# Digital Certificates and Public Key Infrastructure (PKI)

Subhabrata Samajder

CREST CRYPTO SUMMER SCHOOL (CCSS), 2025

tcg crest
Inventing Harmonious Future

$24^{th}$ June, 2025

# Outline

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.
- **Note:** PKC works once public keys are securely distributed.

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.
- **Note:** PKC works once public keys are securely distributed.
- We will show that PKC itself can be used to securely distribute public keys.

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.
- **Note:** PKC works once public keys are securely distributed.
- We will show that PKC itself can be used to securely distribute public keys.
- May sound circular, but it is not.

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.
- **Note:** PKC works once public keys are securely distributed.
- We will show that PKC itself can be used to securely distribute public keys.
- May sound circular, but it is not.
- Once a single public key, belonging to a trusted party, is distributed in a secure fashion, that key can be used to "bootstrap" the secure distribution of arbitrarily many other public keys.

# Digital Signatures

- One of the primary applications is the secure distribution of public keys.
- **Note:** PKC works once public keys are securely distributed.
- We will show that PKC itself can be used to securely distribute public keys.
- May sound circular, but it is not.
- Once a single public key, belonging to a trusted party, is distributed in a secure fashion, that key can be used to "bootstrap" the secure distribution of arbitrarily many other public keys.
- Thus, at least in principle, the problem of secure key distribution need only be solved once.

# Digital Certificates

First described by Kohnfelder in 1978 in his undergraduate thesis.

**Definition (Digital Certificates)**

A ***digital certificate or identity certificate or public key certificate*** is simply a signature binding an entity to some public key.

# PKI

Charlie

# PKI

Charlie

Gen: $(pk_C, sk_C)$

# PKI

Charlie

Gen: $(pk_C, sk_C)$

Bob

# PKI

Charlie
Gen: $(pk_C, sk_C)$

Bob
Gen: $(pk_B, sk_B)$
(can be for sig. or enc. scheme)

# PKI

knows that $pk_B$ is Bob's public key

Charlie

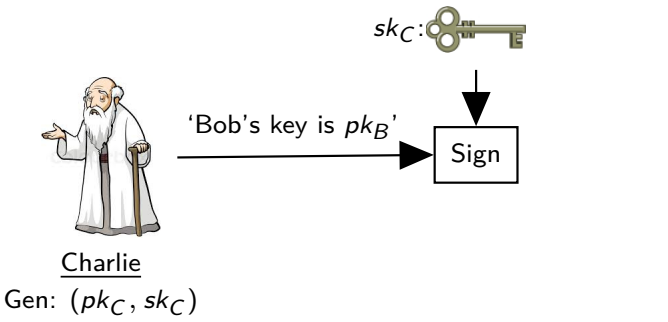
Gen: $(pk_C, sk_C)$

Bob

Gen: $(pk_B, sk_B)$

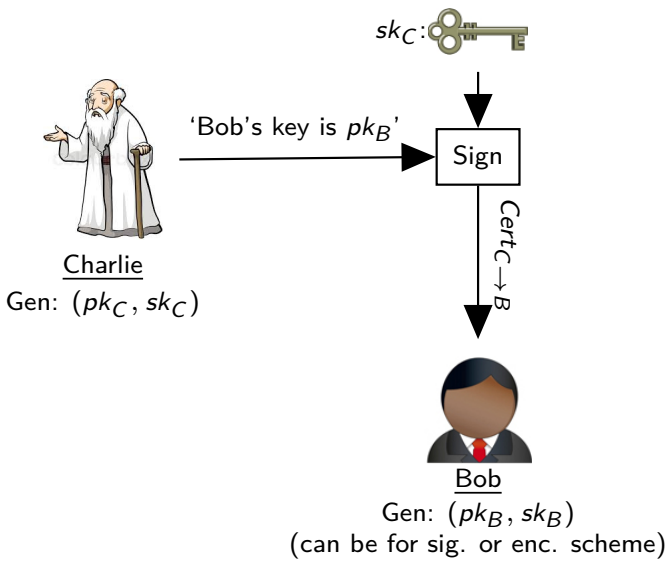
(can be for sig. or enc. scheme)

# PKI



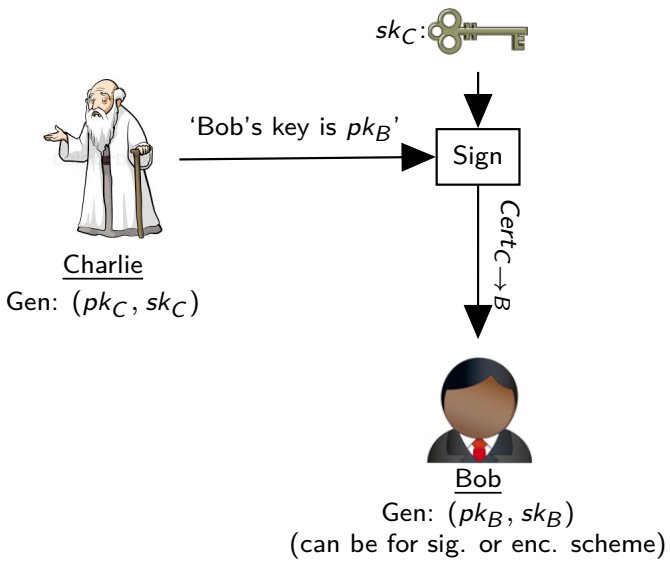

Bob
Gen: $(pk_B, sk_B)$
(can be for sig. or enc. scheme)

# PKI



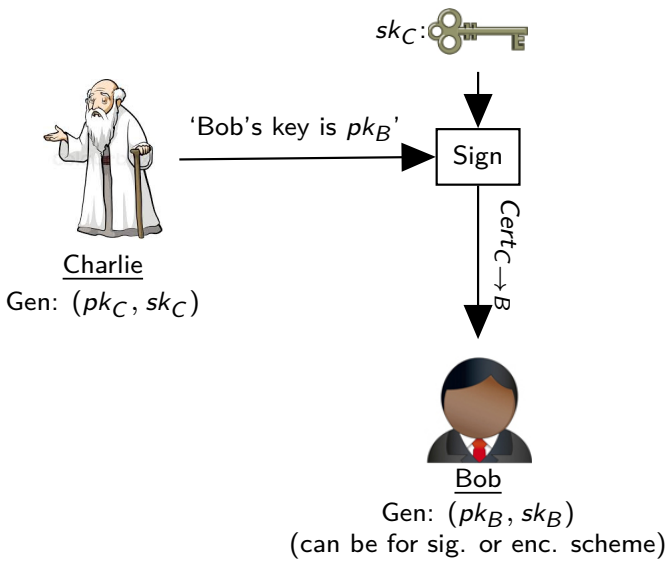

- $Cert_{C \to B}$: A certificate for Bob's public key issued by Charlie.

# PKI



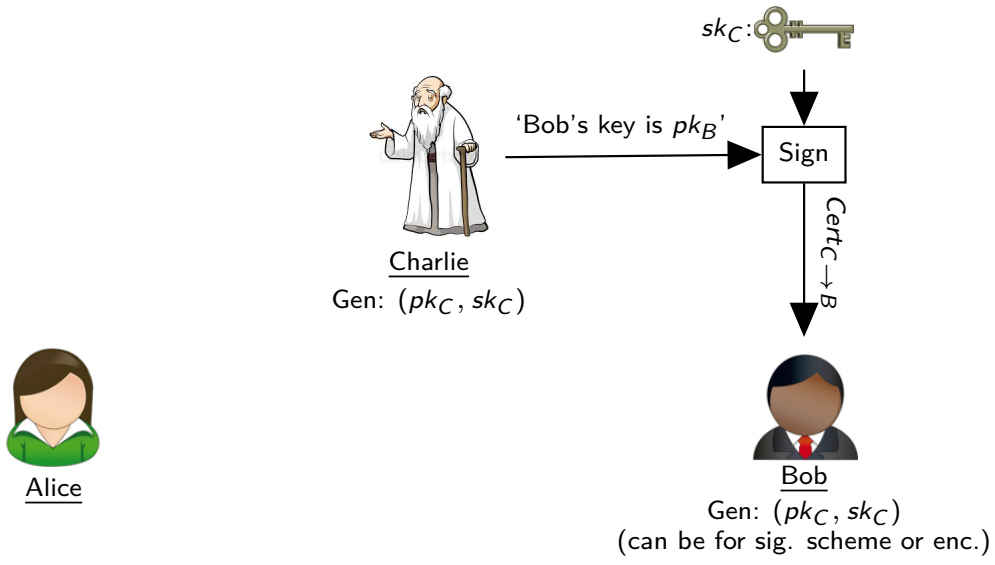

- $Cert_{C \to B}$: A certificate for Bob's public key issued by Charlie.
- A certificate should unambiguously identify the party holding a particular public key rather than "Bob".

# PKI

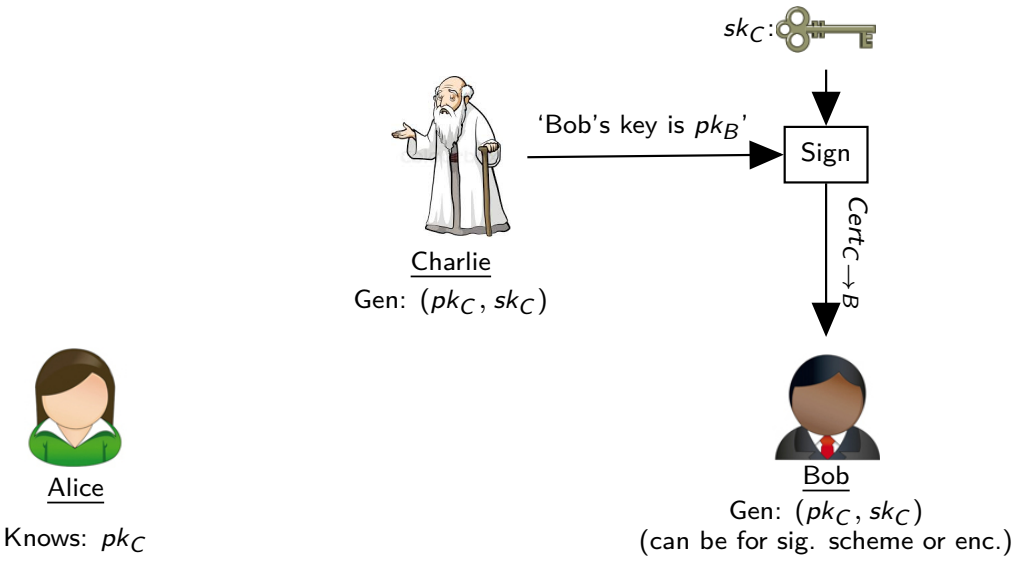



- $Cert_{C \to B}$: A certificate for Bob's public key issued by Charlie.
- A certificate should unambiguously identify the party holding a particular public key rather than "Bob".
- **Example:** Bob's full name and email address, or the URL of Bob's website.

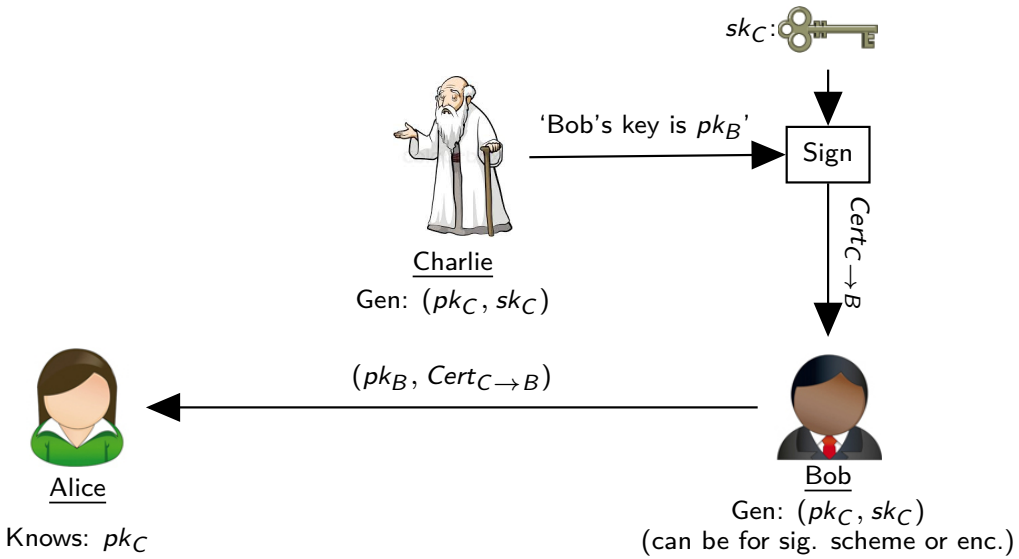
# PKI


$sk_C$:
'Bob's key is $pk_B$'
Sign
$Cert_{C \to B}$
Charlie
Gen: $(pk_C, sk_C)$
Alice
Bob
Gen: $(pk_C, sk_C)$
(can be for sig. scheme or enc.)

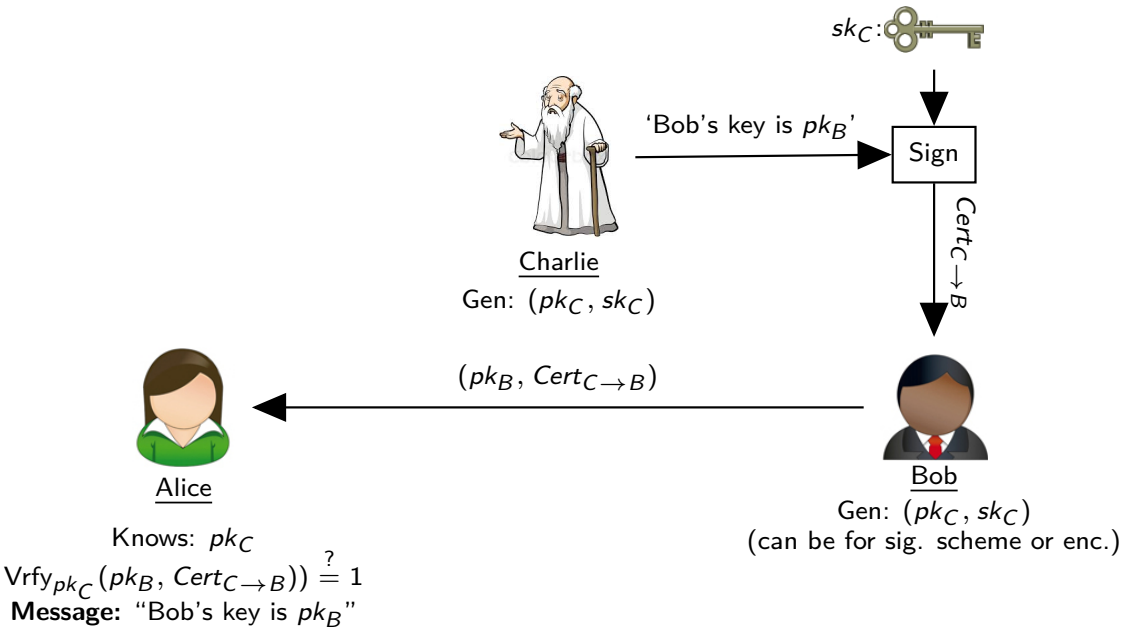
# PKI


$sk_C$:
'Bob's key is $pk_B$'
Sign
$Cert_{C \to B}$
Charlie
Gen: $(pk_C, sk_C)$
Alice
Knows: $pk_C$
Bob
Gen: $(pk_C, sk_C)$
(can be for sig. scheme or enc.)

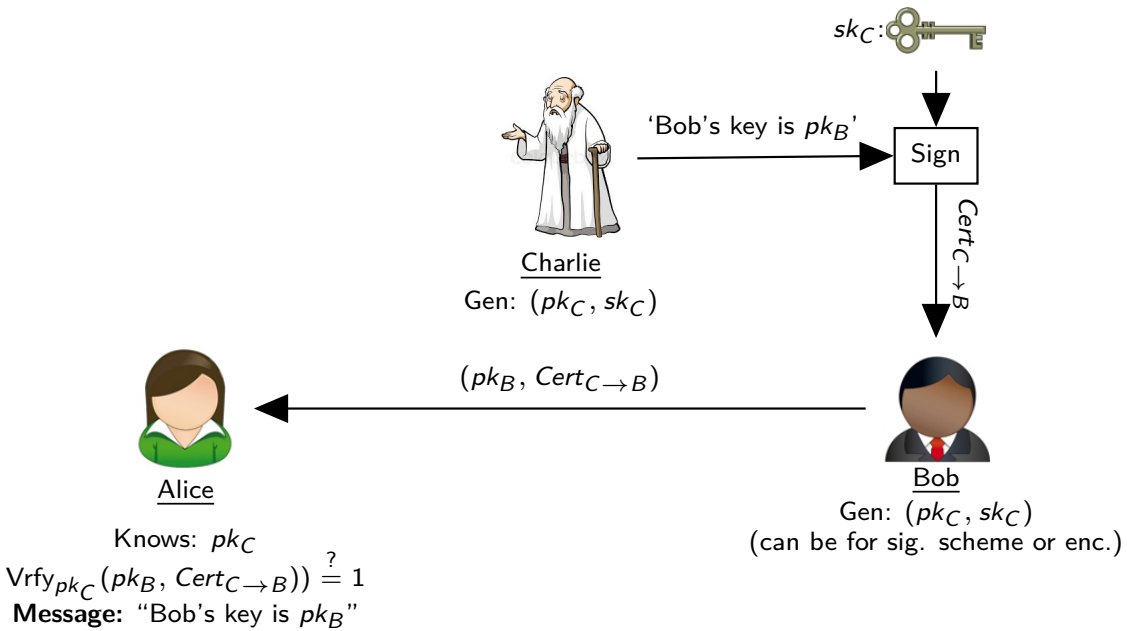
# PKI


$sk_C$:
'Bob's key is $pk_B$'
Sign
$Cert_{C \to B}$
Charlie
Gen: $(pk_C, sk_C)$
$(pk_B, Cert_{C \to B})$
Alice
Knows: $pk_C$
Bob
Gen: $(pk_C, sk_C)$
(can be for sig. scheme or enc.)

# PKI

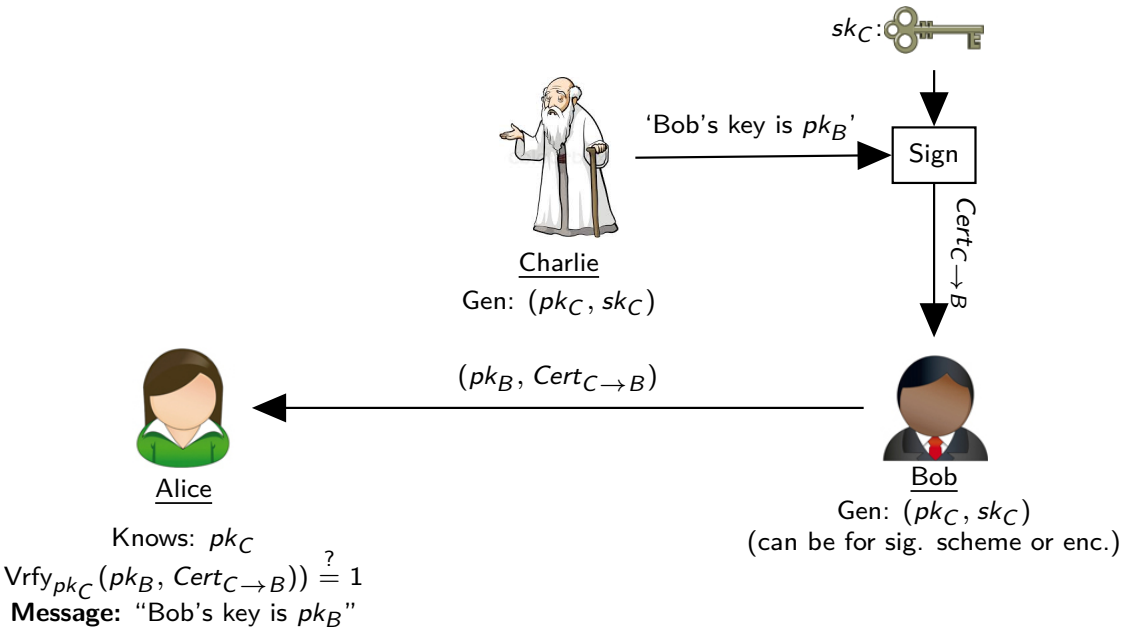

$sk_C$:
'Bob's key is $pk_B$'
Sign
$Cert_{C \to B}$
Charlie
Gen: $(pk_C, sk_C)$
$(pk_B, Cert_{C \to B})$
Alice
Knows: $pk_C$
$\text{Vrfy}_{pk_C}(pk_B, Cert_{C \to B}) \stackrel{?}{=} 1$
**Message:** "Bob's key is $pk_B$"
Bob
Gen: $(pk_C, sk_C)$
(can be for sig. scheme or enc.)

# PKI



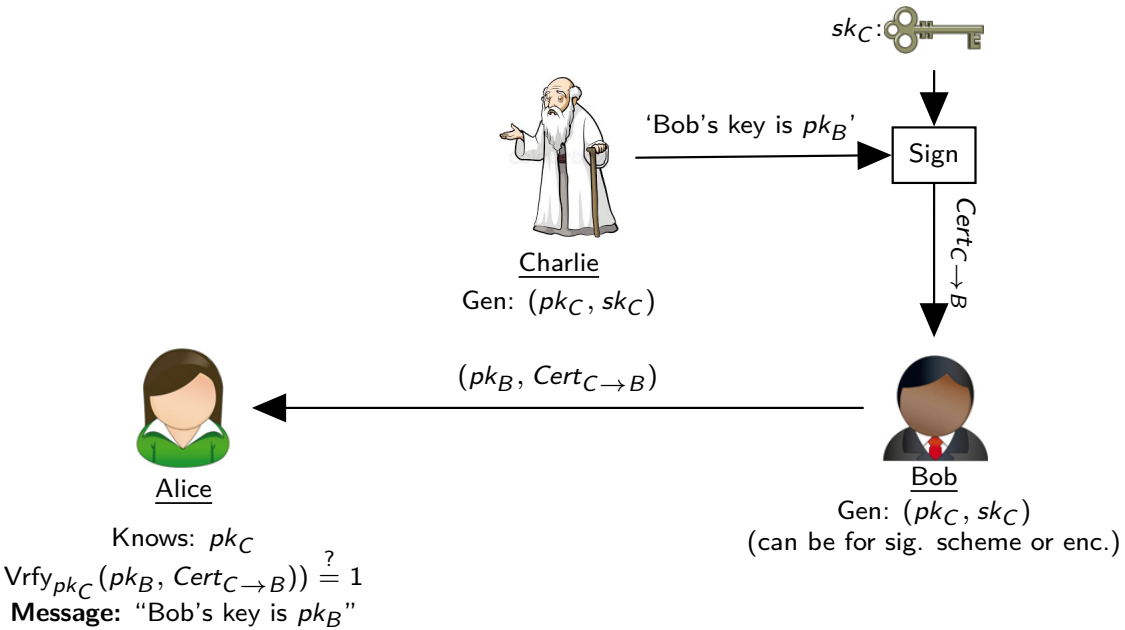

- If verification succeeds, then Alice now knows that Charlie has signed the indicated message.

# PKI




- If verification succeeds, then Alice now knows that Charlie has signed the indicated message.
- If Alice trusts Charlie, she can accept $pk_B$ as Bob's legitimate public key.

# PKI




- If verification succeeds, then Alice now knows that Charlie has signed the indicated message.
- If Alice trusts Charlie, she can accept $pk_B$ as Bob's legitimate public key.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.
- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then
  - the adversary will be unable to generate a valid certificate linking Bob to any other public key $pk'_B$.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then
  - the adversary will be unable to generate a valid certificate linking Bob to any other public key $pk'_B$.
  - Unless Charlie had previously signed some other certificate linking Bob with $pk'_B$.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then
  - the adversary will be unable to generate a valid certificate linking Bob to any other public key $pk'_B$.
  - Unless Charlie had previously signed some other certificate linking Bob with $pk'_B$.
  - In which case this is anyway not much of an attack.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then
  - the adversary will be unable to generate a valid certificate linking Bob to any other public key $pk'_B$.
  - Unless Charlie had previously signed some other certificate linking Bob with $pk'_B$.
  - In which case this is anyway not much of an attack.

- **Assumptions:**
  1. Charlie is honest.

# PKI

- **Note:** All communication between Bob and Alice can occur over an insecure and unauthenticated channel.

- If an active adversary interferes with the transmission of ($pk_B$, $\text{Cert}_{C \to B}$) from Bob to Alice, then
  - the adversary will be unable to generate a valid certificate linking Bob to any other public key $pk'_B$.
  - Unless Charlie had previously signed some other certificate linking Bob with $pk'_B$.
  - In which case this is anyway not much of an attack.

- **Assumptions:**
  1. Charlie is honest.
  2. Charlie's private key has not been compromised.

# PKI

**Note:** Many details have been omitted!

**Example:**

1. How Alice learns $pk_C$ in the first place?

# PKI

**Note:** Many details have been omitted!

**Example:**

1. How Alice learns $pk_C$ in the first place?

2. How Charlie can be sure that $pk_B$ is Bob's public key?

# PKI

**Note:** Many details have been omitted!

**Example:**

1. How Alice learns $pk_C$ in the first place?
2. How Charlie can be sure that $pk_B$ is Bob's public key?
3. How Alice decides whether to trust Charlie?

# PKI

- Public-key infrastructure (PKI): Full specification of these details (and others) in order to enable the widespread distribution of public keys.
- Different PKI models have been suggested.
- We will mention a few of the more popular ones.
- Our treatment will be at a relatively higher level.

# Outline

# Single CA

- The simplest PKI.

# Single CA

- The simplest PKI.
- Assumes a single CA.

# Single CA

- The simplest PKI.
- Assumes a single CA.
  - Completely trusted by everybody.

# Single CA

- The simplest PKI.
- Assumes a single CA.
  - Completely trusted by everybody.
  - Issues certificates for everyone's public key.

# Single CA

- The simplest PKI.
- Assumes a single CA.
  - Completely trusted by everybody.
  - Issues certificates for everyone's public key.
  - Typically not a person.

# Single CA

- The simplest PKI.
- Assumes a single CA.
  - Completely trusted by everybody.
  - Issues certificates for everyone's public key.
  - Typically not a person.
  - **Example:**
    - A company whose business it is to certify public keys.
    - A government agency.
    - Could be a department within an organization (intended to be used by people within the organization).

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.
  - This needs to be carried out in a secure fashion.

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.
  - This needs to be carried out in a secure fashion.
  - Parties having an incorrect version of $pk_{CA}$ will not be able to obtain authentic copies of anyone else's public key.

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.
  - This needs to be carried out in a secure fashion.
  - Parties having an incorrect version of $pk_{CA}$ will not be able to obtain authentic copies of anyone else's public key.
  - Thus $pk_{CA}$ must be distributed over an authenticated channel.

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.
  - This needs to be carried out in a secure fashion.
  - Parties having an incorrect version of $pk_{CA}$ will not be able to obtain authentic copies of anyone else's public key.
  - Thus $pk_{CA}$ must be distributed over an authenticated channel.
  - **Easiest way:** Via physical means.
    - CA is within an organization: Employee can obtain an authentic copy of $pk_{CA}$ directly from the CA on their first day of work.
    - CA is a company: Users need to go to the company at some point and, say, pick up a USB stick that contains the CA's public key.

# Single CA

- For CA's services one would have to obtain a legitimate copy of the CA's public key $pk_{CA}$.
  - This needs to be carried out in a secure fashion.
  - Parties having an incorrect version of $pk_{CA}$ will not be able to obtain authentic copies of anyone else's public key.
  - Thus $pk_{CA}$ must be distributed over an authenticated channel.
  - **A common way:**
    - "Bundle" it with some other software.
      - **Example:** Popular web browsers like Mozilla Firefox (actually falls in "multiple CA" category).
  - This inconvenient step need only be carried out once.

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Example:**
  - Bob may have to show up in person with a copy of his public key $pk_B$ along with identification proving that his name (or his email address) is what he claims.

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Example:**
  - Bob may have to show up in person with a copy of his public key $pk_B$ along with identification proving that his name (or his email address) is what he claims.
  - Only then would the CA issue the certificate.

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Validation levels:**
  1. Domain Validation (DV):
     - Is validated by proving some control over a DNS domain.
  2. Organization Validation (OV):
  3. Extended Validation (EV):

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Validation levels:**
  1. Domain Validation (DV):
  2. Organization Validation (OV):
     - The right to administratively manage the domain.
     - The organization's actual existence as a legal entity.
  3. Extended Validation (EV):

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Validation levels:**
  1. Domain Validation (DV):
  2. Organization Validation (OV):
  3. Extended Validation (EV):
     - Proves the legal entity of the owner.
     - Is signed by a CA key that can issue EV certificates.
     - EV certificates can be issued only by a subset of CAs.

# Single CA: Validation

**Validation:** The mechanism by which a CA issues a certificate to some party Bob must also be very carefully controlled.

- **Validation levels:**
  1. Domain Validation (DV):
  2. Organization Validation (OV):
  3. Extended Validation (EV):

- Parties completely trust the CA to issue certificates only when appropriate.
  - Suppose Alice receives a $\text{Cert}_{CA \to B}$ certifying that $pk_B$ is Bob's public key.
  - Then, Alice accepts this assertion as valid.
  - Uses $pk_B$ as Bob's public key.

- Thus a detailed verification process is very crucial.

# Single CA: Pros and Cons

**Pros:**

- Simple and appealing.

# Single CA: Pros and Cons

**Pros:**

- Simple and appealing.

**Cons:**

- No very Practical.
- Suitable for use within a single organisation.
- Unlikely that *outsiders* will trust the CA $\nRightarrow$ that anyone thinks the CA is corrupt.

# Single CA: Pros and Cons

**Pros:**

- Simple and appealing.

**Cons:**

- No very Practical.
- Suitable for use within a single organisation.
- Unlikely that *outsiders* will trust the CA $\not\Rightarrow$ that anyone thinks the CA is corrupt.
- CA's verification process can be insufficient for Alice.
  - Suppose the CA asks for only one form of identification.
  - But Alice prefers two.

# Single CA: Pros and Cons

**Pros:**

- Simple and appealing.

**Cons:**

- No very Practical.
- Suitable for use within a single organisation.
- Unlikely that *outsiders* will trust the CA $\not\Rightarrow$ that anyone thinks the CA is corrupt.
- CA's verification process can be insufficient for Alice.
- Single point of failure for the entire system.
  - The CA may be *corrupt*, or can be *bribed*, or even if the CA is merely lax with the way it protects its private key.
  - The legitimacy of issued certificates may then be called into question.

# Single CA: Pros and Cons

**Pros:**

- Simple and appealing.

**Cons:**

- No very Practical.
- Suitable for use within a single organisation.
- Unlikely that *outsiders* will trust the CA $\not\Rightarrow$ that anyone thinks the CA is corrupt.
- CA's verification process can be insufficient for Alice.
- Single point of failure for the entire system.
- Inconvenient for all parties to have to contact one CA.

# Outline

# Multiple CA

- Bob can choose a CA from a list of CAs.

# Multiple CA

- Bob can choose a CA from a list of CAs.
- Alice may have multiple certificates issued by different CAs.
- But she can choose which CA's certificates she trusts.

# Multiple CA

- Bob can choose a CA from a list of CAs.
- Alice may have multiple certificates issued by different CAs.
- But she can choose which CA's certificates she trusts.
- Bob also can obtain certificates from more than one CA.

# Multiple CA

- Bob can choose a CA from a list of CAs.
- Alice may have multiple certificates issued by different CAs.
- But she can choose which CA's certificates she trusts.
- Bob also can obtain certificates from more than one CA.
- But Alice must be more careful!
  - The security of her communication is ultimately only as good as the least-secure CA that she trusts.

# Multiple CA

- Bob can choose a CA from a list of CAs.
- Alice may have multiple certificates issued by different CAs.
- But she can choose which CA's certificates she trusts.
- Bob also can obtain certificates from more than one CA.
- But Alice must be more careful!
  - The security of her communication is ultimately only as good as the least-secure CA that she trusts.
- **Example:**
  - Suppose Alice trusts two CAs, $CA_1$ and $CA_2$.
  - Assume $CA_2$ is corrupted by an adversary.
  - This adversary will not be able to forge certificates issued by $CA_1$.
  - However, it will be able to issue fake certificates in the name of $CA_2$ for any identity/public key of its choice.

# Multiple CA

- This is a real problem in current system!

# Multiple CA

- This is a real problem in current system!

- **Remedies:**
  - Operating systems/web browsers typically come pre-configured with many CA's public keys.

# Multiple CA

- This is a real problem in current system!
- **Remedies:**
  - Operating systems/web browsers typically come pre-configured with many CA's public keys.
  - Default setting: Treat all these CAs as equally trustworthy.

# Multiple CA

- This is a real problem in current system!
- **Remedies:**
  - Operating systems/web browsers typically come pre-configured with many CA's public keys.
  - Default setting: Treat all these CAs as equally trustworthy.
  - However, any company willing to pay, can essentially be included as a CA!

# Multiple CA

- This is a real problem in current system!
- **Remedies:**
  - Operating systems/web browsers typically come pre-configured with many CA's public keys.
  - Default setting: Treat all these CAs as equally trustworthy.
  - However, any company willing to pay, can essentially be included as a CA!
  - The list of pre-configured CAs includes some reputable, well-established companies along with other, newer companies whose trustworthiness cannot be easily established.

# Multiple CA

- This is a real problem in current system!
- **Remedies:**
  - Operating systems/web browsers typically come pre-configured with many CA's public keys.
  - Default setting: Treat all these CAs as equally trustworthy.
  - However, any company willing to pay, can essentially be included as a CA!
  - The list of pre-configured CAs includes some reputable, well-established companies along with other, newer companies whose trustworthiness cannot be easily established.
  - *Therefore the user needs to manually configure their settings so as to only accept certificates from CAs the user trusts.*

# Outline

# Delegation and Certificate Chains

- Uses *certificate chains*.

# Delegation and Certificate Chains

- Uses *certificate chains*.

- We present the idea for certificate chains of length 2.

# Delegation and Certificate Chains

- Uses *certificate chains*.
- We present the idea for certificate chains of length 2.
- Can be generalized to chains of arbitrary length.

# Delegation and Certificate Chains (Cont.)

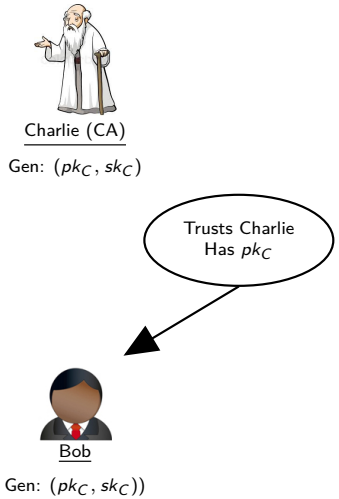
Charlie (CA)

Gen: $(pk_C, sk_C)$

Bob

Gen: $(pk_C, sk_C))$

# Delegation and Certificate Chains (Cont.)



# Delegation and Certificate Chains (Cont.)

Charlie (CA)

Gen: $(pk_C, sk_C)$

$Cert_{C \to B}$

Bob

Gen: $(pk_C, sk_C))$

# Delegation and Certificate Chains (Cont.)

Charlie (CA)

Gen: $(pk_C, sk_C)$

$Cert_{C \to B}$

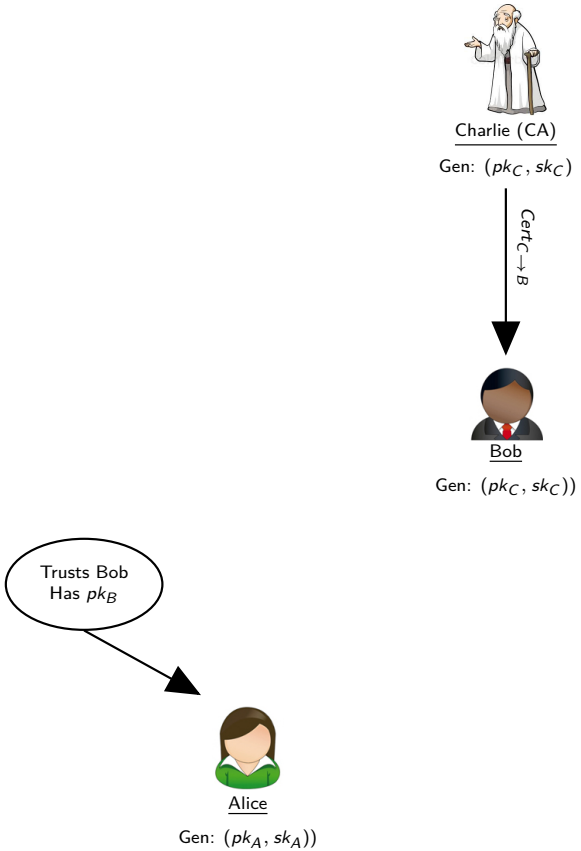
Bob

Gen: $(pk_C, sk_C))$

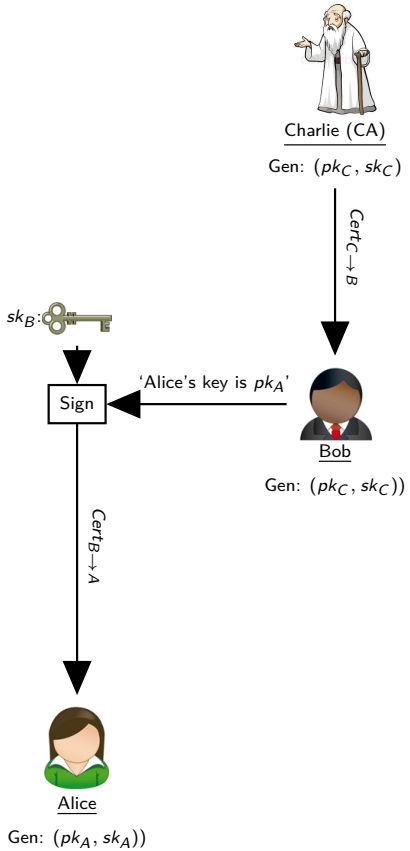
Alice

Gen: $(pk_A, sk_A))$

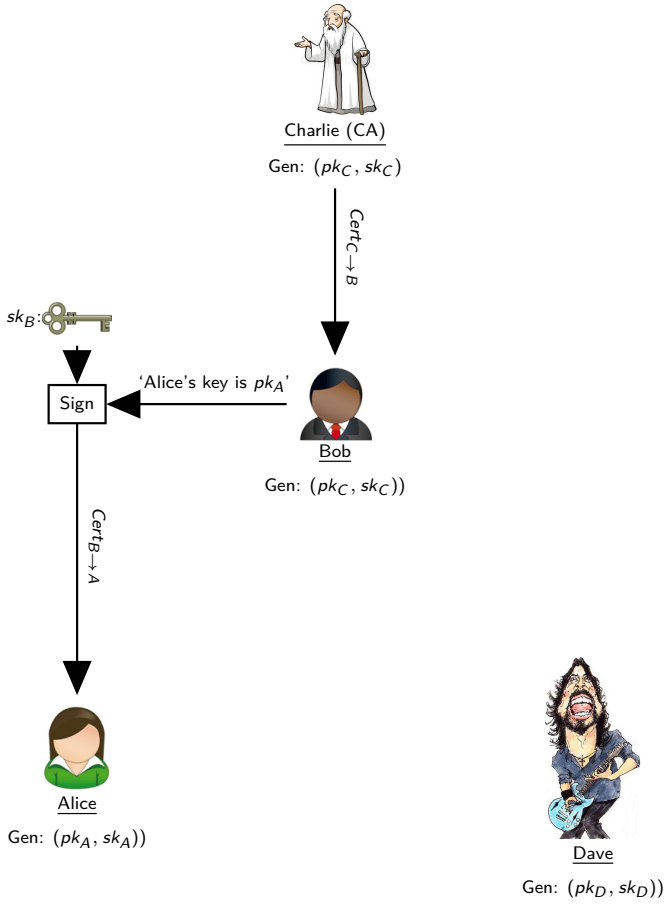
# Delegation and Certificate Chains (Cont.)



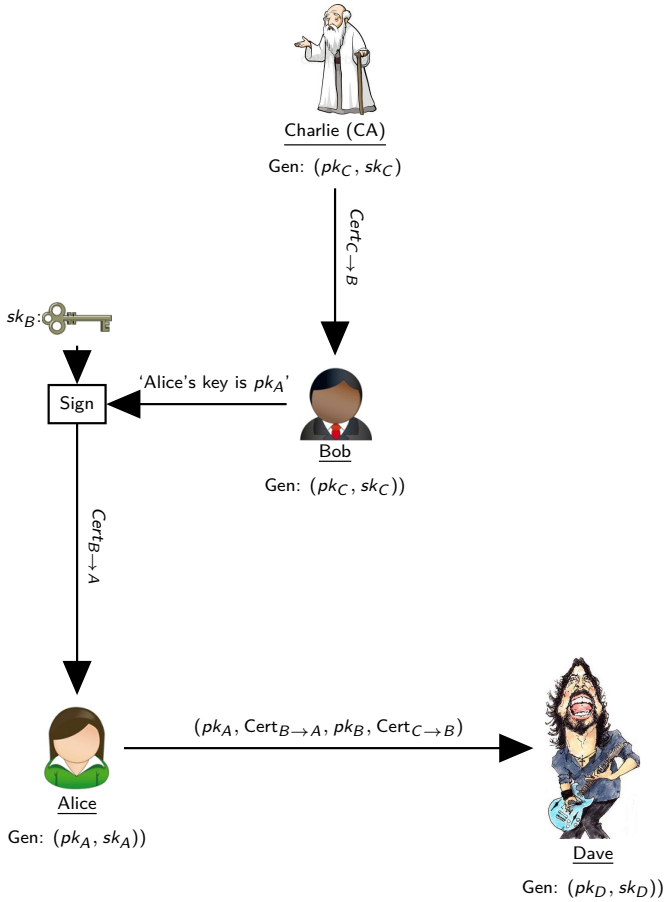
# Delegation and Certificate Chains (Cont.)



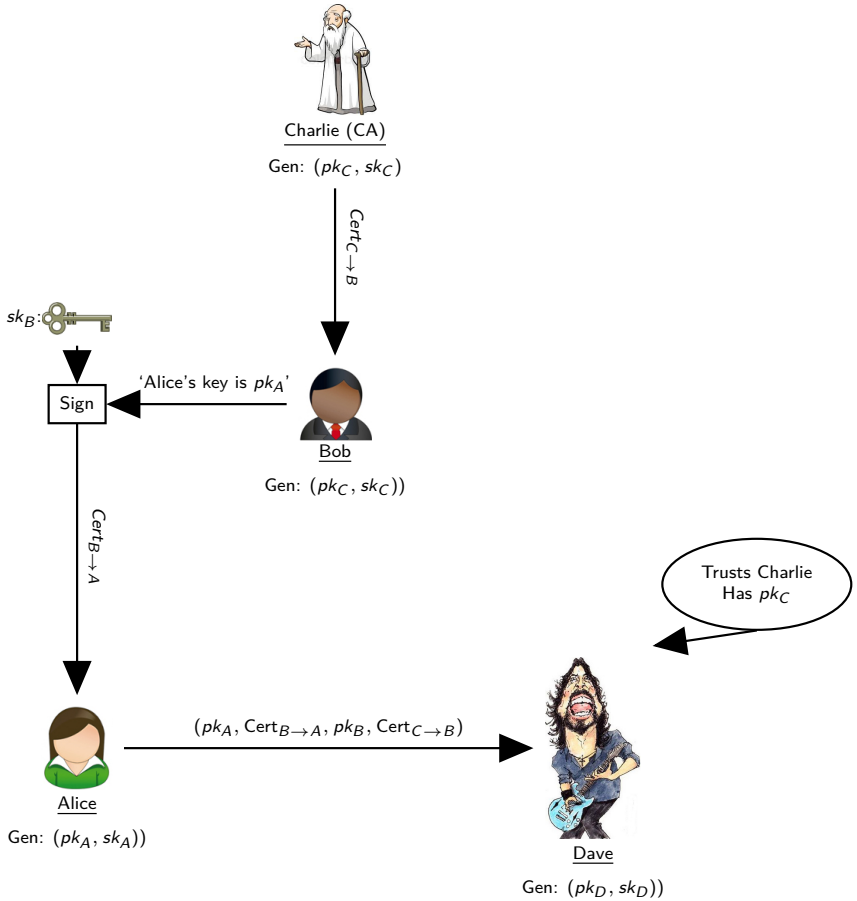
# Delegation and Certificate Chains (Cont.)



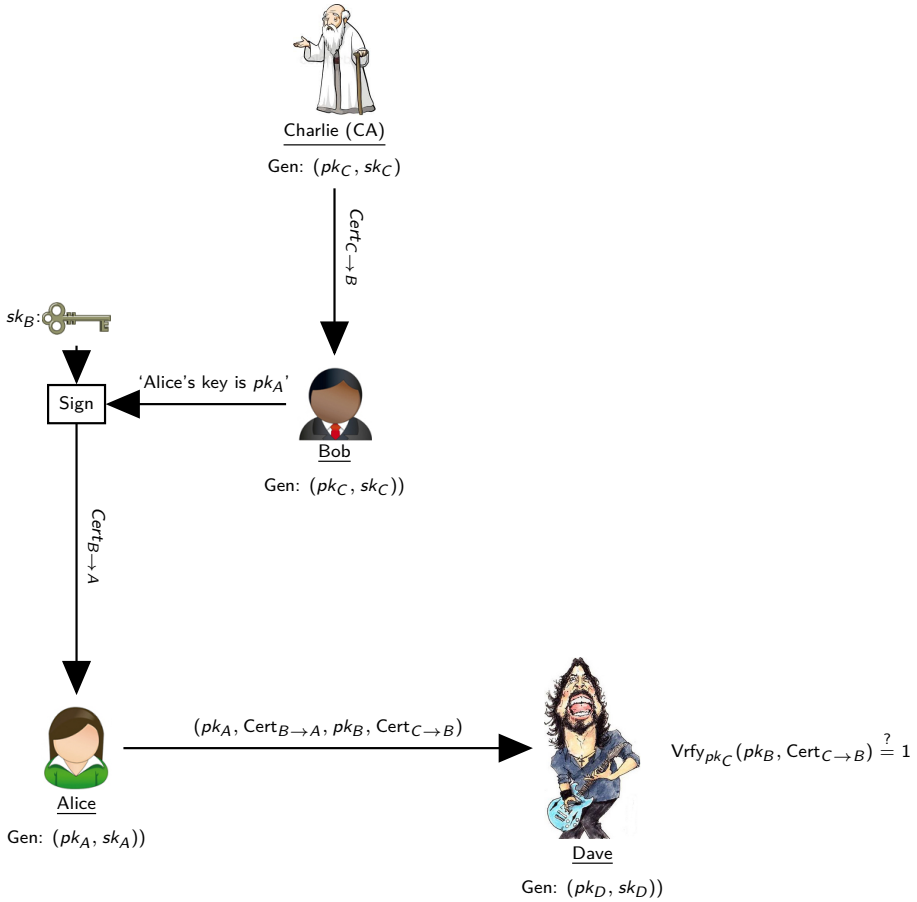
# Delegation and Certificate Chains (Cont.)



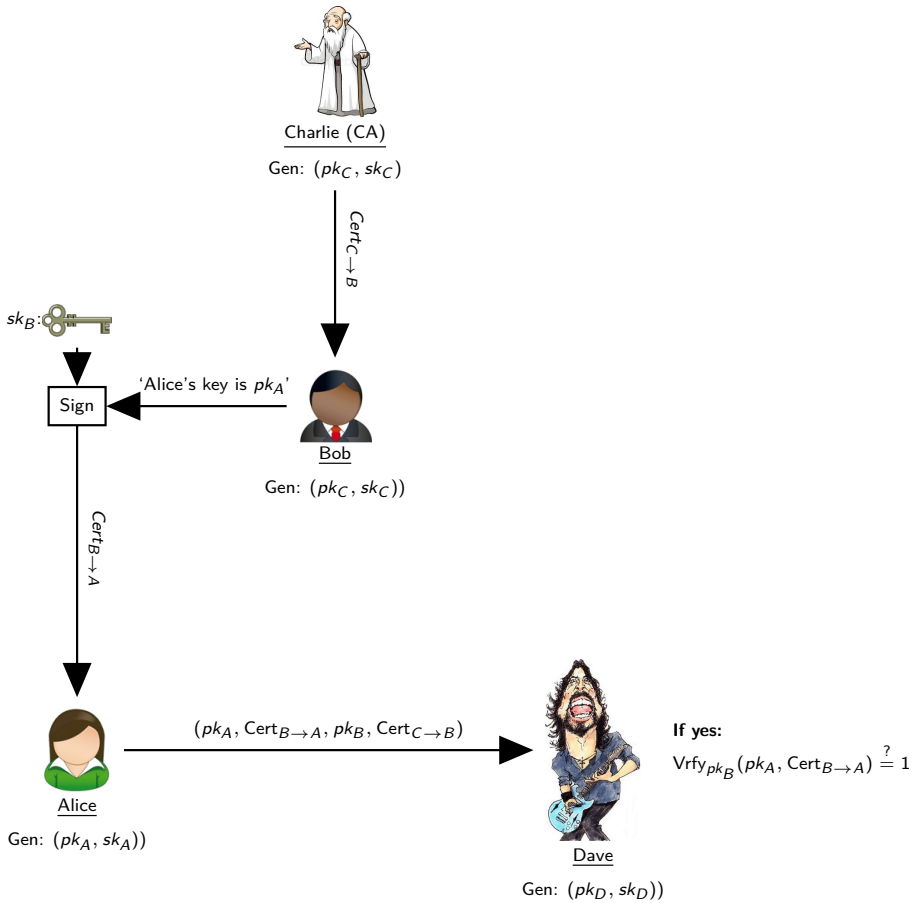
# Delegation and Certificate Chains (Cont.)



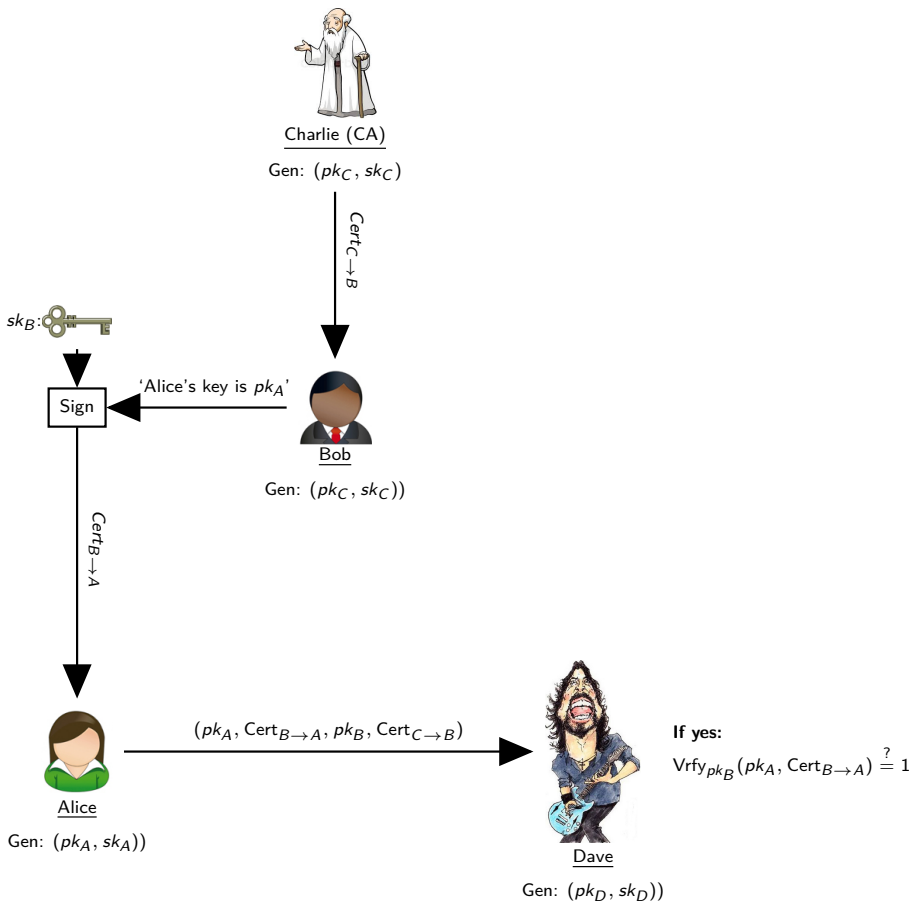
# Delegation and Certificate Chains (Cont.)



# Delegation and Certificate Chains (Cont.)



# Delegation and Certificate Chains (Cont.)




Thus, if Dave trusts Charlie then Dave may accept $pk_A$.

# Delegation and Certificate Chains

**Note:**

- Stronger semantics are now associated with a $\text{Cert}_{C \to B}$.

# Delegation and Certificate Chains

**Note:**

- Stronger semantics are now associated with a $\text{Cert}_{C \to B}$.
  - **Earlier:** $\text{Cert}_{C \to B} \Rightarrow$ an assurance that Bob's public key is $pk_B$.

# Delegation and Certificate Chains

**Note:**

- Stronger semantics are now associated with a $\text{Cert}_{C \to B}$.
  - **Earlier:** $\text{Cert}_{C \to B} \Rightarrow$ an assurance that Bob's public key is $pk_B$.
  - **Now:** Bob holds public key $pk_B$ and Bob is trusted to issue other certificates.

# Delegation and Certificate Chains

**Note:**

- Stronger semantics are now associated with a $\text{Cert}_{C \to B}$.
  - **Earlier:** $\text{Cert}_{C \to B} \Rightarrow$ an assurance that Bob's public key is $pk_B$.
  - **Now:** Bob holds public key $pk_B$ and Bob is trusted to issue other certificates.

- When Charlie signs a certificate for Bob, Charlie is, in effect, delegating his ability to issue certificates to Bob.

# Delegation and Certificate Chains

**Note:**

- Stronger semantics are now associated with a $\text{Cert}_{C \to B}$.
  - **Earlier:** $\text{Cert}_{C \to B} \Rightarrow$ an assurance that Bob's public key is $pk_B$.
  - **Now:** Bob holds public key $pk_B$ and Bob is trusted to issue other certificates.

- When Charlie signs a certificate for Bob, Charlie is, in effect, delegating his ability to issue certificates to Bob.

- Bob can now act as a proxy for Charlie, issuing certificates on Charlie's behalf.

# CA-based PKI

- Imagine one "root" CA and $n$ "second-level" CAs $CA_1, \ldots, CA_n$.
- Root CA: Can issue certificates for each $CA_i$, $1 \leq i \leq n$.
- Each $CA_i$: Then can issue certificates for other entities holding public keys.

# CA-based PKI: Pros and Cons

**Pros:**

- Eases the burden on the root CA.
- Makes it more convenient for parties to obtain certificates.
  - May now contact the second-level CA who is closest to them.

# CA-based PKI: Pros and Cons

**Pros:**

- Eases the burden on the root CA.
- Makes it more convenient for parties to obtain certificates.

**Cons:**

- But managing these second-level CAs may be difficult.
- There are now more points of attack in the system.

# Outline

# Invalidating Certificates

- Certificates generally should not be valid indefinitely.

# Invalidating Certificates

- Certificates generally should not be valid indefinitely.

- **Example 1:**
  - An employee may leave a company.
  - In which case (s)he should no longer receive encrypted communication from others within the company.

# Invalidating Certificates

- Certificates generally should not be valid indefinitely.

- **Example 2:**
  - A user's private key might also be stolen.
  - At which point the user (assuming they know about the theft) will want to generate a new key-pair and have the old public key removed from circulation.

# Invalidating Certificates

- Certificates generally should not be valid indefinitely.
- $\therefore$ we need a way to render previously issued certificates invalid.
- Approaches for handling these issues are varied and complex.
- We will only mention two relatively simple ideas.
- Improving these methods is an active area of real-world network-security research.

# Expiration

- Include an *expiry date* as part of the certificate.

# Expiration

- Include an *expiry date* as part of the certificate.
- Charlie's certificate for Bob's public key now looks like

$$\mathrm{Cert}_{C\to B} \stackrel{\text{def}}{=} \mathrm{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{'}, \text{ exp. date}).$$

# Expiration

- Include an *expiry date* as part of the certificate.
- Charlie's certificate for Bob's public key now looks like

$$\mathsf{Cert}_{C\to B} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{'}, \text{ exp. date}).$$

- $\mathsf{Vrfy}_{pk_C}(pk_B, \mathsf{Cert}_{C\to B}) \stackrel{?}{=} 1$:
  - Is signature valid?
  - Has the expiry date passed?

# Expiration

- Include an *expiry date* as part of the certificate.
- Charlie's certificate for Bob's public key now looks like

$$\mathsf{Cert}_{C\to B} \stackrel{\text{def}}{=} \mathsf{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{'}, \text{ exp. date}).$$

- $\mathsf{Vrfy}_{pk_C}(pk_B, \mathsf{Cert}_{C\to B}) \stackrel{?}{=} 1$:
  - Is signature valid?
  - Has the expiry date passed?
- A user must contact the CA to get a new certificate issued whenever their current one expires.

# Expiration

- Include an *expiry date* as part of the certificate.
- Charlie's certificate for Bob's public key now looks like

$$\mathrm{Cert}_{C\to B} \stackrel{\text{def}}{=} \mathrm{Sign}_{sk_C}(\text{'Bob's key is } pk_B\text{'}, \text{ exp. date}).$$

- $\mathrm{Vrfy}_{pk_C}(pk_B, \mathrm{Cert}_{C\to B}) \stackrel{?}{=} 1$:
  - Is signature valid?
  - Has the expiry date passed?
- A user must contact the CA to get a new certificate issued whenever their current one expires.
- The CA then verifies the identity/credentials of the user again before issuing another certificate.

# Expiration

- Provides a very coarse-grained solution to the problem.
  - Suppose an employee leaves a company the day after getting a certificate.
  - Also assume that the issued certificate expires one year after its issuance date.
  - Then this employee can use his or her public key illegitimately for an entire year until the expiry date passes.
  - For this reason, this approach is typically used in conjunction with other methods such as **revocation**.

# Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like their certificates to become invalid immediately, or at least as soon as possible.

# Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like their certificates to become invalid immediately, or at least as soon as possible.
- In such cases, the CA explicitly revokes the issued certificate.

# Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like their certificates to become invalid immediately, or at least as soon as possible.
- In such cases, the CA explicitly revokes the issued certificate.
- For simplicity, assume a single CA (can be generalised).

# Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like their certificates to become invalid immediately, or at least as soon as possible.
- In such cases, the CA explicitly revokes the issued certificate.
- For simplicity, assume a single CA (can be generalised).
- There are many different ways revocation can be handled.

# Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like their certificates to become invalid immediately, or at least as soon as possible.
- In such cases, the CA explicitly revokes the issued certificate.
- For simplicity, assume a single CA (can be generalised).
- There are many different ways revocation can be handled.
- One possibility: Every certificate includes a serial number, i.e.,

$$\mathrm{Cert}_{C\to B} \stackrel{\mathrm{def}}{=} \mathrm{Sign}_{sk_C}(\text{'Bob's key is } pk_B\text{'}, \#\#\#),$$

where $\#\#\#$ represents the serial number of this certificate.

# Revocation

- Serial number of each certificate should be unique.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.
- If $sk_B$ corresponding to a $pk_B$ is stolen, then alert the CA.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.
- If $sk_B$ corresponding to a $pk_B$ is stolen, then alert the CA.
- **Note:** CA must verify Bob's identity to prevent another user from falsely revoking Bob's certificate.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.
- If $sk_B$ corresponding to a $pk_B$ is stolen, then alert the CA.
- **Note:** CA must verify Bob's identity to prevent another user from falsely revoking Bob's certificate.
- The CA searches its database to find the serial number associated with the certificate issued for Bob and $pk_B$.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.
- If $sk_B$ corresponding to a $pk_B$ is stolen, then alert the CA.
- **Note:** CA must verify Bob's identity to prevent another user from falsely revoking Bob's certificate.
- The CA searches its database to find the serial number associated with the certificate issued for Bob and $pk_B$.
- At the end of each day (say) the CA generates
  - a certificate revocation list (CRL) with the serial numbers of all revoked certificates, and
  - signs the CRL and the current date.

# Revocation

- Serial number of each certificate should be unique.
- CA stores: $(Bob, pk_B, \#\#\#)$ for each certificate it generates.
- If $sk_B$ corresponding to a $pk_B$ is stolen, then alert the CA.
- **Note:** CA must verify Bob's identity to prevent another user from falsely revoking Bob's certificate.
- The CA searches its database to find the serial number associated with the certificate issued for Bob and $pk_B$.
- At the end of each day (say) the CA generates
  - a certificate revocation list (CRL) with the serial numbers of all revoked certificates, and
  - signs the CRL and the current date.
- The signed CRL is then widely distributed or is made available to potential verifiers.

# Revocation

- $\mathrm{Vrfy}_{pk_C}(\mathrm{Cert}_{C \to B}) \stackrel{?}{=} 1$:
  - Is the signature valid?
  - Does the serial number appear on recent CRL?
  - Verify the CA's signature on the revocation list itself.

# Outline

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.
- Used by your browser any time you connect to a website using https rather than http.

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.
- Used by your browser any time you connect to a website using https rather than http.
- We focus on the underlying cryptography used in the core TLS protocol.

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.
- Used by your browser any time you connect to a website using https rather than http.
- We focus on the underlying cryptography used in the core TLS protocol.
- We present a slightly simplified and an abstract version of the protocol in order to convey the main point.

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.
- Used by your browser any time you connect to a website using https rather than http.
- We focus on the underlying cryptography used in the core TLS protocol.
- We present a slightly simplified and an abstract version of the protocol in order to convey the main point.
- We do not formally define or claim security for the protocol.

# SSL/TLS

- Transport Layer Security (TLS) protocol: Used extensively to secure communication over the web.
- Used by your browser any time you connect to a website using https rather than http.
- We focus on the underlying cryptography used in the core TLS protocol.
- We present a slightly simplified and an abstract version of the protocol in order to convey the main point.
- We do not formally define or claim security for the protocol.
- A formal analysis of TLS is the subject of active research.

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).
- SSL was developed by Netscape in the mid-1990s.

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).
- SSL was developed by Netscape in the mid-1990s.
- The last version available was SSL 3.0.

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).
- SSL was developed by Netscape in the mid-1990s.
- The last version available was SSL 3.0.
- TLS version 1.0 was released in 1999, updated to version 1.1 in 2006, updated again to version 1.2 in 2008 and to version 1.3 (current version) in 2018.

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).
- SSL was developed by Netscape in the mid-1990s.
- The last version available was SSL 3.0.
- TLS version 1.0 was released in 1999, updated to version 1.1 in 2006, updated again to version 1.2 in 2008 and to version 1.3 (current version) in 2018.
- Till 2014, approximately 50% of websites used to use TLS 1.0 rather than a more recent version.

# SSL/TLS

- TLS is a standardized protocol based on a precursor called SSL (or Secure Socket Layer).
- SSL was developed by Netscape in the mid-1990s.
- The last version available was SSL 3.0.
- TLS version 1.0 was released in 1999, updated to version 1.1 in 2006, updated again to version 1.2 in 2008 and to version 1.3 (current version) in 2018.
- Till 2014, approximately 50% of websites used to use TLS 1.0 rather than a more recent version.
- As of April 2016, all major web browsers support TLS 1.0, 1.1, and 1.2, and have them enabled by default.

# TLS

- Allows a client (e.g., a web browser) and a server (e.g., a web-site) to agree on a set of shared keys.

# TLS

- Allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys.
- Then uses those keys to encrypt and authenticate their subsequent communication.

# TLS

- Allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys.
- Then uses those keys to encrypt and authenticate their subsequent communication.
- Consists of two parts:
  1. Handshake protocol: Performs authenticated key exchange to establish the shared keys.
  2. Record-layer protocol: Uses those shared keys to encrypt/ authenticate the parties' communication.

# TLS

- Allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys.
- Then uses those keys to encrypt and authenticate their subsequent communication.
- Consists of two parts:
  1. Handshake protocol: Performs authenticated key exchange to establish the shared keys.
  2. Record-layer protocol: Uses those shared keys to encrypt/ authenticate the parties' communication.
- **Note:** TLS allows for clients to authenticate to servers.

# TLS

- Allows a client (e.g., a web browser) and a server (e.g., a website) to agree on a set of shared keys.
- Then uses those keys to encrypt and authenticate their subsequent communication.
- Consists of two parts:
  1. Handshake protocol: Performs authenticated key exchange to establish the shared keys.
  2. Record-layer protocol: Uses those shared keys to encrypt/ authenticate the parties' communication.
- **Note:** TLS allows for clients to authenticate to servers.
- However, it is primarily used only for authentication of servers to clients because only servers typically have certificates.

# TLS: Handshake Protocol

Client ($C$)

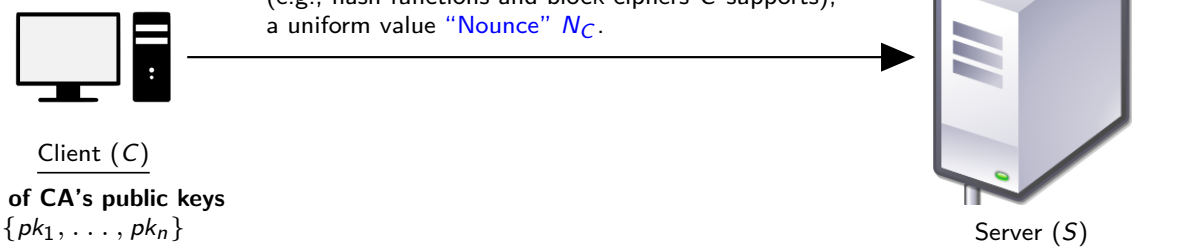
**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\text{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

**KEM:** Key Encapsulation Mechanism

# TLS: Handshake Protocol



# TLS: Handshake Protocol

Client ($C$)

**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

**Message includes:**
latest version of the protocol it supports
as well as an appropriate ciphersuite.
In addition, it sends its key $pk_S$,
its certificate $\text{Cert}_{i \to S}$, and
its own uniform value $N_S$.

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\text{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

# TLS: Handshake Protocol

Client ($C$)

**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

$\mathsf{Vrfy}_{pk_i}(pk_S, \mathsf{Cert}_{i \to S}) \stackrel{?}{=} 1$
**If yes:** verifies the certificate (i.e., expired or revoked)
**If successful:** $pk_S$ is $S$'s key
$(c, \mathsf{pmk}) \leftarrow \mathit{Encaps}_{pk_S}(1^n)$
(pmk: *pre-master key*).

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\mathsf{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

# TLS: Handshake Protocol

Client ($C$)

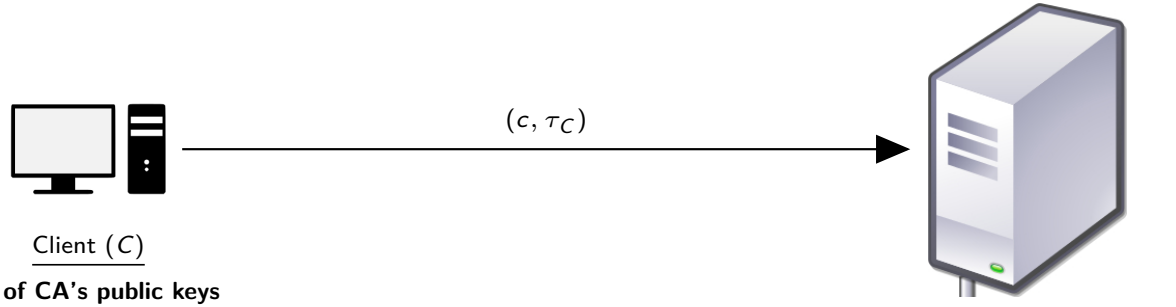
**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

pmk is used to derive a *master key* mk using a key-derivation function applied to pmk, $N_C$, and $N_S$.
$k_C, k'_C, k_S, k'_S \leftarrow \text{PRG}(\text{mk})$
$\tau_C \leftarrow \text{MAC}_{\text{mk}}(\text{transcript})$
transcript: messages exchanged thus far.

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\text{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

# TLS: Handshake Protocol



# TLS: Handshake Protocol

Client ($C$)

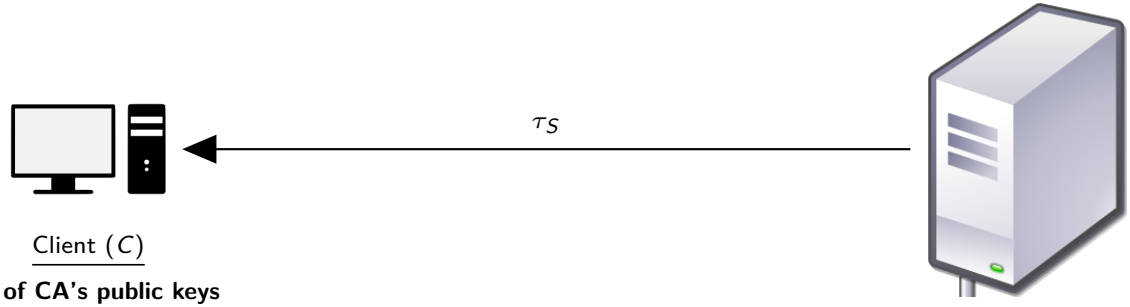
**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\text{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

pmk $:= \text{Decaps}_{sk_S}(c)$
from pmk derive mk and $k_C, k'_C, k_S, k'_S$
as the client did.
**If** $\text{Vrfy}_{\text{mk}}(\text{transcript}, \tau_C) \neq 1$, then **abort**
**Else:** $\tau_S \leftarrow \text{MAC}_{\text{mk}}(\text{transcript}')$
transcript': messages exchanged thus far.

# TLS: Handshake Protocol



# TLS: Handshake Protocol

Client ($C$)

**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

**If** $\mathsf{Vrfy}_{\mathsf{mk}}(\text{transcript}', \tau_S) \neq 1$, then **abort**

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\text{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

# TLS: Handshake Protocol

Client ($C$)

**Set of CA's public keys**
$\{pk_1, \ldots, pk_n\}$

**If** $\mathsf{Vrfy}_{\mathsf{mk}}(\text{transcript}', \tau_S) \neq 1$, then **abort**

Server ($S$)

$(pk_S, sk_S)$ for **KEM**
a certificate $\mathsf{Cert}_{i \to S}$
issued by one of the CAs
whose public key $C$ knows.

At the end of a successful execution of the handshake protocol, $C$ and $S$ share a set of four symmetric keys $k_C, k'_C, k_S, k'_S$.

# TLS: Handshake Protocol

- TLS 1.2 supports two KEMs:
  1. CDH/DDH-based KEM or
  2. the RSA-based encryption scheme PKCS #1 v1.5

# The Record-Layer Protocol

- Once keys have been agreed upon by $C$ and $S$, the parties use those keys to encrypt and authenticate all their subsequent communication.
- $C$ uses $k_C$ (resp., $k'_C$) to encrypt (resp., authenticate) all messages it sends to S.
- Similarly, $S$ uses $k_S$ (resp., $k'_S$) to encrypt (resp., authenticate) all the messages it sends to $C$.
- Sequence numbers are used to prevent replay attacks.

# Books Consulted

1. Chapter 11 & 12 of *Introduction to Modern Cryptography* by Jonathan Katz and Yehuda Lindell, 2nd Edition, Chapman & Hall/CRC.

2. OpenSSL
   - For installation visit this "stackoverflow" page.
   - For generating certificates/basic instructions visit this "stackoverflow" page.

Thank You for your kind attention!

# Questions!!