Cybersecurity: Key Management

Ozalp Babaoglu

Key management

- Issues
  - Distribution of public keys
  - Distributing secret keys through a trusted server
  - Distributing secret keys through public-key protocols

Distribution of public keys

- Public announcement
- Public directory
- Certificates

Public announcement

- The user renders her public key accessible by placing it in a public space
- Examples: the public key is inserted as an attachment to all outgoing mail, the public key is placed in the user's home page or social network profile
- Anyone can publish their (public) key, anyone can access the (public) key of others

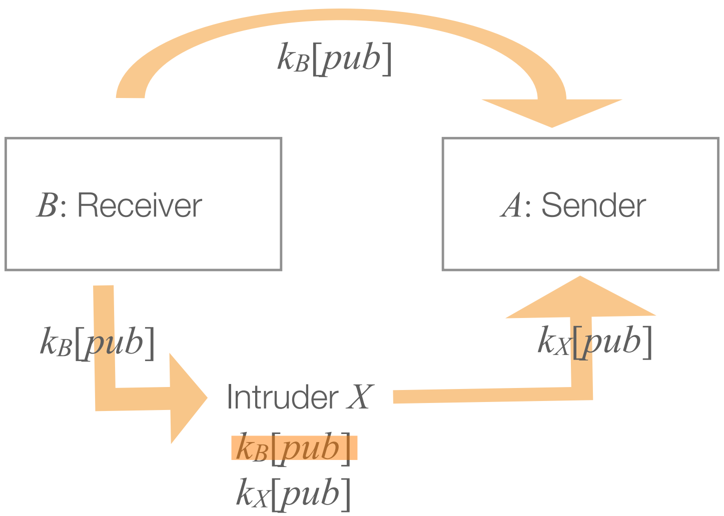
Public announcement

- Advantages
  - Simple, fast, does not require any third party intervention
- Disadvantages
  - No guarantees: the published information can be easily altered
  - An intruder can publish her own public key as if it belonged to someone else — "***man-in-the-middle attack***"

Man-in-the-middle Attack

- Example of an *active attack*
- Takes place during the publication phase of a public key
- X inserts herself in the communication path between A and B
- Towards B she pretends to be A
- Towards A she pretends to be B
- X makes sure that all communication between A and B passes through her

Man-in-the-middle Attack



Man-in-the-middle Attack

- A asks B to send her public key $k_B[pub]$ (for example by email)
- X intercepts $k_B[pub]$ and substitutes it with her own public key $k_X[pub]$
- X intercepts ciphertexts from A to B, decrypts them with $k_X[priv]$, encrypts them with $k_B[pub]$ and forwards them to B
- Works because A and B have no way to distinguish if they are talking with each other or with the intruder

Back to key distribution: Public directory

- The directory is a list of **`<user, public_key>`** pairs
- Directory must be maintained by a trusted party (authority)
- Publication:
  - A user registers (in person or through some other secure method) her public key with the authority for insertion into the directory
  - The user can modify her record through the authority after insertion
- Access:
  - Consult a latest ***local copy*** of the directory received (periodically) from the authority (just like a telephone directory)
  - Consult the copy maintained by the authority ***remotely*** (requires secure and authenticated communication protocols)

Public directory

- Disadvantages
  - Requires a trusted, impartial party — authority
  - The directory can be compromised
  - Requires communication protocols for securely publishing and accessing keys

Certificates

- Authenticity of keys certified by an *authority* by adding her signature
- Guarantees the *identity* of parties and *validity* of public keys (in case they were revoked or had to be regenerated after loss of private keys)
- Eliminates man-in-the-middle attacks: an intruder cannot substitute her own public key for someone else because she cannot sign the modified certificate (without knowing the private key of the authority)
- Requires a trusted, impartial party (the authority)
- (More on certificates later)

Management of (Private) Secret Keys

Management of secret keys

- n parties (clients, server, users, processes, etc.) need to communicate in private
- Use private-key cryptography to establish secure communication channels
- If every pairwise communication is possible and needs to be private, then we need $O(n^2)$ secret keys
- For large n, this may be impractical since secret keys cannot be long lived but should be replaced often
- Can we reduce the number of private keys to $O(n)$?
- Yes, if we can rely on a trusted third party

Management of secret keys

- Assume we have a (trusted) *Key Distribution Server* (KDS) that shares a different secret key with each party
- A and B want to establish a secure communication channel between themselves
- One of them asks the KDS to generate a *one-time session key* to use for the duration of that communication
- Future communications between A and B will generate and use different *session keys*

Management of secret keys: Basic Protocol

- A and *KDS* share K_A
- B and *KDS* share K_B

1. A sends to *KDS*: $\{A,$ "request session key for B"$\}$
2. *KDS* generates new session key K_S and sends to A:

$$C(K_A, \{K_S, C(K_B, K_S)\})$$

3. A stores K_S and sends to B: $C(K_B, K_S)$
4. B stores K_S
5. A and B can exchange confidential messages using K_S

Basic Protocol: Comments

- B does not receive K_S directly from the KDS but from A
- Forms the basis for many other more complex protocols
- Problems:
  - B cannot know for sure if the message was sent by A
  - Subject to "replay attacks"
  - An intruder can record and resend a ciphertext

$$C(K_A, \{K_S, C(K_B, K_S)\})$$

  in the future as if it was new

Management of secret keys: Needham-Schroeder Protocol

- A and *KDS* share K_A
- B and *KDS* share K_B

1. A sends to *KDS*: $\{A$, "request session key for B", $N_1\}$
2. *KDS* generates new session key K_S and sends to A:

$$C(K_A, \{K_S, A, B, N_1, C(K_B, \{K_S, A\})\})$$

3. A stores K_S and sends to B: $C(K_B, \{K_S, A\})$
4. B stores K_S and sends to A: $C(K_S, N_2)$ (challenge)
5. A replies to B: $C(K_S, N_2 + 1)$ (response)
6. A and B can exchange confidential messages using K_S

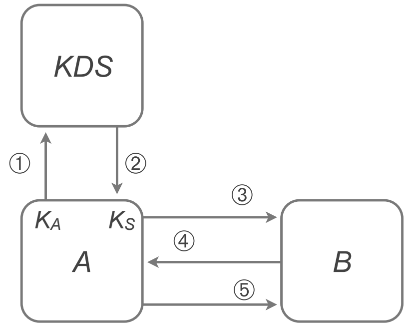
Management of secret keys: Needham-Schroeder Protocol

- N_1 and N_2 are called "nonces" (*number used once*) and prevent replay attacks
- The "challenge-response" handshake in steps 4 and 5 serve to confirm that both A and B are present and willing to communicate as well as to synchronize the communication to using the same session key (messages need not be received in the order in which they are sent)
- Basis for the ***Kerberos authentication protocol***

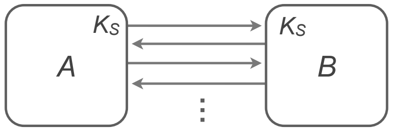
Needham-Schroeder Protocol: Comments

- A trusts *KDS* and is certain to have received the session key from *KDS* because the message is encrypted with K_A
- The nonce N_1 serves to match the session key received to the request made by A in step 1
- A is certain to reveal K_S only to B because it send K_S encrypted with K_B, which only B is able to decrypt
- B trusts *KDS* and *KDS* guarantees B that the key can be used only for communicating with A
- B can detect replay attacks and is sure to be communicating with A

Management of secret keys: Needham-Schroeder Protocol



Management of secret keys: Needham-Schroeder Protocol



Needham-Schroeder Protocol: Possible Attacks

- Message 3 ($C(K_B, \{K_S, A\})$) is not protected by a nonce
- Suppose X cracks the session key K_S from last week's run of the protocol and saves message 3 from that run
- X can now replay that message and make A believe it is talking to B

3. X sends to B: $C(K_B, \{K_S, A\})$
4. B sends to X: $C(K_S, N_2)$ (challenge)
5. X replies to B: $C(K_S, N_2 + 1)$ (response)

- There is no way for B to know if the K_S it receives in message 3 is current
- Fix by adding another nonce to message 3

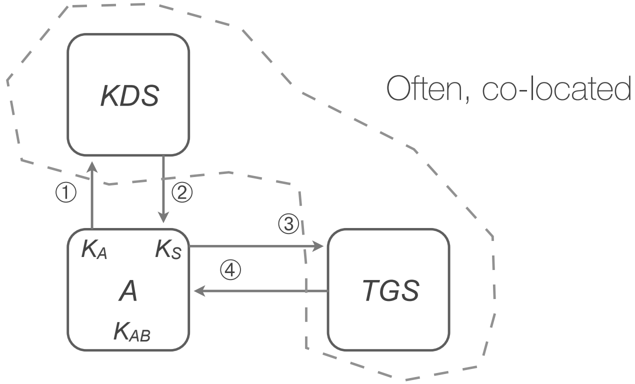
Management of secret keys: Needham-Schroeder Protocol

- If A wants to communicate with a different principal C, it has to restart the protocol with *KDS* to generate a new session key K_S using its shared secret key K_A
- Since the shared key is based on a secret, this results in an increased risk that it may be compromised

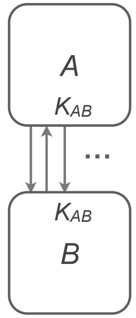
Kerberos

- Developed at MIT during the 1980's to serve as an distributed ***authentication service*** in an academic environment
- Allows ***principals*** (clients running on behalf of users) to prove their identity to ***servers*** in a secure manner
- Each principal initially shares a secret key (password) with the *KDS*
- To reduce the exposure of the secret key, *KDS* used only ***once*** per login session
- All communication within a single session secured through keys obtained from a *Ticket Granting Server* (TGS)

Kerberos

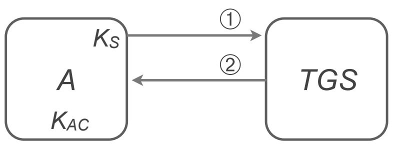


Kerberos

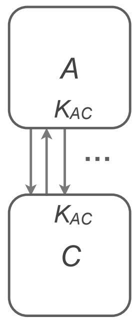


Kerberos

- If A wants to communicate with a different principal C, it has to restart the protocol with *TGS* (*not KDS*) to generate a new session key K_{AC} using the key K_S (*not* the shared secret key K_A



Kerberos



Kerberos

- In a very large system, *KDS* may be a performance and reliability bottleneck
- *KDS* can be replicated to obtain increased performance and reliability using a master-slave scheme
- In a very large systems, a single (or replicated) *KDS* may not be acceptable for administrative reasons

Management of secret keys: Kerberos

- Advantages
  - Guarantees confidentiality and authentication
  - Achieves good performance even in the presence of many parties and frequent key changes
  - For n parties, reduces the number of necessary secret keys from $O(n^2)$ to $O(n)$
- Defects
  - Requires the existence of a trusted (and reliable) *KDS*

Private-Key versus Public-Key Cryptography

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Management of secret keys: Hybrid solutions

- Public-key cryptography is about 1000 times slower than private-key cryptography
- Hybrid solutions:
  - Use ***asymmetric cryptography*** once initially to agree on a secret key
  - Then, switch to ***symmetric cryptography*** (using the agreed upon secret key) for all future communication

Management of secret keys: Hybrid solution

1. A generates $(K_A[pub], K_A[priv])$
2. A sends to B: $\{K_A[pub], A\}$
3. B generates session key K_S
4. B sends to A: $C(K_A[pub], K_S)$
5. A decrypts to obtain $K_S = D(K_A[priv], C(K_A[pub], K_S))$
6. A deletes $(K_A[pub], K_A[priv])$, B deletes $K_A[pub]$
7. A and B then switch to symmetric cryptography using session key K_S

Management of secret keys: Hybrid solution

- Guarantees confidentiality and authentication
- Remains subject to *man-in-the-middle* attacks
- General solution based on ***certificates*** to guarantee mutual authentication while avoiding man-in-the-middle attacks
- Basis for SSL