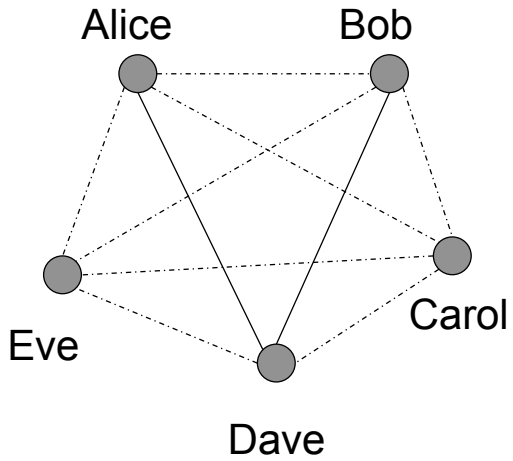
Elements of Applied Cryptography

Key establishment

- Trusted third party: KDC, KTC
- Diffie-Helmann protocol
- The man-in-the-middle attack

Point-to-point key establishment



- Each pair of users must share an *a priori*, long-term secret key
- Each user has $(n-1)$ keys
- The overall number of keys is

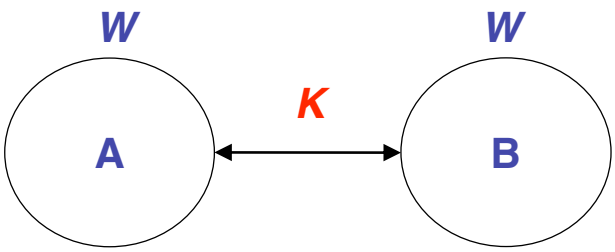
$$\frac{n \times (n-1)}{2} \cong \frac{n^2}{2} \quad \text{or}$$

$$O(n^2)$$

Point-to-point key establishment

- Pros
  - If a subject is compromised only its communications are compromised; communications between two other subjects are not compromised
- Cons
  - Poor scalability: the number of keys is quadratic in the number of subjects
  - Poor scalability: a new member's joining and a member's leaving affect all current members

Establish a session/ephemeral key



- Parties know each other
  e.g., a client *A* has an account on server *B*
- *A* and *B* *a priori* share a long term ***key W***
- *A* and *B* wants to establish a ***session key K***

- Session key is used for a communication session
- Session key is used for bulk encryption
- Long term key is used for key establishment

Establish a session/ephemeral key

one-pass

$M1 \quad A \rightarrow B: \quad E_W \quad t_A, B, K$

- t_A is a timestamp (a "fresh" quantity) requires synchronized clocks

with challenge-response

$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W \quad n_B, B, K$

- n_B is a nonce (a "fresh" quantity)

both parties contribute to the session key

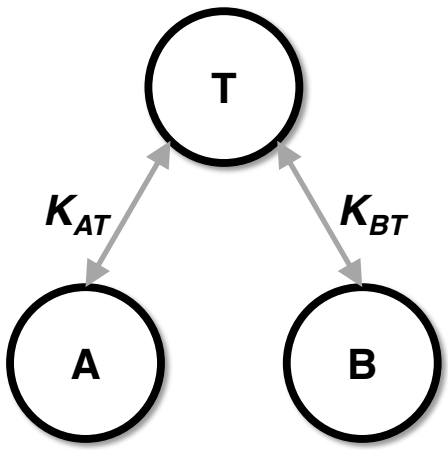
$M1 \quad A \leftarrow B: \quad n_B$

$M2 \quad A \rightarrow B: \quad E_W \quad K_A, n_B, n_A, B$

$M3 \quad A \leftarrow B \quad E_W \quad K_B, n_A, n_B, A$

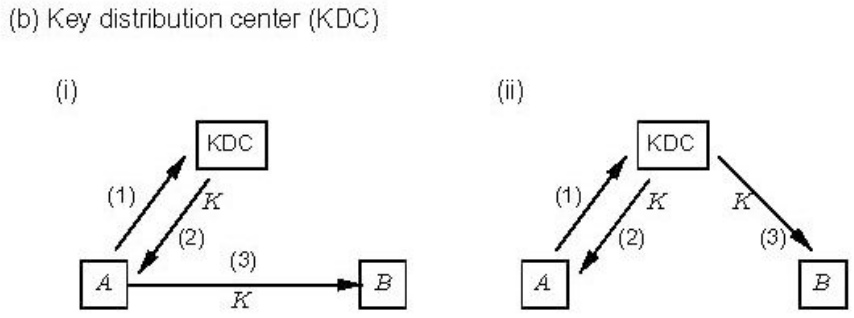
- n_A and n_B are nonces
- K_A and K_B are keying materiale
- $K = f(K_A, K_B)$

Key distribution with Trusted Third Party



- ***T*** allows pair of users to establish a session key
- Each user shares a long-term, a priori key with ***T***
- The overall number of long-term keys is ***O(n)***
- ***T*** is a trusted third-party
  - Maintain a database $<U, K_{TU}>$
  - Guarantee integrity and secrecy of the database
  - Correctly play the key distribution protocol

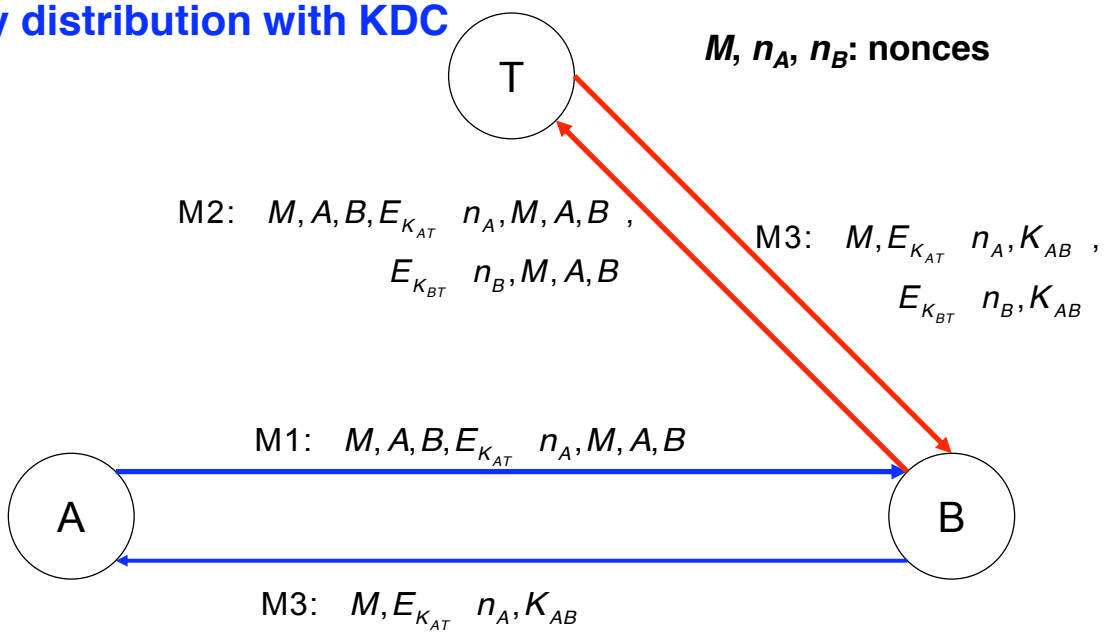
Key Distribution Center



- *A* and *B* share distinct secret keys, K_{AT} and K_{BT}, with KDC
- *KDC* generates the *session key K* and distributes it to *A* and *B*
- *KDC* is trusted to correctly generate the key

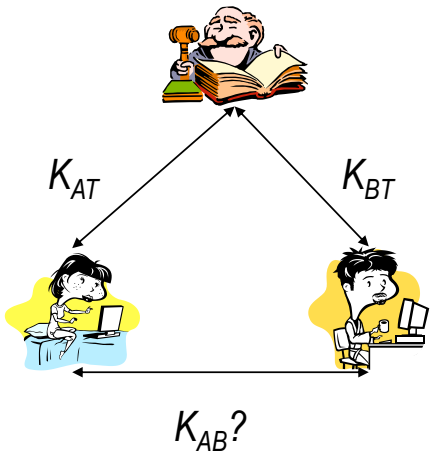
The Otway-Rees protocol (1987)

Key distribution with KDC



Trusted Third Party

***Kerberos* (*Unix*, *Active directory*)**

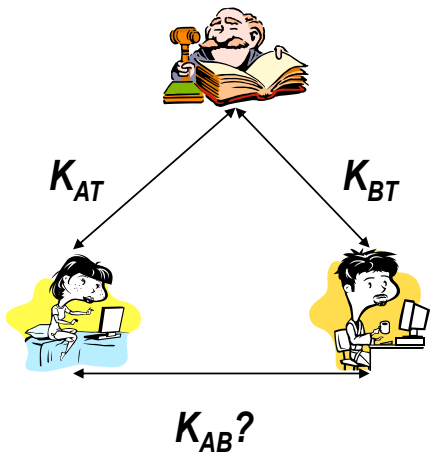


K_{AT}: *shared key between* Trent and Alice

K_{BT}: *shared key between* Trent and Bob

Objective: Alice and Bob establish a secret shared session key K_{AB}

Trusted Third Party: il protocollo



*M*1 $A \rightarrow T$: A, B

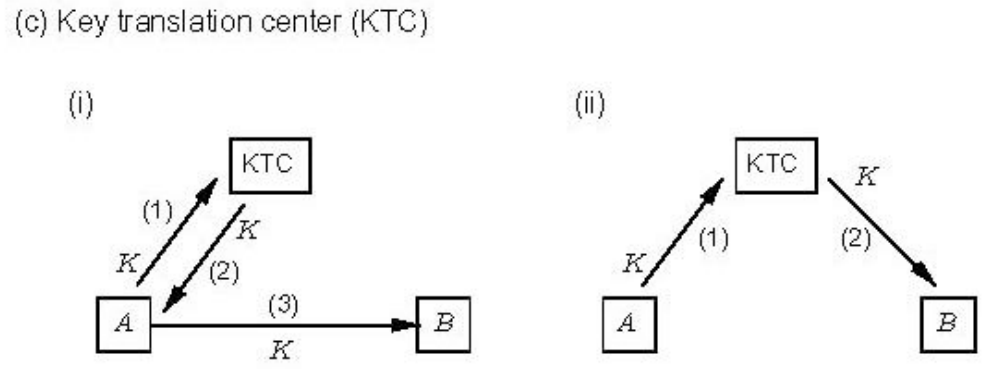
*M*2 $T \rightarrow A$: $E((T, L, K_{AB}, B), K_{AT})$, $E(T, L, K_{AB}, A), K_{BT})$

*M*3 $A \rightarrow B$: $E((A, T), K_{AB})$, $E(T, L, K_{AB}, A), K_{BT})$

M3 $B \rightarrow A$: $E(T+1, K_{AB})$

T: timestamp (nonce)
L: lifetime di K_{AB}

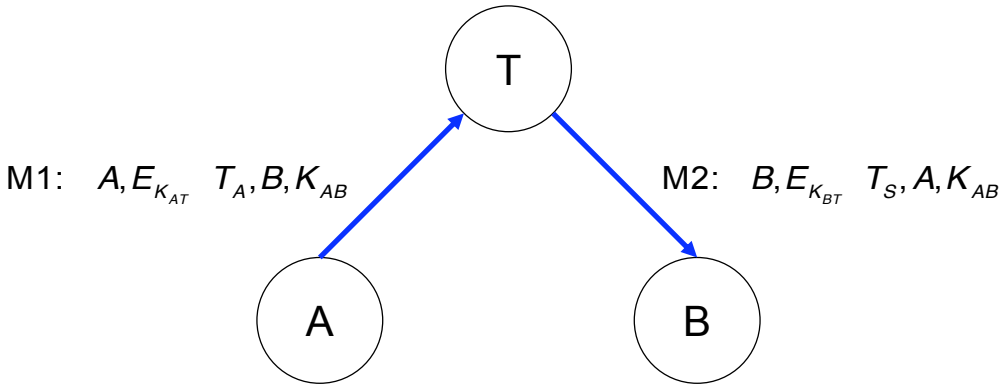
Decentralized key management



- *A* and *B* share distinct secret keys, K_{AT} and K_{BT}, with *KTC*
- One of the parties generates the session key *K*; KTC transmits that key to the other peer
- The party is trusted to correctly generate the key

The Wide-mouthed frog protocol

Key distribution with KTC



- Synchronized clocks
- Bob trusts Alice to be competent in generating keys

Key distribution with symmetric encryption

Pros

- It is easy to add and remove entities from the network
- Each entity needs to store only one long-term secret key

Cons

- All communication require initial interaction with the TTP
- The TTP must store *n* long-term keys
- The TTP has the ability to read all messages
- If the TTP is compromised, all communications are insecure

Public key distribution system

A **public key distribution systems** allows two users to securely exchange a key over ***an insecure channel***

Whitfield Diffie, "The first ten years of public key cryptography," Proceedings of IEEE, Vol. 76, no. 5, May 1988.

Whitfield Diffie and Martin Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, Vol. 22, no. 6, pages 644-654, November 1976.

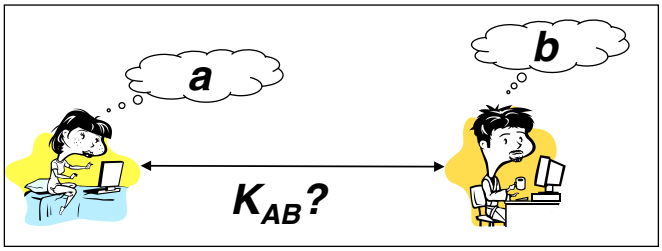
The discrete logarithm problem

- Let p be prime
- Let $1 \leq g < p$ be a generator, i.e.,
  $\forall\ 1 \leq n < p, \exists\ t$ s.t. $g^t \bmod p = n$
- **DISCRETE EXPONENTIATION**

  Given g and x,
  computing $y = g^x \bmod p$ is *computationally easy*
- **DISCRETE LOGARITHM** $x \stackrel{\Delta}{=} \log_g y \bmod p$

  Given g, $1 \leq y \leq p-1$,
  it is *computationally difficult* to determine x $(0 \leq x \leq p-2)$ s.t.
  $y = g^x \bmod p$

Diffie-Hellman protocol: scenario



- Let p be a large prime
- Let $1 \leq g < p$
- Let p e g publicly known

Alice chooses a random number a
Bob chooses a random number b

M1 A → B: A, $Y_A = g^a \bmod p$
M2 B → A: B, $Y_B = g^b \bmod p$

Alice computes $K_{AB} = (Y_B)^a \bmod p = g^{ab} \bmod p$
Bob computes $K_{AB} = (Y_A)^b \bmod p = g^{ab} \bmod p$

Security of Diffie-Hellman

- An adversary can compute K_{AB} from Y_A and Y_B by computing, for example,

$$K_{AB} = Y_A^{\log_g Y_B} \bmod p$$

- If logs mod p are easily computed then the system can be broken
- There is no proof of the converse, i.e., if logs mod p are difficult to compute then the system is secure
- We don't see any way to compute K_{AB} from Y_A and Y_B without first obtaining either a or b

Security of Diffie-Hellman

- Let p be a prime, $p < 2^n$, then

  All quantities are representable as n-bit numbers

  Exponentiation takes at most $2 \times \log_2 p = 2n$ multiplications mod p

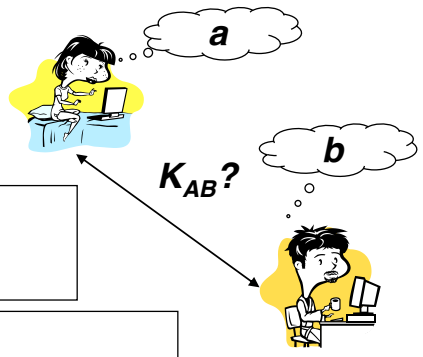
  Taking logs mod p requires $p^{1/2} = 2^{n/2}$ operations

- **Example $n = 512$**

  Exponentiation requires at most 1024 multiplications

  Taking logs mod p requires $2^{256} = 10^{77}$ operations

Diffie-Hellman protocol: an example



Let $p = 11$, $g = 7$

Alice chooses $a = 3$ and computes $Y_A = g^a \bmod p = 7^3 \bmod 11 = 343 \bmod 11 = 2$

Bob chooses $b = 6$ and computes $Y_B = g^b \bmod p = 7^6 \bmod 11 = 117649 \bmod 11 = 4$

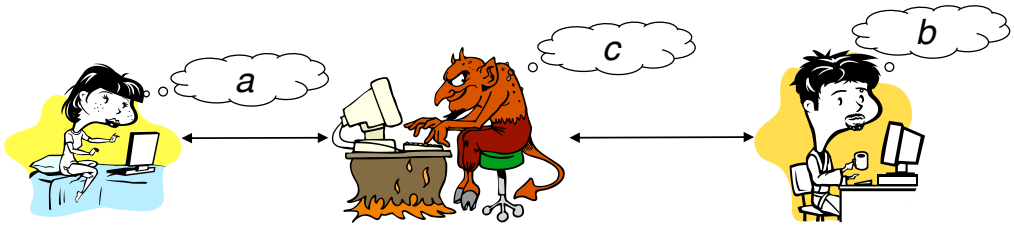
$A \rightarrow B$: 2
$B \rightarrow A$: 4

Alice receives 4 and computes $K_{AB} = (Y_B)^a \bmod p = 4^3 \bmod 11 = 9$

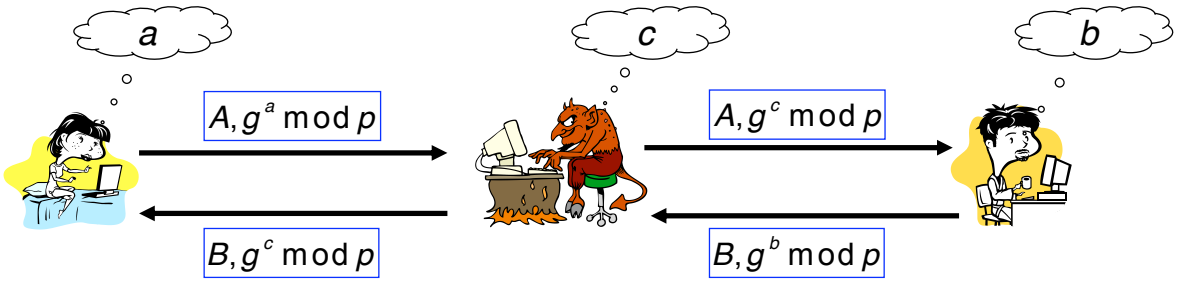
Bob receives 2 and computes $K_{AB} = (Y_A)^b \bmod p = 2^6 \bmod 11 = 9$

The man-in-the-middle attack

Alice has no guarantee that she is actually talking with Bob and vice versa



The man-in-the-middle



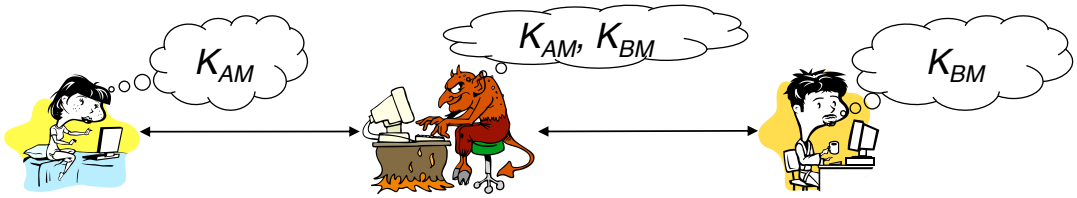
$K_{AM} = g^{ac} \bmod p$

$K_{AM} = g^{ac} \bmod p$, e

$K_{BM} = g^{bc} \bmod p$

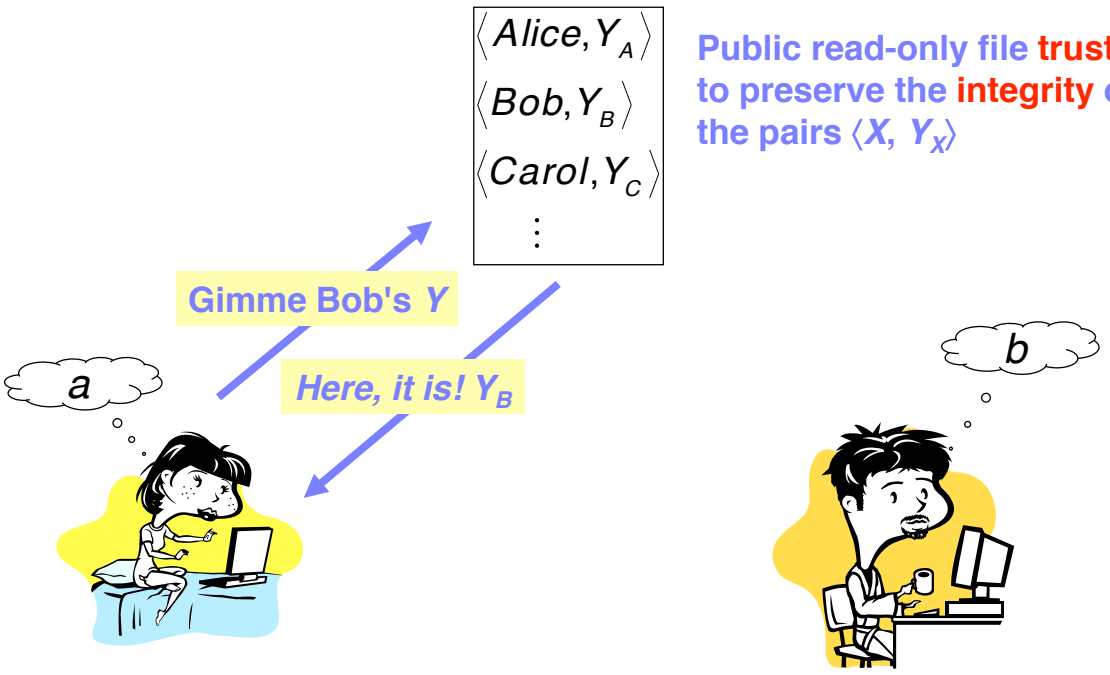
$K_{BM} = g^{bc} \bmod p$

The man-in-the-middle

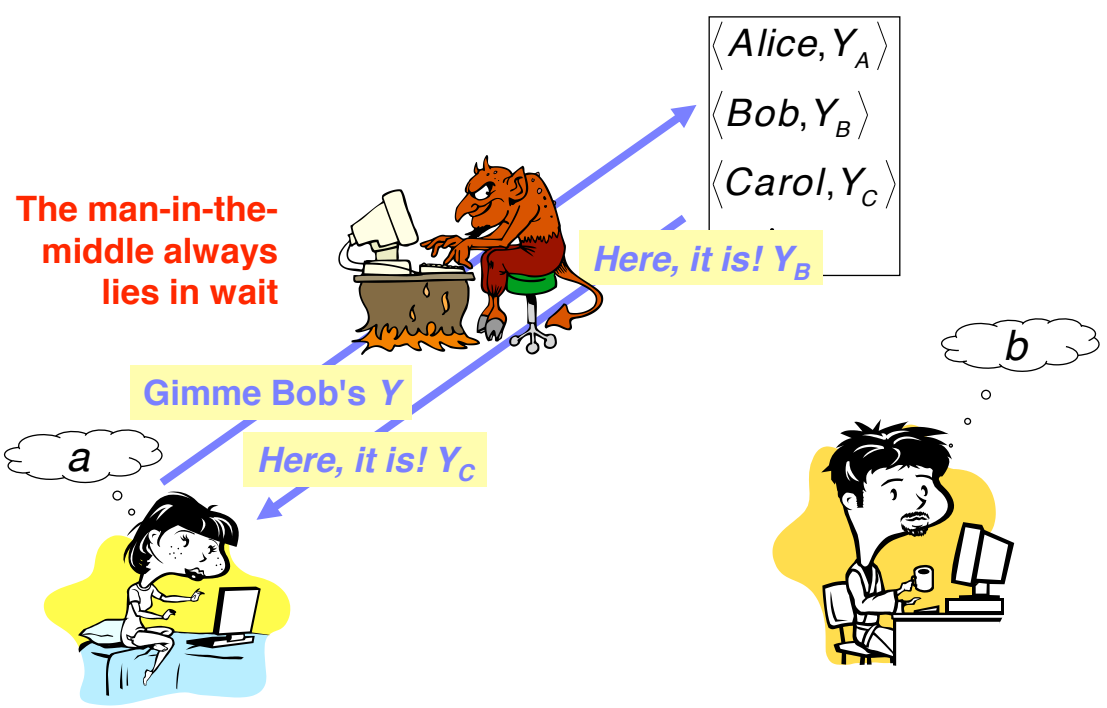


- Alice believes to communicate with Bob by means of K_{AM}
- Bob believes to communicate with Alice by means of K_{BM}
- The adversary can
  - read messages between Alice and Bob
  - inject messages between Alice and Bob (impersonate Alice and Bob)

Diffie-Hellman protocol



Diffie-Hellman protocol

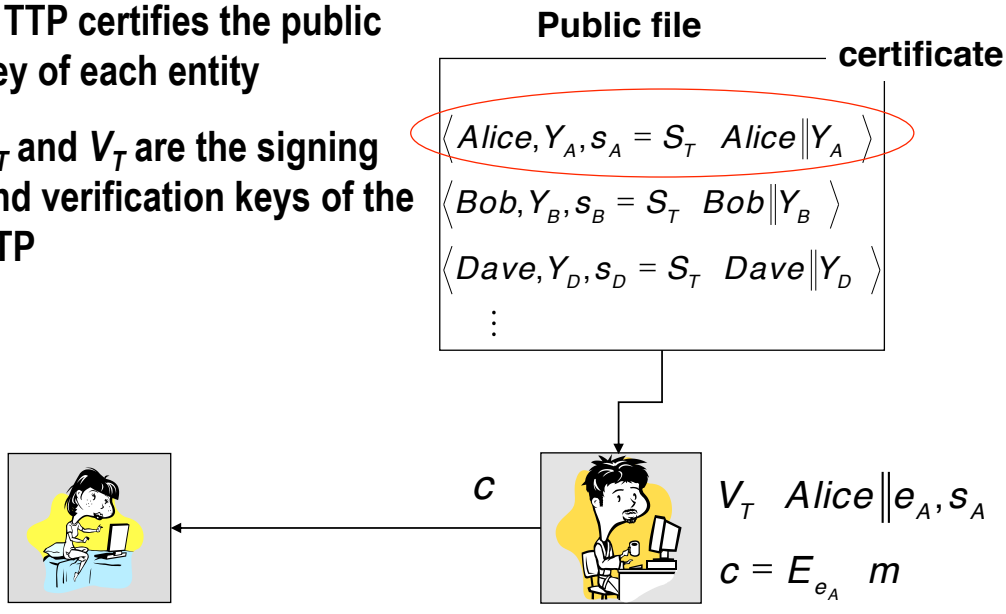


Key distribution with public encryption

- **Pros**
  - No TTP is required
  - The public file could reside with each entity
  - Only *n* public keys need to be stored to allow secure communications between any pair of entities, assuming that the only attack is that by a passive adversary
- **Cons**
  - Key management becomes more difficult in the presence of an active adversary

Key distribution with public keys

- **A TTP certifies the public key of each entity**
- **S_T and V_T are the signing and verification keys of the TTP**



Key distribution with certificates

- **Pros**
  - Prevent an active adversary from impersonation
  - Entities need to trust the TTP only to bind identities to public keys properly
  - Certicates can be stored locally so eliminating per-communication interaction with the public file
    - Uhmmm…not really!

- **Disadvantages**
  - if the signing key of TTP is compromised, all communications become insecure
  - All trust is placed with one entity