CS458: Introduction to Information Security

Notes 9: Key Establishment

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

October 25, 2018

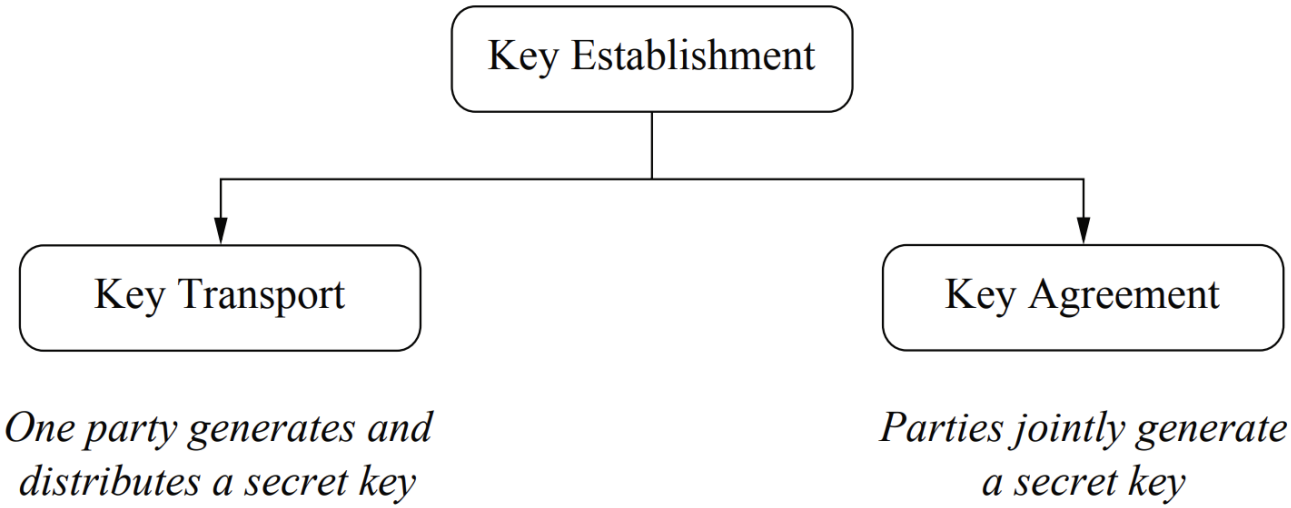
Slides: Modified from Christof Paar and Jan Pelzl & Ewa Syta

Outline

- Introduction
- The n^2 Key Distribution Problem
- Symmetric Key Distribution
- Asymmetric Key Distribution
  - Man-in-the-Middle Attack
  - Certificates
  - Public-Key Infrastructure

Classification of Key Establishment Methods

- key establishment deals with establishing a shared secret between two or more parties
- Methods for this can be classified into key transport and key agreement methods



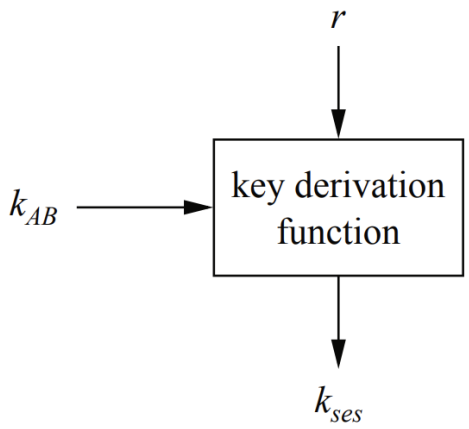
- In an ideal key agreement protocol, no single party can control what the key value will be.
- Additional variations beyond key transport and key agreement exist, including various forms of key update, such as key derivation

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

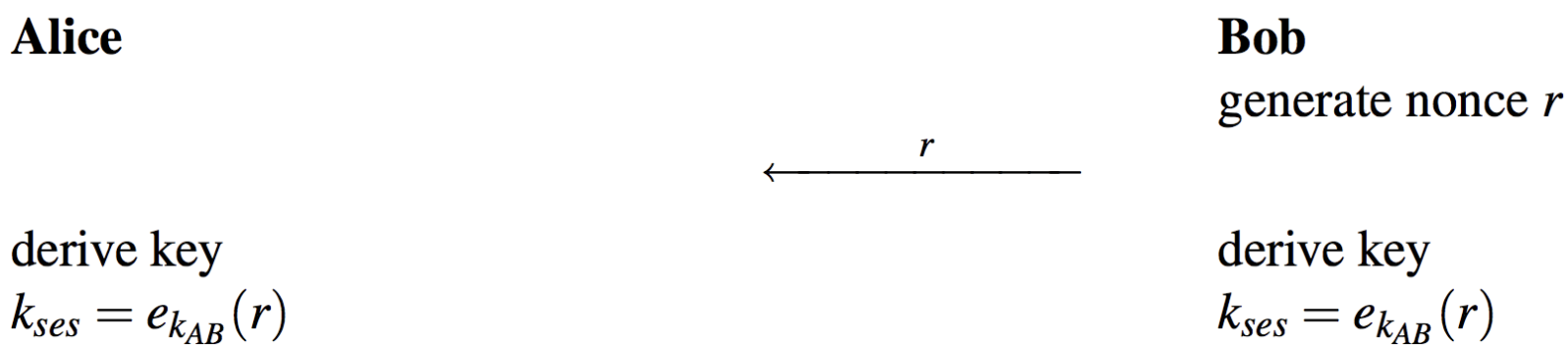
- In order to achieve key freshness, we need to generate new keys frequently.
- Rather than performing a full key establishment every time (which is costly in terms of computation and/or communication), we can derive multiple session keys k_{ses} from a given key k_{AB}.



- 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

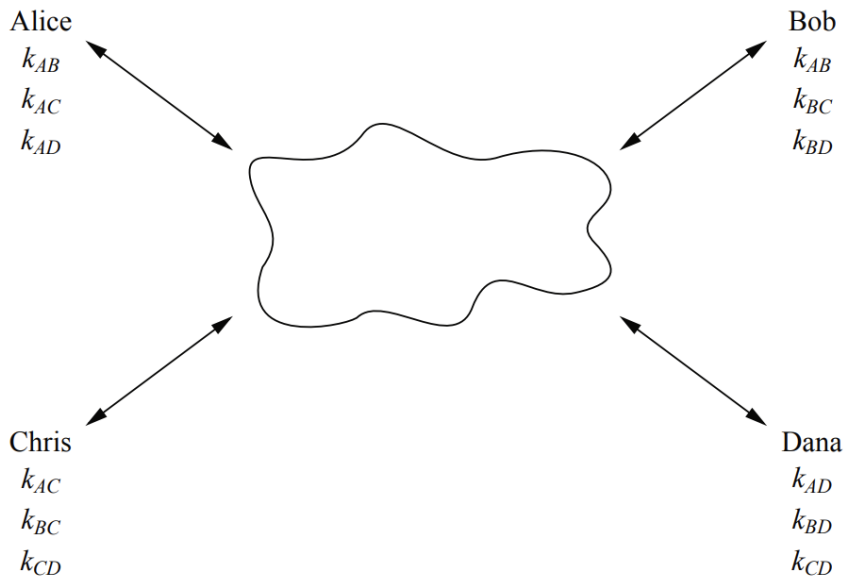
- The key derivation function is a computationally simple function, e.g., a block cipher or a hash function.
- it should be a one-way function
- Example for a basic protocol:



- 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: $K_{ses} = HMAC_{k_{AB}}(cnt)$

Naïve approach: The n^2 Key Distribution Problem

- We assumed the necessary keys for symmetric algorithms are distributed via a "secure channel"
- key predistribution since it involves different mode of communication, e.g., the key is transmitted via a phone line, or in a letter.
- **Simple situation**: Network with n users. Every user wants to communicate securely with every of the other $n-1$ users.
- **Naïve approach**: Every pair of users obtains an individual key pair
- Example: Keys in a network with $n = 4$ users:



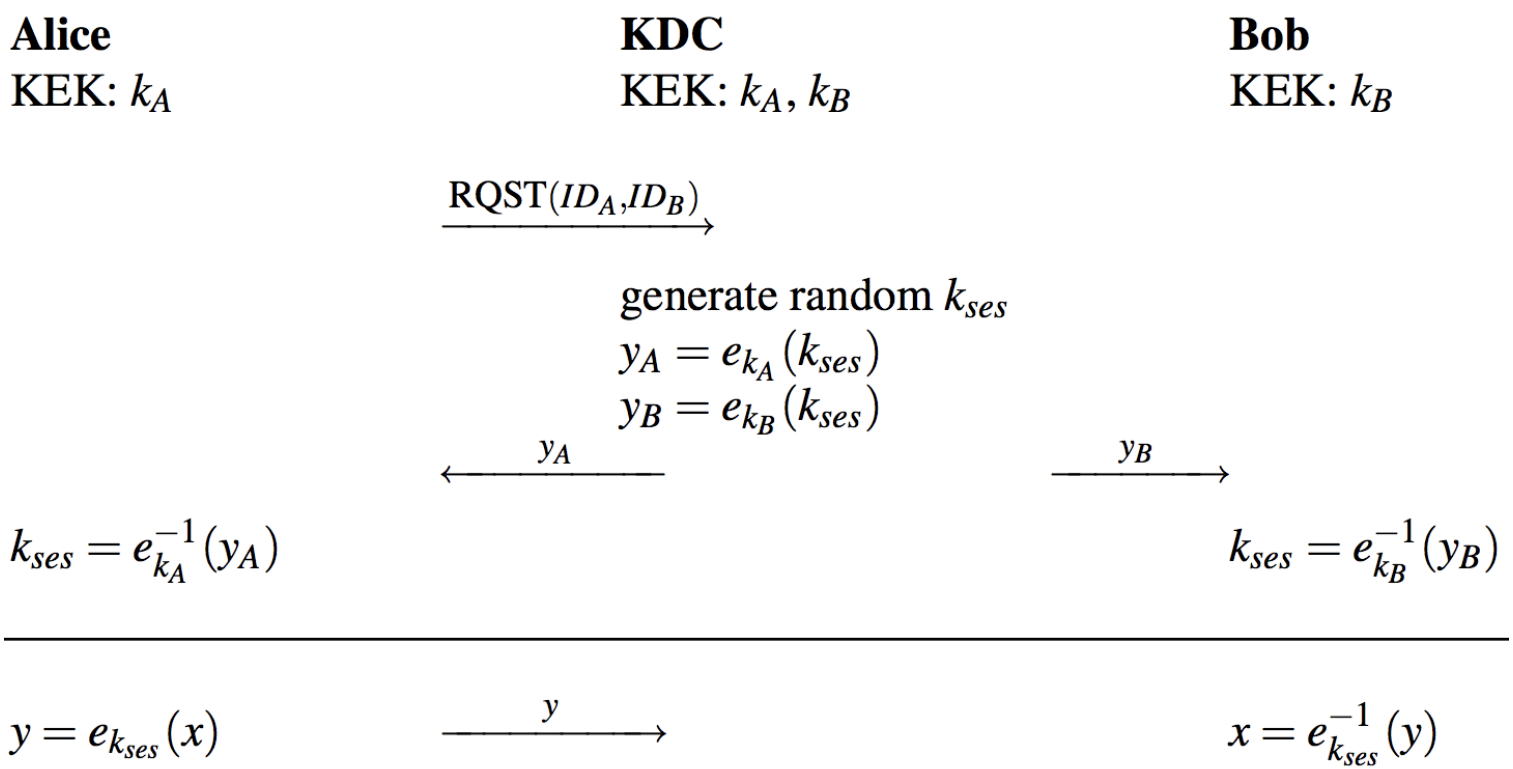
The n^2 Key Distribution Problem

- Shortcomings
  - There are $n(n-1) \approx n^2$ keys stored in the system
  - There are $\frac{n(n-1)}{2}$ pair keys

  - 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
  - Example: mid-size company with 750 employees pause $750 \times 749 = 561,750$ keys must be distributed securely.

Key Establishment with Key Distribution Center (KDC)

- Key Distribution Center (KDC): Central party, trusted by all users
- **Idea**:
  - Central "trusted authority" (KDC) that shares one key key encryption key (*KEK*) with every user
  - KDC sends session keys to users which are encrypted with *KEK*s



- Example: Use AES with *KEK*s and fast stream cipher with **short-lived** k_{ses}

Key Establishment with Key Distribution Center (KDC)

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

Key Establishment with Key Distribution Center (KDC)

- How does Bob know he is talking to Alice?
  - Replay attack: Eve breaks an old session key, resends old messages y_A and y_B, and can then decrypt all messages
  - Key confirmation attack: Eve intercepts Alice's request $RQST(ID_A, ID_B)$ and sends to KDC $RQST(ID_A, ID_e)$. Alice believes that she communicates with Bob and Eve will decrypt her messages - there is **no key confirmation**
- Kerberos (a popular authentication and key distribution protocol) is based on KDCs

Kerberos - Authentication and Key Distribution Standard Protocol

- Advanced protocol that protects against both replay and key confirmation attacks
- Its main purpose is to provide user authentication in computer networks
- Based on a KDC, which is named the “authentication sever” 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
$y = e_{k_{ses}}(x)$	$\xrightarrow{y}$			$x = e^{-1}_{k_{ses}}(y)$

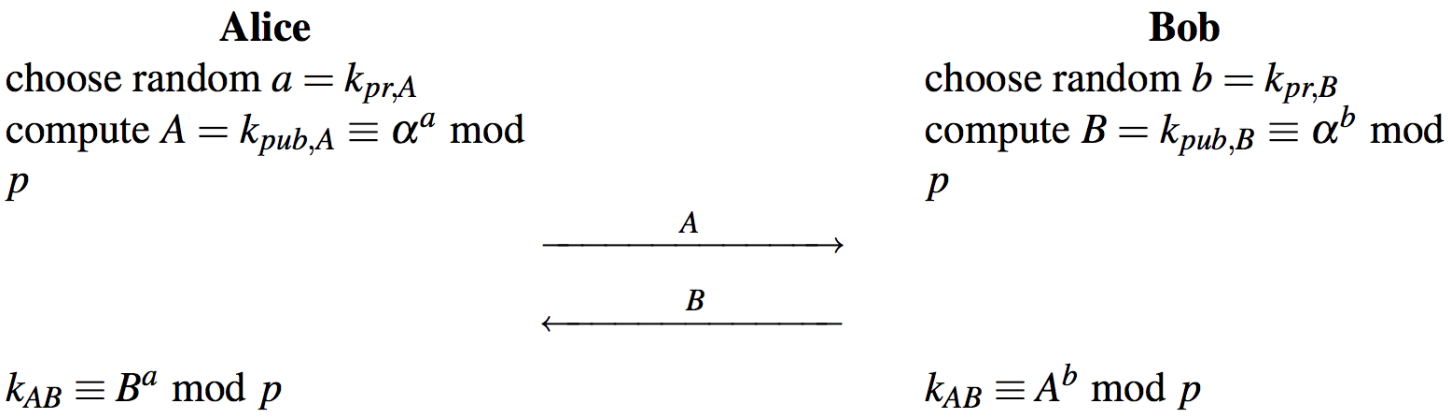
- Protects against both attacks
- Adds **Lifetime** T, **Time stamp** T_s and **Nonce** (only KDC can send new keys)

Key Establishment 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: when a new user joins - public key cipher for new key transport
- 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.

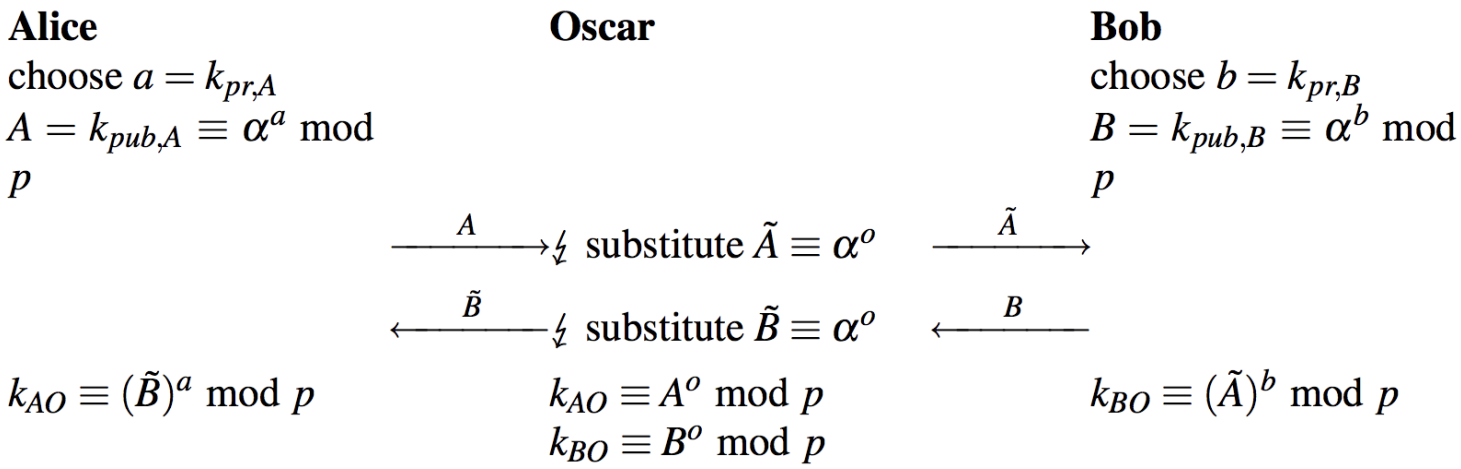
Key Establishment Using Asymmetric Techniques

Recall: Diffie-Hellman Key Exchange (DHKE)



- Widely used in practice
- If the parameters are chosen carefully (especially a prime $p > 2^{1024}$), the DHKE is secure against passive (i.e., listen-only) 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 the Man-in-the-Middle (MITM) 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.
- The attack is also known as Janus attack[1]
- Q: What makes the MITM 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*

[1] in reference to the roman two-headed god of gates

Certificates

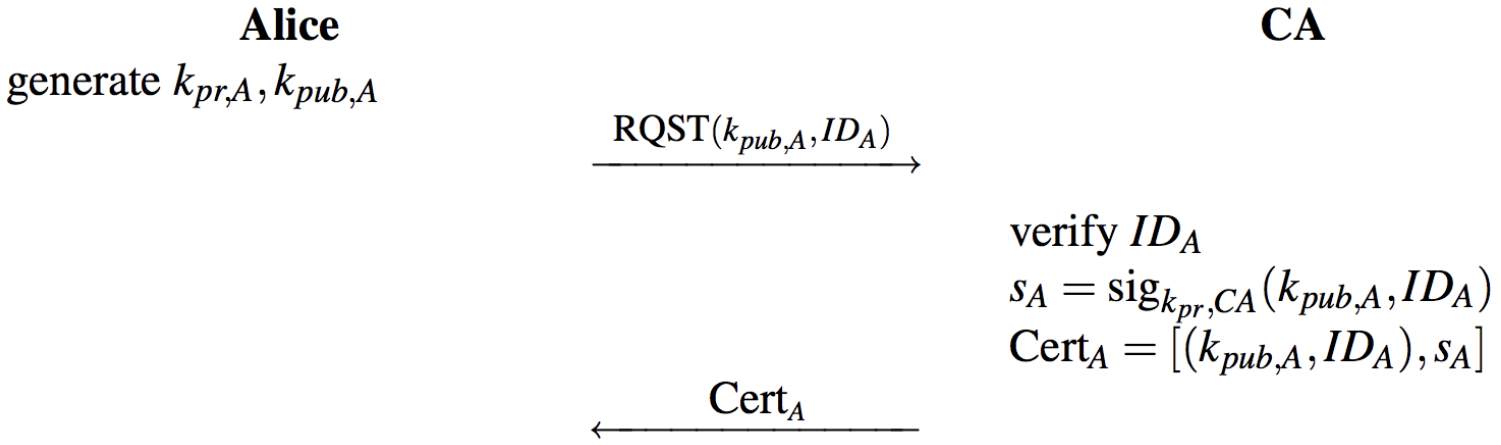
- In order to authenticate public keys (and thus, prevent the MIM attack), all public keys are digitally signed by a central trusted authority.
- Such a construction is called certificate

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

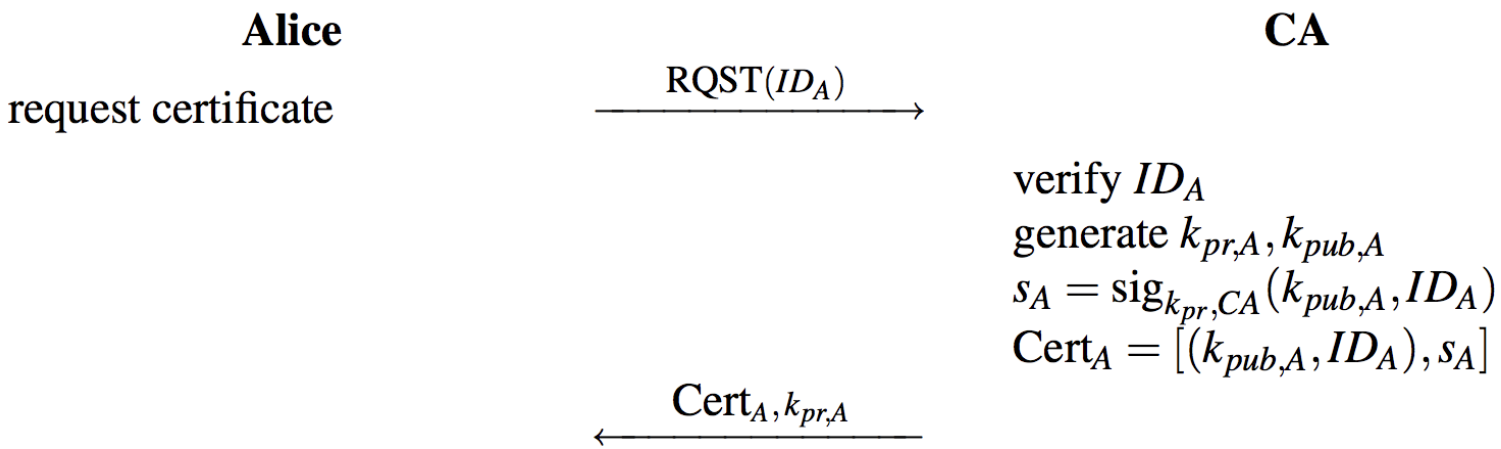
  $Cert(Alice) = (k_{pub}, ID(Alice), sig_{K_{CA}}(k_{pub}, ID(Alice))$
- Certificates bind the identity of user to her public key
- The trusted authority that issues the certificate is referred to as certifying authority (CA)
- "Issuing certificates" means in particular that the CA computes the signature $sig_{K_{CA}}(k_{pub})$ using its (super secret!) private key k_{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**
$a = k_{pr,A}$		$b = k_{pr,B}$
$A = k_{pub,A} \equiv \alpha^a \bmod p$		$B = k_{pub,B} \equiv \alpha^B \bmod p$
$\text{Cert}_A = [(A, ID_A), s_A]$		$\text{Cert}_B = [(B, ID_B), s_B]$
	$\text{Cert}_A \longrightarrow$	
	$\longleftarrow \text{Cert}_B$	
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.

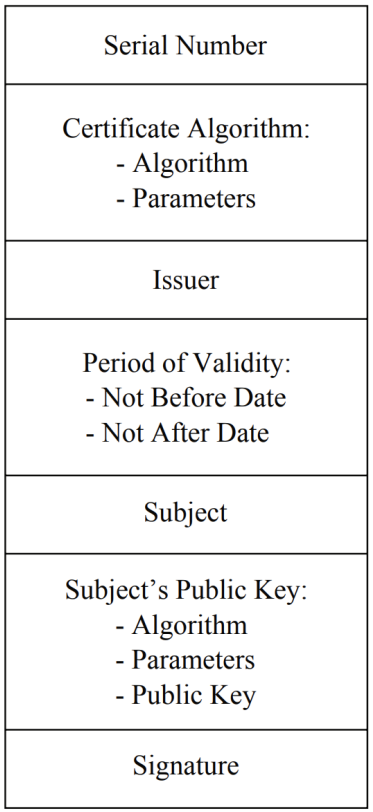
Public-Key Infrastructure

- 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.

Certificates in the Real World: X.509 Certificates

- Certificates contain more information than just a public key and a signature
- X.509 is an important standard for network authentication services, and the corresponding certificates are widely used for Internet communication, i.e., in S/MIME, IPsec and SSL/TLS



Certificates in the Real World: X.509 Certificates

- **Certificate 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:** 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**: 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 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

Remaining Issues with PKIs

- There are additional problems when certificates are to be used in systems with a large number of participants. The more pressing ones are:
  1. Users communicate with other whose certificates are issued by different CAs
     - This requires cross-certification of CAs, e.g., CA1 certifies the public-key of CA2. If Alice trusts "her" CA1, cross-certification ensures that she also trusts CA2. This is called a "chain of trust" and it is said that "trust is delegated".
  2. Certificate Revocation Lists (CRLs)
     - Another real-world problem is that certificates must be revoked, e.g., if a smart card with certificate is lost or if a user leaves an organization. For this, CRLs must be sent out periodically (e.g., daily) which is a burden on the bandwidth of the system.

Lessons Learned

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