Lecture 8
October 10, 2025

ECS 235A, Computer and Information Security

Administrative Stuff

- Because many people were admitted very recently, I am extending the due date for Homework 1 to Friday, October 10
- I am holding an additional in person office hour from 1:30pm to 2:20pm in my office, room 2209 Watershed Sciences.

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
  - basic unit is the bit
  - performs both substitution and transposition (permutation) on the bits
- Cipher consists of 16 rounds (iterations) each with a 48 bit round key generated from the user-supplied key

Structure of the DES

- Input is first permuted, then split into left half (L) and right half (R), each 32 bits
- Round begins; R and round key run through function *f*, then xor'ed with L
  - *f* expands R to 48 bits, xors with round key, and then each 6 bits of this are run through S-boxes (substitution boxes), each of which gives 4 bits of output
  - Those 32 bits are permuted and this is the output of *f*
- R and L swapped, ending the round
  - Swapping does not occur in the last round
- After last round, L and R combined, permuted, forming DES output

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, where x' is the bitwise complement of x
- is the bitwise complement of x
  - $\text{DES}_k(m) = c \Rightarrow \text{DES}_{k'}(m') = c'$
- S-boxes exhibit irregular properties
  - Distribution of odd, even numbers non-random
  - Outputs of fourth box depends on input to third box

Differential Cryptanalysis

- A chosen ciphertext attack
  - Requires 2^{47} plaintext, ciphertext pairs
- Revealed several properties
  - Small changes in S-boxes reduced the number of pairs needed
  - Making every bit of the round keys independent did 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
  - Requires an initialization vector for the first one
- Encrypt-Decrypt-Encrypt (2 keys: k, k')
  - $c = \mathrm{DES}_k(\mathrm{DES}_{k'}^{-1}(\mathrm{DES}_k(m)))$
- Triple DES(3 keys: k, k', k'')
  - $c = \mathrm{DES}_k(\mathrm{DES}_{k'}(\mathrm{DES}_{k''}(m)))$

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
- DES officially withdrawn in 2005

Advanced Encryption Standard

- Competition announces in 1997 to select successor to DES
  - Successor needed to be available for use without payment (no royalties, etc.)
  - Successor must encipher 128-bit blocks with keys of lengths 128, 192, and 256
- 3 workshops in which proposed successors were presented, analyzed
- Rijndael selected as successor to DES, called the Advanced Encryption Standard (AES)
  - Other finalists were Twofish, Serpent, RC6, MARS

Overview of the AES

- A block cipher:
  - encrypts blocks of 128 bits using a 128, 192, or 256 bit key
  - outputs 128 bits of ciphertext
- A product cipher
  - basic unit is the bit
  - performs both substitution and transposition (permutation) on the bits
- Cipher consists of rounds (iterations) each with a round key generated from the user-supplied key
  - If 128 bit key, then 10 rounds
  - If 192 bit key, then 12 rounds
  - If 256 bit key, then 14 rounds

Structure of the AES: Encryption

- Input placed into a state array, which is then combined with zeroth round key
  - Treat state array as a 4×4 matrix, each entry being a byte
- Round begins; new values substituted for each byte of the state array
- Rows then cyclically shifted
- Each column independently altered
  - Not done in last round
- Result xor'ed with round key
- After last round, state array is the encrypted input

Structure of the AES: Decryption

- Round key schedule reversed
- Input placed into a state array, which is then combined with zeroth round key (of reversed schedule)
- Round begins; rows cyclically shifted, then new values substituted for each byte of the state array
  - Inverse rotation, substitution of encryption
- Result xor'ed with round key (of reversed schedule)
- Each column independently altered
  - Inverse of encryption; this is not done in last round
- After last round, state array is the decrypted input

Analysis of AES

- Designed to withstand attacks that the DES is vulnerable to
- All details of design made public, unlike with the DES
  - In particular, those of the substitutions (S-boxes) were described
- After 2 successive rounds, every bit in the state array depends an every bit in the state array 2 rounds ago
- No weak, semi-weak keys

AES Modes

- DES modes also work with AES
- EDE and “Triple-AES” not used
  - Extended block size makes this unnecessary
- New counter mode CTR added

Public Key Cryptography

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

Requirements

1. It must be computationally easy to encipher or decipher a message given the appropriate key
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

RSA

- First described publicly in 1978
  - Unknown at the time: Clifford Cocks developed a similar cryptosystem in 1973, but it was classified until recently
- Exponentiation cipher
- Relies on the difficulty of determining the number of numbers relatively prime to a large integer *n*

Background

- Totient function ϕ(n)
  - Number of positive 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
- Example: $\phi(21) = 12$
  - 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 are relatively prime to 21

Algorithm

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

Example: Confidentiality

- Take $p = 181$, $q = 1451$, so $n = 262631$ and $\phi(n) = 261000$
- Alice chooses $e = 154993$, making $d = 95857$
- Bob wants to send Alice secret message PUPPIESARESMALL (152015 150804 180017 041812 001111); encipher using public key
  - $152015^{154993} \bmod 262631 = 220160$
  - $150804^{154993} \bmod 262631 = 135824$
  - $180017^{154993} \bmod 262631 = 252355$
  - $041812^{154993} \bmod 262631 = 245799$
  - $001111_{154993} \bmod 262631 = 070707$
- Bob sends 220160 135824 252355 245799 070707
- Alice uses her private key to decipher it

Example: Authentication/Integrity

- Alice wants to send Bob the message PUPPIESARESMALL in such a way that Bob knows it comes from her and nothing was changed during the transmission
  - Same public, private keys as before
- Encipher using private key:
  - $152015^{95857} \bmod 262631 = 072798$
  - $150804^{95857} \bmod 262631 = 259757$
  - $180017^{95857} \bmod 262631 = 256449$
  - $041812^{95857} \bmod 262631 = 089234$
  - $001111^{95857} \bmod 262631 = 037974$
- Alice sends 072798 259757 256449 089234 037974
- Bob receives, uses Alice's public key to decipher it

Example: Both (Sending)

- Same *n* as for Alice; Bob chooses *e* = 45593, making *d* = 235457
- Alice wants to send PUPPIESARESMALL (152015 150804 180017 041812 001111) confidentially and authenticated
- Encipher:
  - $(152015^{95857} \bmod 262631)^{45593} \bmod 262631 = 249123$
  - $(150804^{95857} \bmod 262631)^{45593} \bmod 262631 = 166008$
  - $(180017^{95857} \bmod 262631)^{45593} \bmod 262631 = 146608$
  - $(041812^{95857} \bmod 262631)^{45593} \bmod 262631 = 092311$
  - $(001111^{95857} \bmod 262631)^{45593} \bmod 262631 = 096768$
- So Alice sends 249123 166008 146608 092311 096768

Example: Both (Receiving)

- Bob receives 249123 166008 146608 092311 096768
- Decipher:
  - $(249123^{235457} \bmod 262631)^{154993} \bmod 262631 = 152012$
  - $(166008^{235457} \bmod 262631)^{154993} \bmod 262631 = 150804$
  - $(146608^{235457} \bmod 262631)^{154993} \bmod 262631 = 180017$
  - $(092311^{235457} \bmod 262631)^{154993} \bmod 262631 = 041812$
  - $(096768^{235457} \bmod 262631)^{154993} \bmod 262631 = 001111$
- So Alice sent him 152015 150804 180017 041812 001111
  - Which translates to PUP PIE SAR ESM ALL or PUPPIESARESMALL

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 only characters per block, RSA can be broken using statistical attacks (just like symmetric cryptosystems)
- Attacker cannot alter letters, but can rearrange them and alter message meaning
  - Example: reverse enciphered message of text ON to get NO

Checksums

- Mathematical function to generate a set of *k* bits from a set of *n* bits (where $k \leq n$).
  - *k* is smaller than *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 of Cryptographic Checksum

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 at least 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
  - AES 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
  - SHA-512, SHA-3 are examples; older ones include MD4, MD5, RIPEM, SHA-0, and SHA-1 (methods for constructing collisions are known for these)

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*′ ⊕ *opad* || *h*(*k*′ ⊕ *ipad* || *m*))
  - ⊕ exclusive or, || concatenation

Strength of HMAC-*h*

- Depends on the strength of the hash function *h*
- Attacks on HMAC-MD4, HMAC-MD5, HMAC-SHA-0, and HMAC-SHA-1 recover partial or full keys
  - Note all of MD4, MD5, SHA-0, and SHA-1 have been broken

Digital Signature

- Construct that authenticates origin, contents of message in a manner provable to a disinterested third party (a “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

- Symmetric: Alice, Bob share key *k*
  - Alice sends *m* || { *m* } *k* to Bob
  - { *m* } *k* means *m* enciphered with key *k*, || means concatenation

Claim: This is a digital signature

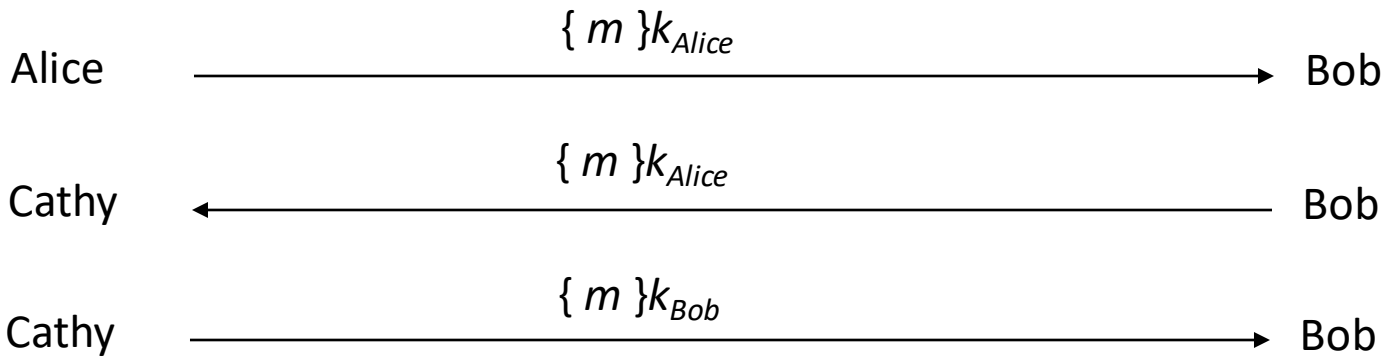
WRONG

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

- Basically, Alice enciphers the message, or its cryptographic hash, with her private key
- In case of dispute or question of origin or whether changes have been made, a judge can use Alice's public key to verify the message came from Alice and has not been changed since being signed

RSA Digital Signatures

- Alice's keys are (e_{Alice},n_{Alice}) (public key), d_{Alice} (private key)
  - In what follows, we use e_{Alice} to represent the public key
- 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
  - Don’t just encipher message and then sign, or vice versa
    - Changing public key and private key can cause problems
    - Messages can be forwarded, so third party cannot tell if original sender sent it to her

Attack #1

- Example: Alice, Bob communicating
  - $n_A = 262631$, $e_A = 154993$, $d_A = 95857$
  - $n_B = 288329$, $e_B = 22579$, $d_B = 138091$
- Alice asks Bob to sign 225536 so she can verify she has the right public key:
  - $c = m^{d_B} \bmod n_B = 225536^{138091} \bmod 288329 = 271316$
- Now she asks Bob to sign the statement AYE (002404):
  - $c = m^{d_B} \bmod n_B = 002404^{138091} \bmod 288329 = 182665$

Attack #1

- Alice computes:
  - new message NAY (130024) by (002404)(225536) mod 288329 = 130024
  - corresponding signature (271316)(182665) mod 288329 = 218646
- Alice now claims Bob signed NAY (130024), and as proof supplies signature 218646
- Judge computes $c^{e_B} \bmod n_B = 218646^{22579} \bmod 288329 = 130024$
  - Signature validated; Bob is toast

Preventing Attack #1

- Do not sign random messages
  - This would prevent Alice from getting the first message
- When signing, always sign the cryptographic hash of a message, not the message itself

Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract LUR (112017)
  - But Bob really wants her to sign contract EWM (042212), but knows she won't
- Alice enciphers, then signs:
  - $(m^{e_B} \bmod n_A)^{d_A} \bmod n_A = (112017^{22579} \bmod 288329)^{95857} \bmod 262631 = 42390$
- Bob now changes his public key
  - Computes r such that $042212^r \bmod 288329 = 112017$; one such $r = 9175$
  - Computes $re_B \bmod \phi(n_B) = (9175)(22579) \bmod 287184 = 102661$
  - Replace public key with (102661,288329), private key with 161245
- Bob claims contract was EWM
- Judge computes:
  - $(42390^{154993} \bmod 262631)^{161245} \bmod 288329 = 042212$, which is EWM
  - Verified; now Alice is toast

Preventing Attack #2

- Obvious thought: instead of encrypting message and then signing it, sign the message and then encrypt it
  - May not work due to surreptitious forwarding attack
  - Idea: Alice sends Cathy an encrypted signed message; Cathy deciphers it, re-enciphers it with Bob's public key, and then sends message and signature to Bob – now Bob thinks the message came from Alice (right) and was intended for him (wrong)
- Several ways to solve this:
  - Put sender and recipient in the message; changing recipient invalidates signature
  - Sign message, encrypt it, then sign the result

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 $p = 262643$, $g = 9563$, $d = 3632$, giving $y = 274598$
- Alice wants to send Bob signed contract PUP (152015)
  - Chooses $k = 601$ (relatively prime to 262642)
  - This gives $a = g^k \bmod p = 9563^{601} \bmod 29 = 202897$
  - Then solving $152015 = (3632 \times 202897 + 601b) \bmod 262642$ gives $b = 225835$
  - Alice sends Bob message $m = 152015$ and signature $(a,b) = (202897, 225835)$
- Bob verifies signature: $g^m \bmod p = 9563^{152015} \bmod 262643 = 157499$ and $y^a a^b \bmod p = 27459^{202897} 202897^{225835} \bmod 262643 = 157499$
  - 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 $k = 5$

  $$m = (da + kb) \bmod p-1 \Rightarrow 152015 = (202897d + 601 \times 225835) \bmod 262642$$

  giving Alice's private key $d = 3632$

Notation

- $X \rightarrow Y : \{ Z \mid\mid W \} k_{X,Y}$
  - 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
- $A \rightarrow T : \{ Z \} k_A \mid\mid \{ W \} k_{A,T}$
  - 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)

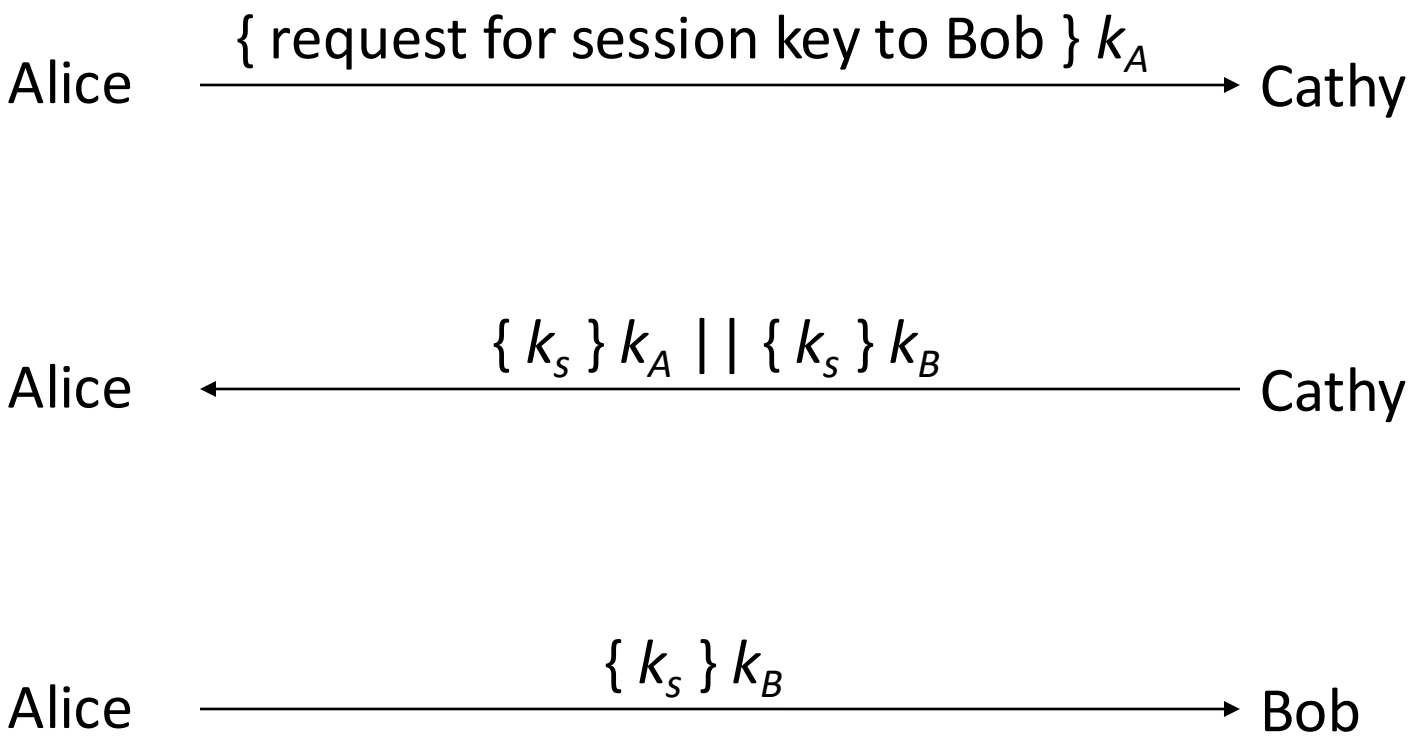
Key Exchange Algorithms

- 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

Symmetric 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



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

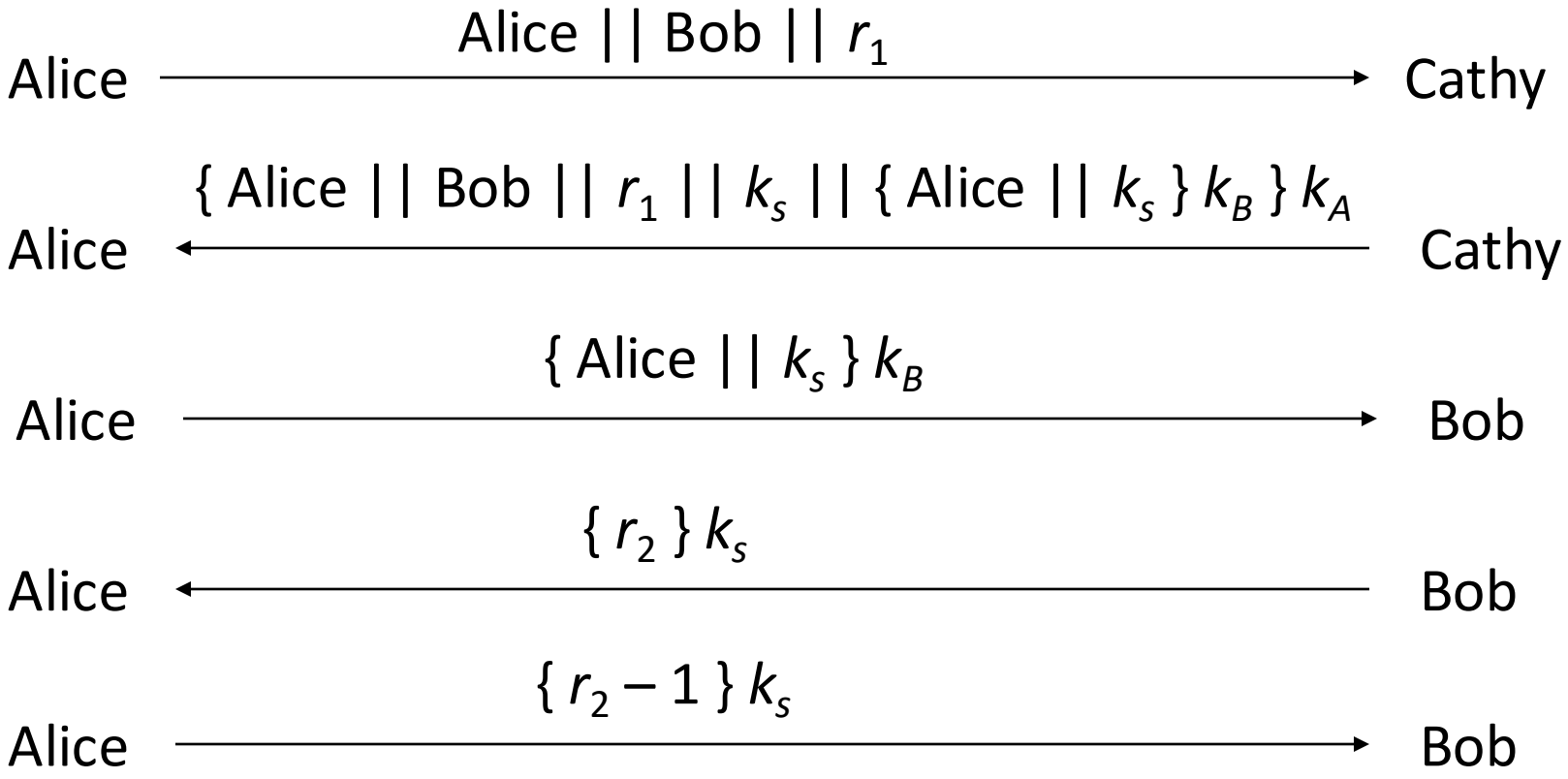
Session, Interchange Keys

- 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

Needham-Schroeder



Argument: Alice talking to Bob

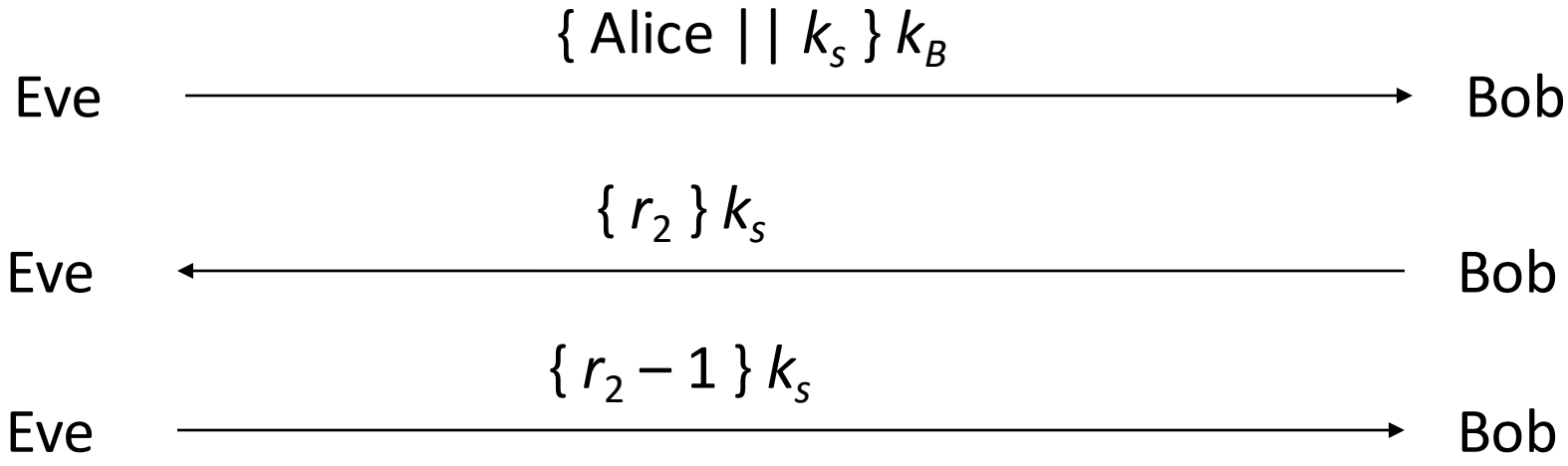
- Second message
  - Enciphered using key only she, Cathy knows
    - So Cathy enciphered it
  - Response to first message
    - As r_1 in it matches r_1 in first message
- Third message
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Any messages enciphered 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

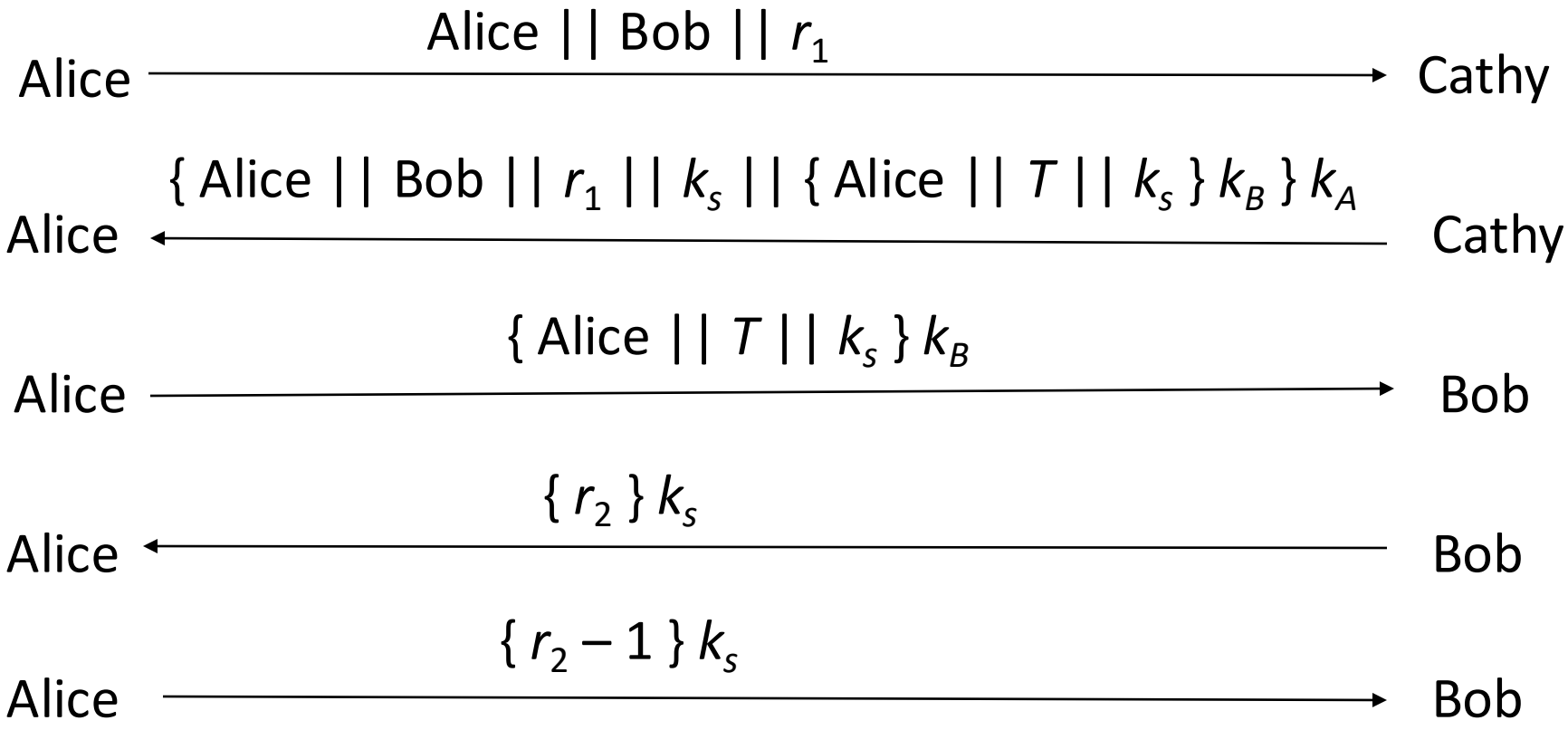
- Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
  - In what follows, Eve knows k_s



Problem and 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



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