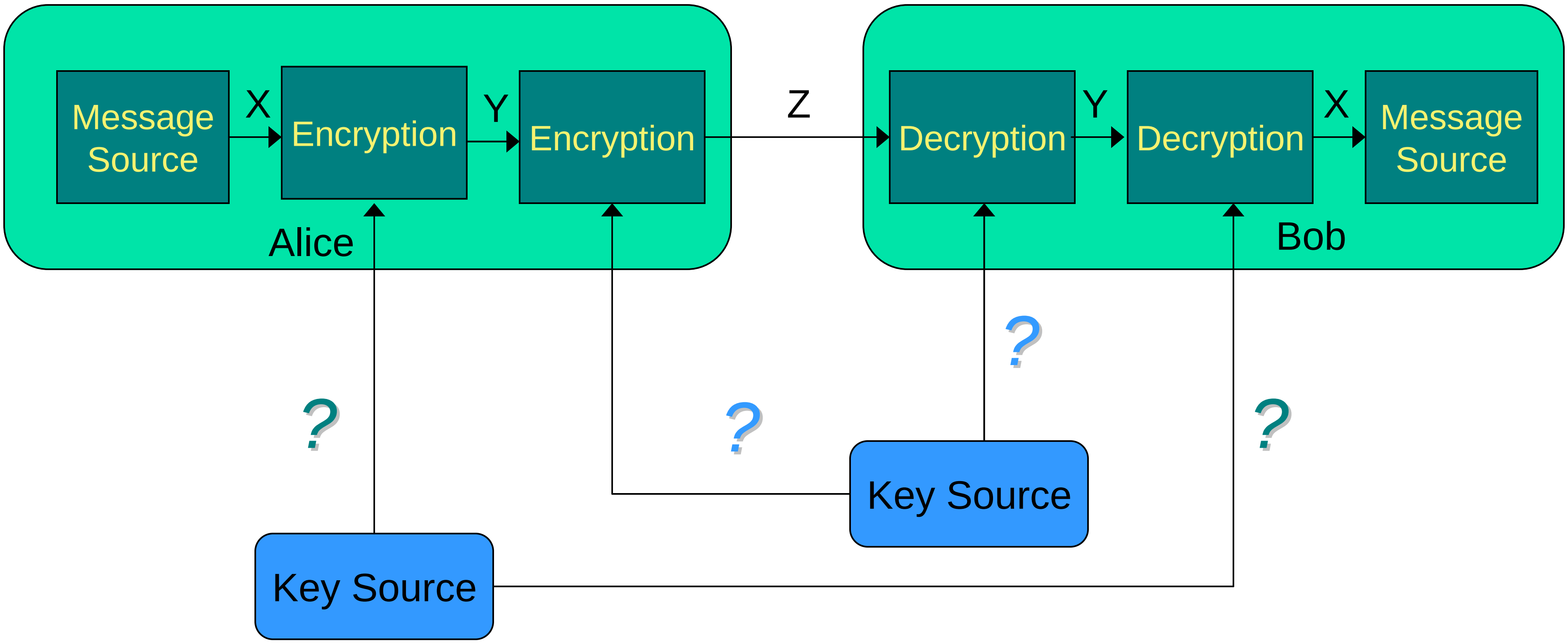
Confidentiality using RSA



Authentication using RSA



Confidentiality + Authentication



Hash Functions

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
  - Keyed CC: requires a cryptographic key

    $h = C_{Key}(M)$
  - Keyless CC: requires no cryptographic key
    - Message Digest or One-way Hash Functions

    $h = H(M)$
- Can be used for message authentication
  - Hence, also called Message Authentication Code (MAC)

Mathematical characteristics

- Every bit of the message digest function potentially influenced by every bit of the function’s input
- If any given bit of the function’s input is changed, every output bit has a 50 percent chance of changing
- Given an input file and its corresponding message digest, it should be computationally infeasible to find another file with the same message digest value

Definition

- Cryptographic checksum function $h: A \to B$:
  1. For any $x \in A$, $h(x)$ is easy to compute
     - Makes hardware/software implementation easy
  2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
     - *One-way property*
  3. It is computationally infeasible to find $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
  4. Alternate form: Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$.

Collisions possible?

Keys

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

Hash Message Authentication Code (HMAC)

- Make keyed cryptographic checksums from keyless cryptographic checksums

- h be keyless cryptographic checksum function
  - takes data in blocks of b bytes and outputs blocks of l bytes.
  - k' is cryptographic key of length b bytes (from k)
    - If short, pad with 0s' to make b 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

Protection Strength

- Unconditionally Secure
  - Unlimited resources + unlimited time
  - Still the plaintext CANNOT be recovered from the ciphertext
- Computationally Secure
  - Cost of breaking a ciphertext exceeds the value of the hidden information
  - The time taken to break the ciphertext exceeds the useful lifetime of the information

Average time required for exhaustive key search

Key Size (bits)	Number of Alternative Keys	Time required at 10^6 Decryption/µs
32	$2^{32} = 4.3 \times 10^9$	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	10 hours
128	$2^{128} = 3.4 \times 10^{38}$	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	5.9×10^{30} years

Key Management

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)

Interchange vs Session Keys

- Interchange Key
  - Tied to the principal of communication
- Session key
  - Tied to communication itself
- Example
  - Alice generates a random cryptographic key k_s and uses it to encipher m
  - She enciphers k_s with Bob's public key k_B
  - Alice sends $\{ m \} k_s \{ k_s \} k_B$
    - Which one is session/interchange key?

Benefits using session key

- In terms of Traffic-analysis by an attacker?
- Replay attack possible?
- Prevents some *forward search attack*
  - 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 to establish a shared key
- Criteria
  - 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 a third party
  - All cryptosystems, protocols assumed to be publicly known
    - Only secret data is the keys, OR ancillary information known only to Alice and Bob needed to derive keys

Classical Key Exchange

- 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 Key Exchange Protocol

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

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

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

$$\text{Alice} \xleftrightarrow{\{m\}k_s} \text{Bob}$$

What can an attacker, Eve, do to subvert it?

Needham-Schroeder

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

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

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

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

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

Questions

- How can Alice and Bob be sure they are talking to each other?
- Is the previous attack possible?
- Key assumption of Needham-Schroeder
  - All keys are secret;
  - What if we remove that assumption?

Needham-Schroeder with Denning-Sacco Modification

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

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

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

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

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

One solution to Needham-Schroeder problem: Use time stamp *T* to detect replay!

Denning-Sacco Modification

- Needs synchronized clocks
- Weaknesses:
  - 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

So use of time stamp adds other problems !!

Otway-Rees Protocol

$$\text{Alice} \xrightarrow{n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{ r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \} k_A} \text{Bob}$$

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

$$\text{Cathy} \xrightarrow{n \,\|\, \{ r_1 \,\|\, k_s \} k_A \,\|\, \{ r_2 \,\|\, k_s \} k_B} \text{Bob}$$

$$\text{Alice} \xleftarrow{n \,\|\, \{ r_1 \,\|\, k_s \} k_A} \text{Bob}$$

Uses integer *n* to associate all messages with a particular exchange

Argument: Alice talking to Bob

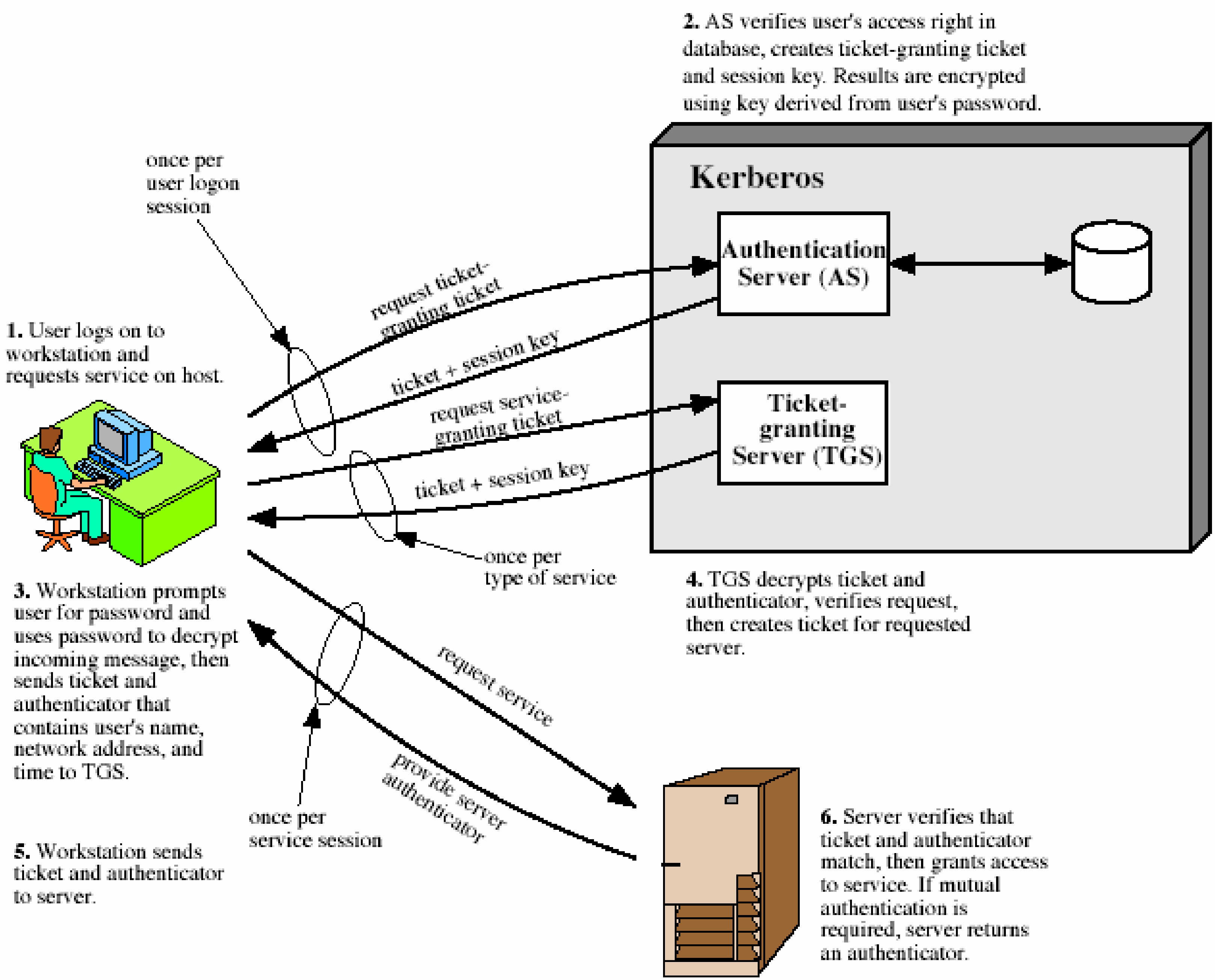
- How does Bob know it is actually Alice he is talking to?
- How does Alice know it is actually Bob she is talking to?

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
  - If Alice has no ongoing key exchange with Bob
    - Accept/reject the message ?
  - Alice has ongoing key exchange with Bob
    - Accept/reject the message ?
- If replay is for the current key exchange, *and* Eve sent the relevant part *before* Bob did,
  - Does replay attack occur?

Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party ("Cathy")
- Ticket (credential)
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender
- Alice must
  1. Authenticate herself to the system
  2. Obtain ticket to use server *S*


2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
Kerberos
Authentication Server (AS)
Ticket-granting Server (TGS)
once per user logon session
request ticket-granting ticket
ticket + session key
request service-granting ticket
ticket + session key
once per type of service
1. User logs on to workstation and requests service on host.
3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.
4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to server.
6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.

Overview

- 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 the ticket is 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

Authentication server

$$user \xrightarrow{\quad user \,\|\, TGS \quad} \mathrm{AS}$$

$$user \xleftarrow{\quad \{\, k_{u,TGS} \,\}\, k_u \,\|\, T_{u,TGS} \quad} \mathrm{AS}$$

$$user \xrightarrow{\quad service \,\|\, A_{u,TGS} \,\|\, T_{u,TGS} \quad} \mathrm{TGS}$$

$$user \xleftarrow{\quad user \,\|\, \{\, k_{u,s} \,\}\, k_{u,TGS} \,\|\, T_{u,s} \quad} \mathrm{TGS}$$

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

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

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

Problem and Solution?

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

Any problem ?

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

What about this?

Public Key Key Exchange

- 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

Man-in-the-Middle Attack

Alice $\xrightarrow{\text{send me Bob's public key}}$ | *Eve intercepts request* $\rightarrow$ Peter

Eve $\xrightarrow{\text{send me Bob's public key}}$ Peter

Eve $\xleftarrow{e_B}$ Peter

Alice $\xleftarrow{e_E}$ Eve

Alice $\xrightarrow{\{ k_s \}\, e_E}$ | *Eve intercepts message* $\rightarrow$ Bob

Eve $\xrightarrow{\{ k_s \}\, e_B}$ Bob

Peter is public server providing public keys

Cryptographic Key Infrastructure

- Goal:
  - bind identity to key
- Classical Crypto:
  - Not possible as all keys are shared
- Public key Crypto:
  - Bind identity to public 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 (identity of signer)

signed by trusted authority (here, Cathy)

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

C_A is A's certificate

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**

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
  - Re-compute hash from certificate and compare
- Problem:
  - Validating the certificate of the issuer and getting issuer's public key

X.509 Chains

- Key certificate fields in X.509v3:
  - Version
  - Serial number   (unique)
  - Signature algorithm identifier
  - Issuer’s name; uniquely identifies issuer
  - Interval of validity
  - Subject’s name; uniquely identifies subject
  - Subject’s public key

  …
  - Signature:
    - Identifies algorithm used to sign the certificate
    - 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
- Re-compute 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 Dan; 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>>
    - represents the certificate that C has generated for A
  - Dan<<Bob> ; Cathy<<Dan>>; Dan<<Cathy>>

- Alice validates Bob’s certificate
  - Alice obtains Cathy<<Dan>>
  - Can Alice validate Cathy<<Dan>> ?

PGP Chains

- Pretty Good Privacy:
  - Widely used to provide privacy for electronic mail and signing files digitally
- 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 using signer’s private key)

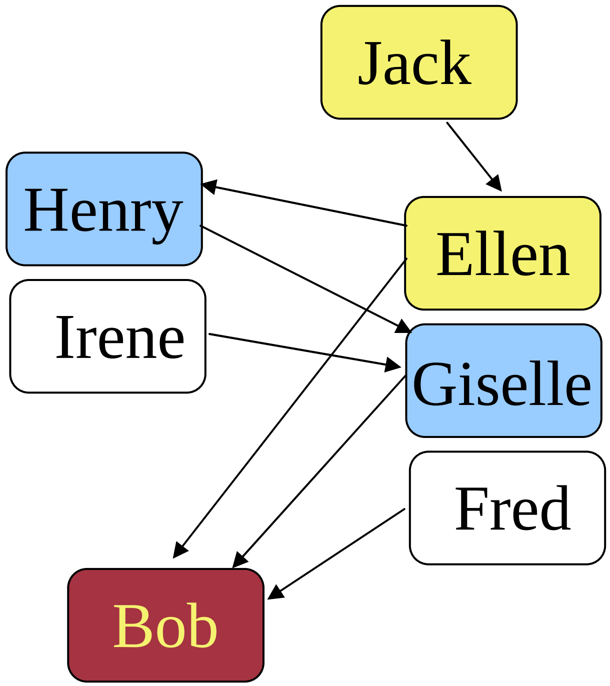
Signing

- 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



Digital Signature

- Construct that authenticates origin, contents of message in a manner provable to a disinterested third party ("judge")
- Sender cannot deny having sent message (which service is this??)
  - 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;

Signature

- Classical: Alice, Bob share key k
  - Alice sends $m \mid\mid \{ m \}k$ to Bob
  - Does this satisfy the requirement for message authentication? How?
  - Does this satisfy the requirement for a digital signature?

Classical Digital Signatures

- Require trusted third party
  - Alice, Bob share keys with trusted party Cathy
- The judge must trust Cathy

$$\text{Alice} \xrightarrow{\{\, m \,\}k_{Alice}} \text{Bob}$$

$$\text{Bob} \xrightarrow{\{\, m \,\}k_{Alice}} \text{Cathy}$$

$$\text{Cathy} \xrightarrow{\{\, m \,\}k_{Bob}} \text{Bob}$$

How can the judge resolve any dispute where one claims that the contract was not signed?

Public Key Digital Signatures (RSA)

- Alice's keys are d_{Alice}, e_{Alice}
- 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
    - 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 $05 \times 17 \bmod 77 = 08$; corresponding signature is $03 \times 19 \bmod 77 = 57$; claims Bob signed 08
  - Note: $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
  - Judge computes $c^{e_B} \bmod n_B = 57^{53} \bmod 77 = 08$
    - Signature validated; Bob is toast!

Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract 06
- Alice enciphers, then signs:
  - Enciper: $c = m^{e_B} \bmod n_B = (06^{53} \bmod 77)^{11}$
  - Sign: $c^{d_A} \bmod n_A = (06^{53} \bmod 77)^{11} \bmod 95 = 63$
- Bob now changes his public key
  - Bob wants to claim that Alice singed N (13)
  - Computes r such that $13^r \bmod 77 = 6$; say, $r = 59$
  - Computes $r.e_B \bmod \varphi(n_B) = 59 \times 53 \bmod 60 = 7$
  - Replace public key e_B with 7, private key $d_B = 43$
- Bob claims contract was 13. Judge computes:
  - $(63^{59} \bmod 95)^{43} \bmod 77 = 13$
  - Verified; now Alice is toast
- Solution: sign first and then enciher!!

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 $k = 5$

$$m = (da + kb) \bmod p{-}1 = 23$$
$$= (11d + 5 \times 25) \bmod 28$$

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

Summary

- Hash functions are key to authenticating data/message
- Session key is better for secret message exchange
- Public key good for interchange key, digital signatures – needs certification system
- Various replay/MITM attacks are possible in key exchange protocols and care is needed.