

# Announcements

- Assignment 1:
  - Any issues so far?
  - Please check with your TAs, discuss in LumiNUS Forum
  - Available ***consultation sessions on A1***
  - Please submit the answers before A1 due date
- Open consultation hours: *to be announced*
  - Wednesday & Friday afternoon during recess week
  - Via Zoom: anyone can just join the discussions
- Last year's mid-term quiz to be uploaded
- *Mid-term e-exam briefing by CIT: 18 Sep, 11:05am*

# Lecture 4: PKI + Channel Security

4.1 Public key distribution

4.2 Public Key Infrastructure (PKI)

4.2.1 Certificate

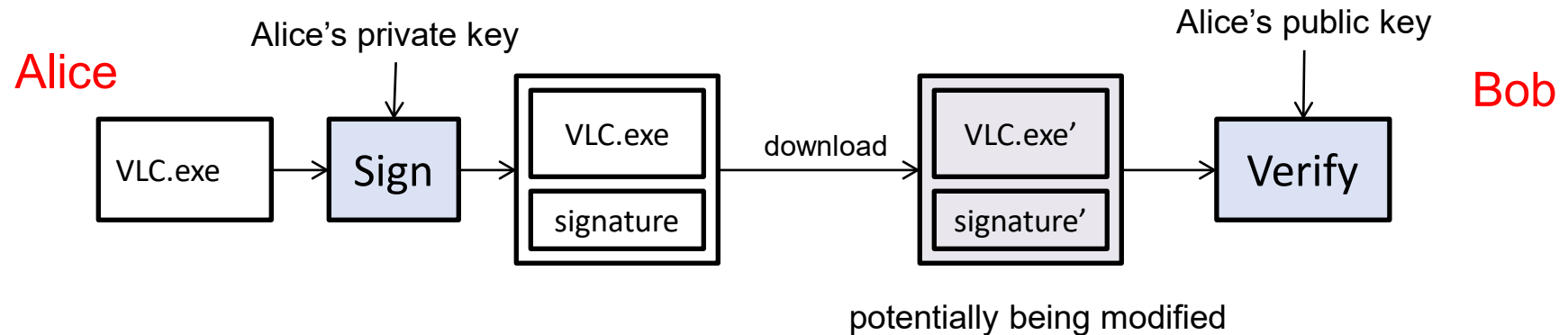
4.2.2 CA & Trust Relationship

4.3 Limitations/attacks on PKI

## **4.1 Public Key Distribution**

# Motivation 1: Signed Application Example

Recall the previous example of **downloading VLC.exe from a website**



As mentioned before, a PKC-based method to assure the **authenticity** of the downloaded program is as follows:

1. The developer, say Alice, signed the VLC.exe file using her **private key**
2. A user, say Bob, who has downloaded the signed file VLC.exe from an unverified source (e.g. CNET download site), can verify the authenticity of the file using **Alice's public key**

## Motivation 2: “Signed” Email (using PGP Public Key) Example

- Alice, with email account `alice@comp.nus.edu.sg`, sent an email to Bob.

Alice has a pair of “**PGP**” **public-private key**.

Alice’s email is signed using her PGP **private key**  
*(see the next slide for the actual email sent).*

- After Bob has received the email, with **Alice’s public key**, he can check its authenticity by verifying the signature
- *Any possible issues for PKC to be used/deployed securely?*
  - To carry out the authenticity, Bob **needs to know Alice’s public key** 😞
  - Now, we are now back to square one:  
**a secure channel is needed** for Alice to send her public key to Bob

# Example of “Signed” Email using PGP Public Key

```
Date: Wed, 07 Mar 2007 03:22:08 +0800
From: Alice Ho <alice@comp.nus.edu.sg>
User-Agent: Thunderbird 1.5.0.10 (Windows/20070221)
MIME-Version: 1.0
To: bob@comp.nus.edu.sg
Subject: My first signed email
X-Enigmail-Version: 0.94.2.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit
```

Header (unsigned,  
i.e. not included in  
computing the signature)

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
```

Dear Bob,

This is my very first signed email and I want you to keep it =)

Regards,

Alice Ho

```
-----BEGIN PGP SIGNATURE-----
```

Version: GnuPG v1.4.3 (MingW32)

Comment: Using GnuPG with Mozilla - <http://enigmail.mozdev.org>

iD8DBQFF7b9XMJcr5kFKO4IRAk+yAKC7JVIleY+aHEAqqCeVdYGOE10PmwCg9DrE

ArgWymKbDnl7m9WlleVeQqM=

=EksE

```
-----END PGP SIGNATURE-----
```

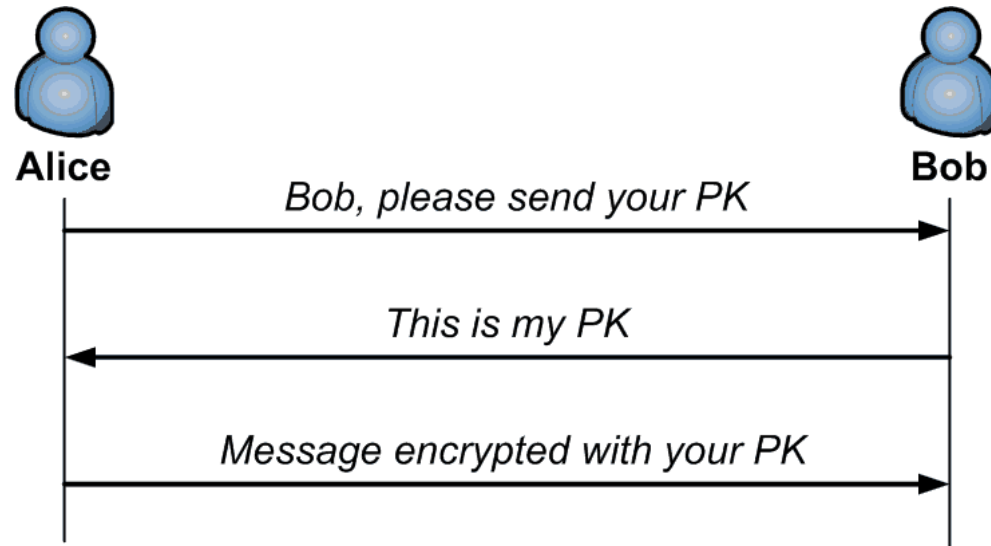
(signed)  
Message

Signature

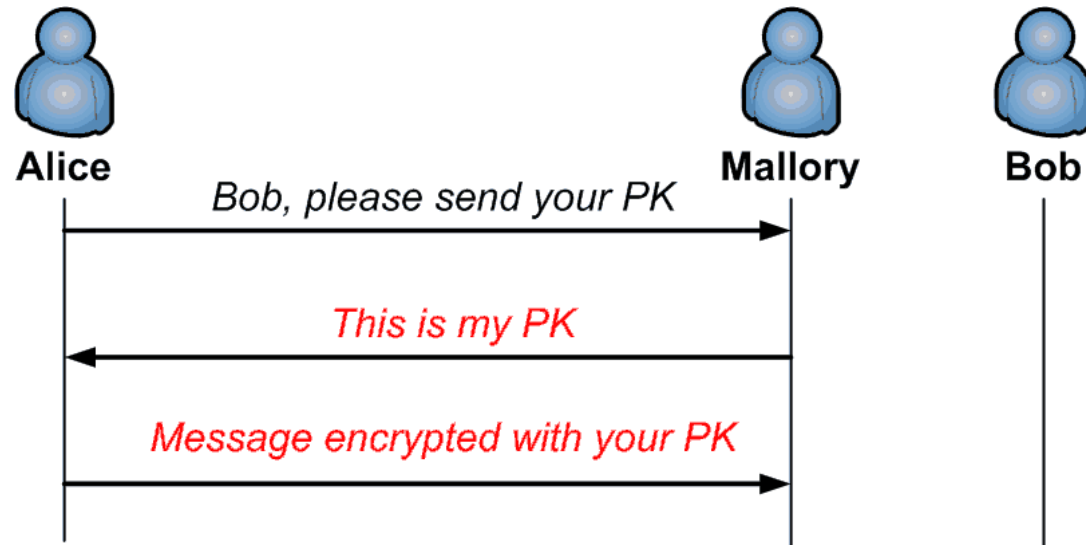
# Motivation: If a Public-Key is Distributed *Insecurely*

When Alice needs  
Bob's public key:

*Intended &  
assumed  
communication*



*Actual  
(intercepted)  
communication*



# Key Distribution Problem and Possible Methods

- The previous two examples illustrate the need for a mechanism to **securely distribute/broadcast public keys**
- With a public key “securely” distributed, we can use it for encryption (**confidentiality**) and signature verification (**authenticity**)
- **Important questions:**
  - How can we use a **secure channel** to address this public-key distribution problem?
  - How different is the **secure channel requirement** between the public-key and secret-key settings?



# Secure Channel Requirement in Public-Key Setting

- Compared to the symmetric-key setting (SK encryption, MAC), public-key setting ***doesn't*** require a secure channel to send **the secret key from Alice to Bob**
- Yet, we still ***need*** a secure channel for Alice to send her **public key** to Bob!
- Nevertheless, the public-key setting is arguably **easier to handle**:

Requirement Aspect	Symmetric-Key Setting	Public-Key Setting
<b>No. of times</b> a secure channel is required	For <b>every pair of entities</b> : $n.(n-1)/2$	<b>Each entity</b> just needs to securely broadcast <b>its public key</b> : $n$
<b>Item</b> to be transmitted	Shared <b>secret</b> key	(Publicly-published) <b>public</b> key
Secure channel <b>timing requirement</b> (e.g. when a new entity needs to securely talk to another entity)	A secure channel is needed to deliver both parties' <b>newly-set secret key</b>	<b>Previously-announced</b> public key(s) just need to be made accessible to a party requiring the key(s)

# Possible Public-Key Distribution Methods

- We would look into **3** different possible methods:
  1. Public announcement
  2. Publicly available directory
  3. Public Key Infrastructure (PKI)

# Public-Key Distribution Method: (1) Public Announcement

- The owner **broadcasts** his/her public key
- For example: by sending it to friends via email, or publishing it on a **website**
- Many owners list their “PGP public key” in a **blog**, personal **webpage**, etc.

For example: Bruce Schneier’s

<https://www.schneier.com/blog/about/contact.html>

## Contact Bruce Schneier

For feedback on content, please e-mail Bruce Schneier: [schneier@schneier.com](mailto:schneier@schneier.com)

For other website issues (browser compatibility problems, etc.), please e-mail: [webmaster@schneier.com](mailto:webmaster@schneier.com)

## Password Safe Support

Password Safe is now an open source project -- please see its [Sourceforge page](#) for feature req and bug reports.

## Keys

### OTR (IM) Fingerprint

8FBB10D4 A2B73FAE 935FF3AE BA5EFE2 9A98966F

### PGP Key

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v2.0.21 (MingW32)

```
mQINBFipG2IBEACuiDv9Lo8UW0eUh9sUvB11tncGMigJczcdSLHXNoApf0uEmTPw
ngIpmkeOdXniLeEHv2eao98I3IjtIfvo2YfnqFQ2lSn+UUfnCf+nh6jYAnyEOCIi
dr8oXN5Lx9lXfRCdU17oGYW6azTIKZqxLQticf0GvCaXYHdBaAqU5E1C20sC6CnV
IlqIxr/kjzvQdhz1Ig8LPu9O17ltsf6BevEI0wSLJFRZXF3mHb9iYntJnz+gWj/S
XBWcgJpFblH0dOo8gyF/K58HBMh8NPo9nQqO9bWmo/TMPzdX5DERGMAz92tg34I6
bFjGj2oflu22o8WlOZn07iXAKJKG6BLeOT4tpqVCWrM2YBr+eD7BR9Q2qRaJQ3T
8fm2ohYHiLjqkvH7/LjpGTilcdwkHmUjr9pD/MJQZR5BsyyWg0a6A35jvViAVaAo
Zkz+wFE6TCIdPGBj9q+vH++F3MZDL/qREiWeUnlcu01JobPJir6b48eyLkxHbeu3
z1GIuzNfC8al/Wr9rPJZpOehf/wodIdkxNvqyyxXo/t7/7ksMJglW6VVVKvG
mWEFHoL93pcKXZdqImSCutK362v8qrb3RlhG/zgFHBRljcvAVbeP+Y7HayeO756i
WewGiy/9Z5dlSMV594fhXM9BzwMwfboS2Bivi1jvOEyTSpm3q0fHx/tQARAQAB
tCBzY2huZWllciA8c2NobmVpZXJAc2NobmVpZXIuY29tPokCOQQOTAIAIwUCUikb
YgIbAwcLCQgHAWIBBHUIAgkKCwQWAgMBAh4BAheAAAoJELS0KiztrOpnODkP/3PA
sx0r2/6D48GLqTmUBWJiK6z4EmNaMmwElvqzeadc7DknzSqHKWDCDCZPxlllIDRV
kdAx7kKq+zuSAfzEtK+KZ4jm0ahn5bpdDzp+j8YHvym+JXcmY+JSIgdTQmCybt0B
1xPvrVpxK7mEr6M+XBxIZ8OfnKflmObR0llwL47eigYGDHP5kX0dMb2hr4QcfncC
```

# Public-Key Distribution Method: (1) Public Announcement

## Limitations:

- Not standardized, and thus there is no systematic way to find/verify the public key when needed
- Eventually, we still need to **trust** the “entity” distributing the public key:
  - In the previous example on Schneier’s PGP key, the **website** needs to be trusted

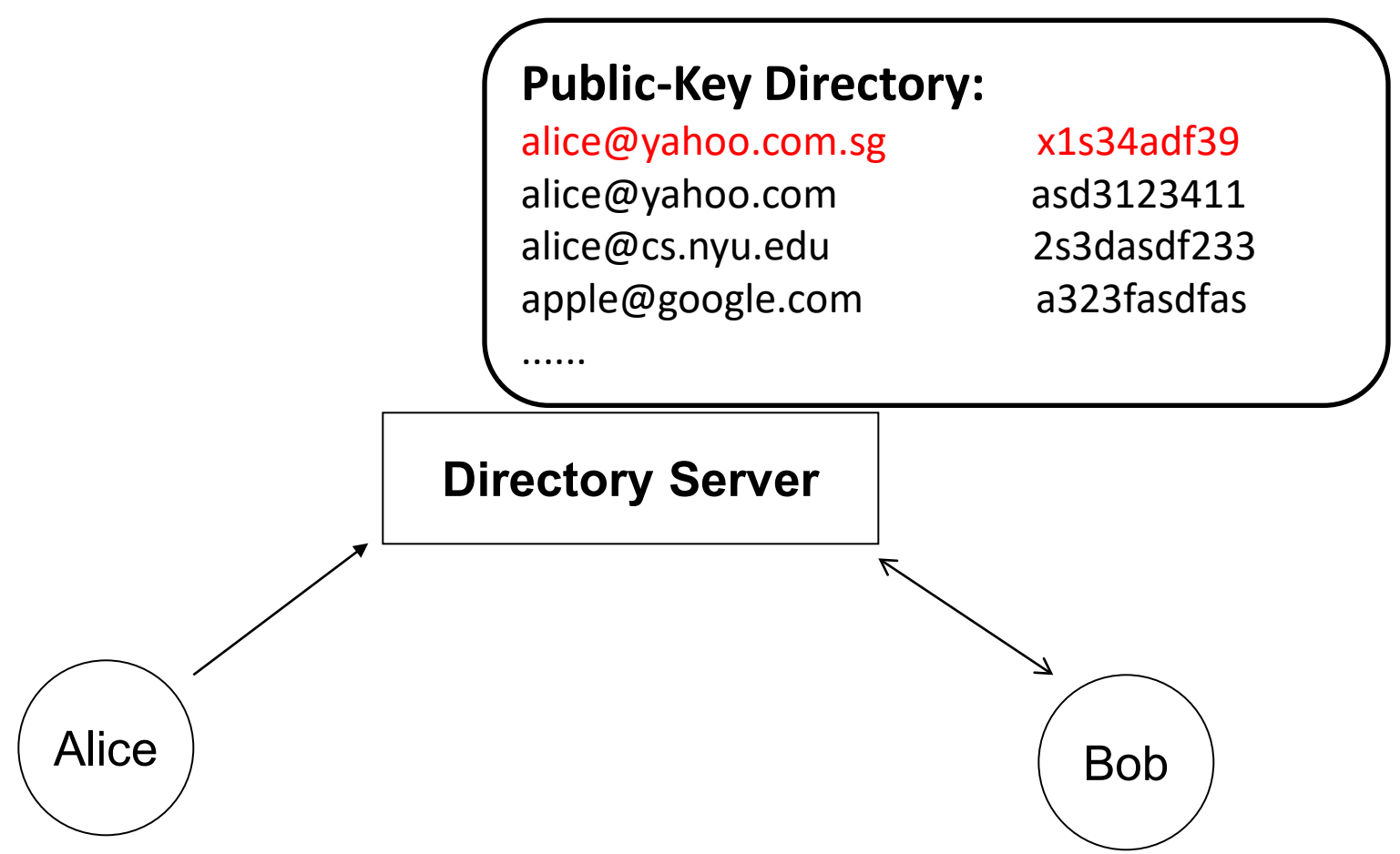
## Exercise:

Get a public key, and send a signed email. (Try PGP.)

# Public-Key Distribution Method: (2) Publicly-Available Directory

If Bob wants to find the public key associated with the name  
`alice@yahoo.com.sg`

he can search the **public directory** by querying a **server**



# Public-Key Distribution Method: (2) Publicly-Available Directory

## Potential issues:

- **Anyone** can post his/her public key into the server, e.g. `https://pgp.mit.edu/`
- It is not easy to have a “secure” **public directory**:  
Suppose the server receives a request to post a public key, how does the server verify that the information is **correct**?
- Eventually, some entity need to be **trusted**:  
in this case, the website `https://pgp.mit.edu/`
- Furthermore, even if a user trusts the website operator, how does the user know that the “website” **visited** is indeed `https://pgp.mit.edu/` as claimed?

# Public-Key Distribution Method: (3) PKI + Certificate

- PKI is a **standardized** system that distributes public keys
- (Again, when reading a document, note that there are different definitions of “Public Key Infrastructure”)
- PKI’s **objectives**:
  - To make public-key cryptography **deployable** on a large scale
  - To make public keys verifiable ***without*** requiring any two communicating parties to **directly trust** each other
  - To manage public & private key pairs throughout their entire key lifecycle

# Public-Key Distribution Method: (3) PKI + Certificate

- PKI is centered around two important components/ notions:
  - ***Certificate***
  - ***Certificate/Certification Authority (CA)***
- PKI provide a mechanism for “trust” to be extended in a **distributed** manner, starting from the “root” CA



## **4.2 Public Key Infrastructure**

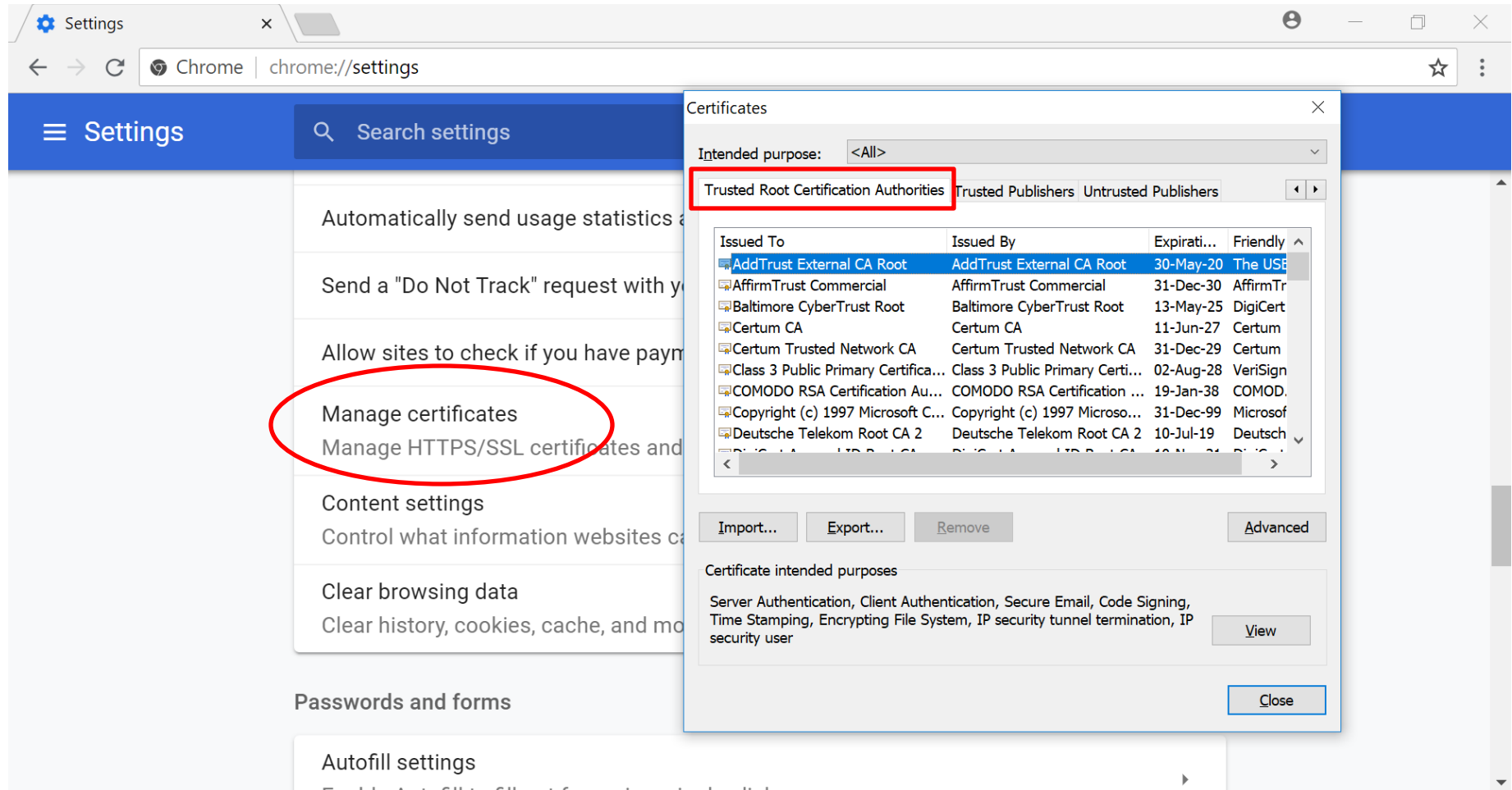
## **4.2.1 Certificate**

# Certificate Authority (CA)

- A **CA**:
  - Issues and signs digital certificates
  - Keeps a directory of public keys (more on this later)
  - It also has its own public-private key pair:  
we assume that the CA's public key **has been securely distributed** to all entities involved
- Most OSes and browsers have a few **pre-loaded CAs' public keys**: they are known as the “root” CAs
- Stringent **operational requirements** for a CA:
  - For example, it must pass **WebTrust audit**  
(<http://www.webtrust.org/homepage-documents/item76002.pdf>)

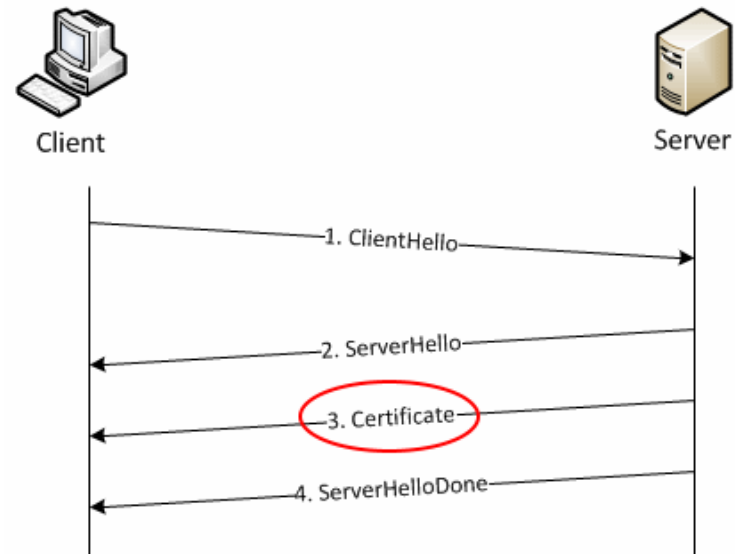
# Certificate Authority (CA)

Example: Root CAs in **Chrome** browser:



# Certificate: Content and Usage

- A **certificate** is a digital document that contains at least the following main items:
  - The identity of an owner, for e.g. [alice@yahoo.com](mailto:alice@yahoo.com)
  - The public key of the owner
  - The time window that this certificate is valid
  - The signature of the CA(It also has additional information like the intended purpose of the certificate: e.g. client authentication, secure email, ...)
- A certificate is **widely used** by Internet applications: SSL/TLS, S/MIME, SSH, ...
- Sample usage in the **SSL/TLS handshake protocol**:



# Role of a Certificate

- **Important question:** *Can a certificate-based PKI work **without** a publicly-available directory server?*
- Recall the method of using publicly-available directory
- Suppose we treat the CA as the **directory server**
- Note that there are **two issues** of retrieving the public key from the directory server whenever needed:
  - Bob, the verifier, needs to have **online access** to the CA at the verification point
  - The CA becomes a **bottleneck**
- Using certificates is a “smart” way of avoiding the above limitations!

# Without Certificate

We assume that Bob has the public key of CA. Hence the authenticity of the messages exchanged between them (i.e. Steps 2,3) can be verified.

alice@yahoo.com.sg	x1s34adf39
alice@yahoo.com	asd3123411
alice@cs.nyu.edu	2s3dasdf233
apple@google.com	a323fasdfas
.....	

Directory Server (CA)

(Step 2) Bob: What is the public key of alice@yahoo.com.sg?

(Step 3) CA: The public key of alice@yahoo.com.sg is x1s34adf39 and it is valid until 1 Sep 2020

Alice

Bob

From: alice@yahoo.com.sg  
Subject: Hello Bob  
Meeting 3pm at the usual place today.  
  
signature: xsdewsd

(Step 1) Alice: This is an email from alice@yahoo.com.sg.  
The email is “**signed**” using my private key.

CA’s public key

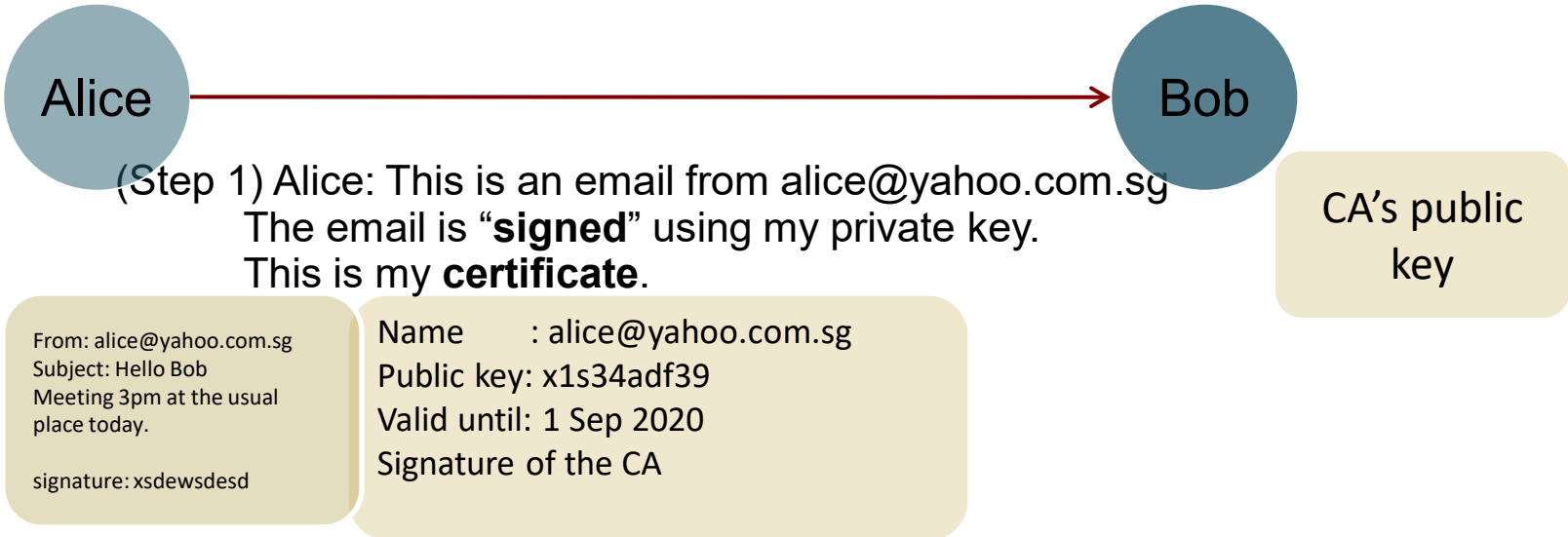
# With a Certificate

An “offline” CA would sign the message **beforehand** and pass it to **Alice**.  
Such a signed message is the certificate.

alice@yahoo.com.sg	x1s34adf39
alice@yahoo.com	asd3123411
alice@cs.nyu.edu	2s3dasdf233
apple@google.com	a323fasdfas
.....	

Directory Server (CA)

(Step 2) Bob **verifies** that the signature in the **certificate** is indeed signed by the CA.  
Since no one except the CA can produce the valid signature, the **authenticity of the information in the certificate** is as good as **coming directly** from the CA.





# Role of a Certificate: No Required Directory Server

- A CA (as certificate issuer) basically **binds** an entity with his/her public key prior to a verification point
- Now, with the certificate, Bob can obtain Alice's public key, and verifies its authenticity, ***even without a connection*** to the CA
- Notice, however, there is still a need to check *that the certificate hasn't been revoked*:
  - Online CRL Distribution Point(s) or OCSP Responder(s): to be discussed later

# X.509 Digital Certificate Standard

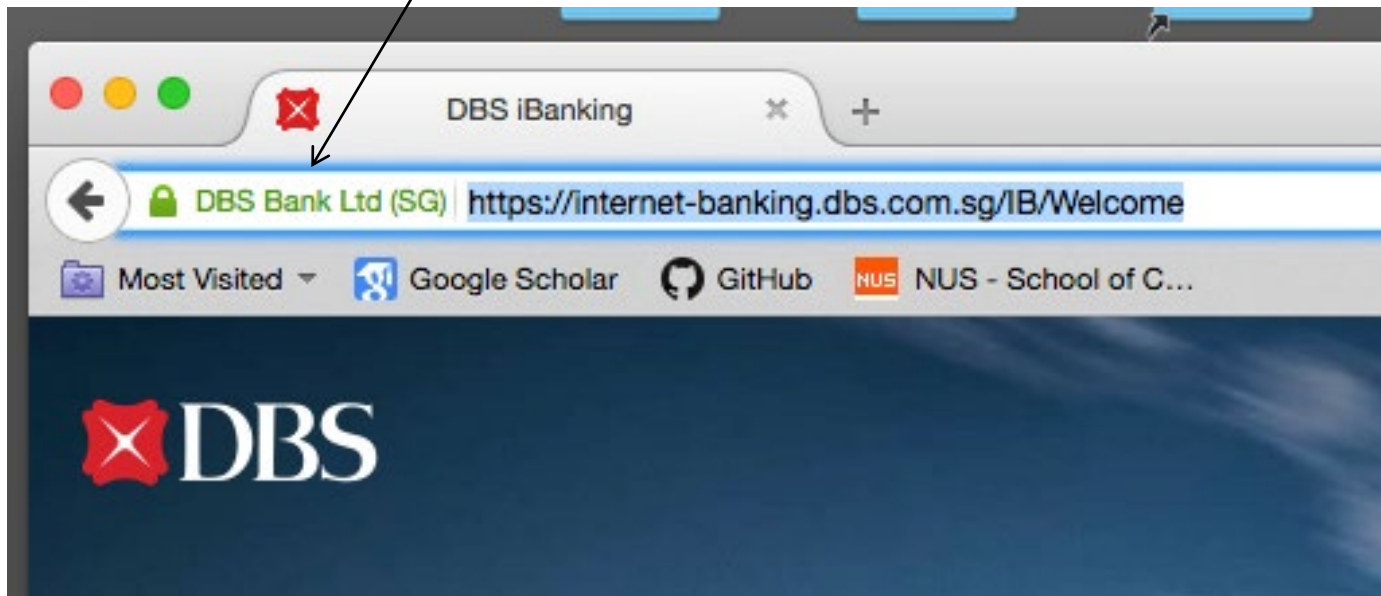
- Standardization **bodies**:
  - ITU-T X.509:  
Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
  - The Public-Key Infrastructure (X.509) Working Group (PKIX):  
IETF working group that creates Internet standards on issues related PKI based on X.509 certificates
- **Structure** of an X.509 v3 digital certificate:
  - Certificate:
    - Version Number
    - Serial Number
    - Signature Algorithm ID (*Note Signature Algorithm below too*)
    - **Issuer Name**
    - Validity period: Not Before, Not After

# X.509 Digital Certificate Standard

- **Subject Name**
  - Subject Public Key Info: Public Key Algorithm, Subject Public Key
  - Issuer Unique Identifier (optional)
  - Subject Unique Identifier (optional)
  - Extensions (optional)
- Certificate Signature Algorithm
- Certificate Signature
- ***Distinguished Name*** (DN) to identify an entity (e.g. **issuer and subject names**):
  - Common attribute types:  
Country (C), State (S), Locale (L), Organization name (O), Organizational unit name (OU), ***Common name*** (CN)
  - ***Common name*** can be an individual user or any other entity, e.g. a web server

# Example of Certificate

- Visit <https://internet-banking.dbs.com.sg/IB/Welcome>
- Check its certificate's detail  
(For Firefox, click the address bar)



# Example of Certificate

Certificate Viewer:"internet-banking.dbs.com.sg"

GeneralDetails

This certificate has been verified for the following uses:

SSL Client Certificate

SSL Server Certificate

Issued To

Common Name (CN)internet-banking.dbs.com.sg

Organisation (O)DBS Bank Ltd

Organisational Unit (OU)Technology and Operations

Serial Number66:77:FC:1D:EA:3E:C0:39:2A:9D:4A:8E:75:55:0C:83

Issued By

Common Name (CN)Symantec Class 3 EV SSL SGC CA - G2

Organisation (O)Symantec Corporation

Organisational Unit (OU)Symantec Trust Network

Period of Validity

Begins On15/12/14

Expires On11/1/16

Fingerprints

SHA-256 Fingerprint2A:6B:45:E2:C7:6E:69:8B:1C:9A:71:20:06:DD:67:FE:B2:28:F0:8E:13:43:4D:25:05:20:A6:60:83:D4:37:DC

SHA1 FingerprintFA:78:AD:48:D7:21:31:E2:65:3E:38:90:FB:E3:C8:E9:15:D4:C4:B9

Close

Certificate Viewer:"internet-banking.dbs.com.sg"

GeneralDetails

Certificate Hierarchy

VeriSign Class 3 Public Primary Certification Authority - G5

Symantec Class 3 EV SSL SGC CA - G2

internet-banking.dbs.com.sg

Certificate Fields

Issuer

Validity

Subject

Subject Public Key Info

Subject Public Key Algorithm

Subject's Public Key

Extensions

Certificate Signature Algorithm

Certificate Signature Value

Field Value

Modulus (2048 bits):

d3 ea 34 0b ba 79 70 14 2e e5 24 d2 dc 6c 98 85

8e 18 a1 51 e9 b0 e8 23 7a 2c 9b c4 cc 37 d2 f9

a5 bb 66 7c a1 14 68 54 76 3f 01 95 4b 75 5b 55

3f fe d6 a6 a6 cd 62 39 62 aa 2f 8f 50 d0 35 cc

7c fb 51 89 e6 61 99 8f b2 5d c0 1f 9c 13 67 26

6a 19 ed a4 e2 9e 2c f2 71 e8 c4 d7 07 b4 f5 65

af e8 82 f9 be ad 8b bc 6d c9 b7 28 4f 56 a1 9e

0e 44 49 10 29 4a 79 bf 31 17 16 73 50 a7 fc 8e

98 aa c5 e1 99 ca ea 29 8f 98 68 d2 88 0b 01 c8

Export...

Close

# How Do I Get a Certificate?

- Get a **root CA** to issue you one:
  - Paid ones: \$10 - \$50 / year (not costly)
- “*Let's Encrypt*” provides (basic) TLS certs at **no charge**:
  - Launched in April 2016
  - A certificate is valid for 90 days
  - Its renewal can take place at anytime
  - Automated process of cert creation, validation, signing, installation, and renewal
  - No of issued certs: **1M** (March 2016) to **380M** (Sept 2018)
- Firefox Telemetry:
  - **77%** of all page loads via Firefox are now encrypted
  - It is predicted that it will reach **90%** by the end of 2019

# Certificate: Summary

- A certificate is simply a document signed by a CA that specifies:
  1. An identity
  2. The associated public key
  3. The time window that this certificate is valid
  4. The signature of the CA
- The certificate “certifies” that the public key indeed belongs to the stated identity
- We assume that Bob already has the CA’s public key installed in his machine

## **4.2.2 Certificate Authority & Trust Relationship**



# Responsibility of a CA

- The CA, besides issuing certificate, is also responsible to **verify** that the information is correct
- For instance, if someone wants request for a certificate for [www.nus.edu.sg](http://www.nus.edu.sg), the CA should check that the applicant indeed **owns** the above **domain name**
- This may involve some manual checking and thus it could be costly, especially for the **Extended Validation SSL (EV SSL)** certificates:
  - Latest initiative by CA/Browser Forum
  - The highest “class” of SSL certificates with more stringent checks done
  - Activate both the padlock and **the green address bar** in major browsers!

# What are Checked by a CA before a Certificate Issuance?

- **Domain Validation (DV)** SSL certificate:
  - Issued if the purchaser can demonstrate the right to administratively **manage a domain name**, (e.g. response to email sent to the email contact in whois details, publishing a DNS TXT record)
- **Organization Validation (OV)** SSL certificate:
  - Issued if the purchaser additionally has an organization's actual **existence as a legal entity**
- **Extended Validation (EV)** SSL certificate:
  - Issued if the purchaser can persuade the cert provider of its legal identity, including **manual verification checks** by a human

# Types of SSL Certificates

- Read: <https://www.ssl.com/article/dv-ov-and-ev-certificates/>
- Summarized below:

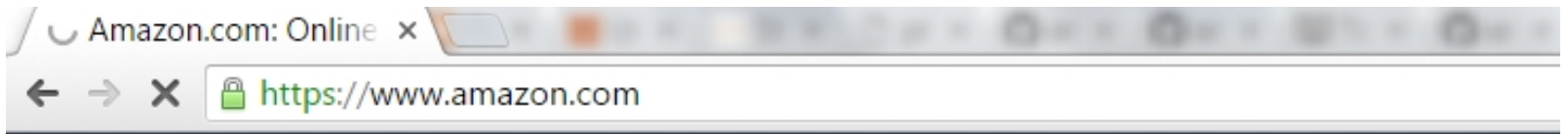
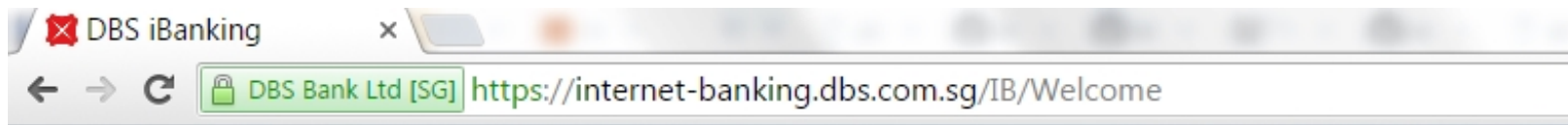
**TLS Certificate Level Summaries**

Certificate type	HTTPS encrypted?	Padlock displayed?	Domain validated?	Address validated?	Identity validation	Green address bar?
DV	Yes	Yes	Yes	No	None	No
OV	Yes	Yes	Yes	Yes	Good	No
EV	Yes	Yes	Yes	Yes	Strong	Yes

Source: PCI Security Standards Council, "Best Practices for Securing E-commerce", [https://www.pcisecuritystandards.org/pdfs/best\\_practices\\_securing\\_ecommerce.pdf](https://www.pcisecuritystandards.org/pdfs/best_practices_securing_ecommerce.pdf)

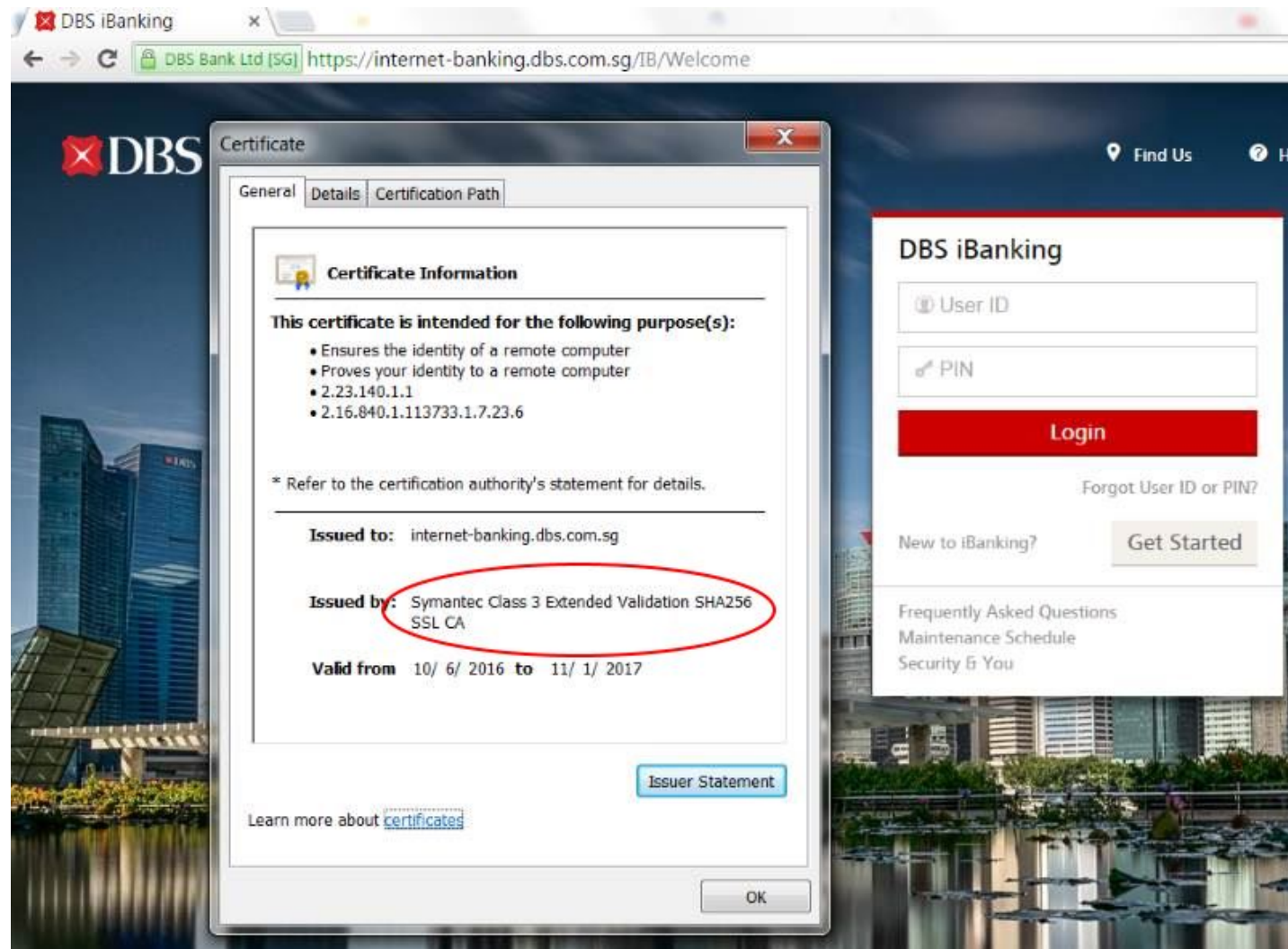
# Browser UI Security Indicators

- Browsers offer users different **visual-based security indicators** on different types of certificates
- Examples: Two different domains as shown by Chrome browser below
- Can you guess which types of certificates used?



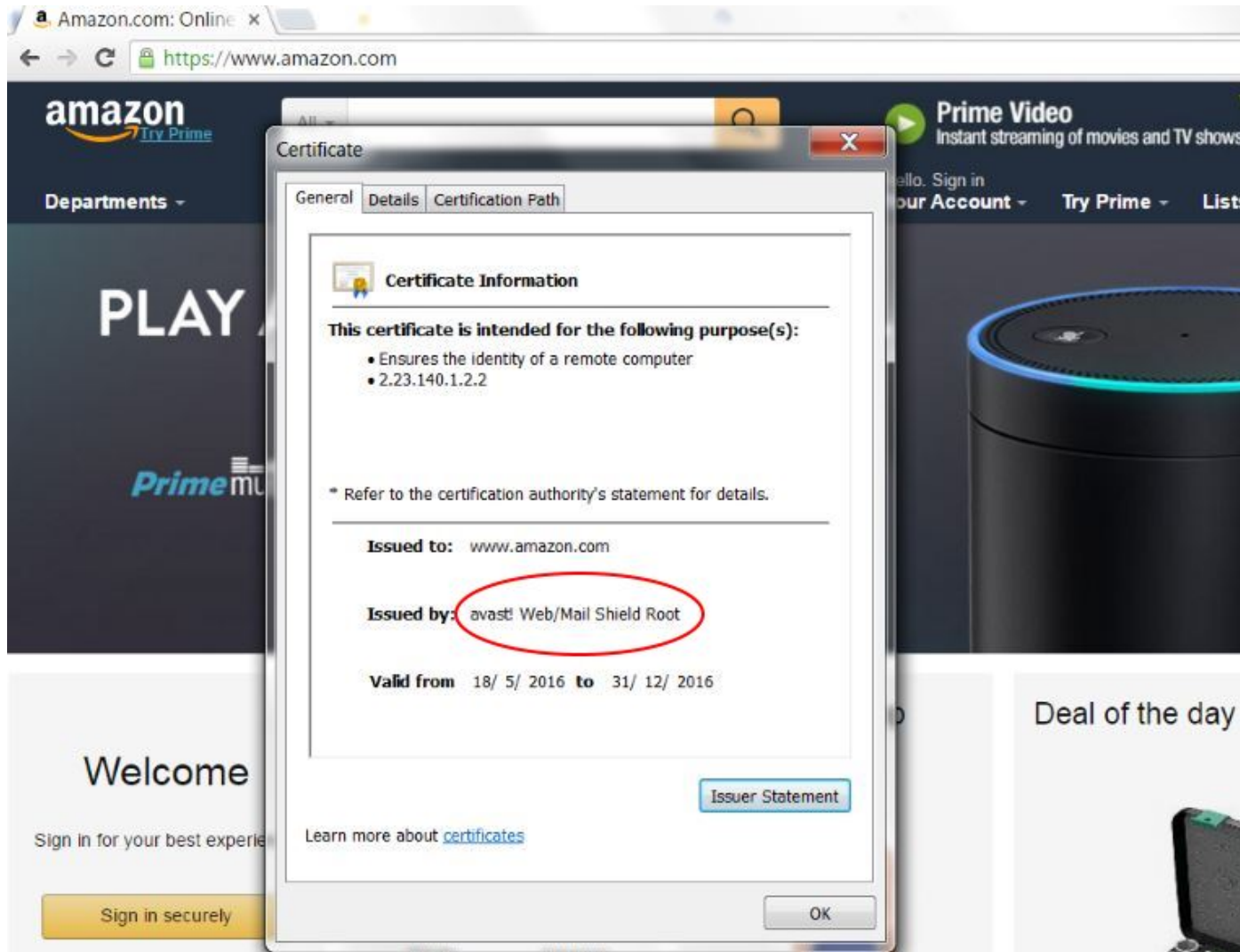
# EV SSL Certificate

- DBS Internet banking:



# Standard SSL Certificate

- www.amazon.com:





# Browser UI Security Indicators

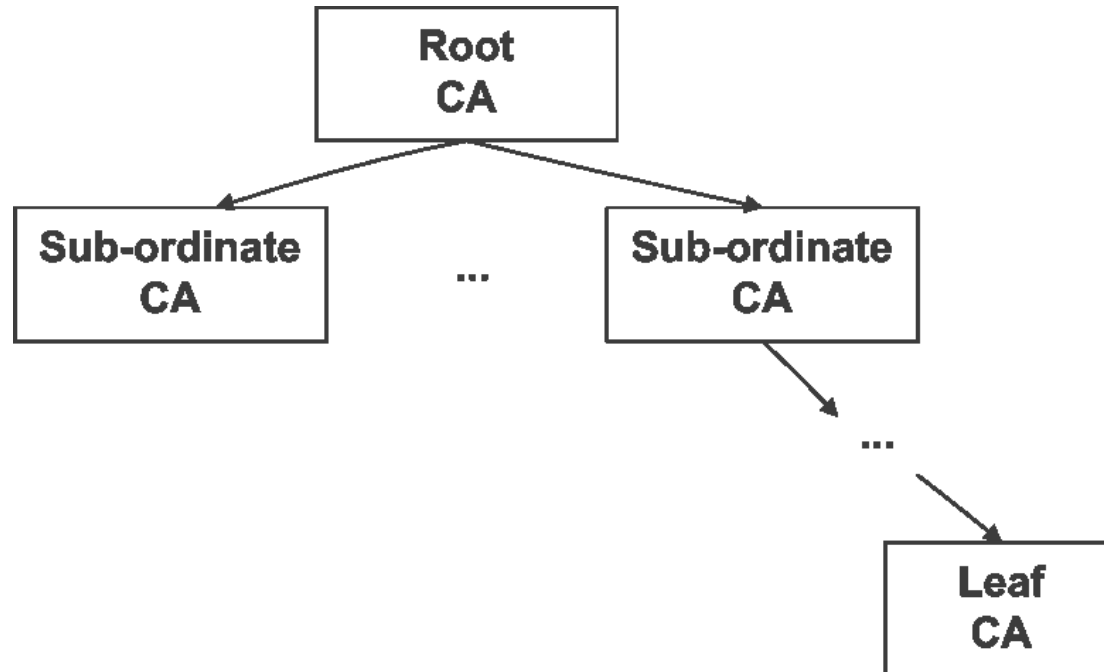
✓ Browser UI Security INDICATORS as of March 2017: Updated URL UIs indicated by orange outlined box

Browser UI Security Indicator:	HTTP only (no certificate)	DV certificate	OV certificate	EV certificate
Chrome 56 ( Windows )	www.example.com	Secure   https://case	Secure   https://www	Trustwave Holdings, Inc. [US]   https://www.trust
Chrome 56 ( Android )	www.example.com	https://casecurity.cor	https://www.example	https://www.entrust.com
Edge 20 ( Windows )	example.com	casecurity.org	example.com	Symantec Corporation [US] symantec.com
Firefox 51 ( Windows )	www.example.com	https://casecurity	https://www.exai	COMODO CA Limited (GB)   https://crt.sh
Safari 10 ( Mac )	www.example.com	https://casecurity	https://www.exai	GMO GlobalSign Inc
Safari 10 ( iOS )	example.com	casecurity.org	example.com	DigiCert, Inc.
OperaMini 23 ( Android )	www.example.com	casecurity.org	www.example.cor	www.globalsign.com/en/
UC Mini 10 ( Android )	Example Domain	CA Security Council	Example Domain	https://www.digicert.com
UC Browser 10.8.6.889 ( iOS )	example.com	CA Security Council	example.com	SSL & Digital Certificates by GlobalSign

Source <https://casecurity.org/wp-content/uploads/2017/03/CASC-Browser-UI-Security-Indicators.pdf>

# Types of CA

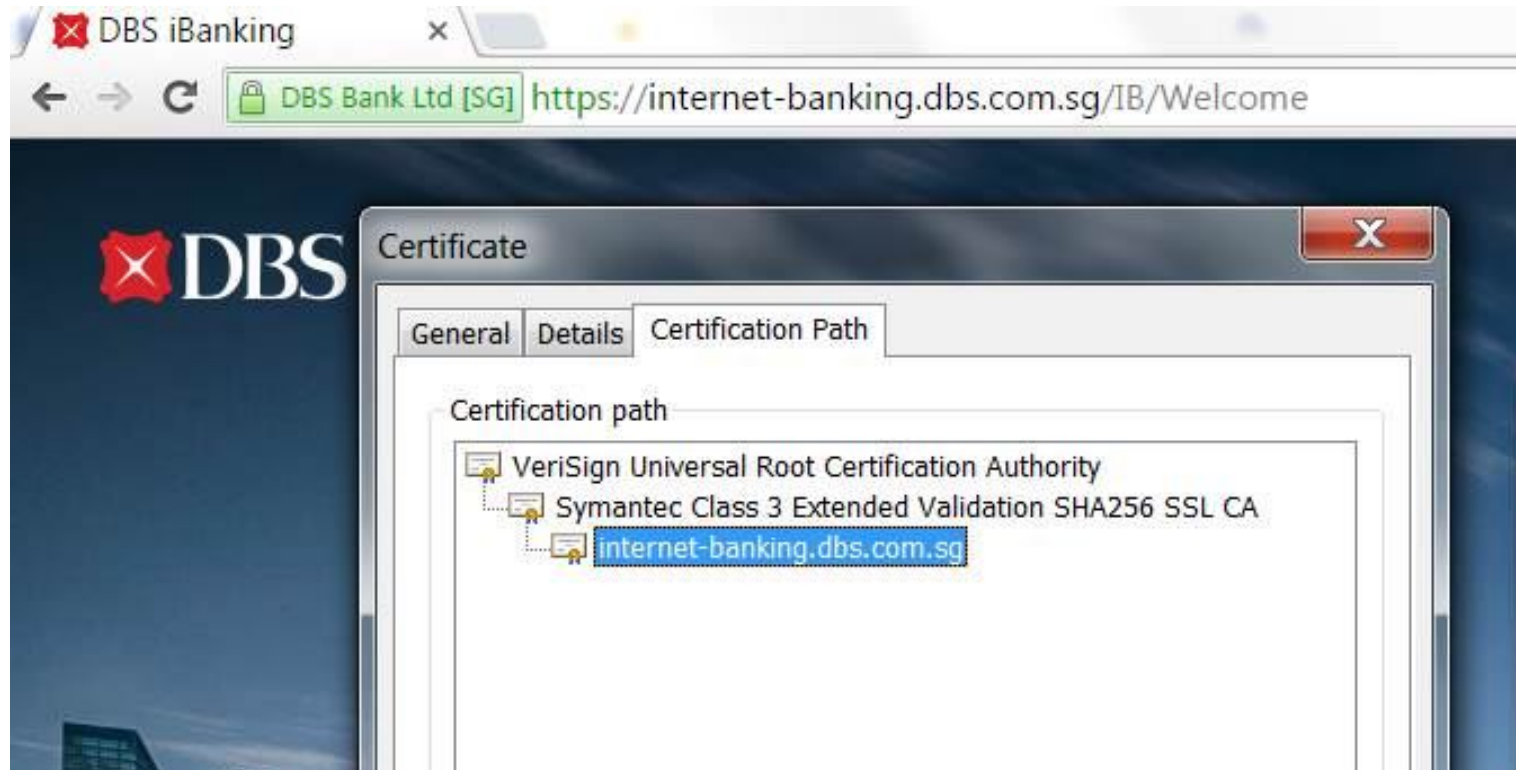
- There are **3 different types** of CA:
  - *Root CA*: whose certificate is self-signed
  - *Sub-ordinate/intermediate CA*: Tier 1, 2, ...
  - *Leaf CA*





# Types of CA: A Real Example

- DBS Internet banking website:



# Hierarchy of Trust

## Trust inference:

**Bob trusts CA #1:** because Bob trusts the *Root CA*, and the *Root CA* certifies CA #1

**Bob also trusts CA #3** and all certificates signed by CA #3: since *CA #1* certifies CA #3.

Hierarchy of Trust

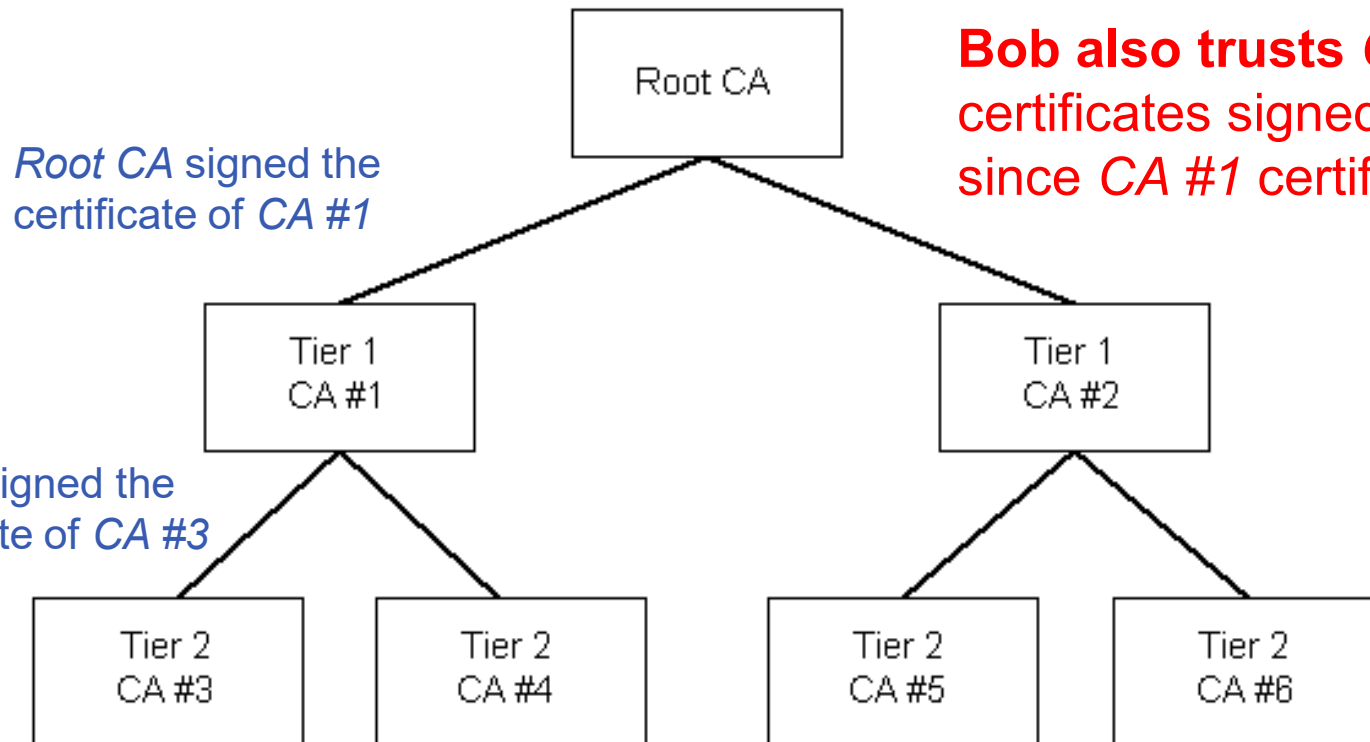


image from  
<https://msdn.microsoft.com/en-us/library/windows/desktop/aa382479%28v=vs.85%29.aspx>

See [PF] pages 117-121 for a detailed explanation of **Trust**

# Certification Chain/Path Verification: An Example

- Suppose Alice's certificate is issued & signed by  $CA_1$ , which is a tier-1 intermediate CA
- Bob **doesn't have** the public key of  $CA_1$
- *Why should Bob do?*
- In the first place, Alice, anticipating the Bob might not have  $CA_1$ 's public key, can send Bob her email, her certificate, and  $CA_1$  certificate (*see the next slide*)
- Now, Bob can:
  - Verify  $CA_1$ 's certificate: using root CA's public key
  - Verify Alice's certificate: using the verified  $CA_1$ 's public key
  - Verify Alice's email: using Alice's verified public key
- If Alice doesn't attach  $CA_1$ 's certificate, then Bob has to obtain it from other sources

# Certification Chain/Path Verification: An Example

- Illustration:

From: alice@yahoo.com.sg  
Subject: Hello Bob  
Meeting 3pm at the usual  
place today.

Signature: xsdewsdesd

Name : alice@yhoo.com.sg  
Public key: x1s34adf39  
Valid until: 1 Sep 2019  
Signature of the CA<sub>1</sub>

Name : CA<sub>1</sub>  
Public key: x3141342  
Valid until: 1 Sep 2020  
**Note: CA<sub>1</sub> can issue certificates**  
Signature of the Root CA

- In our example, CA<sub>1</sub>'s certificate clearly indicates that CA<sub>1</sub> is a CA that can issue certificate
- Without that "Note" portion, the certificate owner can't issue other certificates

# Certification Chain: Definition

## *Certification chain/path:*

- A **list of certificates** starting with an **end-entity certificate** followed by **one or more CA certificates** (with the last one being a self-signed **root certificate**)
- For each certificate (except the last one):
  - The **issuer** matches the **subject** of the next certificate in the list
  - It is **signed** by the **private key** of the next certificate in the list
- The last certificate in the list, i.e. the root CA's, is the **trust anchor**

# Certification Chain: Diagram and Verification

- How does a certificate chain get verified?

## End-entity Certificate

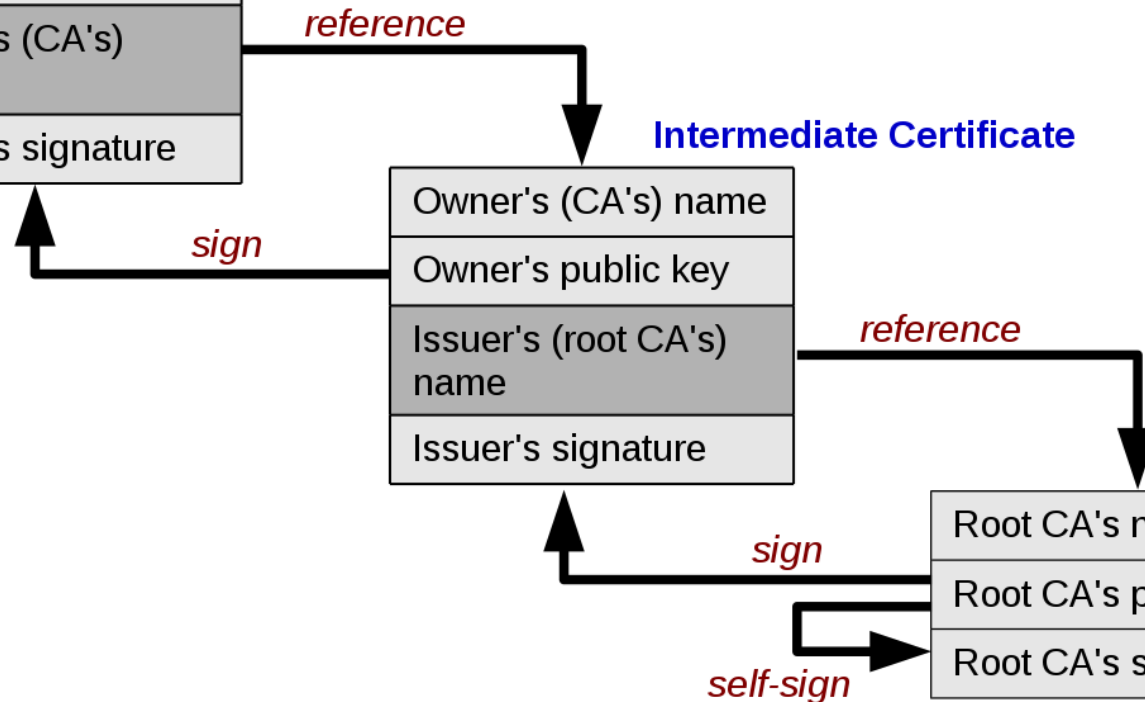
Owner's name
Owner's public key
Issuer's (CA's) name
Issuer's signature

## Intermediate Certificate

Owner's (CA's) name
Owner's public key
Issuer's (root CA's) name
Issuer's signature

## Root Certificate

Root CA's name
Root CA's public key
Root CA's signature



Source: Wikipedia

# Some Questions:

- Occasionally, while surfing the web, you may encounter this **warning** message:

*www.example.com uses an invalid security certificate.*

*The certificate is not trusted because the issuer certificate is unknown.*

*Option 1: Get me out of here.*

*Option 2: I know the risk. Accept the certificate.*

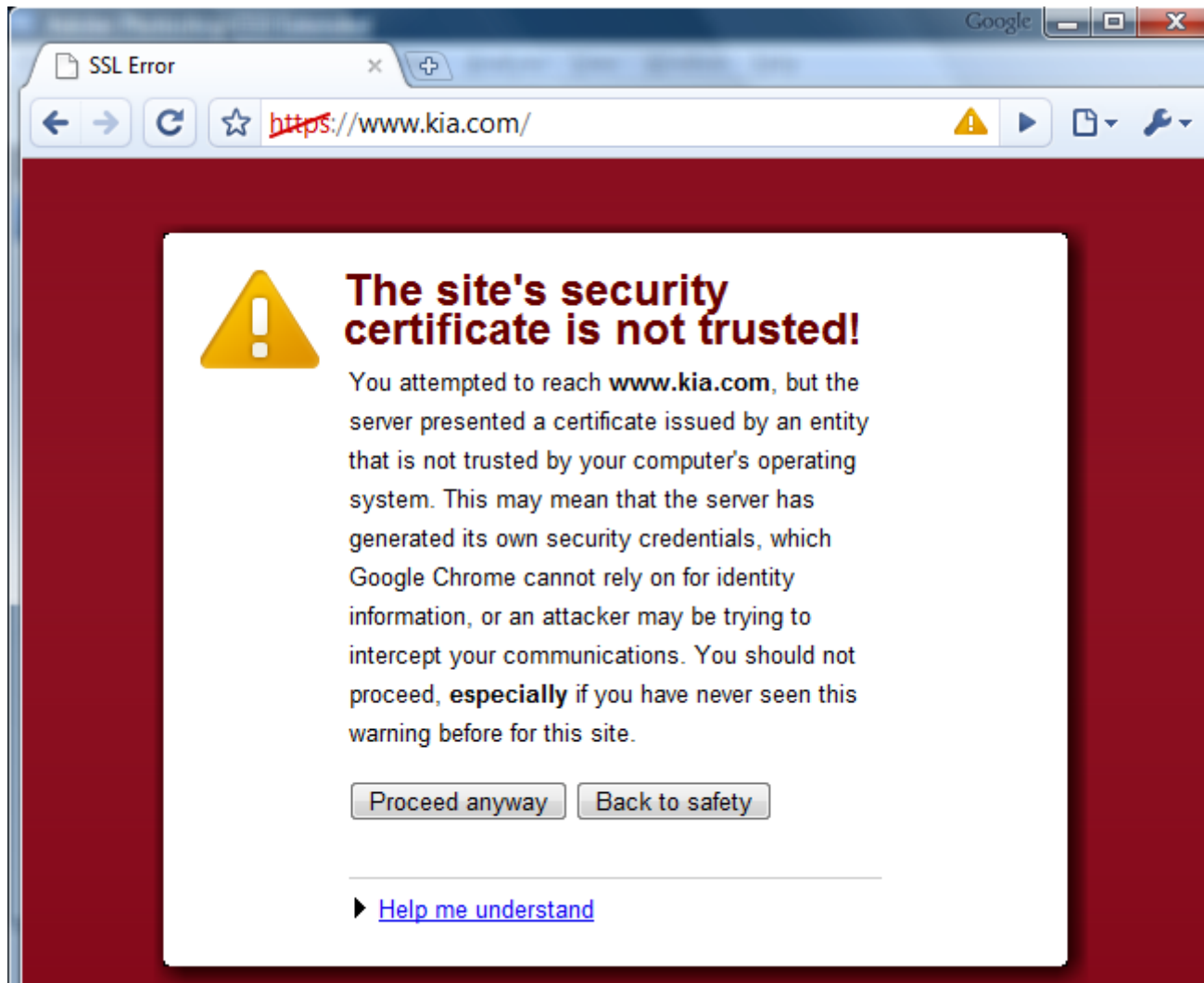
What is going on here? What are the security implications?

- While **installing** a new package using a package manager (this applied to MAC OS, Linux, cgywin, etc.), say `apt-get`, you may also encounter similar message:

*Packages server certificate verification failed.*

What is going on here? Should you continue the installation?

# Some Questions: Sample Alert Box





# Another Issue: Certificate Revocation

- **Non-expired certificates** can be **revoked** for different reasons:
  - Private key was compromise
  - Issuing CA was compromise
  - Entity left an organization
  - Business entity closed
- A verifier needs to check if a certificate in question is **still valid**, although the certificate is not expired yet
- Different **approaches** to certificate revocation:
  - **Certificate Revocation List (CRL):**  
CA periodically signs and publishes a revocation list
  - **Online Certificate Status Protocol (OCSP):**  
OCSP Responder validates a cert in question
- An **online** CRL Distribution Point or OCSP Responder is needed

## Another Issue: Certificate Revocation

- As of Firefox 28, Mozilla have announced they are **deprecating** CRL in favor of OCSP
- Some OCSP problems:
  - **Privacy**: OCSP Responder knows certificates that you are validating
  - **Soft-fail validation**: Some browsers proceed in the event of no reply to an OCSP request (no reply *is* a “good” reply)
- Solution/improvement?
  - **OCSP stapling**: allows a certificate to be accompanied or “stapled” by a (time-stamped) **OCSP response** signed by CA
  - **Part of TLS handshake**: clients **do not** need to contact CA or OCSP Responder
  - **Drawback**: increased network cost

## **4.3 Limitations/Attacks on PKI**

# Compromised CAs

## CA breach incidents:

### Four CAs Have Been Compromised Since June

Posted by **Soulskill** on Friday October 28 2011 , @04:08PM  
from the four-whole-californias-woww dept.

- DigiNotar (Netherlands):
  - Resulted in 500+ fraudulent certificates, including for \*.google.com, \*.mozilla.com, \*.windowsupdate.com, \*.torproject.org, in Sept 2011
  - Immediately removed by major browsers
  - Declared bankrupt within the same month
- Turktrust (Turkey):
  - Its sub-ordinate CA, \*.EGO.GOV.TR, issued \*.gmail.com certificates
  - Fraudulent certificates were used against Google Web properties

# Abuse by CA

- There are so many CAs: Some of them could be **malicious**
- A **rogue CA** can practically forge any certificate.  
Here is a well-known incident.
- **Trustwave** issued a “sub-ordinate root certificate”, which can then issue other certificates, to an organization for monitoring the network.  
With this certificate, the organization can “**spoof**” X.509 certificates, and hence is able to act as the **man-in-the-middle** of any SSL/TLS connection.
- See:  
ComputerWorld, *Trustwave admits issuing man-in-the-middle digital certificate; Mozilla debates punishment*, Feb 8 2012.  
<http://www.computerworld.com/article/2501291/internet/trustwave-admits-issuing-man-in-the-middle-digital-certificate--mozilla-debates-punishment.html>

# Another Famous Case of CA's Abuse (or Ignorance?)

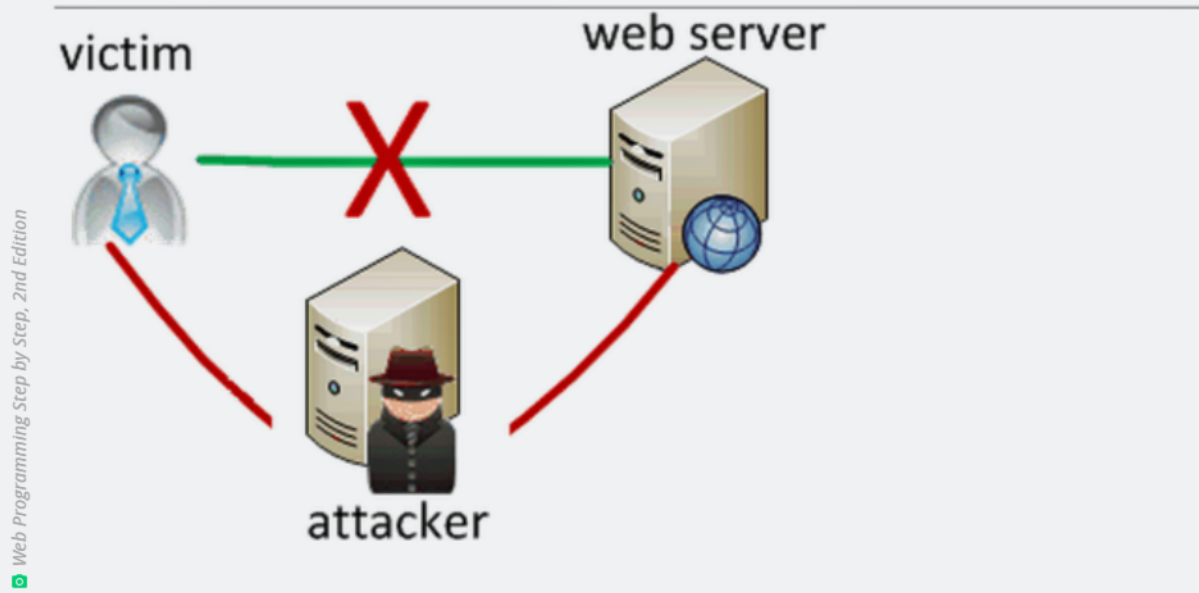
- Lenovo's **SuperFish** scandal (*reserved for class presentation*)

BIZ & IT —

## Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections [Updated]

Superfish may make it trivial for attackers to spoof any HTTPS website.

DAN GOODIN - 2/20/2015, 12:36 AM

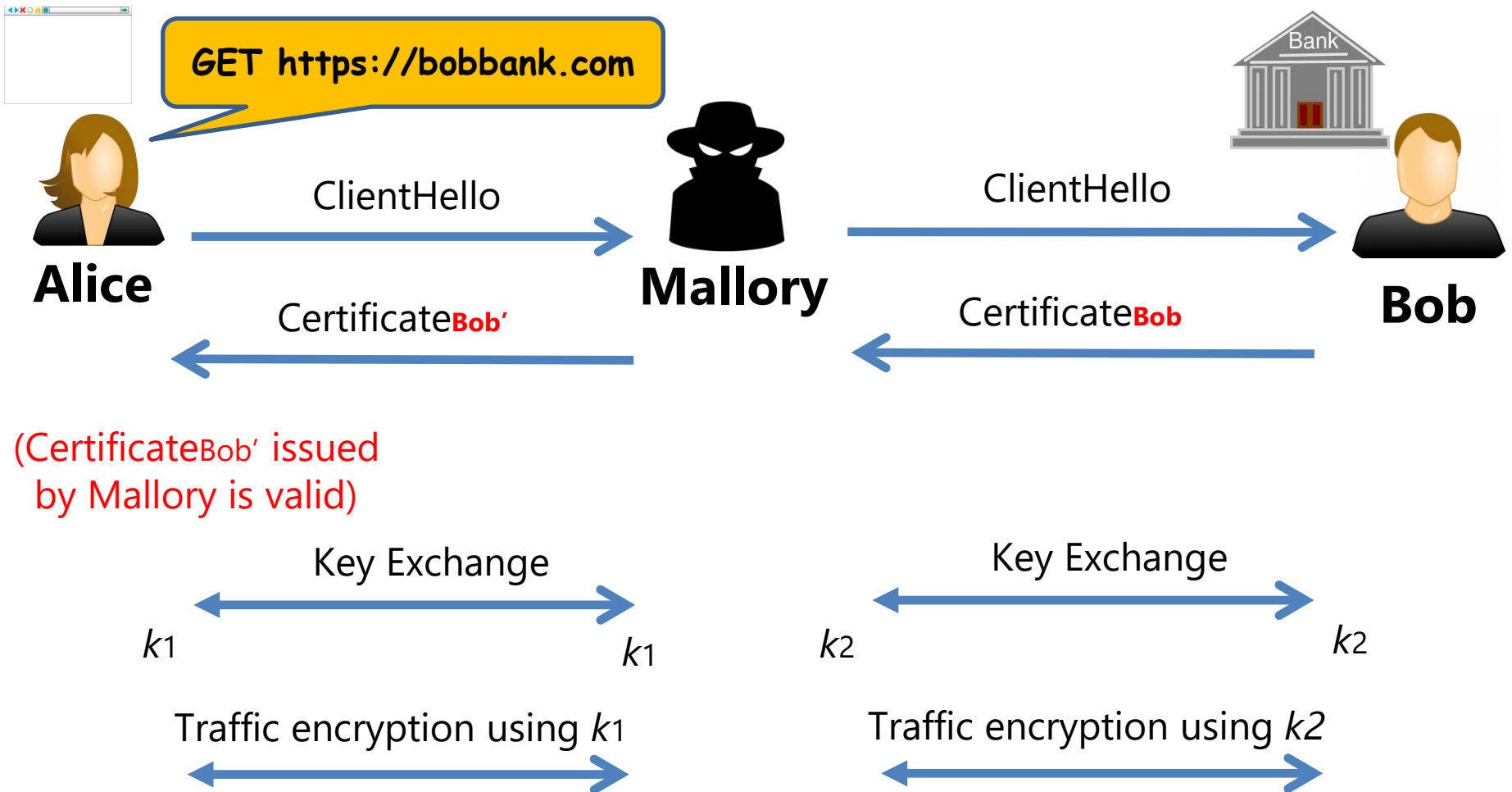


Source: <https://arstechnica.com>

# Weak Browser Trust Model

- **Browser trust model:**
  - A **pre-loaded list** of widely-used root CAs compiled by web browser developers
  - An form of *Certificate Trust List (CTL) approach*, where a list of CAs' certificates are compiled by a “trusted” authority
- **Security issue:**
  - Trust anchor: the **union** of all root CAs
  - Question: **which root CA** is the one used from the root-CA list?
  - Certification is only as *strong as the weakest root CA*!
- *See real-world analyses in:*
  - Peter Eckersley, Jesse Burns, “An observatory for the SSLiverse”, Defcon 18, 2010.
  - Peter Eckersley Jesse Burns, “Is the SSLiverse a Safe Place?”, 27<sup>th</sup> Chaos Communication Congress (CCC), 2010.

# MITM Attack by a Rouge CA



Mallory performs a *proxy re-encryption*.  
He can see all traffic, and also modify data!  
(To be discussed in Tutorial)



# Implementation Bugs: E.g. Null-byte Injection Attack

- There are quite a number of well-known implementation bugs leading to severe vulnerability. Here is one example.
- Some browsers ignore the substrings in the entity's identity/name field **after** the null characters **when displaying it in the address bar, but** include them **when verifying the certificate**.

The null character is displayed as the string “\0”

(a) The common name in the cert when it is **being verified**:

“www.comp.nus.edu.sg\0.hacker.com”

(b) The browser displays it as:

“www.comp.nus.edu.sg”

- As a result, the user **thought** he/she is connecting via https to www.comp.nus.edu.sg, **but in fact** to www.comp.nus.edu.sg\0.hacker.com.
- See also:

[www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/](http://www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/)

**Question (Terminologies):** What is CVE?

# Social Engineering

Malicious hackers may carry out ***typosquatting***.

For example:

1. A hacker **registered** for a domain name `luminus.nvs.edu.sg`, and **obtained a valid certificate** of the name
2. The hacker employs a **phishing attack**, tricking a victim to click on the above link, which is a spoofed site of `luminus.nus.edu.sg`
3. The address bar of the victim's browser **correctly displays** `https://luminus.nvs.edu.sg`, but the victim doesn't notice that, and log in using the victim's credential

It is also possible that the hacker doesn't carry out step 2.

He just waits and hopes that some students accidentally type the wrong address `luminus.nvs.edu.sg`.

**Read** <http://en.wikipedia.org/wiki/Typosquatting>