Contents

- Introduction
- Symmetric-key cryptography
  - Block ciphers
  - Symmetric-key algorithms
  - Cipher block modes
  - Stream cipher
- Public-key cryptography
  - RSA
  - Diffie-Hellman
  - ECC
  - Digital signature
  - Public key Infrastructure
- Cryptographic hash function
  - Attack complexity
  - Hash Function algorithm
- Integrity and Authentication
  - Message authentication code
  - Authentication encryption
  - Digital signature
- Symmetric Key establishment
  - Public-key based
  - Key agreement (Diffie-Hellman)
  - server-based

Key Establishment

Key establishment

- establishing symmetric key
  - How are the secret keys in the symmetric key encryption distributed and managed?
- distributing public key
  - When a public key is known in the public domain, how can I trust that the key is really his or her public key to be claimed?
  - For this topic, we already discuss how public keys are distributed in a trusted way in real world.

Symmetric key establishment

- Key transportation
  - Using public key encryption
- Key agreement
  - Diffie-Hellman
- Key establishment using symmetric encryption
  - Based on KDC

Using public key encryption

- We already learned that one of the public key applications is to use for establishing symmetric keys.
- Drawback
  - Must trust the public key.
  - To do that, we need PKI.

Symmetric key transportation using RSA

Alice

Bob

K_{+A}, K_{-A}

K_{+A}

K_{+A}

generate session(sym) key: K_{AB}

Message: x

encrypt message: $c=E_{K_{AB}}(x)$

encrypt sym key: $ckey=E_{K\text{-}A}(K_{AB})$

(c, ckey)

decrypt sym key

$K_{AB} = D_{K+A}(ckey))$

decrypt message

$x = D_{K_{AB}}(c)$

Session key

- Session key is an ephemeral key to be used for encrypting messages belonging to one session.
- A session key is generated and used during a session. After that, it is thrown away.
- So, a user has a master key which is used permanently until it is updated, and a session key for encryption for temporary use.
- Why do they need session keys, instead of one key?
- How can they have mater keys?

Perfect Forward Secrecy

- Consider this “issue”
  - Alice encrypts message with shared key K and sends ciphertext to Bob
  - Trudy records ciphertext and later attacks Alice’s (or Bob’s) computer to recover K
  - Then Trudy decrypts recorded messages
- **Perfect forward secrecy (PFS):**
  - Even if Trudy gets key K or other secret(s) later, he should not decrypt all past communicated messages.
- Is PFS possible?

Perfect Forward Secrecy

- Suppose Alice and Bob share a key K
- For perfect forward secrecy, Alice and Bob don't use K to encrypt.
- Instead they must use a session key K_S and forget it after it's used.
- Is a session key K_S enough to ensure PFS?

Key agreement

- Use Diffie-Hellman(D-H) or EC-DH algorithm for Alice and Bob to share a secret key.
- D-H key agreement
  - Alice and Bob choose *p*, a large prime numbers p and g, a generator *g* of order p-1, letting them known in public.
  - Then do the procedures in the following slide.
  - The final result, $g^{ab} \bmod p$, can be used directly as a sym key or as secret information to compute a sym key.
  - They destroy a and b after computing a sym key. So, guarantee "Perfect Forward Securecy (PFS)."

D-H key exchange

Alice | **Bob**

p, g : public

---

choose $a \in \{2,3,\ldots,p-2\}$
compute $A = g^a \bmod p$

A →

choose $b \in \{2,3,\ldots,p-2\}$
compute $B = g^b \bmod p$

← B

$K_{AB} = B^a \bmod p = g^{ab} \bmod p$

$K_{AB} = A^b \bmod p = g^{ab} \bmod p$

Message x
Encrypt: $Y = E_{KAB}(x)$

y →

Decrypt: $x = D_{KAB}(y)$

Security of D-H key agreement

- We already discussed the security of D-H algorithm.
  - It depends on the parameters, especially the size of p.
- Aside from the algorithm attack, D-H key agreement protocol is subject to the man-in-the-middle attack.

Man-in-the-middle(MIM) attack

Alice | **Curry** | **Bob**

choose a
compute $A = g^a \bmod p$

A →

choose c
compute $C = g^c \bmod p$

← C   C →

choose b
compute $B = g^b \bmod p$

← B

$K_{AT} = C^a \bmod p = g^{ac} \bmod p$

$K_{AC} = A^C \bmod p = g^{ac} \bmod p$

$K_{BC} = C^C \bmod p = g^{bc} \bmod p$

$K_{BC} = C^b \bmod p = g^{bc} \bmod p$

How to prevent MIM attack

- Encrypt DH exchange with symmetric key
  - Sound like silly answer
- Encrypt DH exchange with public key
- Sign DH values with private key(digital signature)
- Any other?

Alice

Bob

p, g : public

choose $a \in \{2,3,\ldots,p-2\}$
compute $A = g^a \bmod p$

A →

choose $b \in \{2,3,\ldots,p-2\}$
compute $B = g^b \bmod p$

$K_{AB} = A^b \bmod p = g^{ab} \bmod p$

← B, Bob's certificate, $Sign_{K\text{-}B}(Alice|A|B)$

$K_{AB} = B^a \bmod p = g^{ab} \bmod p$
$Verify_{K+B}(Alice|A|B)$

Alice's certificate, $Sign_{K\text{-}A}(Bob|A|B)$ →

$Verify_{K+A}(Bob|A|B)$

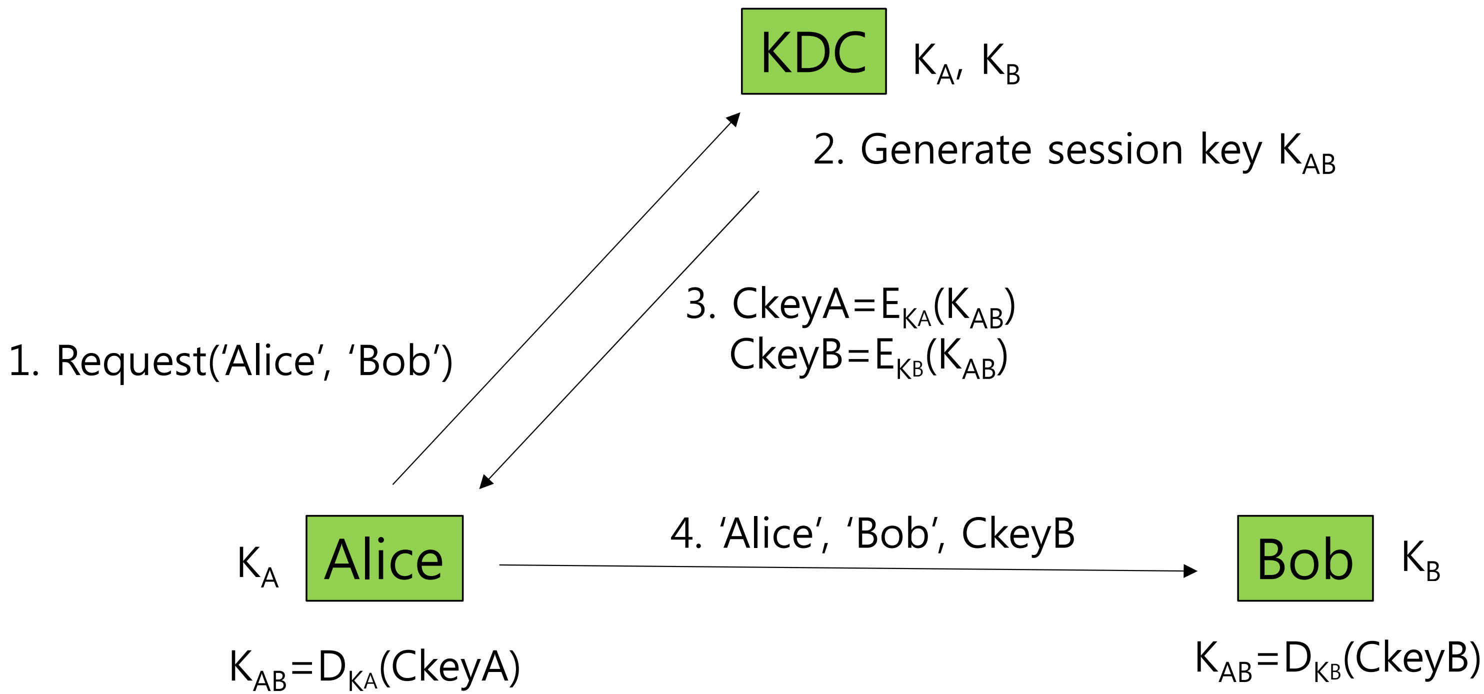
Remark:

- After all, in order to establish symmetric keys, we need public keys, which also bring about secure distribution of public keys.
- Then, the question is how we can establish symmetric keys without resort to public keys.

Key establishment using symmetric key

- Decentralized scheme
  - Establish key pairs between all users at initialization time
  - Drawback:
    - Large number of keys: keys pairs = n(n-1)/2
    - Adding new users is complex
- Centralized scheme
  - A central trusted authority(or authorities) that shares a key with every user distributes a key pair when requested.
  - A central trusted authority is often called a key distribution center(KDC).

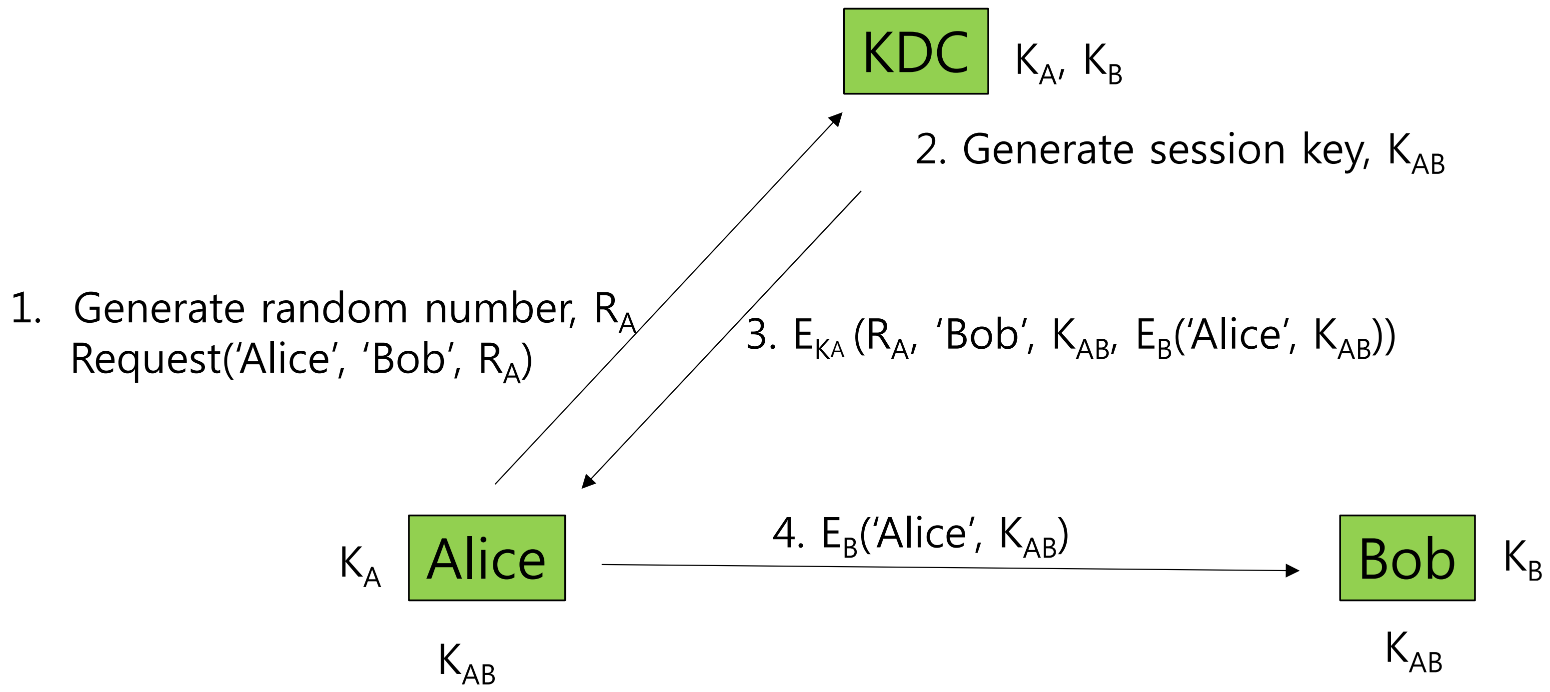
simple key establishment using KDC



simple key establishment using KDC

- The keys, K_A and K_B are pre-installed at KDC and users.
- # of keys
  - When n users, there are n keys.
- Adding a new user only requires secure channel between KDC and a new user at setup time.
- Drawbacks
  - KDC is a single point of failure.
  - No perfect forward secrecy
  - Replay attack

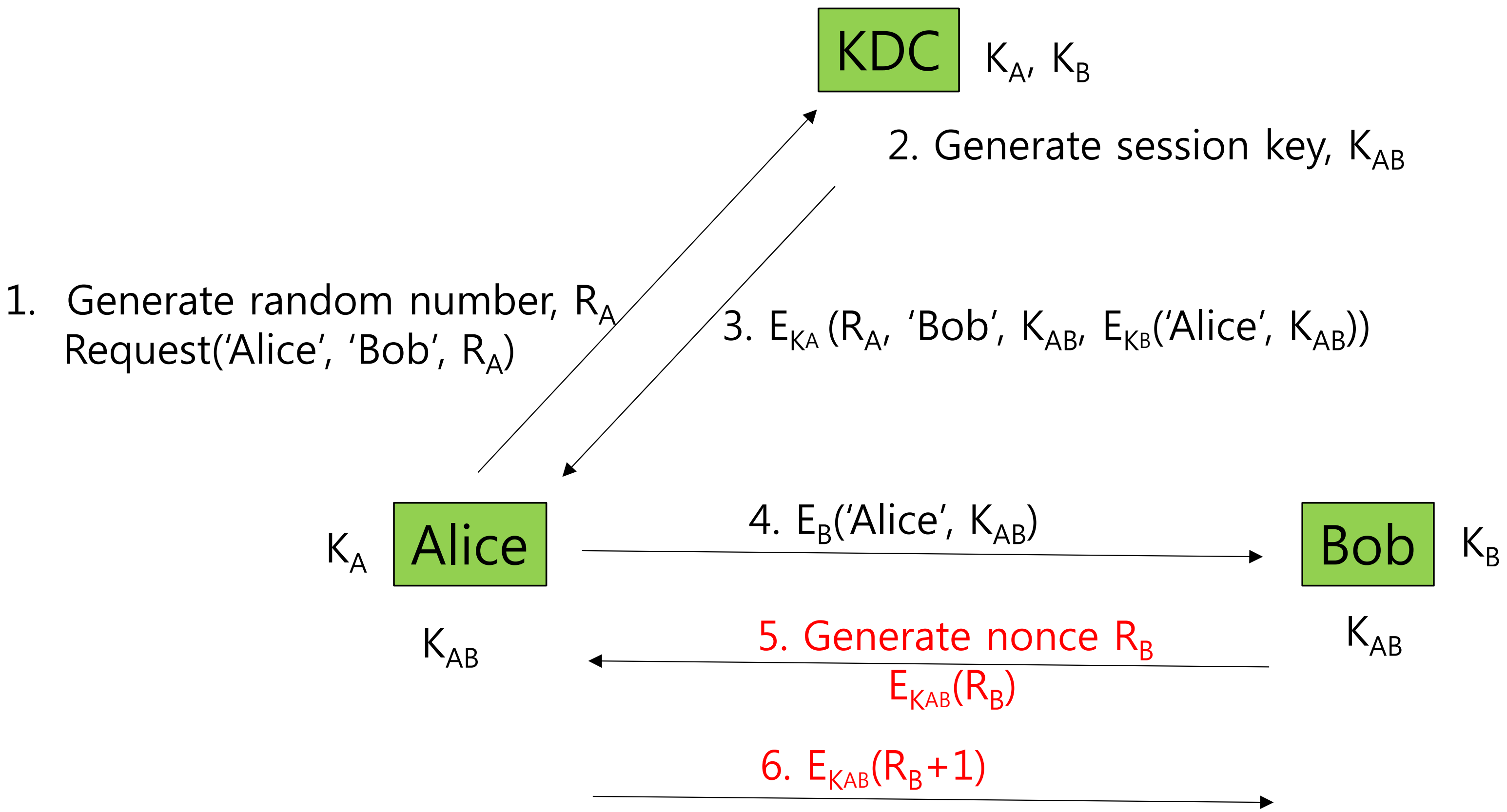
Elaborated establishment using KDC



Key establishment + mutual authentication

- In the protocol of previous slide, nonce(one time random number) is used to prevent replay attack.
- What about PFS?
- When Bob receives the message, he can be assured the other party is really Alice if he trusts KDC.
- But Bob doesn’t authenticate himself to Alice.
- How can they mutually authenticate themselves?
  - Challenge-response scheme can be used for this purpose.

+ mutual authentication



Remarks:

- Session key, K_{AB}, can make them authenticate themselves to the other party.
- Nonce R_B is used for preventing replay attack.
- Why $E_{K_{AB}}(R_B+1)$?
  - Someone can reuse $E_{K_{AB}}(R_B)$.
- Timestamp often replaces nonce.
  - But when using timestamp, the clocks at both users must be synchronized within permissible time difference.
- Kerberos is slightly complex version of this protocol.

Kerberos KDC

- Kerberos **Key Distribution Center** or **KDC**
  - KDC acts as the TTP
  - TTP is trusted, so it must not be compromised
- KDC shares symmetric key K_A with Alice, key K_B with Bob, key K_C with Carol, etc.
- And a master key K_{KDC} known ***only*** to KDC
- KDC enables authentication as well as establish session keys
  - Session key for confidentiality and integrity

Kerberos Tickets

- KDC issue **tickets** containing info needed to access network resources
- KDC also issues **Ticket-Granting Tickets** (**TGTs)** that are used to obtain tickets
- Each TGT contains
  - Session key
  - User's ID
  - Expiration time
- Every TGT is encrypted with K_{KDC}
  - So, TGT can only be read by the KDC

Kerberized Login

- Alice enters her password
- Then Alice's computer does following:
  - Derives K_A from Alice's password
  - Uses K_A to get TGT for Alice from KDC
- Alice then uses her TGT (credentials) to securely access network resources
- **Plus:** Security is transparent to Alice
- **Minus:** KDC ***must*** be secure — it's trusted!

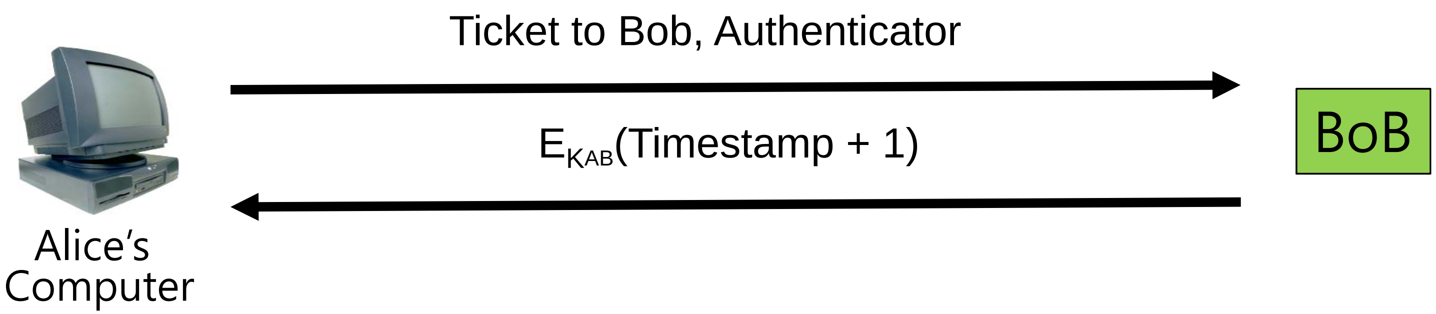
Kerberized Login

- Key K_A = h(Alice's password)
- KDC generates a session key S_A
- Alice's computer decrypts S_A and TGT
  - Then it forgets K_A
- TGT = $E_{K_{KDC}}$("Alice", S_A)

Alice Requests “Ticket to Bob”

- REQUEST = (TGT, Authenticator)
  - authenticator = E_{S_A}(Timestamp)
- REPLY = E_{S_A} (“Bob”, K_{AB}, Ticket to Bob)
  - Ticket to Bob = E_{K_B}(“Alice”, K_{AB})
- KDC gets S_A from TGT to verify timestamp

Alice Uses Ticket to Bob



- Ticket to Bob = E_{K_B}("Alice", K_{AB})
- Authenticator = $E_{K_{AB}}$(Timestamp)
- Bob decrypts "Ticket to Bob" to get K_{AB} which he then uses to verify timestamp

Remark:

- Key S_A used in authentication for Alice
- Timestamps for replay protection
  - Reduce the number of messages—like a nonce that is known in advance
  - But, “time” is a security-critical parameter
- KDC could have remembered session key instead of putting it in a TGT
  - Then no need for TGT
  - But **stateless** KDC is major feature of Kerberos

Key management

- In Kerberos, K_A = h(Alice's password)
- Could instead generate random K_A
  - Compute K_h = h(Alice's password)
  - And Alice's computer stores $E_{K_h}(K_A)$
- Then K_A need not be changed when Alice changes her password
  - But $E_{K_h}(K_A)$ must be stored on computer
- This alternative approach is often used
  - But not in Kerberos

Kerberos Questions

- When Alice logs in, KDC sends $E_{K_A}(S_A, TGT)$ where TGT = E("Alice", S_A, K_{KDC})

  Q: Why is TGT encrypted with K_A?

  A: Extra work for no added security!

- In Alice's "Kerberized" login to Bob, can Alice authenticate herself?
- Why is "ticket to Bob" sent to Alice?
  - Why doesn't KDC send it directly to Bob?

Key Wrap Algorithm

Key wrap algorithm

- Key wrap algorithm
  - Even when a user encrypts message by using symmetric key algorithm, he has two keys; one is called key encryption key(KEK) which is used for encrypting the content encryption key(CEK) which is use for encrypting message.
  - And then send encrypted CEK and encrypted message.
- Types of key wrap algorithms
  - AESKW(AES key wrapping algorithm)
  - TDKW (TDES key wrapping algorithm)
  - AKW1
  - AKW2

Simplified AESKW

Alice

KEK_{AB}

generate CEK_{AB}
encrypt CEK_{AB} : $Ckey=E_{KEK_{AB}}(CEK_{AB})$
Message: x
encrypt message: $c=E_{CEK_{AB}}(x)$

Bob

KEK_{AB}

(Ckey, c) →

decrypt Ckey : $CEK_{AB}=E_{KEK_{AB}}(Ckey)$
decrypt message: $x=E_{CEK_{AB}}(c)$

Purpose of key wrapping

- For more security?
  - In my opinion, there is no point of key wrapping for providing more security.
  - If KEK is revealed, so is the message.
- But there is one advantage:
  - Suppose Bob maintains encrypted data communicated up to now.
  - Even if KEK is revealed, he doesn't need to change the CEK.
  - Instead, Alice re-encrypts the same CEK with new KEK and sends the newly encrypted CEK to Bob.

Random Number Generation (RNG)

Types of RNG

- True RNG
  - Random numbers are generated from physical process in real life.
    - Eg, coin flipping, lottery, thermal noise, mouse movement, etc.
- Pseudo RNG
  - Random numbers are computed, i.e. they are deterministic.
  - Typical algorithm for computing PRNG
    - S_0=seed, $S_{i+1} = F(S_i)$
  - Eg, RAND() function in ANSI C
    - S_0=12345, $S_{i+1} = 1103515245\ S_i + 12345 \pmod{2^{31}}$

Types of RNG

- Cryptography PRNG (CPRNG)
  - CPRNGs are PRNG with one additional property; generated numbers are unpredictable.
  - Given n output bits

    $S_i, S_{i+1}, \ldots, S_{i+n-1}$

    It is computationally infeasible to generate S_n.