Introduction to
Computer Security

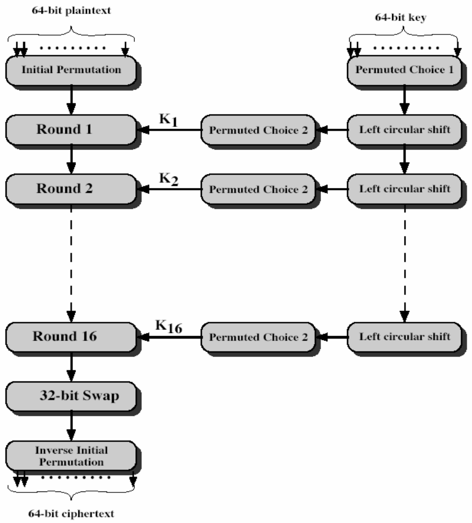
Lecture 6
Cryptography

October 2, 2003

Overview of the DES

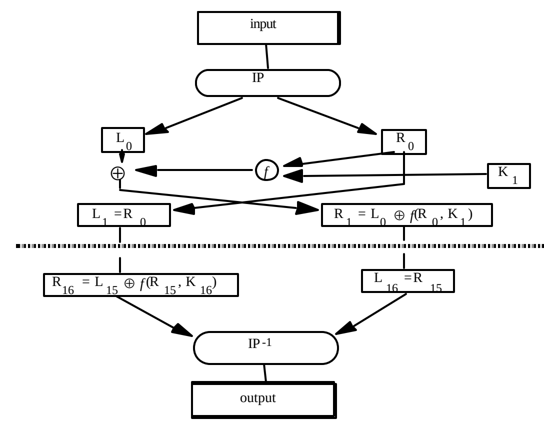
- A block cipher:
  - encrypts blocks of 64 bits using a 64 bit key
  - outputs 64 bits of ciphertext
  - A product cipher
    - performs both substitution and transposition (permutation) on the bits
  - basic unit is the bit
- Consists of 16 rounds (iterations) each with a round key generated from the user-supplied key

DES

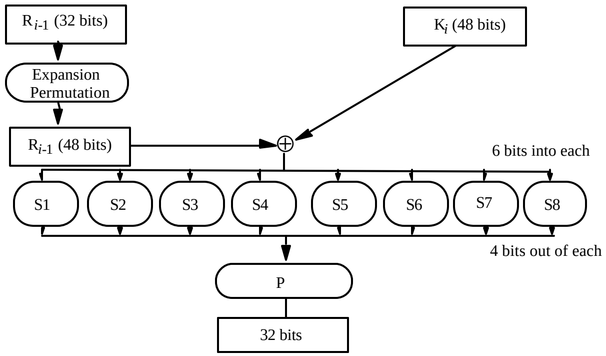


- Round keys are 48 bits each
  - Extracted from 64 bits
  - Permutation applied
- Deciphering involves using round keys in reverse

DES Encryption



The *f* function



Controversy

- Considered too weak
  - Diffie, Hellman said in a few years technology would allow DES to be broken in days
    - Design using 1999 technology published
  - Design decisions not public
    - S-boxes may have backdoors

Undesirable Properties

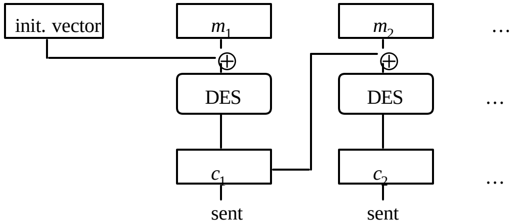
- 4 weak keys
  - They are their own inverses
- 12 semi-weak keys
  - Each has another semi-weak key as inverse
- Complementation property
  - $\mathrm{DES}_k(m) = c \Rightarrow \mathrm{DES}_{k'}(m') = c'$
- S-boxes exhibit irregular properties
  - Distribution of odd, even numbers non-random
  - Outputs of fourth box depends on input to third box
  - Reasons for structure were suspicious

Differential Cryptanalysis

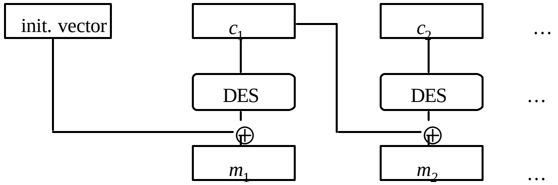
- A form of chosen plaintext attack
  - Involves encrypting many texts that are only slightly different from one another and comparing results
  - Requires 2^{47} plaintext, ciphertext pairs
- Revealed several properties
  - Small changes in S-boxes reduce the number of pairs needed
  - Making every bit of the round keys independent does not impede attack
- Linear cryptanalysis improves result
  - Requires 2^{43} plaintext, ciphertext pairs

DES Modes

- Electronic Code Book Mode (ECB):
  - Encipher each block independently
- Cipher Block Chaining Mode (CBC)
  - XOR each block with previous ciphertext block
  - Uses an initialization vector for the first one



CBC Mode Decryption



- CBC has self healing property
  - If one block of ciphertext is altered, the error propagates for at most two blocks

Self-Healing Property

- Initial message
  - `3231343336353837 3231343336353837 3231343336353837 3231343336353837`
- Received as (underlined 4c should be 4b)
  - `ef7c4cb2b4ce6f3b f6266e3a97af0e2c 746ab9a6308f4256 33e60b451b09603d`
- Which decrypts to
  - `efca61e19f4836f1 3231333336353837 3231343336353837 3231343336353837`
  - Incorrect bytes underlined; plaintext “heals” after 2 blocks

Current Status of DES

- Design for computer system, associated software that could break any DES-enciphered message in a few days published in 1998
- Several challenges to break DES messages solved using distributed computing
- NIST selected Rijndael as Advanced Encryption Standard, successor to DES
  - Designed to withstand attacks that were successful on DES

Public Key Cryptography

- Two keys
  - *Private key* known only to individual
  - *Public key* available to anyone
- Idea
  - Confidentiality:
    - encipher using public key,
    - decipher using private key
  - Integrity/authentication:
    - encipher using private key,
    - decipher using public one

Requirements

1. Given the appropriate key, it must be computationally easy to encipher or decipher a message
2. It must be computationally infeasible to derive the private key from the public key
3. It must be computationally infeasible to determine the private key from a chosen plaintext attack

Diffie-Hellman

- Compute a common, shared key
  - Called a *symmetric key exchange protocol*
- Based on discrete logarithm problem
  - Given integers n and g and prime number p, compute k such that $n = g^k \bmod p$
  - Solutions known for small p
  - Solutions computationally infeasible as p grows large – hence, choose large p

Algorithm

- Constants known to participants
  - prime p; integer g other than 0, 1 or p–1
- Alice: (private = k_A, public = K_A)
- Bob: (private = k_B, public = K_B)
  - $K_A = g^{kA} \bmod p$
  - $K_B = g^{kB} \bmod p$
- To communicate with Bob,
  - Anne computes $S_{A, B} = K_B^{kA} \bmod p$
- To communicate with Alice,
  - Bob computes $S_{B, A} = K_A^{kB} \bmod p$
- $S_{A, B} = S_{B, A}$?

Example

- Assume p = 53 and g = 17
- Alice chooses k_A = 5
  - Then K_A = 17^5 mod 53 = 40
- Bob chooses k_B= 7
  - Then K_B= 17^7 mod 53 = 6
- Shared key:
  - K_B^{kA} mod p = 6^5 mod 53 = 38
  - K_A^{kB} mod p = 40^7 mod 53 = 38

Let p = 5, g = 3
k_A = 4, k_B = 3

K_A = ?, K_B = ?,
S = ?,

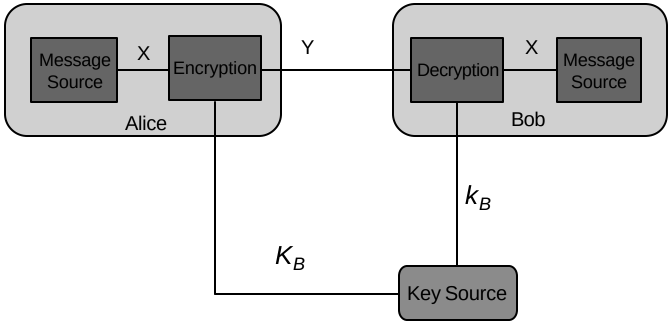
RSA

- Relies on the difficulty of determining the number of numbers relatively prime to a large integer n
- Totient function $\phi(n)$
  - Number of + integers less than n and relatively prime to n
    - Relatively prime means with no factors in common with n
- Example: $\phi(10)$ = 4
  - 1, 3, 7, 9 are relatively prime to 10
- $\phi(77)$?
- $\phi(p)$?
  - When p is a prime number
- $\phi(pq)$?
  - When p and q are prime numbers

Algorithm

- Choose two large prime numbers p, q
  - Let $n = pq$; then $\phi(n) = (p-1)(q-1)$
  - Choose $e < n$ relatively prime to $\phi(n)$.
  - Compute d such that $ed \bmod \phi(n) = 1$
- Public key: (e, n); private key: d (*or* (d, n))
- Encipher: $c = m^e \bmod n$
- Decipher: $m = c^d \bmod n$

Confidentiality using RSA



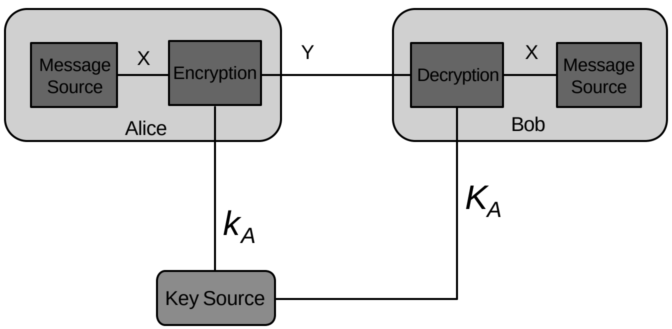
Example: Confidentiality

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Say Bob chooses (K_B) $e = 17$, making (k_B) $d = 53$
  - 17 x 53 mod 60 = ?
- Alice wants to send Bob secret message HELLO [07 04 11 11 14]
  - 07^{17} mod 77 = 28
  - 04^{17} mod 77 = 16
  - 11^{17} mod 77 = 44
  - 11^{17} mod 77 = 44
  - 14^{17} mod 77 = 42
- Alice sends ciphertext [28 16 44 44 42]

Example

- Bob receives [28 16 44 44 42]
- Bob uses private key (k_B), $d = 53$, to decrypt the message:
  - 28^{53} mod 77 = 07 H
  - 16^{53} mod 77 = 04 E
  - 44^{53} mod 77 = 11 L
  - 44^{53} mod 77 = 11 L
  - 42^{53} mod 77 = 14 O
- No one else could read it, as only Bob knows his private key and that is needed for decryption

Authentication using RSA



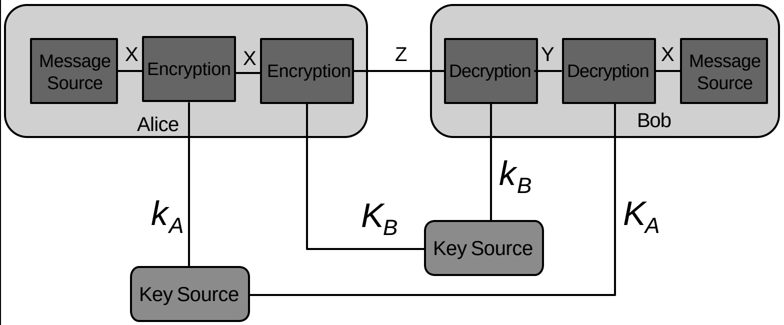
Example: Origin Integrity/Authentication

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses (K_A) $e = 17$, making (k_A) $d = 53$
- Alice wants to send Bob message HELLO [07 04 11 11 14] so Bob knows it is what Alice sent and there was no changes in transit
  - $07^{53} \bmod 77 = 35$
  - $04^{53} \bmod 77 = 09$
  - $11^{53} \bmod 77 = 44$
  - $11^{53} \bmod 77 = 44$
  - $14^{53} \bmod 77 = 49$
- Alice sends [35 09 44 44 49]

Example

- Bob receives 35 09 44 44 49
- Bob uses Alice's public key (KA), *e* = 17, *n* = 77, to decrypt message:
  - $35^{17} \bmod 77 = 07$ H
  - $09^{17} \bmod 77 = 04$ E
  - $44^{17} \bmod 77 = 11$ L
  - $44^{17} \bmod 77 = 11$ L
  - $49^{17} \bmod 77 = 14$ O
- Alice sent it as only she knows her private key, so no one else could have enciphered it
- If (enciphered) message's blocks (letters) altered in transit, would not decrypt properly

Confidentiality + Authentication



Example:
Confidentiality + Authentication

- Alice wants to send Bob message HELLO both enciphered and authenticated (integrity-checked)
  - Alice's keys: public (17, 77); private: 53
  - Bob's keys: public: (37, 77); private: 13
- Alice enciphers HELLO [07 04 11 11 14]:
  - $(07^{53} \bmod 77)^{37} \bmod 77 = 07$
  - $(04^{53} \bmod 77)^{37} \bmod 77 = 37$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(11^{53} \bmod 77)^{37} \bmod 77 = 44$
  - $(14^{53} \bmod 77)^{37} \bmod 77 = 14$
- Alice sends [07 37 44 44 14]

Example:
Confidentiality + Authentication

- Alice's keys: public (17, 77); private: 53
- Bob's keys: public: (37, 77); private: 13
- Bob deciphers (07 37 44 44 14):
  - $(07^{13} \bmod 77)^{17} \bmod 77 = 07$ H
  - $(37^{13} \bmod 77)^{17} \bmod 77 = 04$ E
  - $(44^{13} \bmod 77)^{17} \bmod 77 = 11$ L
  - $(44^{13} \bmod 77)^{17} \bmod 77 = 11$ L
  - $(14^{13} \bmod 77)^{17} \bmod 77 = 14$ O

Security Services

- Confidentiality
  - Only the owner of the private key knows it, so text enciphered with public key cannot be read by anyone except the owner of the private key
- Authentication
  - Only the owner of the private key knows it, so text enciphered with private key must have been generated by the owner

More Security Services

- Integrity
  - Enciphered letters cannot be changed undetectably without knowing private key
- Non-Repudiation
  - Message enciphered with private key came from someone who knew it

Warnings

- Encipher message in blocks considerably larger than the examples here
  - If 1 character per block, RSA can be broken using statistical attacks (just like classical cryptosystems)
  - Attacker cannot alter letters, but can rearrange them and alter message meaning
    - Example: reverse enciphered message of text ON to get NO

Cryptographic Checksums

- Mathematical function to generate a set of *k* bits from a set of *n* bits (where $k = n$).
  - *k* is smaller then *n* except in unusual circumstances
  - Keyed CC: requires a cryptographic key

    $h = C_K(M)$
  - 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 $h: A \rightarrow B$:
  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 proerpty*
  3. It is computationally infeasible to find $x, x' \in A$ such that x ? x' and $h(x) = h(x')$

  3'. 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 ? 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: suppose n = 5 and k = 3. Then there are 32 elements of A and 8 elements of B, so at least one element of B has at least 4 corresponding elements of A

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

Message Digest

- MD2, MD4, MD5 (Ronald Rivest)
  - Produces 128-bit digest;
  - MD2 is probably the most secure, longest to compute (hence rarely used)
  - MD4 is a fast alternative; MD5 is modification of MD4
- SHA, SHA-1 (Secure Hash Algorithm)
  - Related to MD4; used by NIST's Digital Signature
  - Produces 160-bit digest
  - SHA-1 may be better
- SHA-256, SHA-384, SHA-512
  - 256-, 384-, 512 hash functions designed to be use with the Advanced Encryption Standards (AES)
- Example:
  - MD5(There is $1500 in the blue bo) = f80b3fde8ecbac1b515960b9058de7a1
  - MD5(There is $1500 in the blue box) = a4a5471a0e019a4a502134d38fb64729

Hash Message Authentication Code (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

Security Levels

- 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

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

Key Management

Issues

- 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

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
- Makes replay attack less effective
- 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

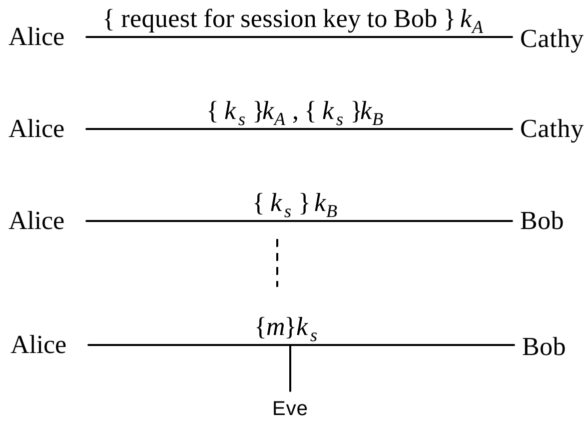
Key Exchange Algorithms

- Goal: Alice, Bob use a shared key to communicate secretely
- 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 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

- 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



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

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

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

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

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

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

Problem with Needham-Schroeder

- 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 —— $\{ \text{Alice} \,||\, k_s \} k_B$ [Replay] ——> Bob

Eve <—— $\{ r_3 \} k_s$ [Eve intercepts] —— Bob

Eve —— $\{ r_3 - 1 \} k_s$ ——> Bob

Solution: Denning-Sacco Modification

- In protocol above, Eve impersonates Alice
- Problem: replay in third step
  - First in previous slide
- Solution: use time stamp *T* to detect replay
  - Needs synchronized clocks
- 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

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

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

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

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

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

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 a particular exchange

The Protocol

Alice $\xrightarrow{n \parallel \text{Alice} \parallel \text{Bob} \parallel \{ r_1 \parallel n \parallel \text{Alice} \parallel \text{Bob} \} k_A}$ Bob

Cathy $\xrightarrow[\{ r_2 \parallel n \parallel \text{Alice} \parallel \text{Bob} \} k_B]{n \parallel \text{Alice} \parallel \text{Bob} \parallel \{ r_1 \parallel n \parallel \text{Alice} \parallel \text{Bob} \} k_A \parallel}$ Bob

Cathy $\xrightarrow{n \parallel \{ r_1 \parallel k_s \} k_A \parallel \{ r_2 \parallel k_s \} k_B}$ Bob

Alice $\xrightarrow{n \parallel \{ r_1 \parallel k_s \} k_A}$ 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
  - $n \parallel \{ r_1 \parallel k_s \} k_A \parallel \{ r_2 \parallel k_s \} k_B$
- 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, *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 (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*

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

user	$user \parallel TGS$	Cathy
Cathy	$\{ k_{u,TGS} \} k_u \parallel T_{u,TGS}$	*user*
user	$service \parallel A_{u,TGS} \parallel T_{u,TGS}$	TGS
user	$user \parallel \{ k_{u,s} \} k_{u,TGS} \parallel T_{u,s}$	TGS
user	$A_{u,s} \parallel T_{u,s}$	*service*
user	$\{ t + 1 \} k_{u,s}$	*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

Alice $\xrightarrow{\{\,\{\,k_s\,\}\,d_A\,\}\,e_B}$ 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)

Man-in-the-Middle Attack

Alice —send me Bob's public key→ | *Eve intercepts request* Cathy

Eve —send me Bob's public key→ Cathy

Eve ←e_B— Cathy

Alice ←e_E— Eve

Alice —$\{ k_s \} e_E$→ | *Eve intercepts message* Bob

Eve —$\{ k_s \} e_B$→ 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 ransom numbers*: a sequence of numbers n_1, n_2, … such that for any integer $k > 0$, an observer cannot predict n_k even if all of n_1, …, n_{k-1} are known
  - Best: physical source of randomness
    - 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 [$n_k = (an_{k-1} + b) \bmod n$] broken (a, b and n are relatively prime)
    - Polynomial congruential generators [$n_k = (a_j n_{k-1}^j + \ldots + a_1 n_{k-1}\, a_0) \bmod n$] 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

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 (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;

Common Error

- Classical: Alice, Bob share key *k*
  - Alice sends $m \parallel \{ m \}k$ 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

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

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

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

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}!