CS458: Introduction to Information Security

Notes 8: Key Management and Distribution

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

March 20th 2024

Slides: Modified from "Cryptography and Network Security", 7/e, by William Stallings, "Computer Security: Principles and Practice", 4th Edition, Christof Paar and Jan Pelzl, Ion Petre, University of Turku, Finland, & Steven Gordon at Thammasat University

Outline

- Cryptographic Key Management
  - *Cryptographic key management involves the secure handling and distribution of keys used for encryption and decryption.*
- Symmetric Key Distribution using Symmetric Encryption
- Symmetric Key Distribution using Asymmetric Encryption
- Distribution of Public Keys
- Certificates
- Public-Key Infrastructure

Cryptographic Key Management

- Essential for secure data handling.
- Ensure keys are protected from:
  - Unauthorized access and disclosure.
  - Modification and tampering.
- Key management involves:
  - Generation of strong, random keys.
  - Secure storage, minimizing exposure.
  - Safe key exchange with authorized parties.
  - Regular key replacement to mitigate risks.
- Access control, monitoring, and auditing are crucial.
  - Access control ensures that only authorized entities can access and modify keys, reducing the risk of misuse.
  - Monitoring detects unusual activities in real-time, aiding in security incident detection and response.
  - Auditing records key-related actions, enabling compliance, accountability, and post-incident analysis, enhancing overall system security.
- The security of the entire cryptosystem relies on effective key management.

Key Management: Challenges

- Key management in secure communication systems involves several challenges
  - *How to share a secret key?*
    - When two parties need to communicate securely, they must share a secret key.
    - The challenge is to ensure that the key is not intercepted by an attacker.
  - *How to obtain someone else's public key?*
    - In a public key cryptography system, users need to obtain the public key of the party they want to communicate with.
    - The challenge is to ensure that the public key is authentic and has not been tampered with.
  - *When to change keys?*
    - Keys need to be changed periodically to ensure that they have not been compromised.
    - The challenge is to determine the frequency of key changes and to manage the distribution of new keys.

Key Management: Assumptions and Principles

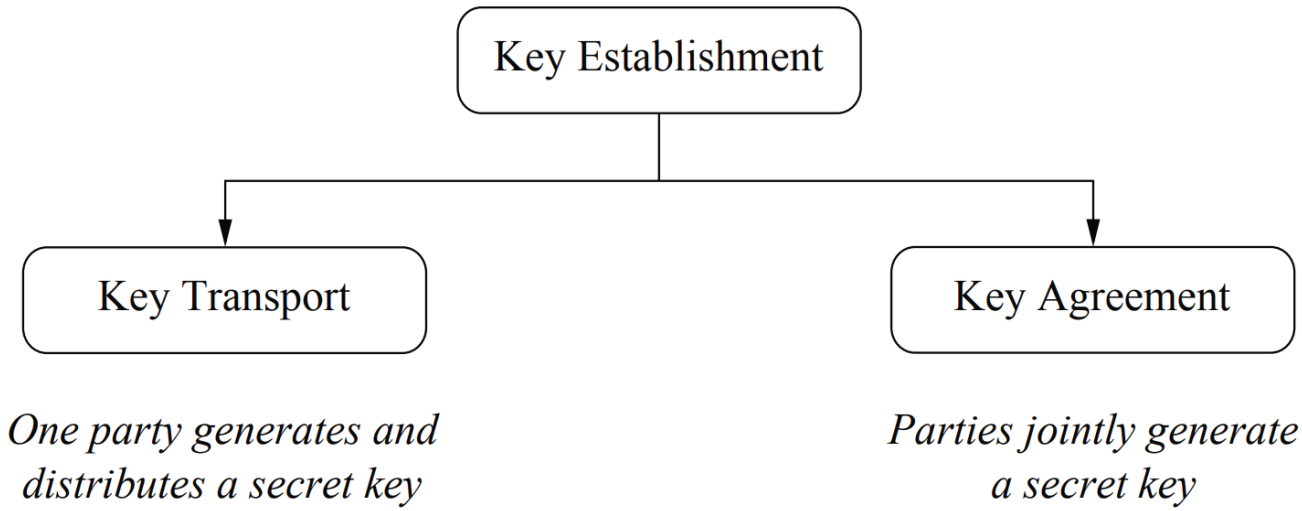
- To address these challenges, there are some assumptions and principles that are commonly followed:
  - Secure communication is a requirement for many users
    - It acknowledges that many users require secure communication.
    - Systems and protocols should be designed with the expectation that security is a fundamental necessity
  - Attacker can intercept any location in the network
    - It assumes that malicious actors can intercept data at any point within a network.
    - This assumption implies that all communication must be treated as potentially compromised.
  - Manual interactions between users are undesirable
    - It discourages relying on physical or manual interactions between users for key exchange.
    - Instead, automated processes and protocols should be in place to streamline key management.
  - Key reuse increases the risk of compromise
    - The more times a key is used, the greater the chance that an attacker will discover the key.
    - To enhance security, it is advisable to change or "rotate" encryption keys regularly.

Key Management: Assumptions and Principles

- *These principles guide the design and implementation of cryptographic systems to ensure secure communication while considering the constant threat of network interception and the need for automated, efficient, and secure key management practices.*

Classification of Key Exchange Methods

- Key exchange (also key establishment) deals with establishing a shared secret between two or more parties
  - *Key establishment includes the generation and sharing of cryptographic keys and other keying material between entities.*
- Methods classified into key transport and key agreement methods



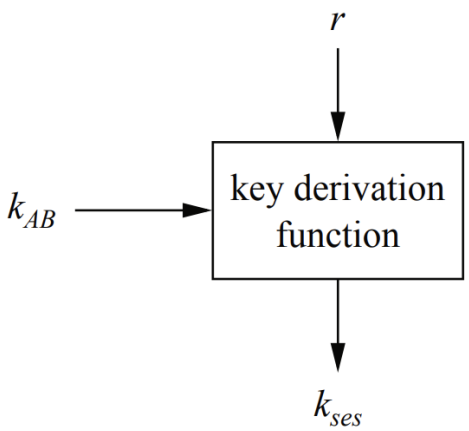
- In an ideal key agreement protocol, no single party can control what the key value will be.
- Additional variations beyond key transport and key agreement exist, including various forms of key update, such as key derivation.

Key Freshness

- It is often desirable to frequently change the key in a cryptographic system.
- Reasons for key freshness include:
  - If a key is exposed (e.g., through hackers), there is limited damage if the key is changed often
  - Some cryptographic attacks become more difficult if only a limited amount of ciphertext was generated under one key
  - If an attacker wants to recover long pieces of ciphertext, he has to recover several keys which makes attacks harder.

Key Derivation

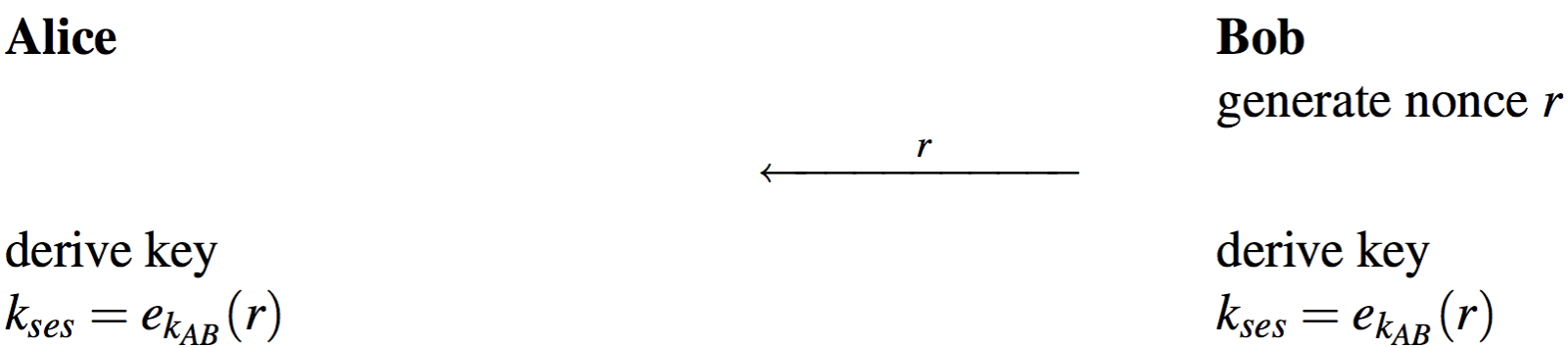
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can derive multiple session keys k_{ses} from a given key k_{AB}.



- The key k_{AB} is fed into a key derivation function (KDF) together with a nonce r ("number used only once").
- Every different value for r yields a different session key

Key Derivation

- The key derivation function is a computationally simple function, e.g., a block cipher or a hash function.
- It should be a one-way function
- Example for a basic protocol:



- Other alternatives:
  - hashing $nonce$ together with k_{AB}: $K_{ses} = HMAC_{k_{AB}}(r)$
  - Rather sending a $nonce$, Alice and Bob can also simply encrypt a counter cnt periodically: $K_{ses} = e_{k_{AB}}(cnt)$ or
  - Compute the HMAC of the counter: $K_{ses} = HMAC_{k_{AB}}(cnt)$

Key Distribution Technique

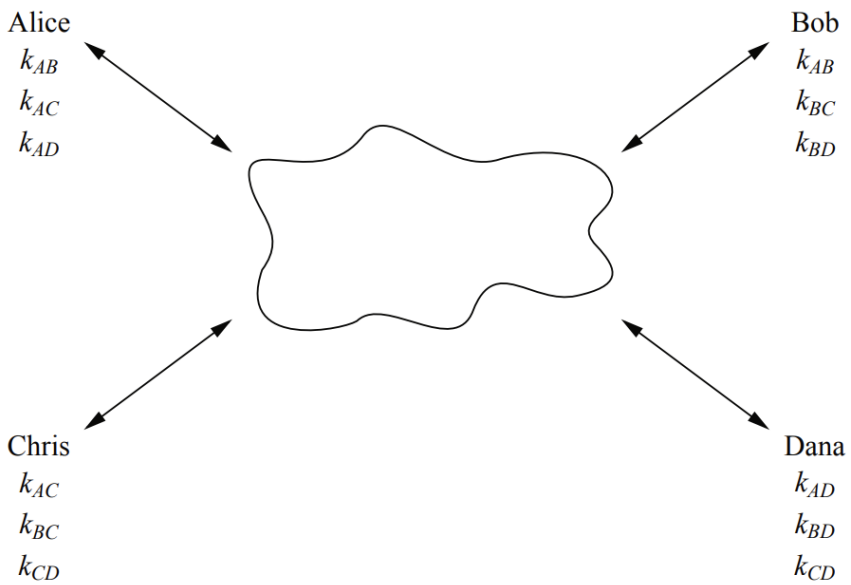
- For symmetric encryption to work, the two parties engaged in an exchange must share the same key, and that key must be protected from access by others.
- Frequent key changes are desirable as they limit the amount of data that could be compromised if an attacker were to learn the key.
- *The robustness and security of any cryptographic system largely depend on the key distribution techniques employed.*
- Key distribution
  - refers to the methods used to securely deliver a key to two or more parties who need to exchange data without exposing the key to unauthorized access.

Exchanging Secret Keys

- Option 1: Manual Exchange of All Keys
  - All users exchange secret keys with all other users manually (e.g., face-to-face)
  - *Can be inconvenient and time-consuming, especially for large groups of users.*
- Option 2: Manual Exchange of Master Keys
  - All users exchange master key with trusted, central entity (e.g., Key Distribution Center)
  - Session keys automatically exchanged between users via `KDC`
  - *Security and performance bottleneck at the `KDC`, as all session keys must be exchanged through it.*
- Option 3: Public Key Cryptography to Exchange Secrets
  - Use public-key cryptography to securely and automatically exchange secret keys
  - Example 1: `Alice` encrypts secret with `Bob`'s public key; sends to `Bob`
  - Example 2: Diffie-Hellman secret key exchange
  - Related issue: *How to obtain someone else's public key?*

Naïve approach: The n^2 Key Distribution Problem

- Let a network with n users where every user wants to communicate securely with every other user ($n-1$ users), and the group of users is dynamic, meaning individuals can join or leave at any time.
- **Naïve approach** (Separate Channel): Every pair of users obtains an individual key pair
  - Ideally we've separate secure channel for exchanging keys (e.g., a phone line, separate data network, or in a letter.)
- Example: Keys in a network with $n = 4$ users:



The n^2 Key Distribution Problem

- Shortcomings
  - There are $n(n-1) \approx n^2$ keys stored in the system
  - There are $\frac{n(n-1)}{2}$ pair keys
  - Large number of keys
    - Example: mid-size company with $1{,}000$ employees
      $1{,}000 \times 999 = 999{,}000$ keys must be distributed securely.
  - If a new user `Sammy` joins the network, new keys k_{XS} have to be transported via secure channels to each of the existing users
    $\Rightarrow$ Only works for small networks which are relatively static
- Direct secret sharing grows at n^2. *So how can we solve that?*

Symmetric Key Distribution using Symmetric Encryption

Symmetric Key Distribution using Symmetric Encryption

- **Objective**: for two entities to share same secret key
- **Principle**: should be changed keys frequently to limit the risk of compromise
- *How to exchange a secret key?*
- There are two primary methods for exchanging secret keys:
  1. Decentralized Key Distribution
     - manual distribution of master keys between all entities, automatic distribution of session keys
     - *this method can be inconvenient and impractical, especially for larger networks.*
  2. Key Distribution Center (KDC)[1]
     - manual distribution of master keys with `KDC`, automatic distribution of session keys
     - *while this method can add some performance overhead, it is more efficient than decentralized key distribution and is commonly used in practice.*

[1] *A trusted third party, another server in the network that support a key distribution*

Key Hierarchy and Lifetimes

- Master keys (Key Encryption Key) used to securely exchange session keys
- Session keys used to securely exchange data
- Change session keys automatically and regularly
- Change master keys manually and seldom
- Session key lifetime
  - The lifetime of a session key in cryptographic communication plays a crucial role in balancing security and operational efficiency.
    - Shorter lifetime enhances security but increases overhead.
    - *Connection-oriented protocols (TCP)* use a new session key for each connection.
    - *Connection-less protocols (UDP)* change session keys after a fixed period or a certain number of packets sent.

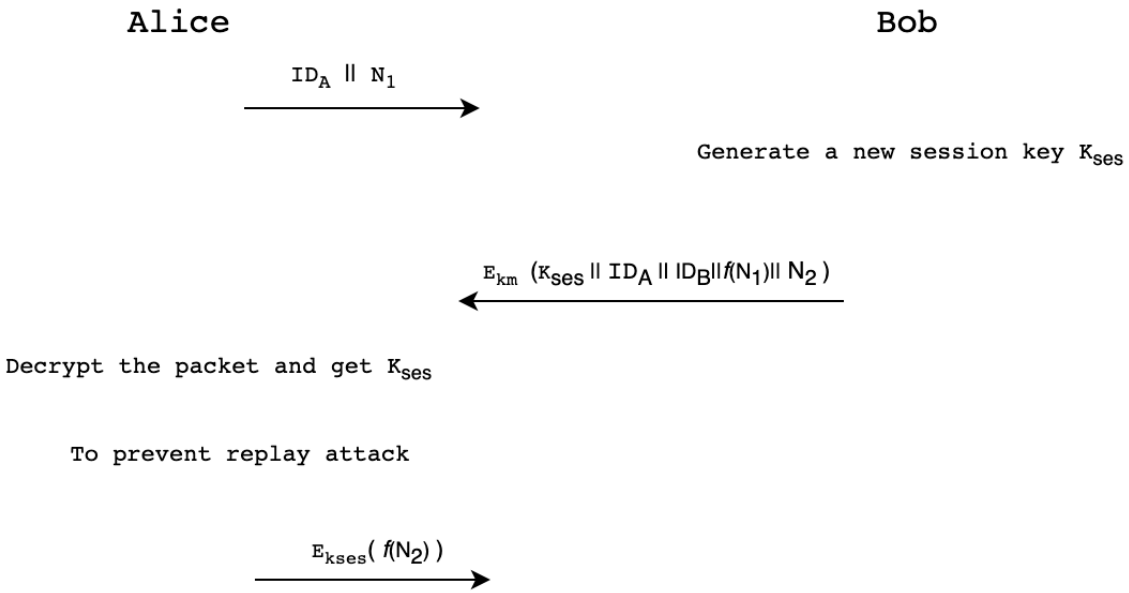
Notation

- End-systems: *Alice* and *Bob*, identified by ID_A and ID_B
- Master key (between *Alice* and *Bob*): K_m
- Master keys specific to user: K_A, K_B
- Session key (between *Alice* and *Bob*): K_{ses}
- Nonce values: N_1, N_2 (Number used only once)
  - Nonces N_1 and N_2 are unique values used for security purposes.
  - Generated using time-stamps, counters, or random values.
  - Must be different for each request to prevent replay attacks.
  - Nonces should be difficult for attackers to guess or predict.

Decentralized Key Distribution

- Each end-system must manually exchange $n-1$ master keys (K_m) with all other end-systems in the network.
  - *This can be a tedious and impractical process, especially for larger networks.*
- Once the master keys are exchanged, session keys can be automatically generated and exchanged between end-systems as needed.
  - When we want to encrypt the data, we generate session key and then exchange it automatically
- Does not rely on trusted-third party and allows for secure communication between entities without exposing the master keys to an outside party.
- However, the need for manual key exchange and management can become a significant burden as the network grows.

Decentralized Key Distribution



- Problem: *Does not work on a large network*.
  - Each communication pair needs to share a master key.
  - Still have to exchange many master keys manually.

Using a Key Distribution Center

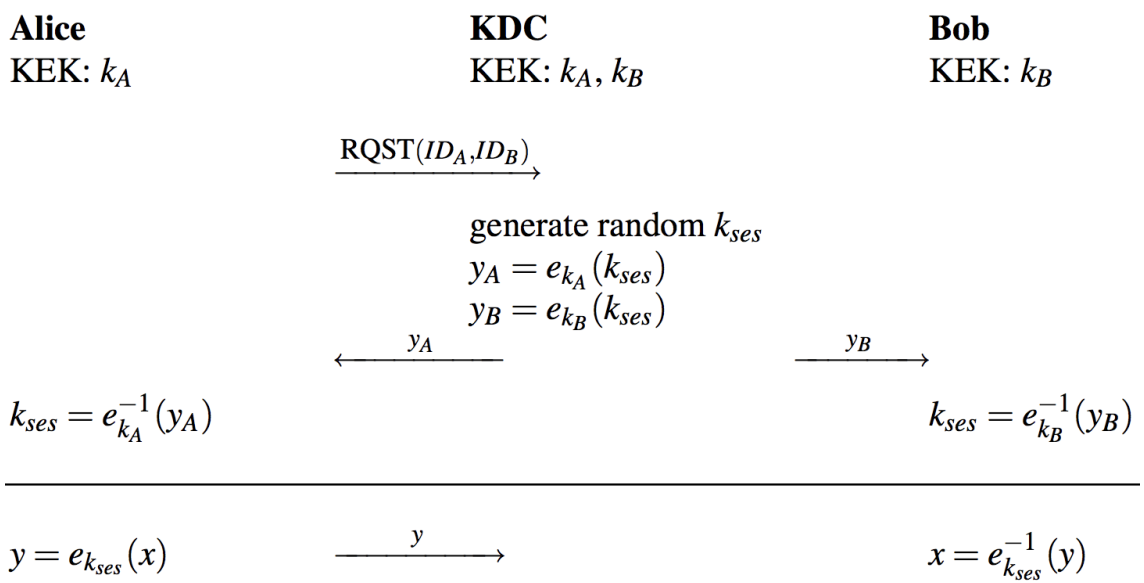
- Key Distribution Center (`KDC`) is trusted third party
- Users manually exchange master keys with `KDC`
  - As a result, the `KDC` has many master keys, one for each party.
  - *KDC reduces the number of master keys necessary.*
- Users automatically obtain session key (via `KDC`) to communicate with other users

Using a Key Distribution Center

- Key Distribution Center (KDC): Central party, trusted by all users, generates and distributes session keys
- **Idea**:
  - Central trusted authority (KDC) that shares one key key encryption key (*KEK*) with every user
  - A key distribution center (KDC) generates and distributes session keys
  - Principle
    - `Alice` sends a request to the KDC for a symmetric key to be used as a session key for communication with `Bob`.
    - The KDC generates a symmetric session key, encrypts it with the master key it shares with `Alice` and sends it to `Alice`.
    - The KDC also encrypts the session key with the master key it shares with `Bob` and sends it to `Bob`.
    - Alternatively, the KDC sends both encrypted key values to `Alice`, and `Alice` forwards the session key encrypted with the master key shared by the KDC and `Bob` to `Bob`.

Using a Key Distribution Center

- KDC sends session keys to users which are encrypted with *KEK*s



Using a Key Distribution Center

- Advantages over previous approach:
  - Only n **long-term key pairs** are in the system

n users
n master key between users and KDC
-- linear complexity
# of keys in system: $2n$
e.g., Two users: Alice, Bob:
K_A, at Alice, K_B at Bob,
Two keys K_A, K_B at KDC
==>Total four keys.

  - If a new user is added, a secure key is only needed between the user and the KDC (the other users are not affected)
  - Scales well to moderately sized networks

Using a Key Distribution Center

Remaining problems:

- How does Bob know he is talking to Alice?
  - Replay attack: An attack in which attacker repeats or delays a valid transmission and fraudulently re-transmits it. Oscar records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't.
  - Key confirmation attack:
    - Oscar intercepts Alice's request $RQST(ID_A, ID_B)$ and sends to KDC $RQST(ID_A, ID_o)$.
    - Alice believes that she communicates with Bob and Oscar will decrypt her messages - there is **no key confirmation**
- *Protocols must provide authentication and defense against replay*
- Kerberos (a popular authentication and key distribution protocol) is based on KDCs

Kerberos : Authentication and Key Distribution Standard Protocol

- Widely used authentication and key distribution protocol that provides strong security against both replay and key confirmation attacks
- It's main purpose is to provide user authentication in computer networks
- Based on a KDC, which is named the "authentication server" in Kerberos terminology.

Key Establishment Using a Simplified Version of Kerberos

Alice		**KDC**		**Bob**
KEK: k_A		KEK: k_A, k_B		KEK: k_B
generate nonce r_A				
	$\xrightarrow{\text{RQST}(ID_A, ID_B, r_A)}$			
		generate random k_{ses} generate lifetime T $y_A = e_{k_A}(k_{ses}, r_A, T, ID_B)$ $y_B = e_{k_B}(k_{ses}, ID_A, T)$		
	$\xleftarrow{y_A, y_B}$			
$k_{ses}, r'_A, T, ID_B = e^{-1}_{k_A}(y_A)$ verify $r'_A = r_A$ verify ID_B verify lifetime T generate time stamp T_S $y_{AB} = e_{k_{ses}}(ID_A, T_S)$				
	$\xrightarrow{y_{AB}, y_B}$			
				$k_{ses}, ID_A, T = e^{-1}_{k_B}(y_B)$ $ID'_A, T_S = e^{-1}_{k_{ses}}(y_{AB})$ verify $ID'_A = ID_A$ verify lifetime T verify time stamp T_S
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$			$x = e^{-1}_{k_{ses}}(y)$

- Protects against both attacks

Key Establishment Using a Simplified Version of Kerberos

- Kerberos assures the timeliness of the protocol through two measures
  1. KDC specifies a lifetime T for the session key
     - The lifetime is encrypted with both session keys, i.e., it is included in y_A and y_B.
     - Hence, both Alice and Bob are aware of the period during which they can use the session key
  2. Alice uses a time stamp T_S, through which Bob can be assured that Alice's messages are recent and are not the result of a replay attack.
     - For this, Alice's and Bob's system clocks must be synchronized, but not with a very high accuracy

Key Establishment Using a Simplified Version of Kerberos

- Kerberos provides key confirmation and user authentication.
  - Alice sends a random nonce r_A to the **KDC**.
  - This can be considered as a *challenge* because she challenges the **KDC** to encrypt it with their joint **KEK** k_A.
  - If the returned *challenge* r'_A matches the sent one, Alice is assured that the message y_A was actually sent by the **KDC**.
  - This method to authenticate users is known as *challenge-response protocol* and is widely used, e.g., for authentication of smart cards.

Key Establishment Using a Simplified Version of Kerberos

- Kerberos provides key confirmation and user authentication.
  - Through the inclusion of Bob's identity ID_B in y_A, Alice is assured that the session key is actually meant for a session between herself and Bob
  - With the inclusion of Alice's identity ID_A in both y_B and y_{AB} , Bob can verify that
    i. **KDC** included a session key for a connection between him and Alice, and
    ii. he is currently actually talking to Alice

Key Distribution with Key Distribution Center

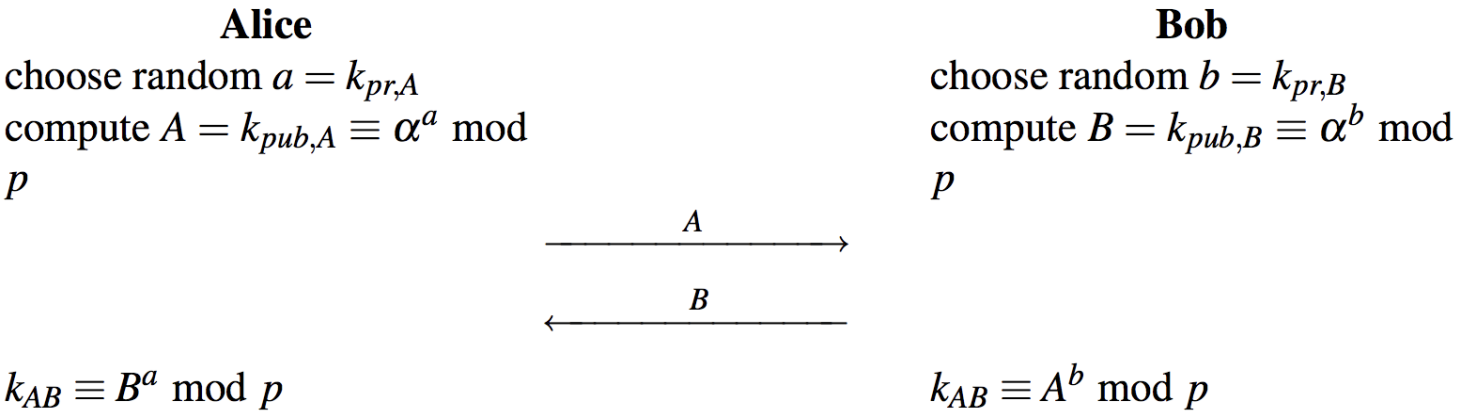
- Remaining problems:
  - No Perfect Forward Secrecy: If the *KEK*s are compromised, an attacker can decrypt past messages if he stored the corresponding ciphertext
  - Single point of failure: The KDC stores all *KEK*s. If an attacker gets access to this database, all past traffic can be decrypted.
  - Communication bottleneck: The KDC is involved in every communication in the entire network (can be countered by giving the session keys a long life time)
  - Secure channel during initialization: All KDC-based protocols require a secure channel at that time a new user joins the network for transmitting that user's KED.
- A cryptographic protocol has perfect forward secrecy (PFS) if the compromise of **long-term keys** does not allow an attacker to obtain **past session keys**
- The main mechanism to assure PFS is to employ public-key techniques.

Symmetric Key Distribution using Asymmetric Encryption

Key Establishment Using Asymmetric Techniques

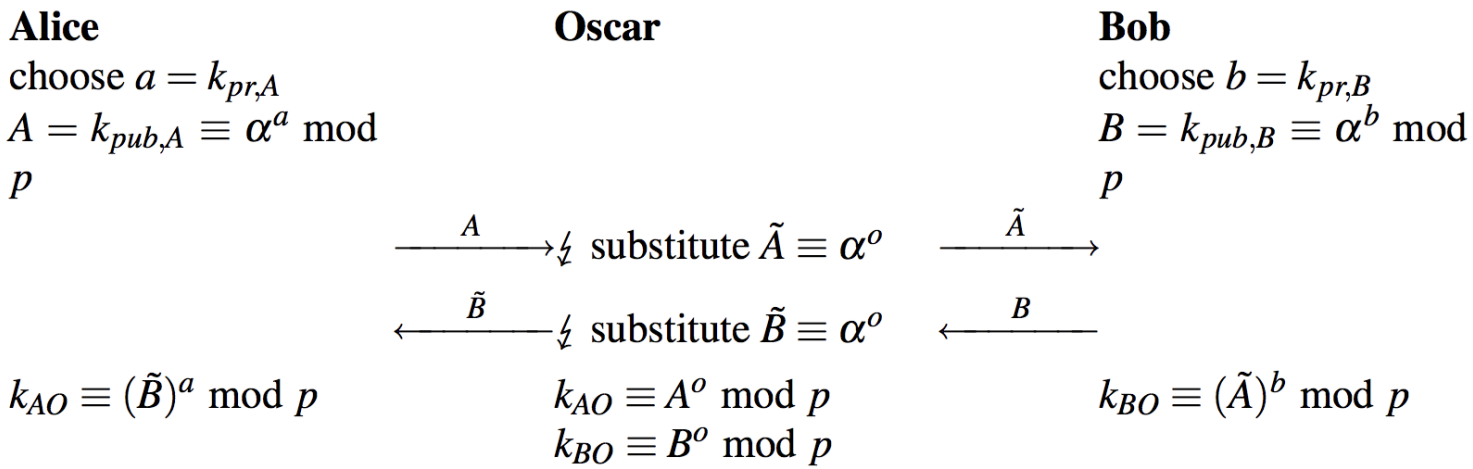
- Public-key algorithms are especially suited for key establishment protocols since they don't share most of the drawbacks that symmetric key approaches have
- Next to digital signatures, key establishment is the other major application domain of public-key schemes.
- Can be used for both key transport (e.g., RSA, ElGamal) and key agreement (e.g., DHKE) .
- However, they require an authenticated channel to distribute the public keys.
- Later: Solve the problem of authenticated public key distribution

Recall: Diffie-Hellman Key Exchange (DHKE)



- Widely used in practice
- If the parameters are chosen carefully (especially a prime p with a length of *1024* or more bit, the DHKE is secure against eavesdropping, i.e., passive attacks.
- However: If the attacker can actively intervene in the communication, the man-in-the-middle attack becomes possible.

Man-in-the-Middle Attack



- Oscar computes a session key k_{AO} with Alice, and k_{BO} with Bob
- However, Alice and Bob think they are communicating with each other!
- The attack efficiently performs 2 DH key-exchanges: Oscar-Alice and Oscar-Bob
- Here is why the attack works:

  Alice computes: $k_{AO} = (B')^a = (\alpha^o)^a$ Bob computes: $k_{BO} = (A')^b = (\alpha^o)^b$

  Oscar computes: $k_{AO} = A^o = (\alpha^a)^o$ Oscar computes: $k_{BO} = B^o = (\alpha^b)^o$
- Oscar has now complete control over the channel, e.g., if Alice wants to send an encrypted message x to Bob, Oscar can read the message

Important facts about Man-in-the-Middle Attack

- The man-in-the-middle-attack is not restricted to DHKE; it is applicable to any public-key scheme, e.g. RSA encryption, etc.
- Attack works always by the same pattern: Oscar replaces the public key from one of the parties by his own key.
- Q: What makes the MIM attack possible?
  - The public keys are not authenticated: When Alice receives a public key which is allegedly from Bob, she has no way of knowing whether it is in fact his.

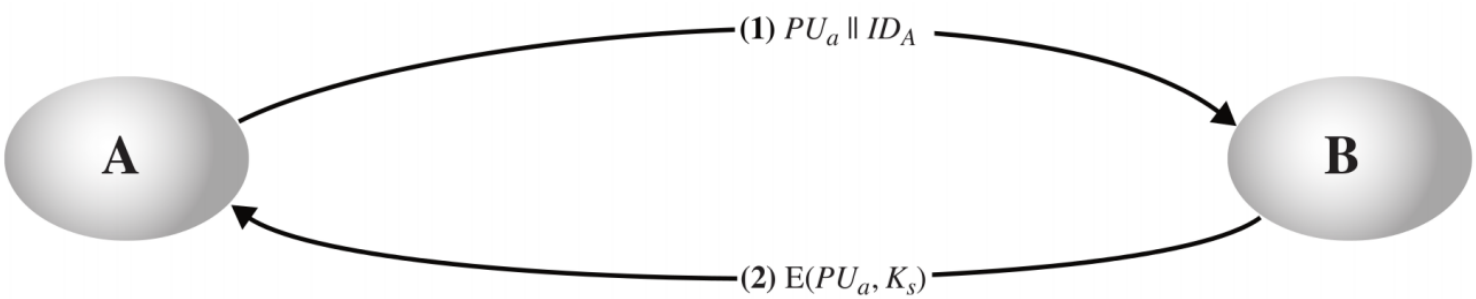
- *Even though public keys can be sent over insecure channels, they require authenticated channels*

Symmetric Key Distribution using Asymmetric Encryption

- Asymmetric encryption generally too slow for encrypting large amount of data
- Common application of asymmetric encryption is exchanging secret keys
- Three examples
  1. Simple Secret Key Distribution
  2. Secret Key Distribution with Confidentiality and Authentication
  3. Hybrid Scheme: Public-Key Distribution of KDC Master Keys

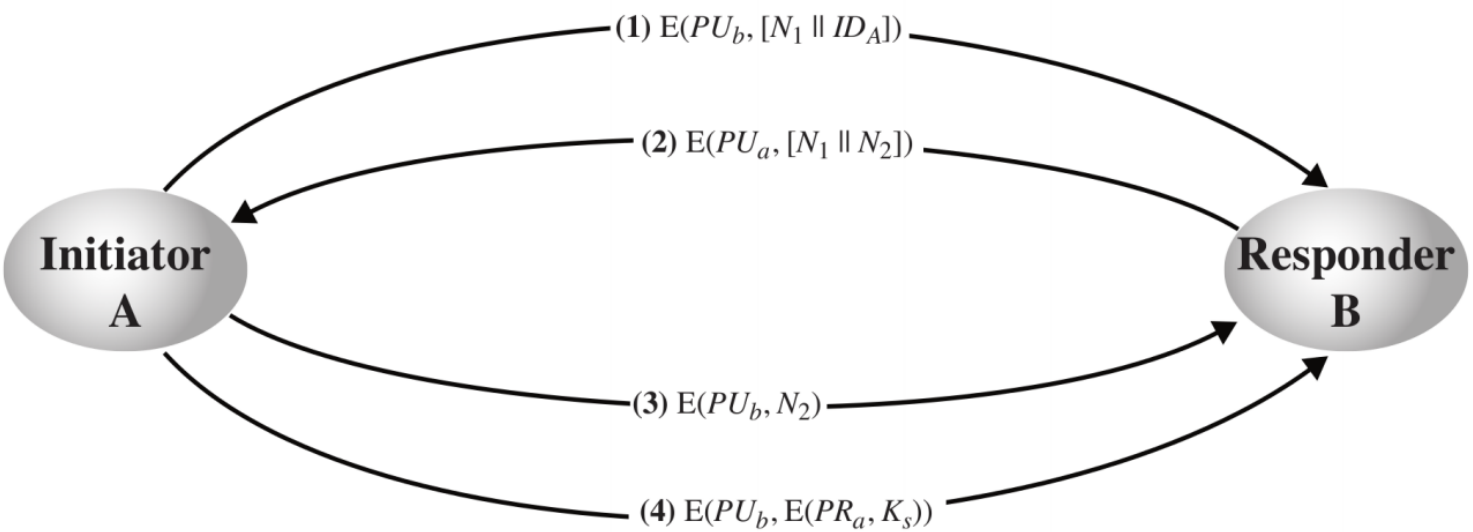
Simple Secret Key Distribution

- Simple: no keys prior to or after communication
  $\Rightarrow$ *the risk of compromise of the keys is minimal*
- Provides confidentiality for session key
- Only useful if attacker cannot modify/insert messages (passive attacks).
- Subject to man-in-the-middle attack



Secret Key Distribution with Confidentiality and Authentication

- Provides both confidentiality and authentication in exchange of secret key
- *Provide protection against both active and passive attacks.*



Hybrid Scheme: Public-Key Distribution of KDC Master Keys

- Another way to use public-key encryption to distribute secret keys is a hybrid approach
  - Retain the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key.
  - Public key used to distribute master keys
  - Efficient method of delivering master keys (rather than manual delivery)
  - Useful for large networks, widely distributed set of users with single KDC

Distribution of Public Keys

Distribution of Public Keys

- By design, public keys are made public
- Issue
  - *how to ensure public key of `A` actually belongs to `A` (and not someone pretending to be `A`)*
- Four approaches for distributing public keys
  1. Public announcement
  2. Publicly available directory
  3. Public-key authority
  4. Public-key certificates

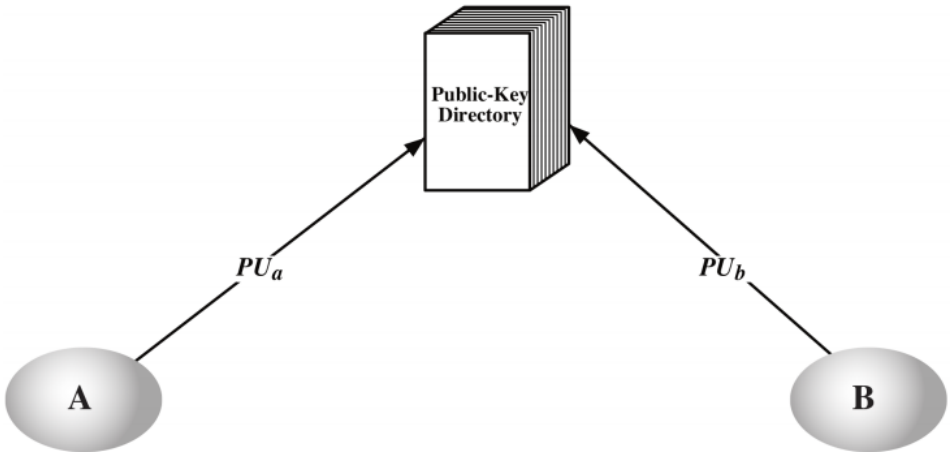
Public Announcements

- The point of public-key encryption is that the public key is public.
- Users distribute public keys to recipients or broadcast to community at large
  - Make public key available in open forum: newspaper, email signature, website, conference, ...
- Weakness: Major weakness is forgery
  - Anyone can announce a key pretending to be another user[2]
  - Until forgery is discovered, can masquerade as claimed user

[2] *anyone can create a key claiming to be someone else and broadcast it*

Publicly Available Directory

- Can obtain greater security by registering keys with a public directory of public keys.
- All users publish keys in central directory
- Directory must be trusted with properties
  - Contains *{name, public key}*
  - Participants register securely with directory
  - Participants can replace key at any time
  - Directory is periodically published
  - Directory can be accessed electronically
- Weakness: Still vulnerable to tampering or forgery[3].



[3] *When Alice uploaded here public key, how Bob knows that she uploaded her public key and no one else does that.*

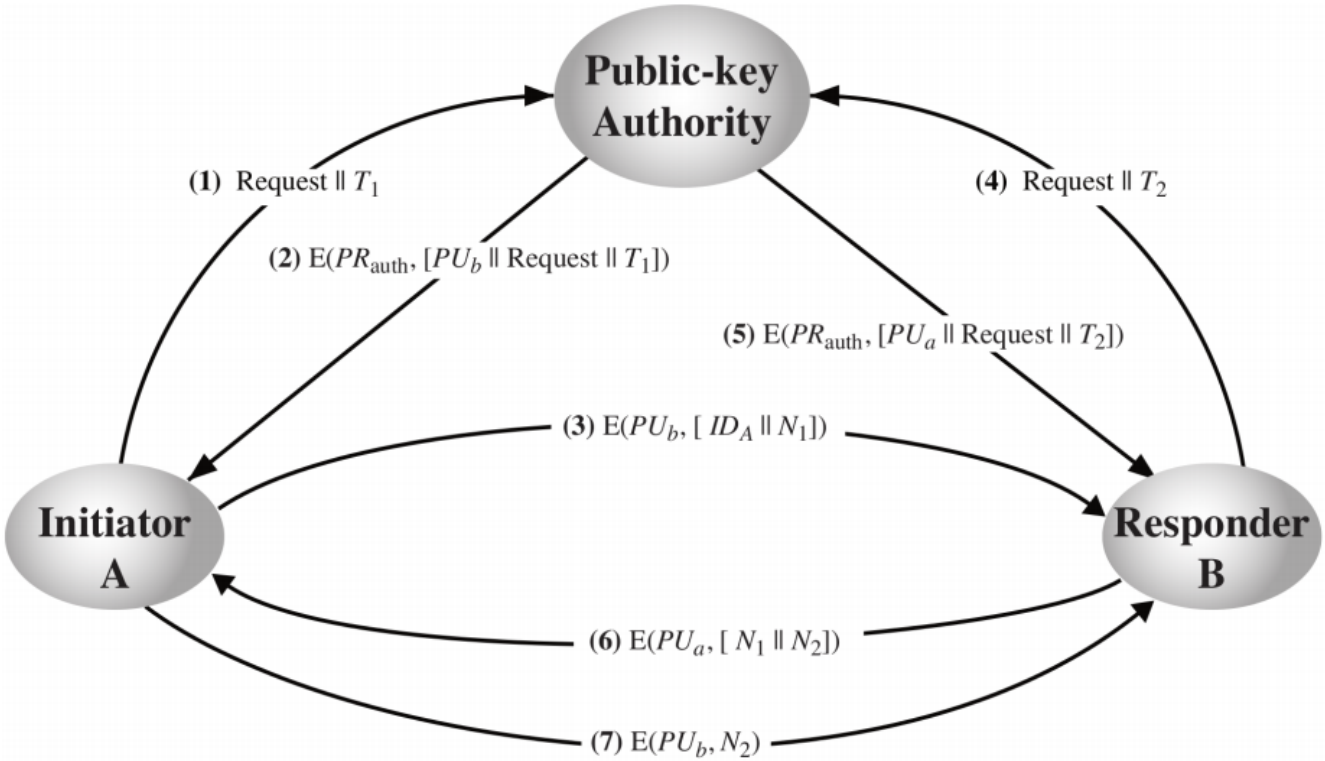
Public-Key Authority

- Similar to the public directory but, improve security by tightening control over distribution of keys from directory
- This scenario assumes the existence of a public/central authority that maintains a dynamic directory of public keys of all participants.
  - The public authority has its own (private key, public key) that it is using to communicate to users
- Assume each user has already securely published public-key at authority; each user knows authorities public key
- It requires users to know the public key for the directory, and that they interact with directory in real-time to obtain any desired public key securely

Public-Key Authority

- Whenever Alice wants to communicate with Bob she will go through the following protocol:
  1. Alice sends a timestamped message to the central authority with a request for Bob's public key (the time stamp is to mark the moment of the request)
  2. The authority sends back a message encrypted with its private key (for authentication) - message contains Bob's public key and the original message of Alice - this way Alice knows this is not a reply to an old request
  3. Alice starts the communication to Bob by sending him an encrypted message containing her name and a random number (to uniquely identify this transaction)
  4. Bob gets Alice's public key in the same way (step 1)
  5. Bob gets Alice's public key in the same way (step 2)
  6. Bob replies to Alice by sending an encrypted message with Alice's random number plus another random number (to uniquely identify the transaction)
  7. Alice replies once more encrypting Bob's random number

Public-Key Authority



Public-Key Authority

- First *5* messages are for key exchange; last *2* are authentication of users
- Although *7* messages, public keys obtained from authority can be cached
  - the initial five messages need be used only infrequently because both *A* and *B* can save the other's public key for future use
- Drawbacks: authority can be bottleneck
  - For any communication between any two users, the central authority must be consulted by both users to get the newest public keys
  - The central authority must be online 24 hours/day
  - If the central authority goes offline, all secure communications halt
  - This clearly leads to an undesirable bottleneck
- Alternative: public-key certificates

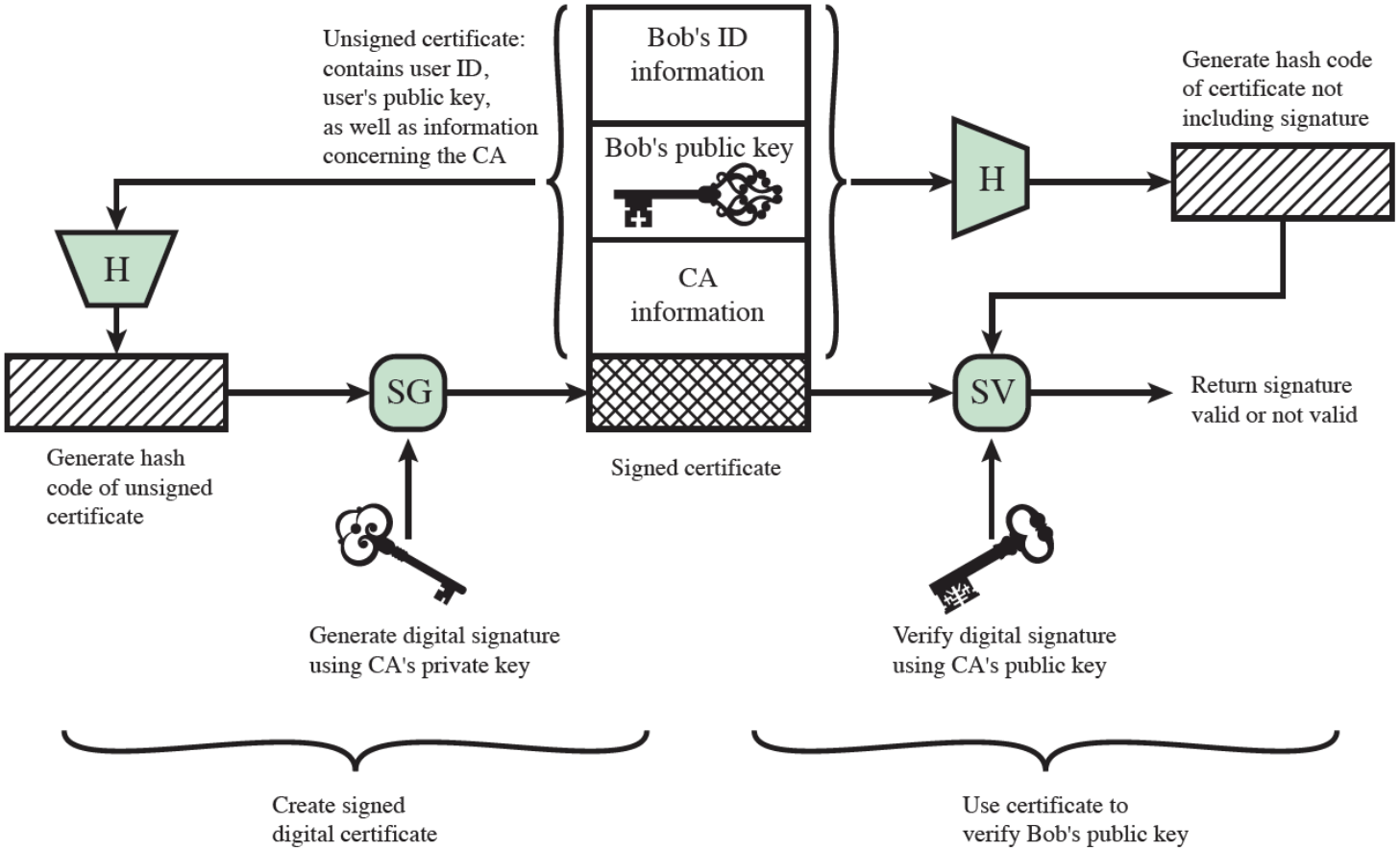
Certificate Authority (CA)

- Idea: have a trusted authority to certify one's own public key
  - Whenever Alice wants to start secure communication with Bob she sends him her public key certified by the central authority (encrypted with its private key) - Bob will know that it is indeed Alice because he will see her name in the certificate (he can decrypt it using the authority's public certificate)
  - To get her own certificate she must visit the authority (with her passport) or otherwise use some type of e-security - after that she may place it on the web because it is unforgeable
  - The certificate can be used for a period of time after which it must be changed - think of it as a credit card with an expiration date
  - The central authority does not have to be online all the time - it may not be online at all

Certificate Authority (CA)

- Certificate links a public key with the identity of the key's owner
- Certificate consists of:
  - Public key with the identity of the key's owner
  - Signed by a trusted third party
  - Typically the third party is certificate authority (CA) that is trusted by the user community
    - such as a government agency, telecommunications company, financial institution, or other trusted peak organization
- User can present their public key to the authority in a secure manner and obtain a certificate
  - User can then publish the certificate or send it to others
  - Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature.
- i.e., public keys sent to *CA* can be authenticated by *CA*; each user has certificate of *CA*

Public-Key Certificates



Certificates

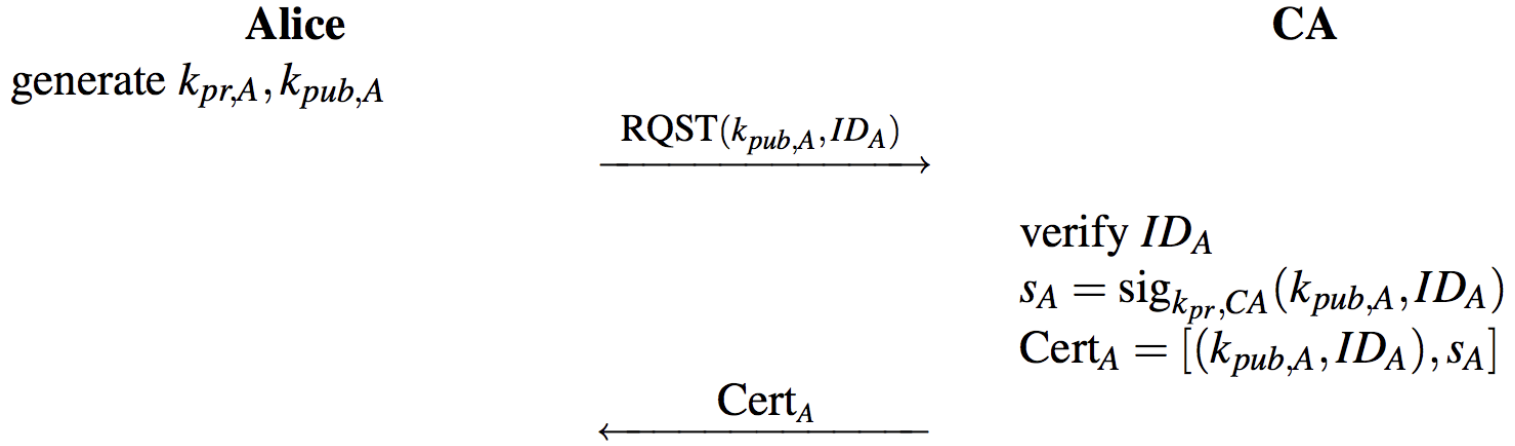
- In order to authenticate public keys (and thus, prevent the MIM attack), all public keys are digitally signed by a central trusted authority.
- Such a construction is called certificate

  certificate = public key + ID(user) + digital signature over public key and ID
- In its most basic form, a certificate for the key $k_{pub,A}$ of user Alice is:

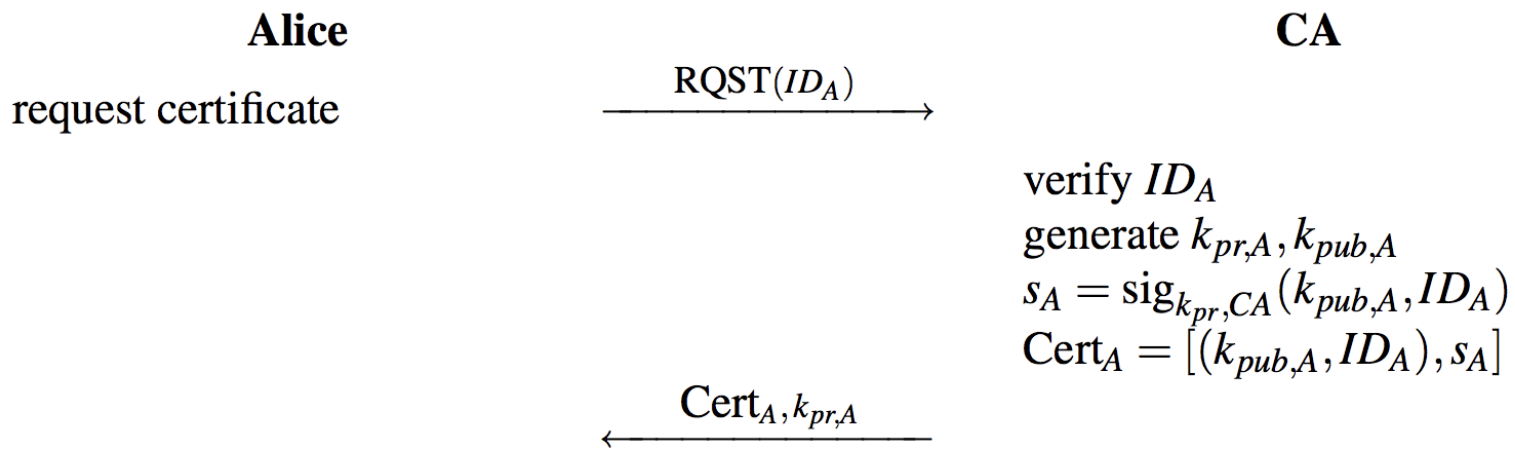
  $Cert(Alice) = (k_{pub,A}, ID(Alice), sig_{K_{pr,CA}}(k_{pub,A}, ID(Alice))$
- Certificates bind the identity of user to her public key
- The trusted authority that issues the certificate is referred to as certificate authority (CA)
- "Issuing certificates" means in particular that the CA computes the signature $sig_{K_{pr,CA}}(k_{pub,A})$ using its (super secret!) private key $k_{pr,CA}$.
- The party who receives a certificate, e.g., Bob, verifies Alice's public key using the public key of the CA

Certificate generation

- User provided keys:



- CA generated keys:



Diffie-Hellman Key Exchange (DHKE) with Certificates

Alice		**Bob**
$a = k_{pr,A}$		$b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$		$B = k_{pub,B} \equiv \alpha^B \bmod p$
$\text{Cert}_A = [(A, ID_A), s_A]$		$\text{Cert}_B = [(B, ID_B), s_B]$
	$\xrightarrow{\text{Cert}_A}$	
	$\xleftarrow{\text{Cert}_B}$	
verify certificate:		verify certificate:
$\text{ver}_{k_{pub,CA}}(\text{Cert}_B)$		$\text{ver}_{k_{pub,CA}}(\text{Cert}_A)$
compute session key:		compute session key:
$k_{AB} \equiv B^a \bmod p$		$k_{AB} \equiv A^b \bmod p$

Certificates

- Note that verification requires the public key of the CA for $ver_{k_{pub,CA}}$
- In principle, an attacker could run a MIM attack when $k_{pub,CA}$ is being distributed

  $\Rightarrow$ The public CA keys must also be distributed via an authenticated channel!
- Q: So, have we gained anything?

  After all, we try to protect a public key (e.g., a DH key) by using yet another public-key scheme (digital signature for the certificate)?
- YES! The difference from before (e.g., DHKE without certificates) is that we only need to distribute the public CA key once, often at the set-up time of the system
- Example: Most web browsers are shipped with the public keys of many CAs. The "authenticated channel" is formed by the (hopefully) correct distribution of the original browser software.

Certificates in the Real World: X.509 Certificates

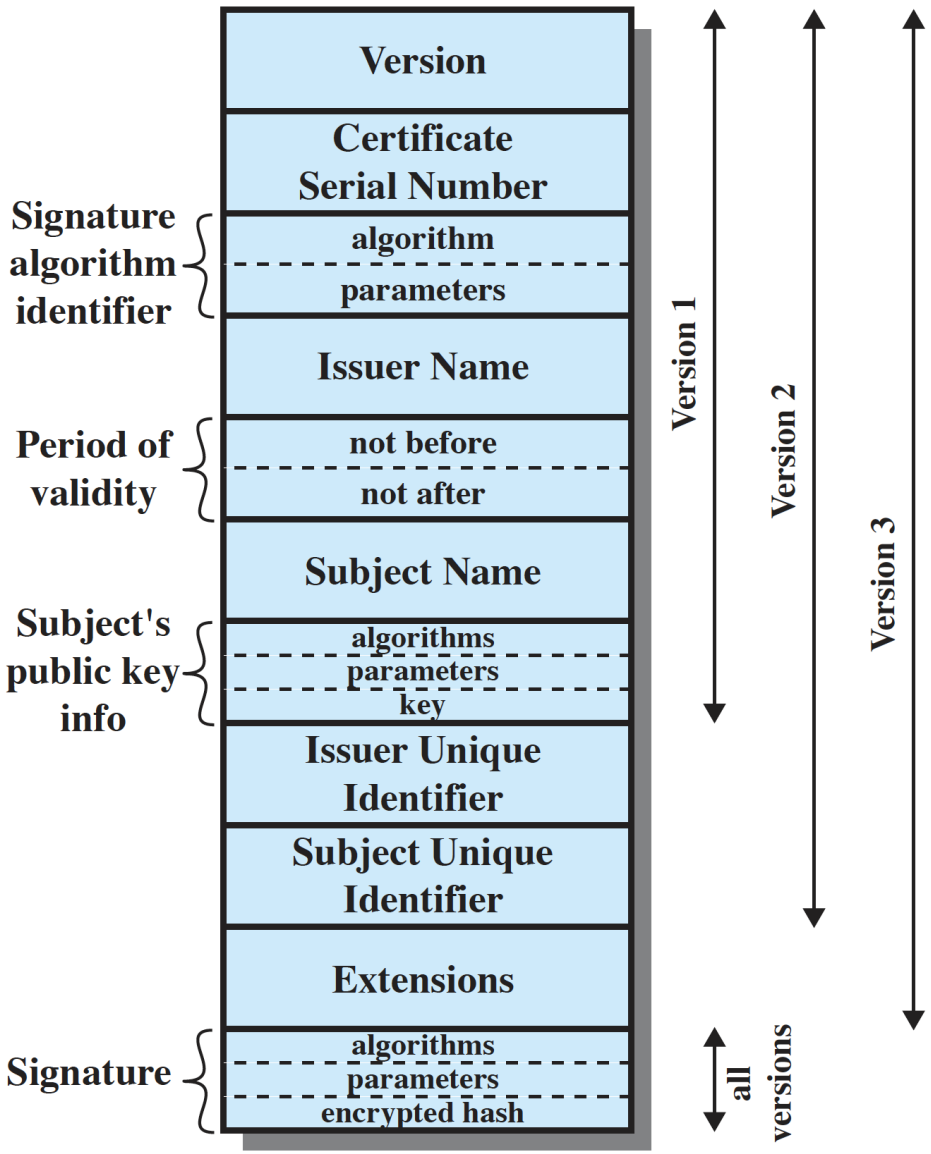
- Each user has a certificate, created by the Certificate Authority (CA)
- Certificates are stored in a public directory
- Certificates contain more information than just a public key and a signature
- X.509 is the most widely accepted format for public-key certificates
  - Provide access to the subject's public key
  - Confirm certificate's public key belongs to the certificate's subject
- Certificates are used in most network security applications, including:
  - IP security (IPSEC)
  - Secure sockets layer (SSL)/Transport layer security (TLS)
  - Secure electronic transactions (SET)
  - S/MIME (secure email)
  - eBusiness applications

- *X.509 refers to a widely used standard for digital certificates, specifically the format used in Public Key Infrastructure (PKI).*
- *Secure websites (HTTPS): The padlock you see in your browser bar for secure websites relies on X.509 certificates to verify the website's identity and encrypt communication.*

Certificates in the Real World: X.509 Certificates

- Created by a trusted Certification Authority (CA) and have the following elements:
  - Version
  - Serial number
  - Signature algorithm identifier
  - Issuer name
  - Period of validity
  - Subject name
  - Subject's public-key information
  - Issuer unique identifier
  - Subject unique identifier
  - Extensions
  - Signature

Certificates in the Real World: X.509 Certificates



Certificates in the Real World: X.509 Certificates

- *X.509* is based on the use of public-key cryptography and digital signatures.
- **Signature Algorithm:** Specified which signature algorithm is being used, e.g., RSA with SHA-1 or ECDSA with SHA-2, and with which parameters, e.g., the bit lengths.
- **Issuer Name:** There are many companies and organizations that issue certificates. This field specifies who generated the one at hand.
- **Period of Validity**: public key is not certified indefinitely but rather for a limited time
- **Subject Name**: This field contains what was called ID_A or ID_B in our earlier examples. It contains identifying information such as names of people or organizations.
- **Subject's Public Key**: The public key that is to be protected by the certificate. In addition, the algorithm and its parameters are stored.
- **Signature**: The signature of hash over all other fields of the certificate
- Note that there are two public-key schemes involved in every certificate: the one whose public key is protected by the certificate and the algorithm with which the certificate is signed

X.509- Version 3: Extensions

- X.509 Version 3 supports the notion of extensions, whereby anyone can define an extension and include it in the certificate.
- These extensions allow for more flexibility by including additional information within the certificate itself, such as key usage, subject alternative names, and certificate policies.
- Some common extensions in use today are:
  - **Key Usage**: Defines the intended purpose of the key (e.g., signing only, encryption).
  - **Subject Alternative Names (SAN)**: Allows associating additional identities (like domain names or email addresses) with the public key in the certificate. This is particularly useful for websites with multiple domains.
  - **Critical Extensions**: An extension can be marked "critical", indicating that software using the certificate must understand and process that extension. If not understood, the certificate should be rejected.

  - For example, if a certificate has the KeyUsage extension marked critical and set to "keyCertSign" then if this certificate is presented during SSL communication, it should be rejected, as the certificate extension indicates that the associated private key should only be used for signing certificates and not for SSL use.

Obtaining a Certificate

- User certificates generated by a *CA* have the following characteristics:
  - Any user with access to the public key of the *CA* can verify the user public key that was certified
  - No party other than the certification authority can modify the certificate without this being detected
- Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them
  - In addition, a user can transmit his or her certificate directly to other users
- Once *B* is in possession of *A*'s certificate, *B* has confidence that messages it encrypts with *A*'s public key will be secure from eavesdropping and that messages signed with *A*'s private key are unforgeable

Hierarchical CAs: Establishing Trust Levels

- Single CA System: Everyone trusts the one central authority.
- Hierarchical CAs: Introduce a layered trust model.
  - Root CA: The ultimate trusted entity at the top of the hierarchy.
  - Intermediate CAs:
    - CAs issued certificates by the root CA
    - They can further issue certificates to other entities (users, websites) or other subordinate CAs.
  - Notation: `Y<<X>>` certificate of `X` issued by `CA Y`

- *Single CA system is simple but not scalable. Hierarchies distribute trust and improve scalability.*

CA Directory with Forward and Reverse Certificates

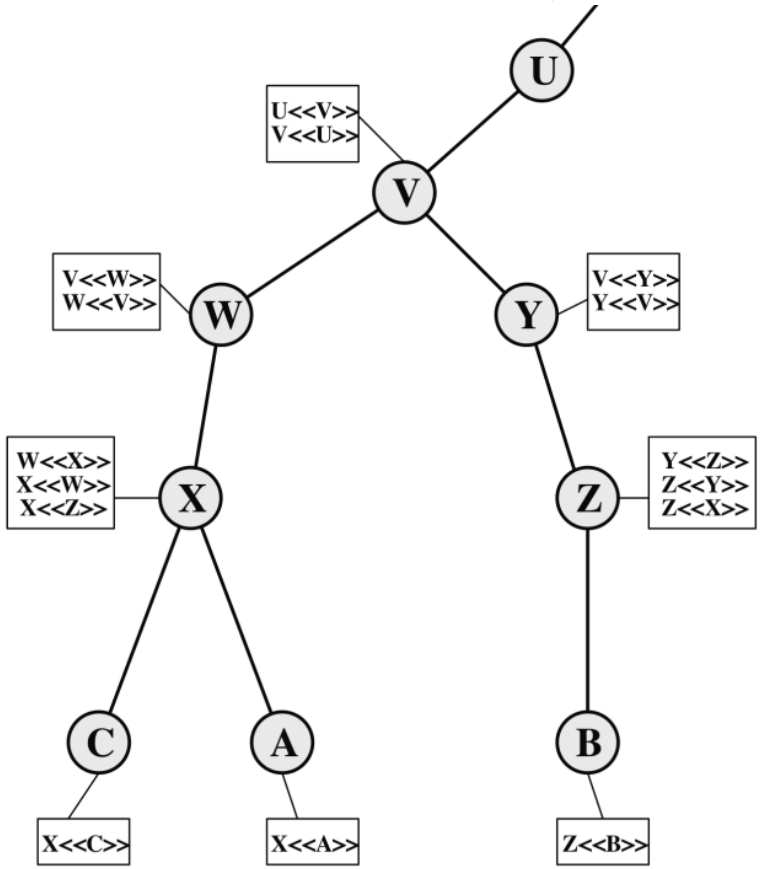
- Directory/system that stores information information about CAs and the certificates they hold.
- It typically includes:
  - CA details: Information about each CA, like its name, validity period, and capabilities
  - Forward Certificates: Certificates issued for a CA by other CAs (usually parent CA) to verify the CA's legitimacy.
  - Reverse Certificates: Certificates issued by a CA to other CAs (usually subordinates) to establishes chain of trust.
- This directory is not stored on each CA, but rather on a central server or a distributed system for:
  - Centralized management: Easier administration and information consistency.
  - Improved security: Enhanced control and protection of critical information.
  - Accessibility: Simplified access for users and CAs to verify trust relationships.

Certificate Authority (CA) Hierarchy

- If both users share a common CA, then they are assumed to know its public key.
- Otherwise, CAs must form a hierarchy.
- Use certificates linking members of the hierarchy to validate other CAs.
  - Each CA has certificates for clients (forward) and parent (backward).
  - Each client trusts parent's certificates.
- Enable verification of any certificate from one CA by users of all other CAs in the hierarchy.

Multiple Certificate Authorities

- Multiple *CA*'s can be arranged in hierarchy
- Notation: *Y*<<*X*>> certificate of *X* issued by *CA* *Y*

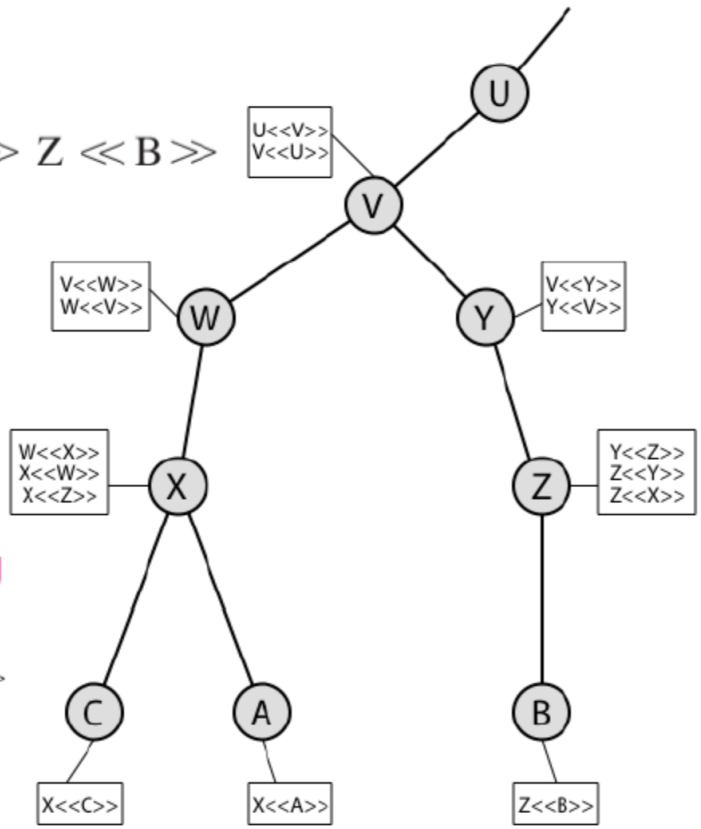


- The directory entry for each *CA* includes two types of certificates:
  - Forward certificates: Certificates of *X* generated by other *CA*s
  - Reverse certificates: Certificates generated by *X* that are the certificates of other *CA*s

CA Hierarchy Use

A can acquire the following certificates from the directory to establish a certification path to B:

X ≪W≫ W ≪V≫ V ≪Y≫ Y ≪Z≫ Z ≪B≫



B obtains A's public key from the following certification path:

Z ≪Y≫ Y ≪V≫ V ≪W≫ W ≪X≫ X ≪A≫

Certificate Revocation

- Each certificate includes a period of validity
  - Typically a new certificate is issued just before the expiration of the old one
- It may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:
  - The user's private key is assumed to be compromised
  - The user is no longer certified by this *CA*
  - The *CA*'s certificate is assumed to be compromised
- Each *CA* must maintain a list consisting of all revoked but not expired certificates issued by that *CA*
  - the Certificate Revocation List (CRL)
  - These lists should be posted on the directory
- Users should check certificates with CA's CRL

Major Players of CA

- As of November 2021 the survey company W3Techs, which collects statistics on certificate authority usage among the Alexa top 10 million and the Tranco top 1 million websites, lists the five largest authorities by absolute usage share as below.[4]

Rank	Issuer	Usage
1	IdenTrust	36.0%
2	DigiCert	16.9%
3	Sectigo (Comodo Cybersecurity)	15.3%
4	Let's Encrypt	11.1%
5	GoDaddy	5.6%

[4] https://en.wikipedia.org/wiki/Certificate_authority#Issuing_a_certificate

Public-Key Infrastructure

- Public Key Infrastructure (PKI): all you need to securely use public key crypto.

  i.e., the entire system that is formed by CAs together with the necessary support mechanisms
  - Key generation and management.
  - Certificate authority (CA) or authorities or other trust model.
  - Certificate revocation lists (CRLs), etc.

Lessons Learned

- Key agreement vs. Key transport
- Key freshness
- Key derivation
- Key establishment can be done using symmetric or asymmetric techniques
- Key establishment with a Key Distribution Center
- Certificates

Reading

- `Computer Security: Principles and Practice`
  - Chapter 2 (sections 2.4-2.6) and chapter 23 (section 23.2)
- `Understanding Cryptography: A Textbook for Students and Practitioners` available online
  - Chapter 13: Key Establishment