

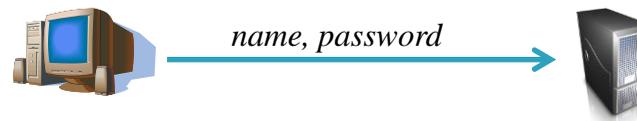
Remote Authentication and Key Establishment

Content

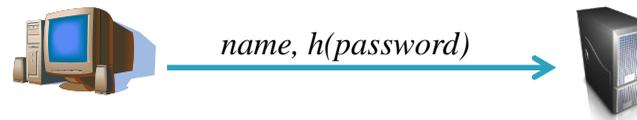
- ▶ Remote authentication
- ▶ Key establishment (and authentication)
- ▶ We look at two main key establishment problems:
 - A and B share a long term key and want to negotiate a session key.
 - A wants to have a shared key with B. Both trust a third party C.

Remote Authentication

- ▶ Authentication over a network
- ▶ Trivial variant: Send name and password just as in OS login
 - ▶ Used by Basic Access Authentication in HTTP



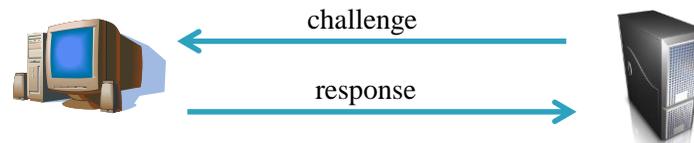
- ▶ Variant: Send name and the hash of the password



- ▶ **Replay attack:** Resending an eavesdropped hash will authenticate anyone with the hash
- ▶ Do the two methods differ in security in any way?

Avoid Sending Password

- ▶ Challenge response protocol
 - Server sends challenge, client sends response
 - Response depends on challenge



- ▶ **Example 1:** Encrypt challenge using (hash of) password as key
 - NTLM uses block cipher DES
- ▶ **Example 2:** Use a hash function including both challenge and password
 - Digest Access Authentication in HTTP uses a variant of this
- ▶ **Replay attack:** If same challenge is used twice, an attacker can replay an eavesdropped response to get authenticated
 - Solution 1: challenge is a "number used once", a *nonce*
 - Solution 2: (part of) challenge is a time stamp
- ▶ More details in the course "Web Security"

Key Establishment and Authentication

Different keys

- ▶ **Long term keys (Permanent key)** – Rarely or never changed. Use sparingly.
- ▶ **Session keys** – Often changed. If lost or broken, only current session is affected.
 - Each key is used to encrypt a limited amount of data
 - Asymmetric long term keys can be used to negotiate symmetric keys.

Slow encryption → fast encryption

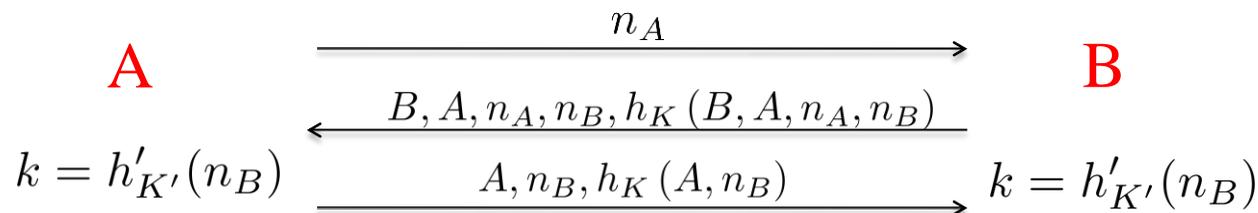
- Key is not valid for a long time → **key freshness**
- ▶ Common to separate keys depending on application
 - Symmetric: One for encryption, one for message authentication
 - Asymmetric: Different key pairs for encryption and digital signatures
- ▶ We want to know *who* we are establishing keys with so authentication is included
 - Mutual vs. Unilateral authentication

Key Establishment

- ▶ Key Establishment divided into
 - **Key Transport** – one party creates/obtains secret key and securely transfers it to the other party (also called key distribution)
 - **Key Agreement** – Both parties contribute to the generation of the secret key
- ▶ Other terms
 - **(Implicit) Key Authentication** – One party knows that no one besides a specifically identified second party may gain access to a secret key
 - **Key Confirmation** – One party is assured that the second party has possession of a secret key (but identity of the other party may not be known)
 - **Explicit Key Authentication** – Both implicit key authentication and key confirmation

Authenticated Key Exchange Protocol 2

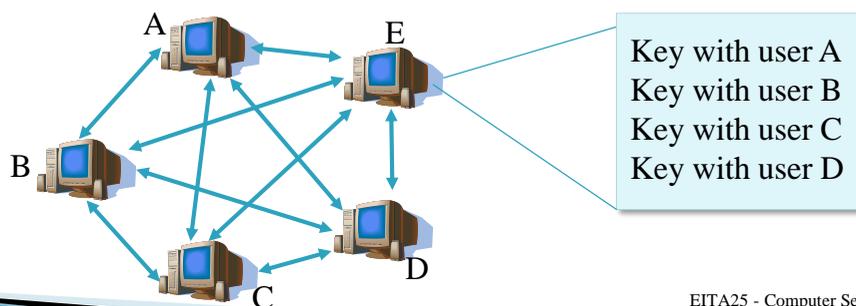
- ▶ Bellare and Rogaway, 1994
- ▶ No trusted third party involved
- ▶ A and B shares two common *symmetric* keys, K and K' and wish to negotiate a session key.
- ▶ h and h' are keyed hash functions (MACs), n is a nonce (number used once)



Protocol provides (implicit) key authentication and mutual entity authentication

Pre-shared Keys

- ▶ Consider a system of n users, everyone having pre-shared key with each other
- ▶ There are $n(n-1)/2$ different keys
- ▶ Some problems:
 - Each user needs to securely store $n-1$ keys
 - Distribution of pre-shared keys require distribution of about n^2 keys
 - Must be done using a secure channel



Without Pre-Shared Secret

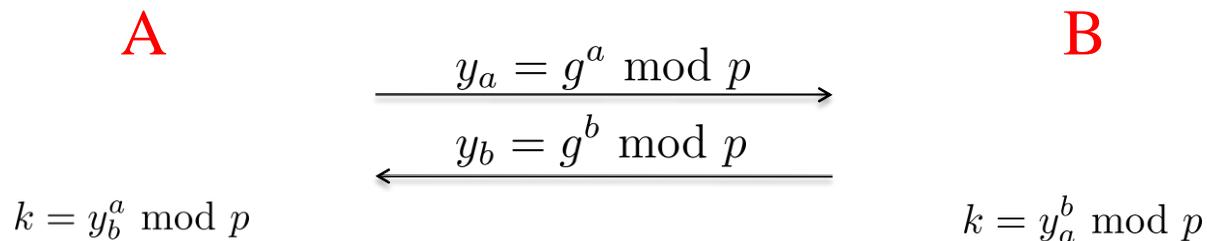
Can two parties agree on a key without having a previously established secret?

Assume anyone can eavesdrop on the communication when they agree on the key!



Diffie-Hellman Protocol

- ▶ Diffie and Hellman
- ▶ Key agreement protocol
- ▶ A and B do not share any secret (long term key) in advance
- ▶ p is a large prime, g is element of large order in multiplicative group mod p .

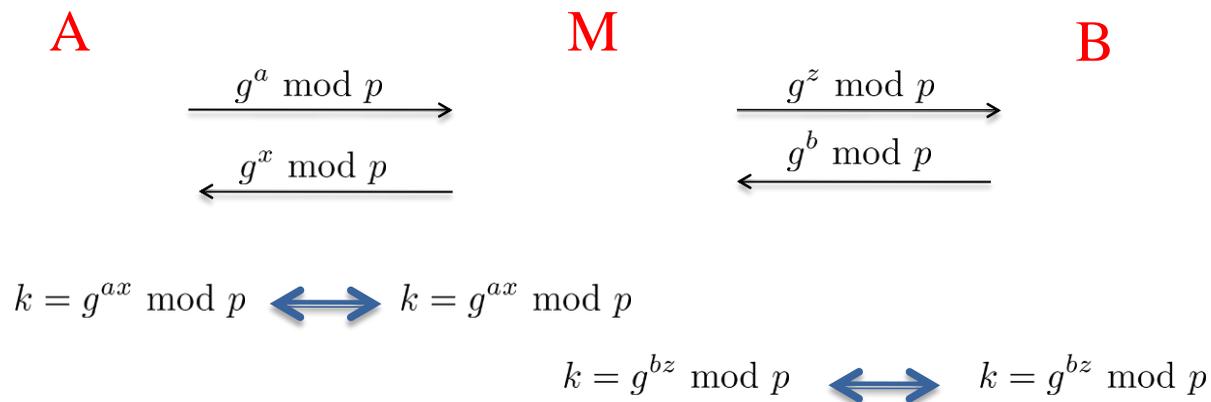


Based on the DLP problem (discrete logarithm problem)

This works against eavesdroppers, but what about active attackers?

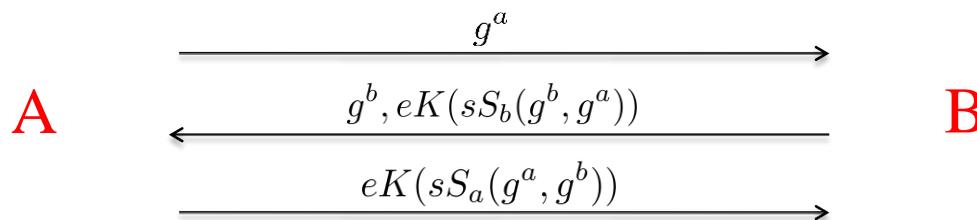
Problem with Diffie-Hellman

- ▶ No key authentication – no party knows with whom they share the secret
- ▶ **Man-in-the-middle attack**



Station-to-Station (STS) Protocol

- ▶ Authentication added to Diffie-Hellman
- ▶ S_x is x's signature key and sS_x is the signature produced by S_x .



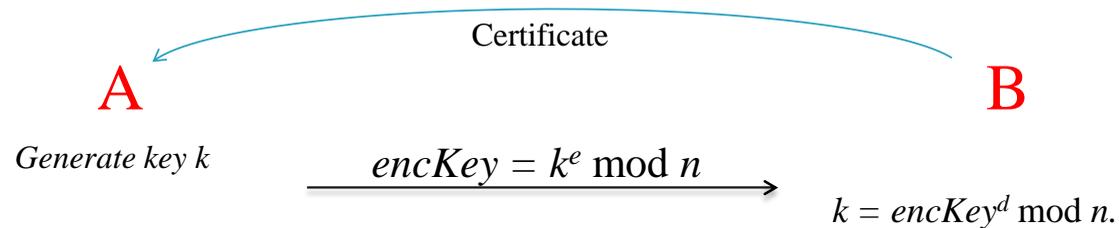
As before, $K = g^{ab} \text{ mod } p$

Provides *mutual entity authentication* and *explicit key authentication*

A PKI (Public Key Infrastructure) is needed

Agree on a Key, Another Variant

- ▶ Encrypt a key using receiver's public key (and e.g., RSA)



Why do we encrypt keys? We could just encrypt data using recipients public key.

1. A may not have a certificate
2. Asymmetric encryption is very slow

Again, a PKI is needed!

Which One is Best?

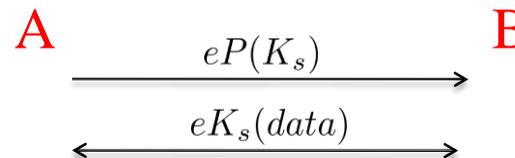
- ▶ Diffie-Hellman with PKI or RSA with PKI?
- ▶ Answer: Diffie-Hellman!
- ▶ **Perfect Forward Secrecy:** If a long-term key is stolen or compromised, previous session keys are not compromised!
 - Diffie-Hellman with signed messages: Nothing is encrypted → PFS
 - Session key encryption with public key: Session key can be decrypted and eavesdropped traffic can be decrypted → No PFS

Password-based Protocols

- ▶ Long-term keys need to be stored on clients
- ▶ A password can represent a key
- ▶ Convenient for human interaction – Easier to remember a password
- ▶ P is password, eP is encryption with password (mapped to encryption key), K_s is session key, eK_s is encryption with session key

Problem: Offline dictionary attacks or brute force attacks on password using data redundancy possible.
Passwords are often badly chosen

Simple protocol:

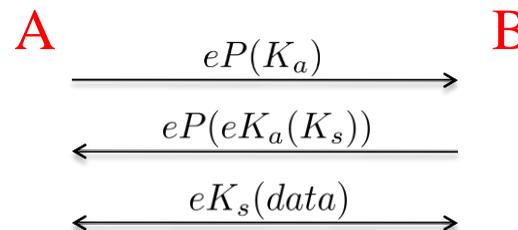


Password-based Protocols

- ▶ Encrypted Key Exchange (EKE) (*Bellare and Merritt 1992*)
- ▶ Use a temporary public key K_a encrypted with password to encrypt session key

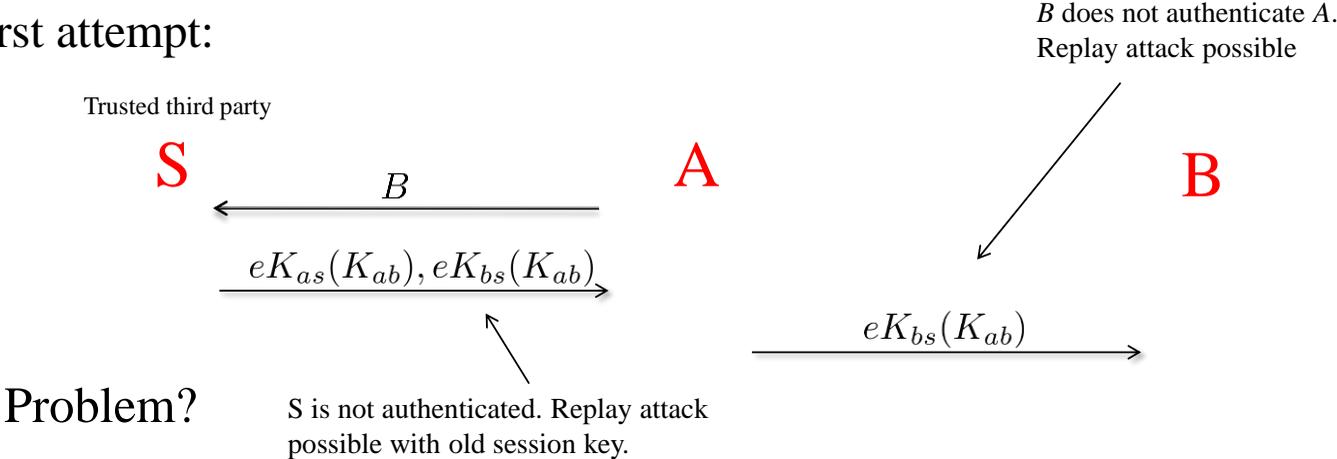
Eavesdropper can see $eP(K_a)$
and $eP(eK_a(K_s))$

Guess P' gives K'_a and $eK'_a(K'_s)$, now either
1. Brute force K_s and check if $eK'_a(K'_s) = eK_a(K_s)'$
OR
2. Find private key corresponding to K'_a



Using a Trusted Third Party

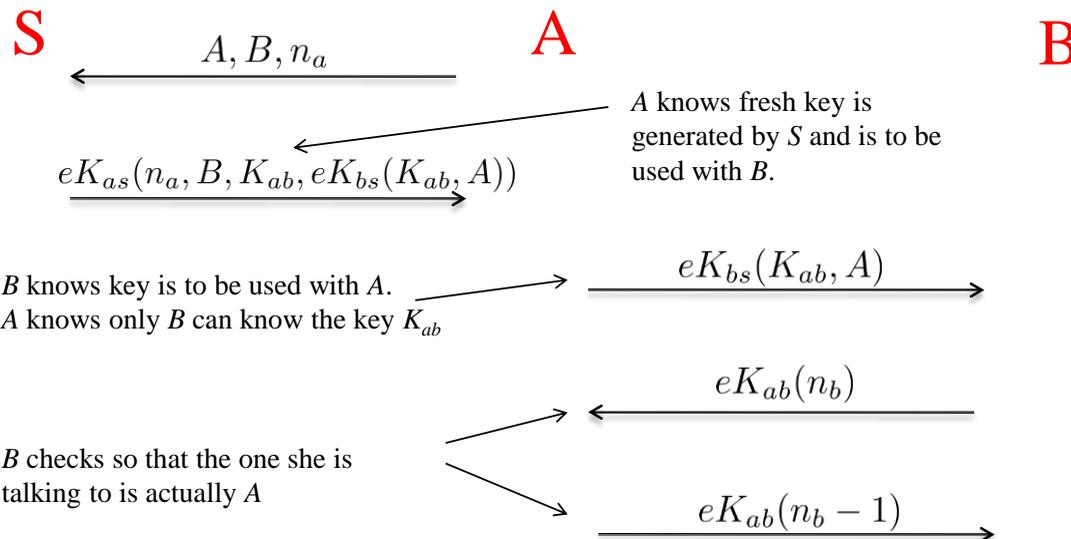
- ▶ *A* and *B* each share a secret key with server *S*.
 - K_{as} : secret key shared between *A* and *S* (long term)
 - K_{bs} : secret key shared between *B* and *S* (long term)
- ▶ **Goal:** Obtain, from *S*, secret key shared between *A* and *B*
 - K_{ab} : session key created by *S*, for use between *A* and *B*
- ▶ **First attempt:**



Needham-Schroeder Protocol

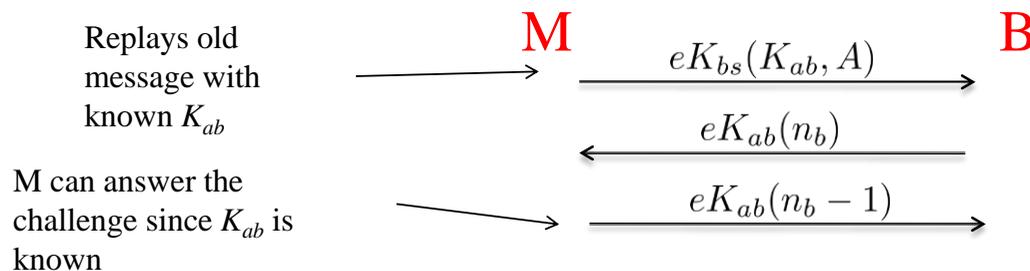
- ▶ Key transport protocol, 1978
- ▶ n_a, n_b : Nonces generated by A and B . Used to prevent replay attacks

Trusted third party



Problem with Needham-Schroeder

- ▶ B does not know if K_{ab} is fresh or not!
- ▶ What if we can break one session key?
- ▶ Then replay attack is possible (Denning – Sacco 1981)
- ▶ Assume adversary M breaks K_{ab} , and enter protocol at message 3

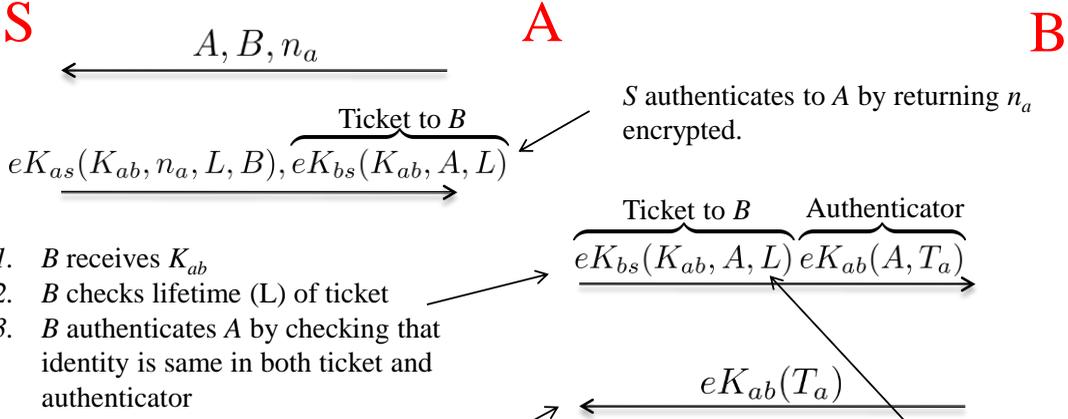


Solution: Include lifetimes for session keys

Kerberos

- ▶ Basically Needham-Schroeder with timestamps and limited lifetimes for session keys

Core protocol:



1. *B* receives K_{ab}
2. *B* checks lifetime (*L*) of ticket
3. *B* authenticates *A* by checking that identity is same in both ticket and authenticator

B authenticates himself to *A*.

Lifetime will prevent replay of broken K_{ab}

Kerberos

- ▶ A Kerberos Authentication Server (KAS) is used together with one or several Ticket Granting Servers TGS.
- ▶ A principal is a user or a server.
- ▶ KAS authenticates principals at login and issues Ticket Granting Tickets (TGTs), which enable principals to obtain other tickets from TGSs.
- ▶ TGSs issues tickets that give principals access to network services demanding authentication.
- ▶ Kerberos 4 uses DES as symmetric cipher, Kerberos 5 can use other algorithms
- ▶ Users authenticate using passwords

Kerberos

1: A, TGS, n_a

2: $eK_{as}(K_{a,tgs}, n_a, L_1, TGS), \overbrace{eK_{tgs}(K_{a,tgs}, A, L_1)}^{TGT}$

3: $\overbrace{eK_{tgs}(K_{a,tgs}, A, L_1)}^{TGT}, \overbrace{eK_{a,tgs}(A, T_a)}^{Authenticator}, B, n'_a$

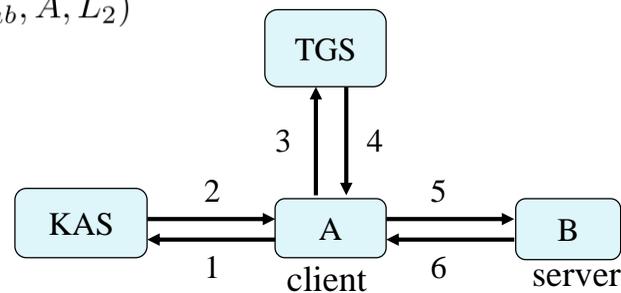
4: $eK_{a,tgs}(K_{ab}, n'_a, L_2, B), \overbrace{eK_{bs}(K_{ab}, A, L_2)}^{Ticket_B}$

5: $\overbrace{eK_{bs}(K_{ab}, A, L_2)}^{Ticket_B}, \overbrace{eK_{ab}(A, T'_a)}^{Authenticator}$

6: $eK_{ab}(T'_a)$

KAS and TGS knows each other's keys

A cache protects authenticators to be used twice



Kerberos

- ▶ Revocation – access rights are revoked by updating KAS, TGS databases. However, issued tickets are valid until they expire.
- ▶ A realm has a KAS, one or more TGSs and a set of servers. It is possible to get tickets for other realms. KAS_x and KAS_y must share keys.
- ▶ Limitations of Kerberos:
 - synchronous clocks.
 - servers must be on-line, trust in servers.
 - password attacks still possible, implementation errors.
- ▶ Secure protocol is not enough, implementation also has to be secure