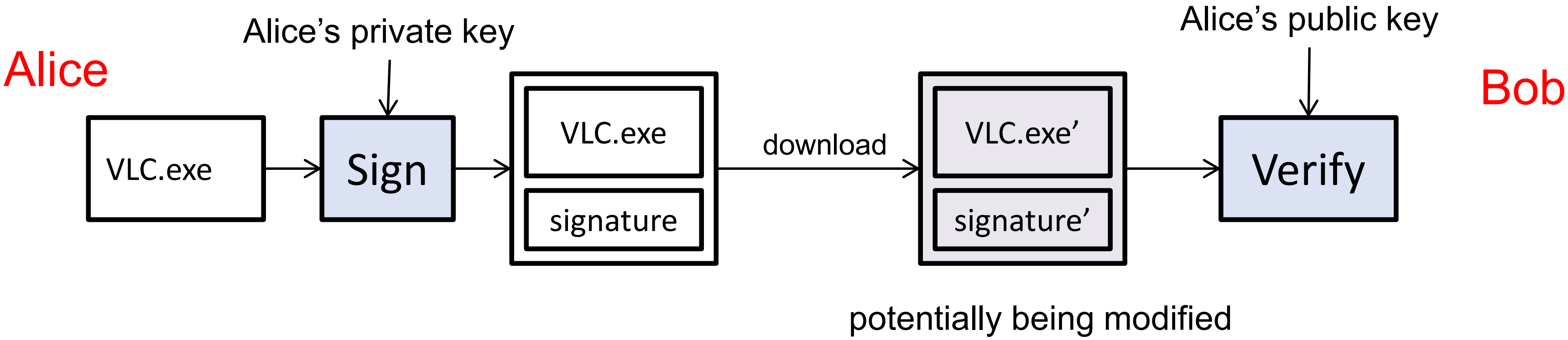
Lecture 4: PKI + Channel Security

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 method to assure the authenticity of the downloaded program VLC.exe 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

-----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+yAKC7JVI1eY+aHEAqqCeVdYGOE10PmwCg9DrE
ArgWymKbDnl7m9W1leVeQqM=
=EksE
-----END PGP SIGNATURE-----
```

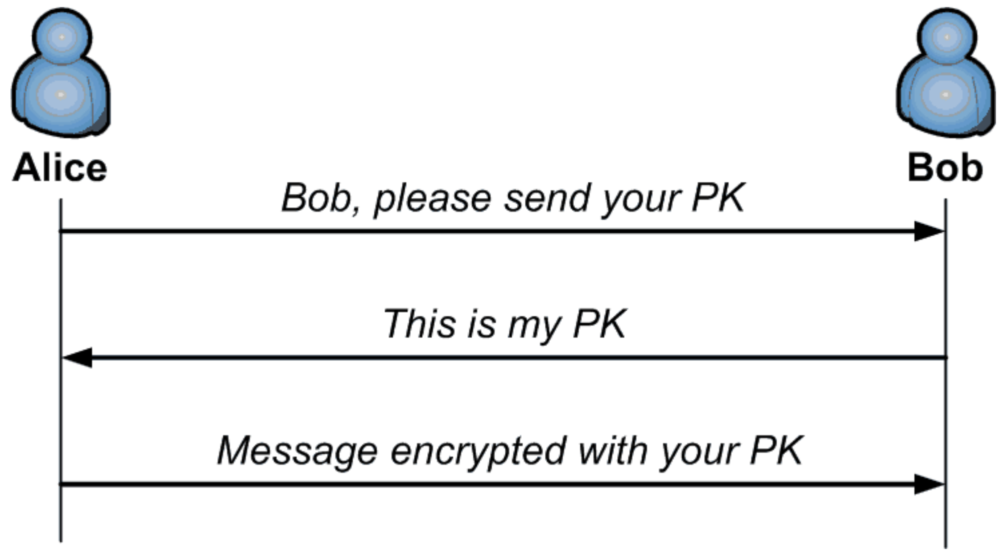
Header (unsigned, i.e. not included in computing the signature)

(signed) Message

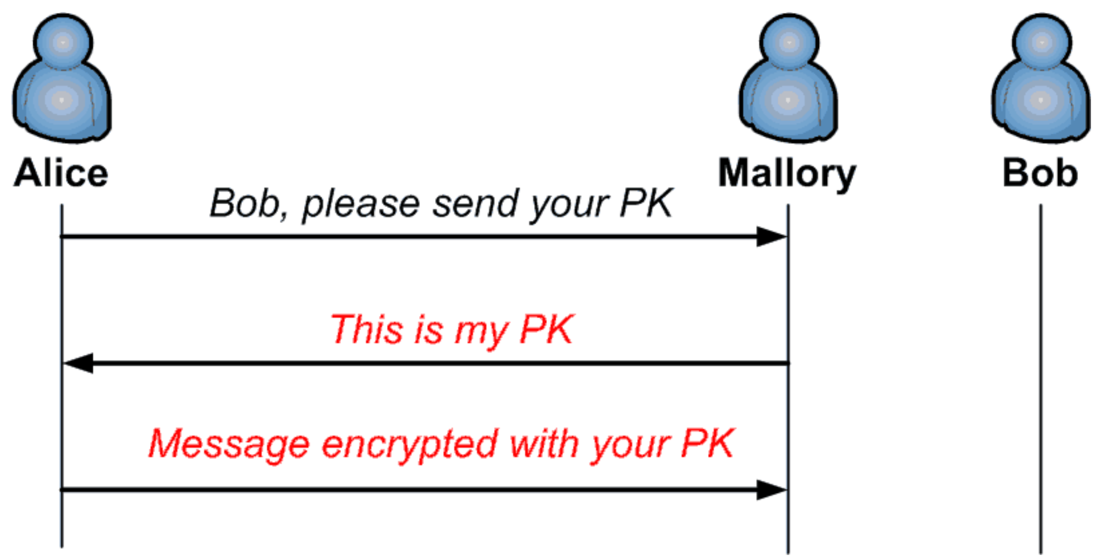
Signature

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

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

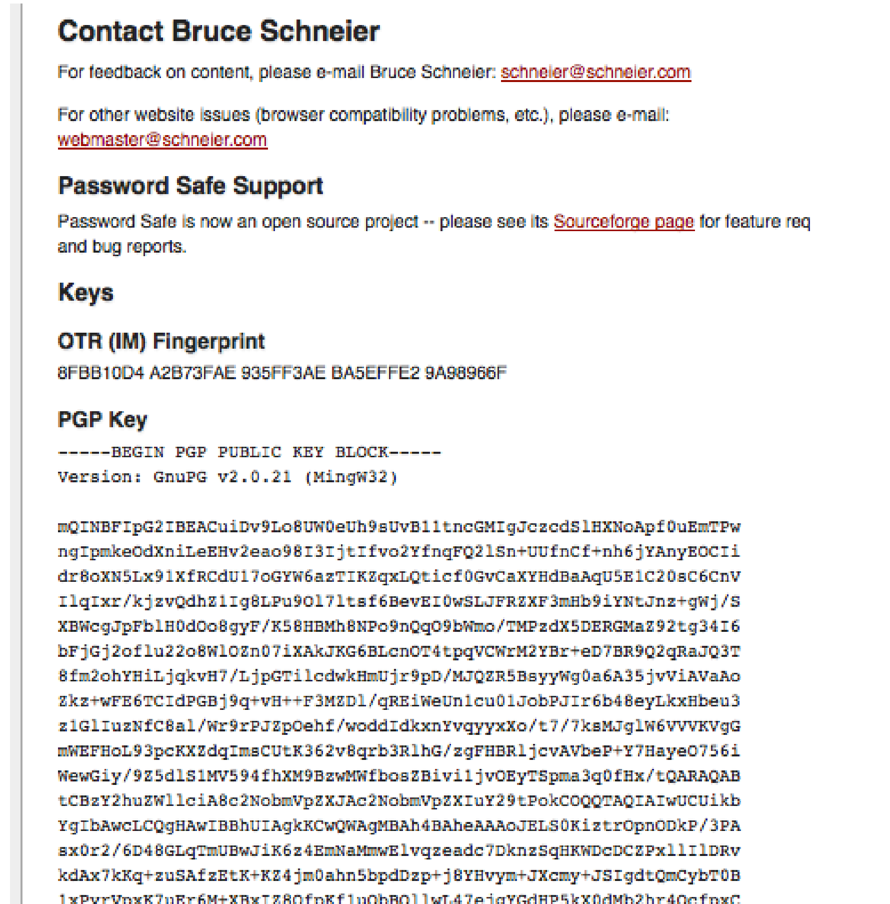
Possible Public-Key Distribution Methods

- 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

For other website issues (browser compatibility problems, etc.), please e-mail: 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 BA5EFFE2 9A98966F

PGP Key


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.21 (MingW32)

mQINBFIpG2IBEACuiDv9Lo8UW0eUh9sUvB11tncGMIgJczcdSlHXNoApf0uEmTPw
ngIpmkeOdXniLeEHv2eao98I3IjtIfvo2YfnqFQ2lSn+UUfnCf+nh6jYAnyEOCIi
dr8oXN5Lx91XfRCdU17oGYW6azTIKZqxLQticf0GvCaXYHdBaAqU5E1C20sC6CnV
IlqIxr/kjzvQdhZ1Ig8LPu9Ol7ltsf6BevEI0wSLJFRZXF3mHb9iYNtJnz+gWj/S
XBWcgJpFblH0dOo8gyF/K58HBMh8NPo9nQqO9bWmo/TMPzdX5DERGMaZ92tg34I6
bFjGj2oflu22o8WlOZn07iXAkJKG6BLcnOT4tpqVCWrM2YBr+eD7BR9Q2qRaJQ3T
8fm2ohYHiLjqkvH7/LjpGTilcdwkHmUjr9pD/MJQZR5BsyyWg0a6A35jvViAVaAo
Zkz+wFE6TCIdPGBj9q+vH++F3MZDl/qREiWeUn1cu01JobPJIr6b48eyLkxHbeu3
z1GlIuzNfC8al/Wr9rPJZpOehf/woddIdkxnYvqyyxXo/t7/7ksMJglW6VVVKVgG
mWEFHoL93pcKXZdqImsCUtK362v8qrb3RlhG/zgFHBRljcvAVbeP+Y7HayeO756i
WewGiy/9Z5dlS1MV594fhXM9BzwMWfbosZBivi1jvOEyTSpma3q0fHx/tQARAQAB
tCBzY2huZWllciA8c2NobmVpZXJAc2NobmVpZXIuY29tPokCOQQTAQIAIwUCUikb
YgIbAwcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJELS0KiztrOpnODkP/3PA
sx0r2/6D48GLqTmUBwJiK6z4EmNaMmwElvqzeadc7DknzSqHKWDcDCZPxllIlDRv
kdAx7kKq+zuSAfzEtK+KZ4jm0ahn5bpdDzp+j8YHvym+JXcmy+JSIgdtQmCybT0B
```

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

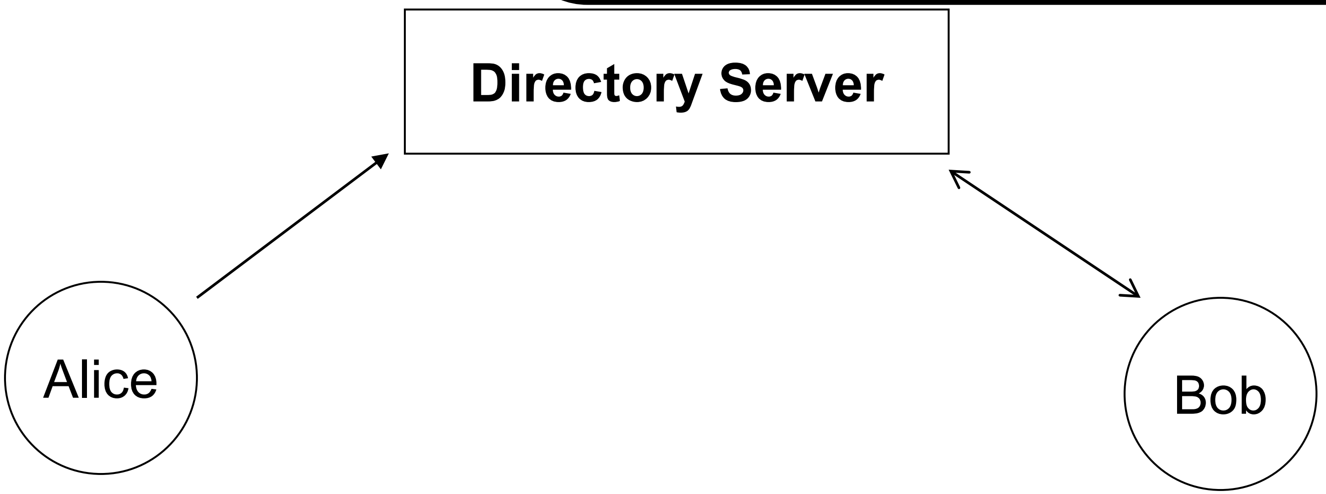
If Bob wants to find the public key associated with the name

`alice@yahoo.com.sg`

he can search the public directory by querying a server

Public-Key Directory:

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



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

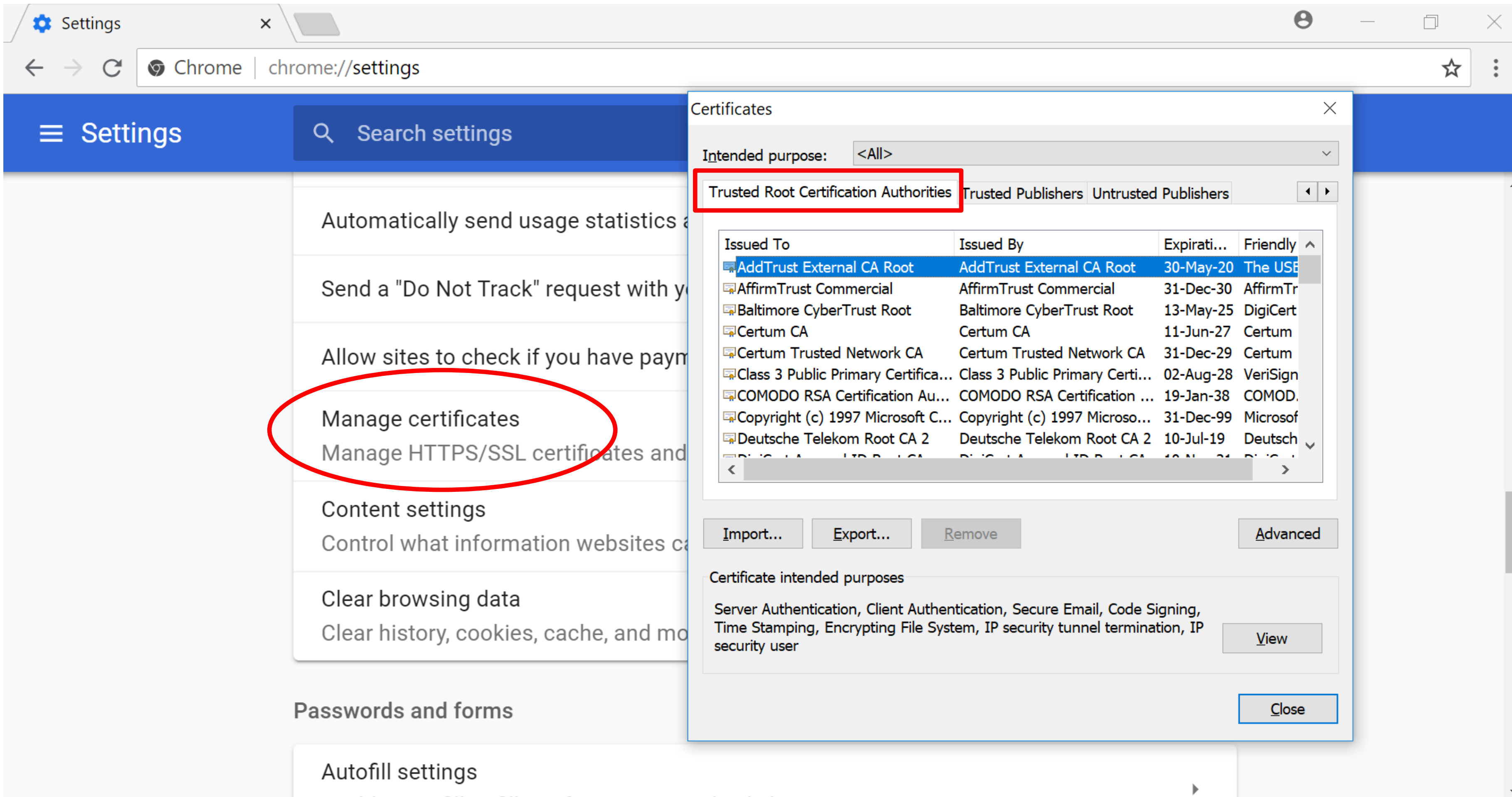
4.2.1 Certificate

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:

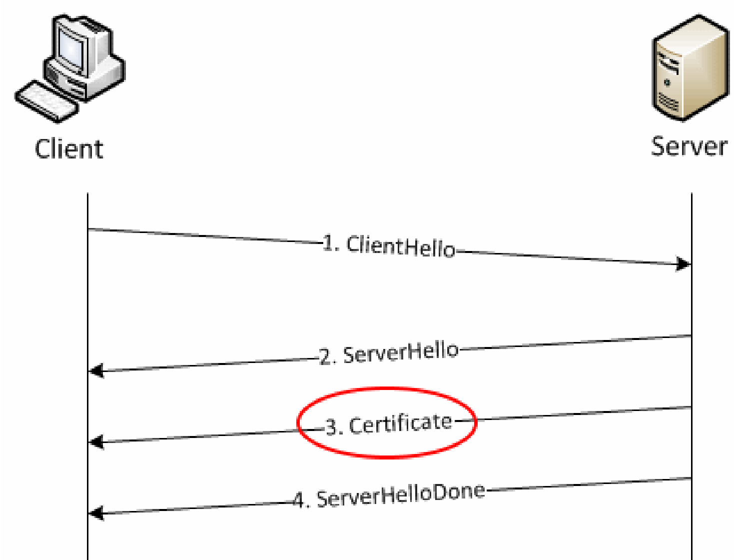


Certificate: Content and Usage

- 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
  - 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
- Now, treat CA as the **directory server**
- Note that there are **two issues** of retrieving the public key from the directory server whenever needed:
  - The CA becomes a bottleneck
  - Bob, the verifier, needs to has online access to the CA at the verification point
- 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

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

CA's public key

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

signature: xsdewsdesd

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.

Alice → Bob

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

CA’s public key

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

signature: xsdewsdesd

Name : alice@yahoo.com.sg
Public key: x1s34adf39
Valid until: 1 Sep 2020
Signature of the CA

Role of a Certificate: No Required Directory Server

- A CA (as certificate issuer) basically binds an entity with his/her public key
- 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 or OCSP Responder (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

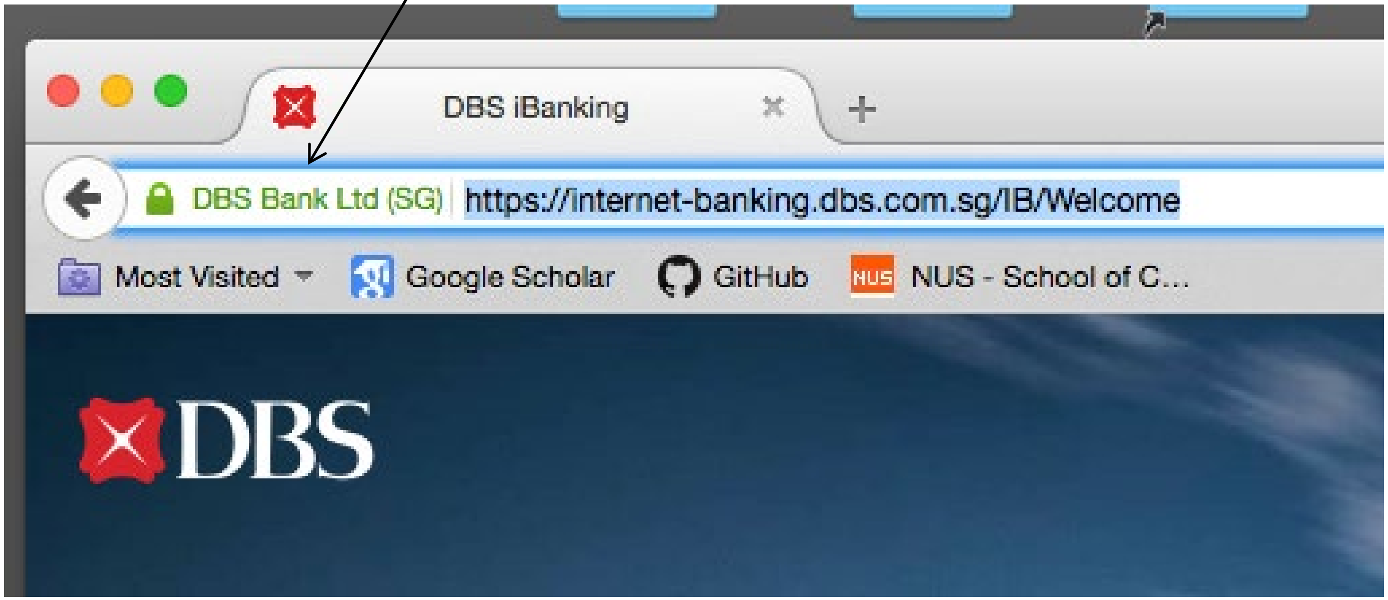
    - Subject Name
    - Subject Public Key Info: Public Key Algorithm, Subject Public Key
    - Issuer Unique Identifier (optional)
    - Subject Unique Identifier (optional)
    - Extensions (optional)
  - Certificate Signature Algorithm
  - Certificate Signature

- ***Distinguished Name*** (DN) to identify an entity (e.g. issuer and subject names):
  - Common attribute types:

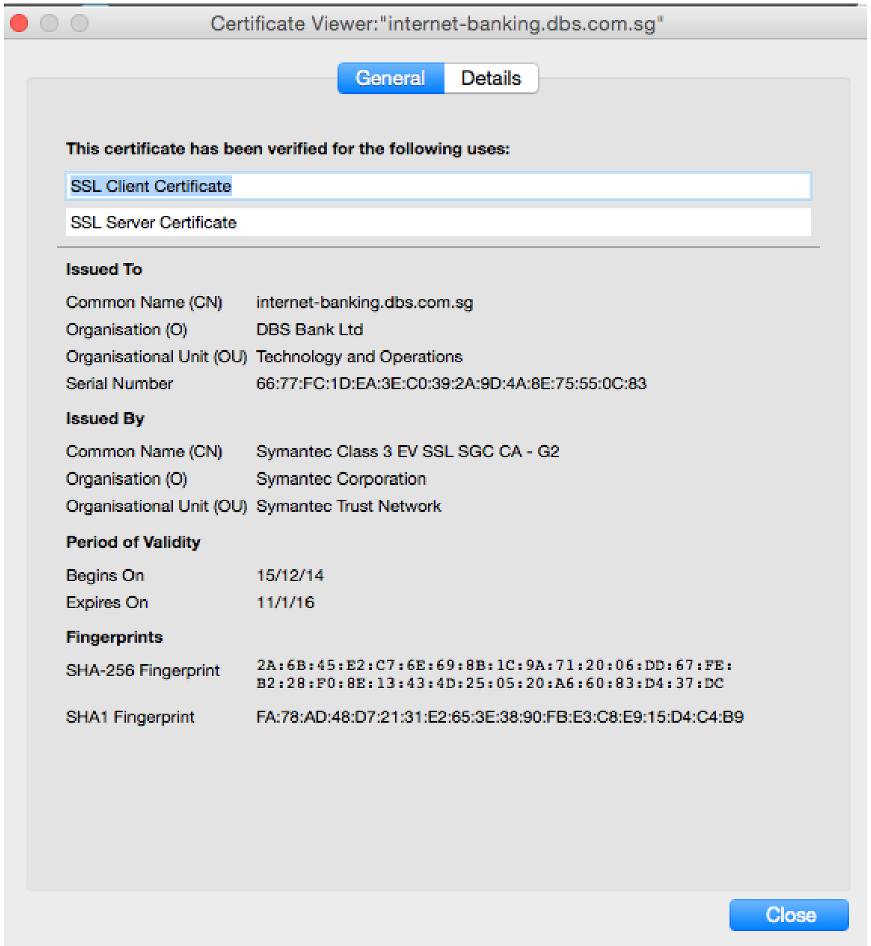
    Country (C), State (S), Locale (L), Organization name (O), Organizational unit name (OU), ***Common name*** (CN)
  - ***Common name*** can be an individual user or any other entity, e.g. a web server

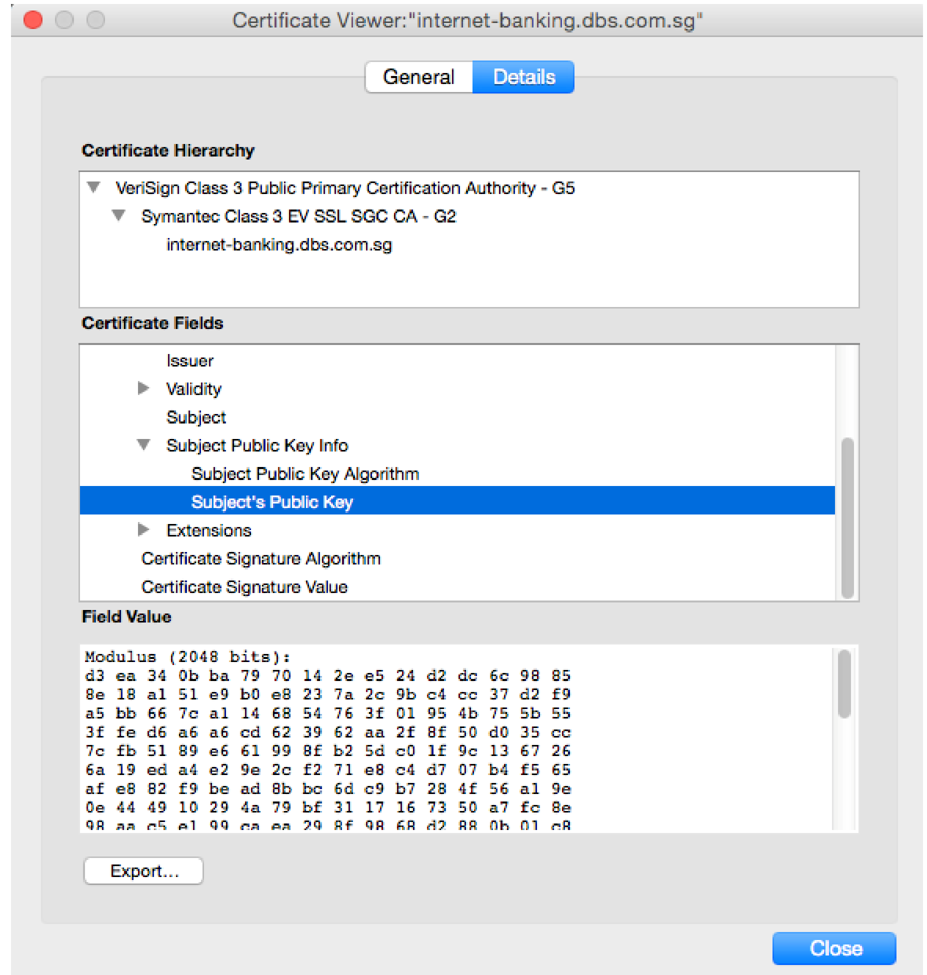
Example of Certificate

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



Example of Certificate





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, 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:
  - 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:

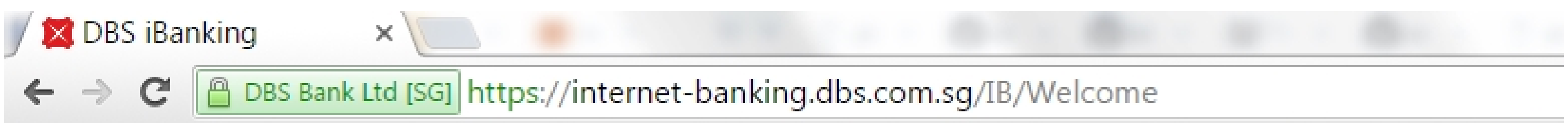
TLS Certificate Level Summaries

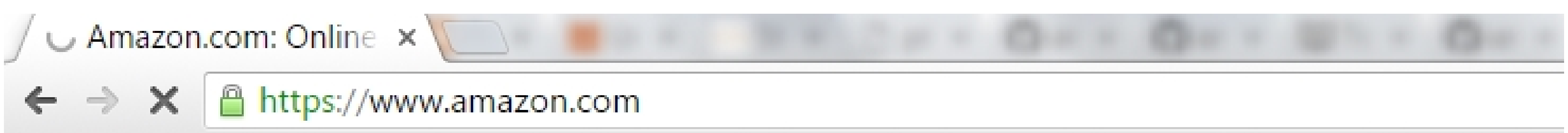
Certificate type	HTTPS encrypted?	Padlock displayed?	Domain validated?	Address validated?	Identity validation	Green address bar?
DV	Yes	Yes	Yes	No	None	No
OV	Yes	Yes	Yes	Yes	Good	No
EV	Yes	Yes	Yes	Yes	Strong	Yes

Source: PCI Security Standards Council, "Best Practices for Securing E-commerce", https://www.pcisecuritystandards.org/pdfs/best_practices_securing_ecommerce.pdf

Browser UI Security Indicators

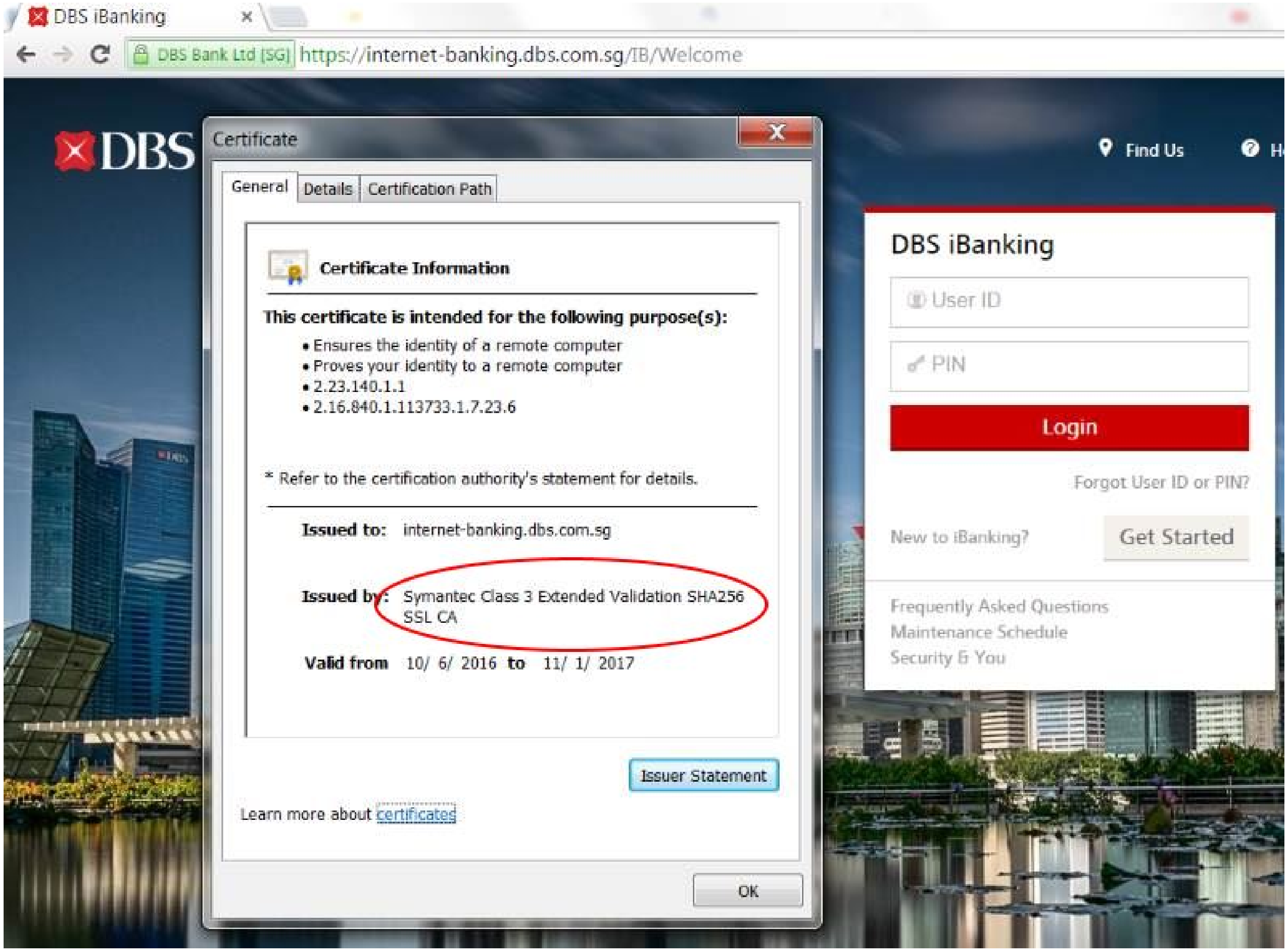
- Browsers offer users different **visual-based security indicators** on different types of certificates
- Examples: Two different domains as shown by Chrome browser below
- Can you guess which types of certificates used?





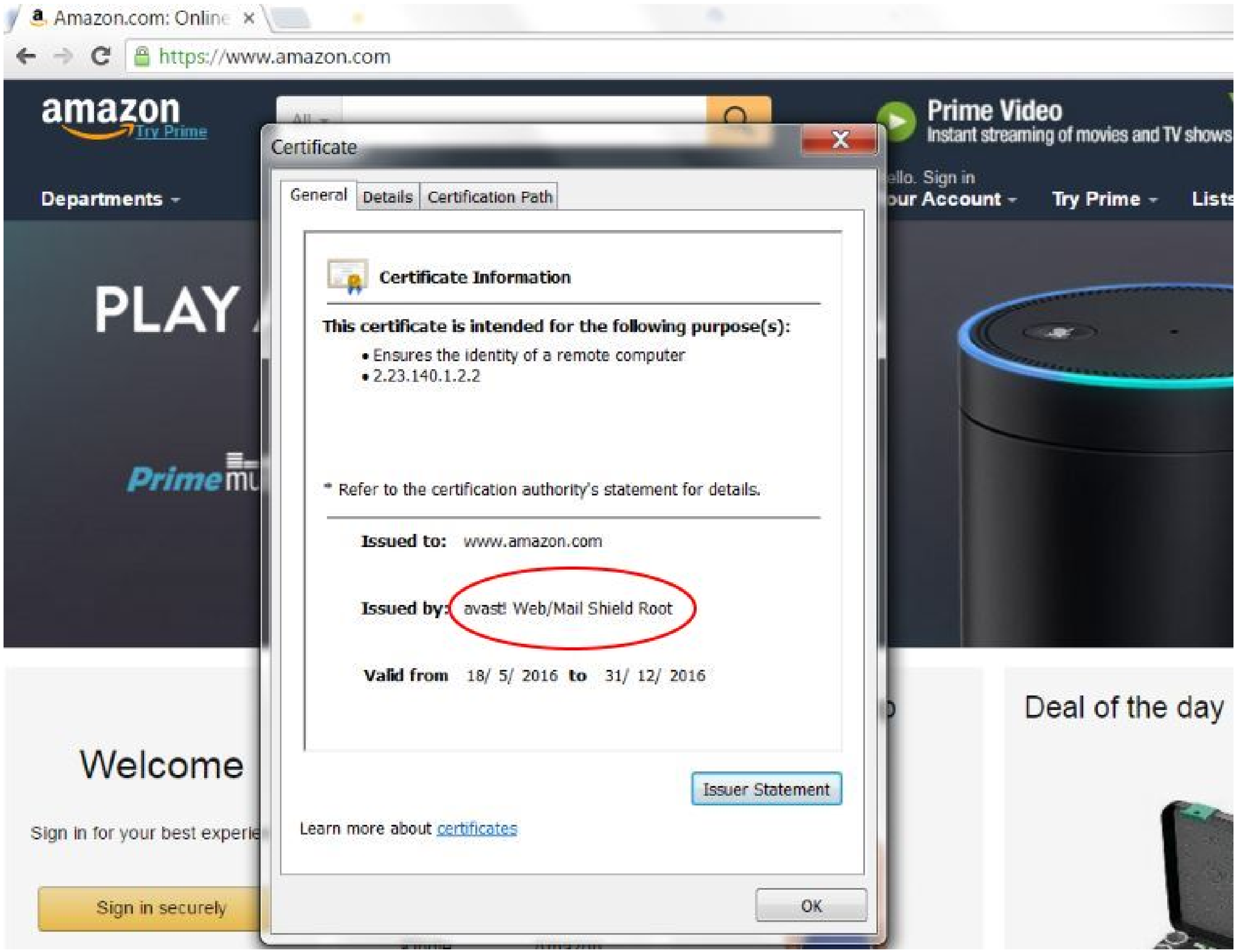
EV SSL Certificate

- DBS Internet banking:



Standard SSL Certificate

- www.amazon.com:



Browser UI Security Indicators

Browser UI Security INDICATORS as of March 2017:

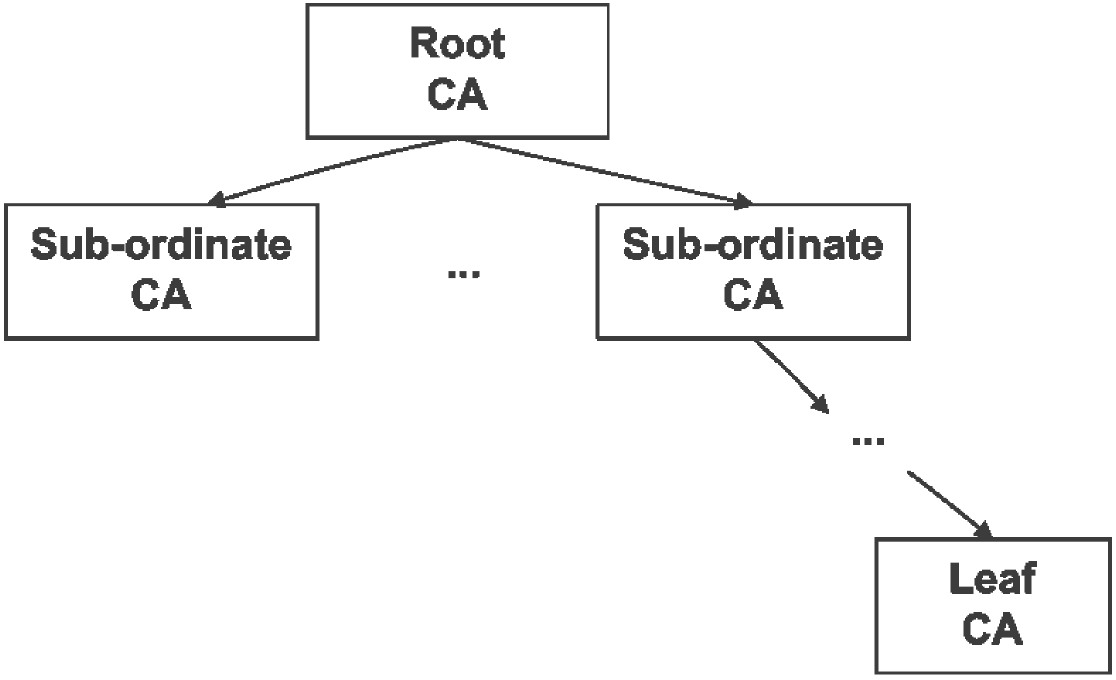
Updated URL UIs indicated by orange outlined box

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.coi	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

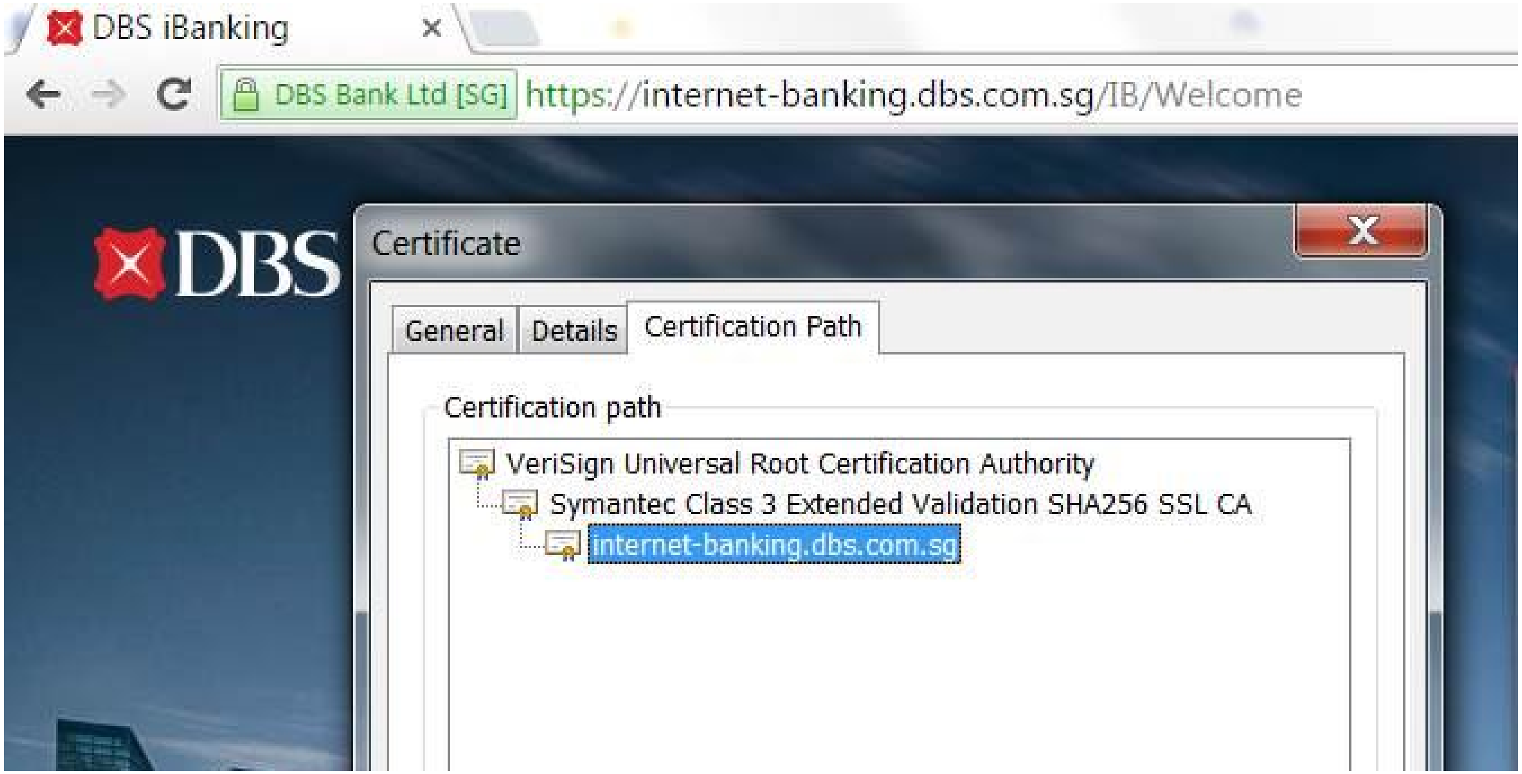
Types of CA

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

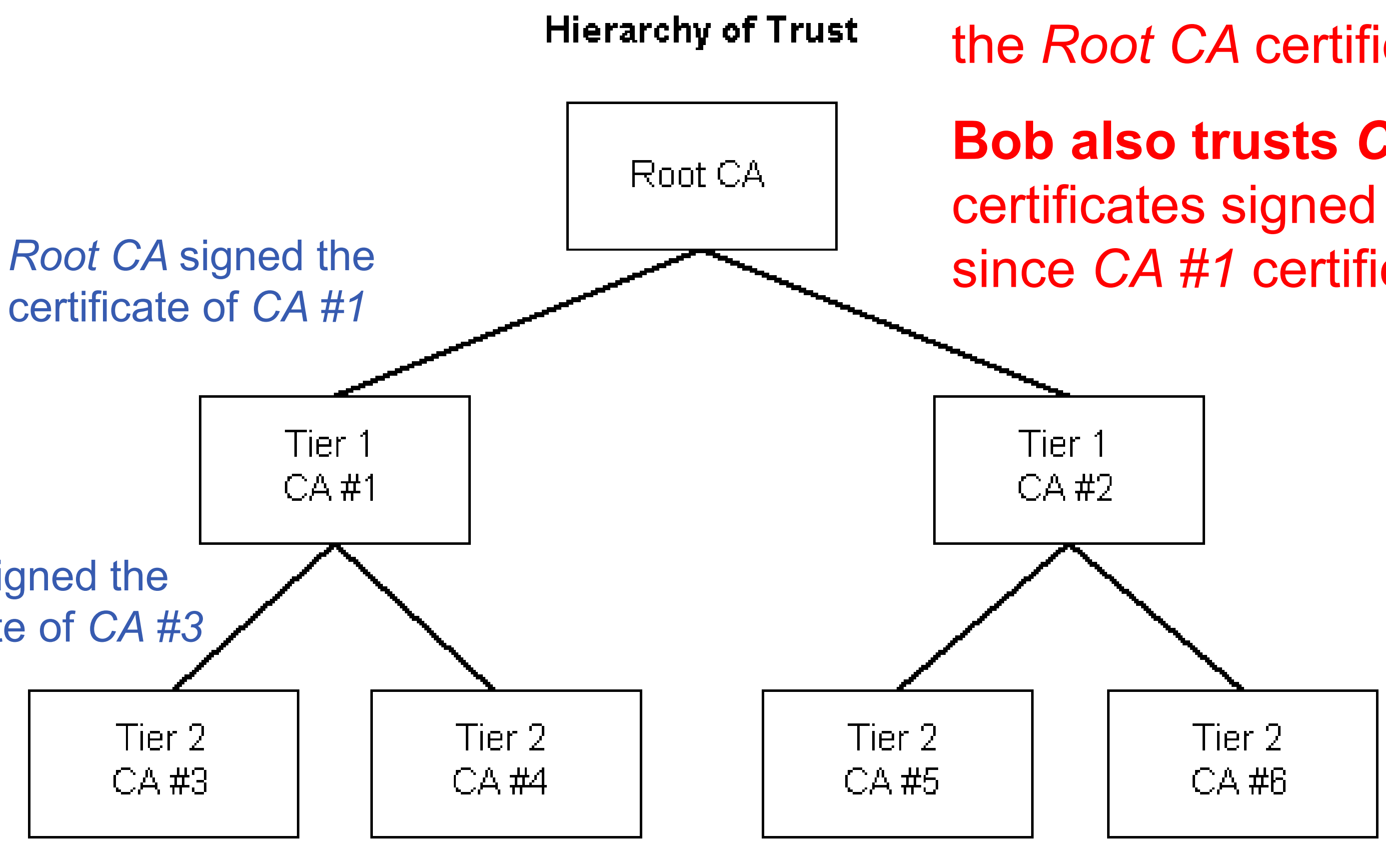


Types of CA: A Real Example

- DBS Internet banking website:



Hierarchy of Trust



Root CA signed the certificate of *CA #1*

CA #1 signed the certificate of *CA #3*

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

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.*

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_1

Name : CA_1
Public key: x3141342
Valid until: 1 Sep 2020
Note: CA_1 can issue certificates
Signature of the Root CA

- In our example, CA_1's certificate clearly indicates that CA_1 is a CA that can issue certificate
- Without that "Note" portion, the certificate owner can't issue other certificates

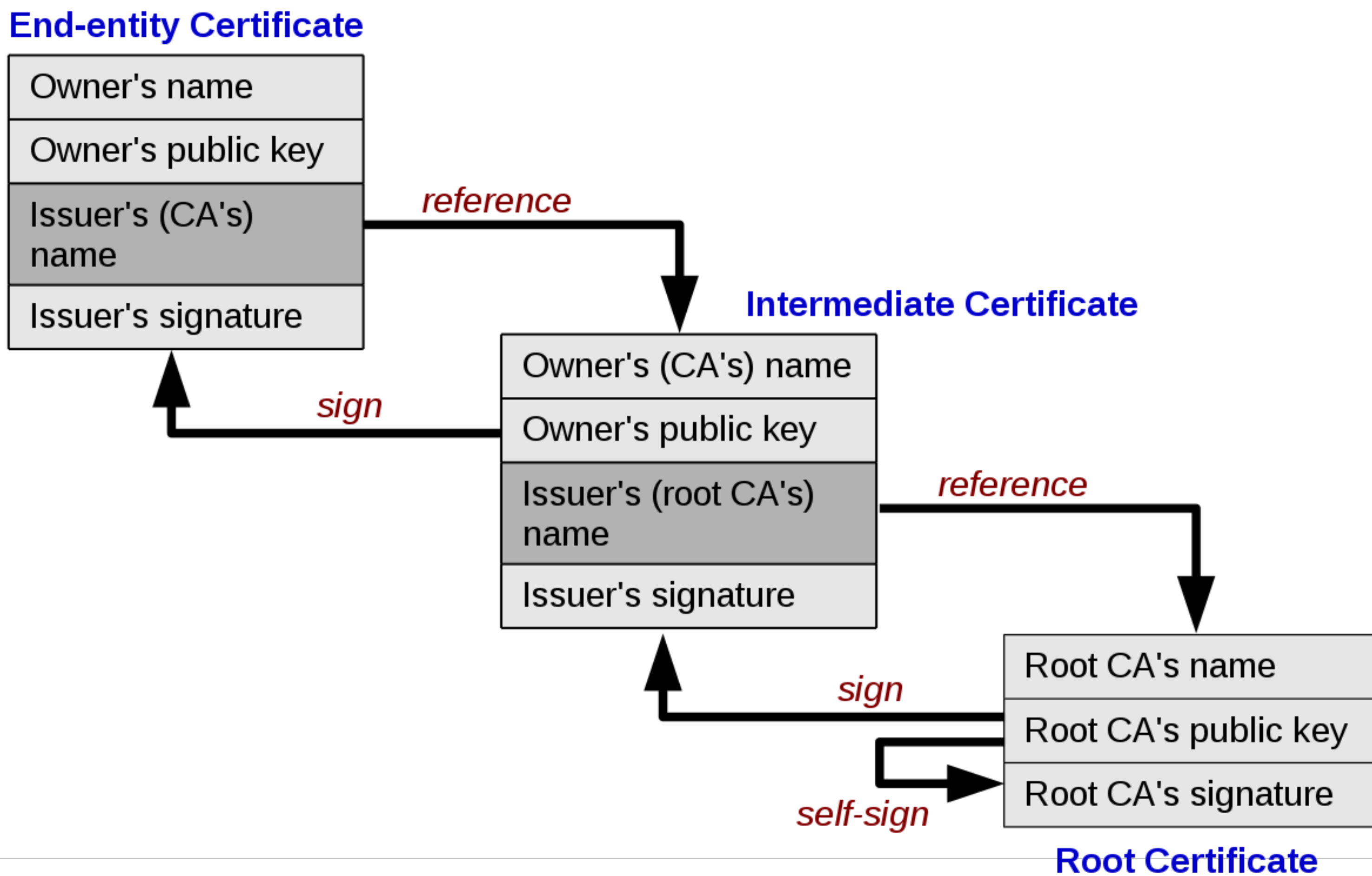
Certification Chain: Definition

***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 secret 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?



Souce: 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.

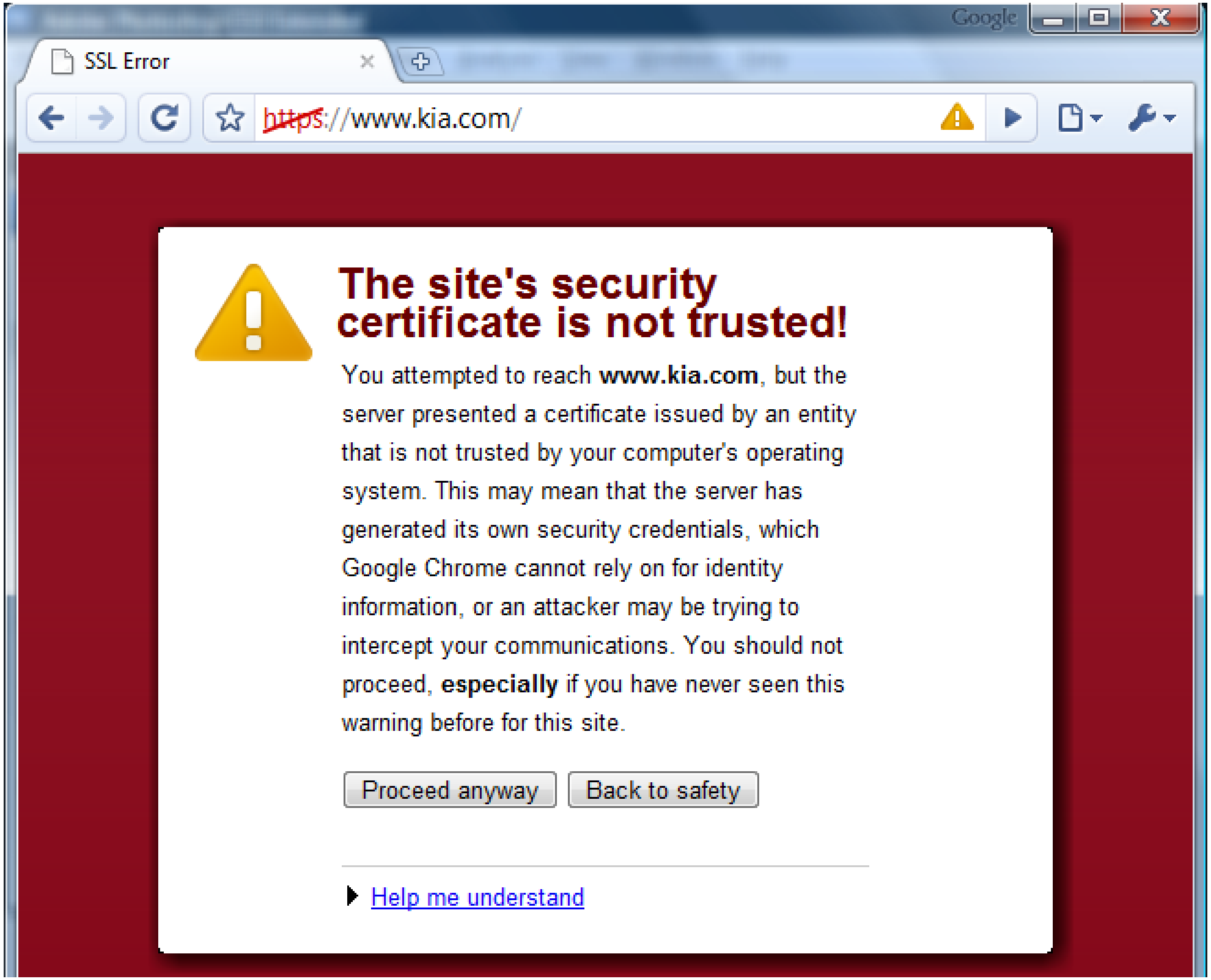
  What is going on here? What are the security implications?

- 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-wow 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?)

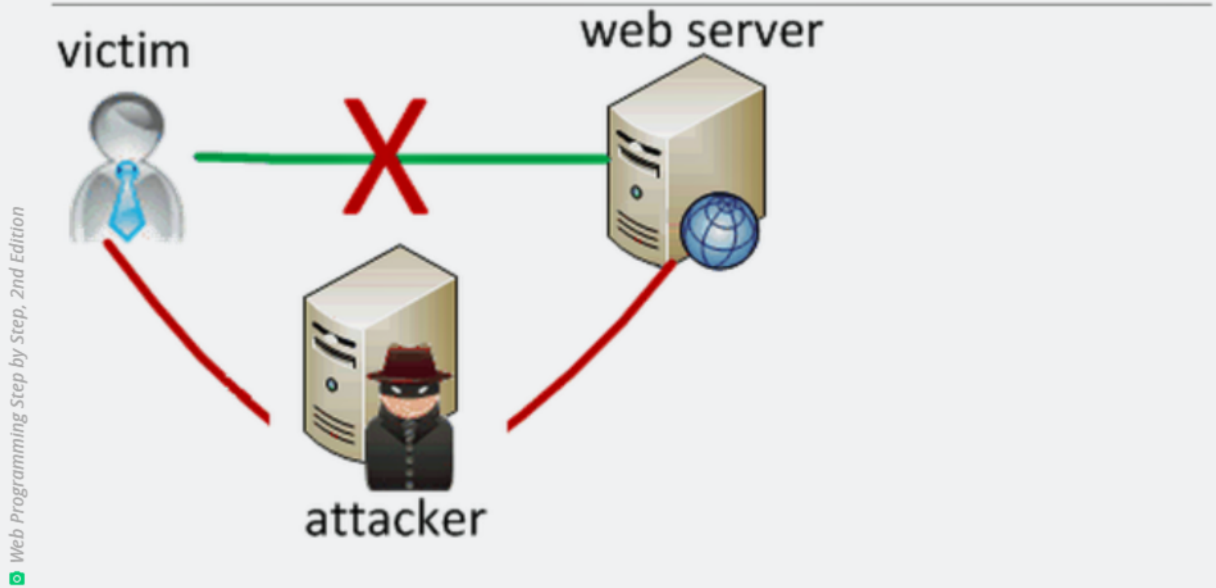
- Lenovo's **SuperFish scandal** *(reserved for class presentation)*

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

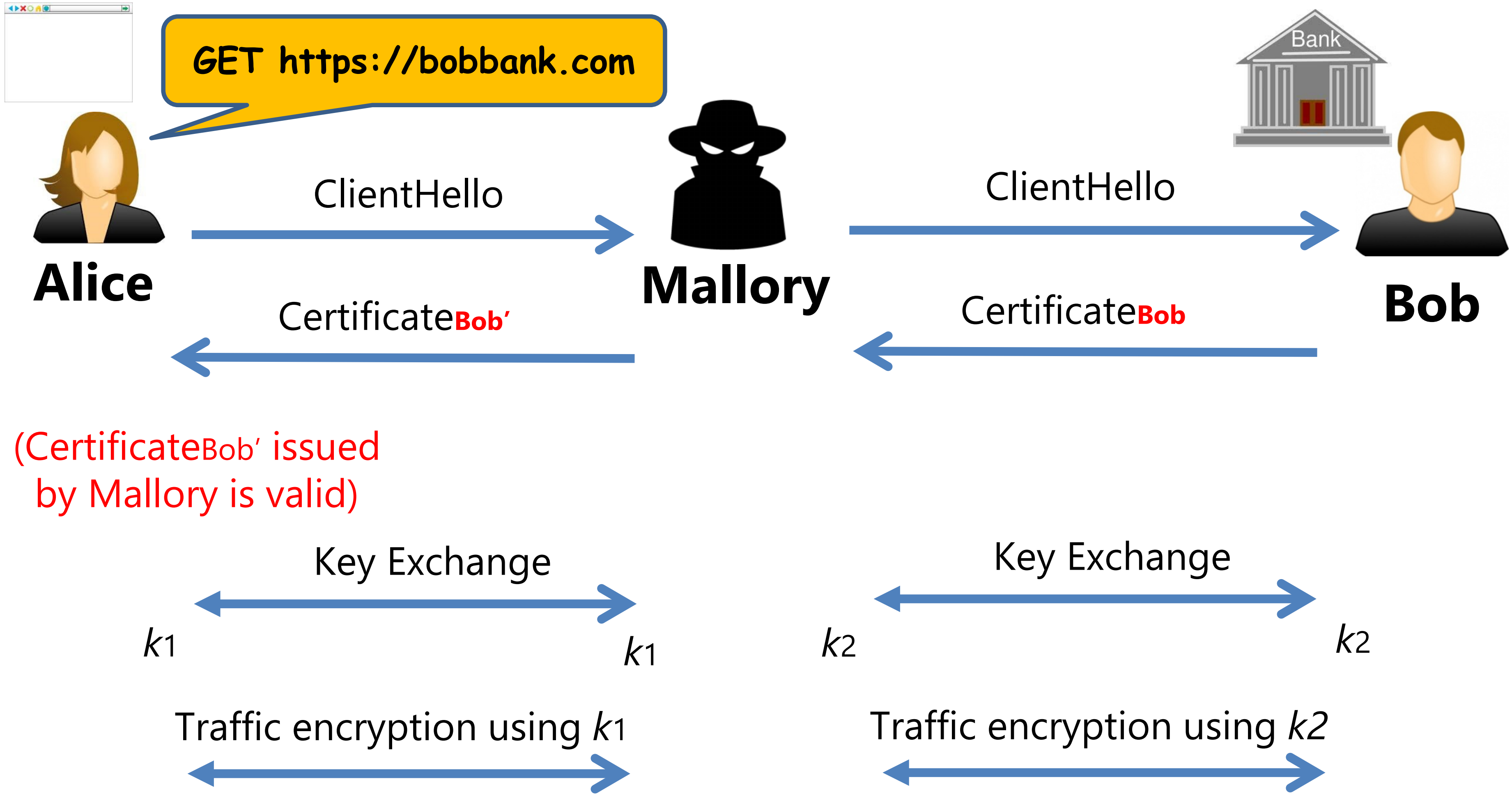


Souce: https://arstechnica.com

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?", 27th 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/

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`.

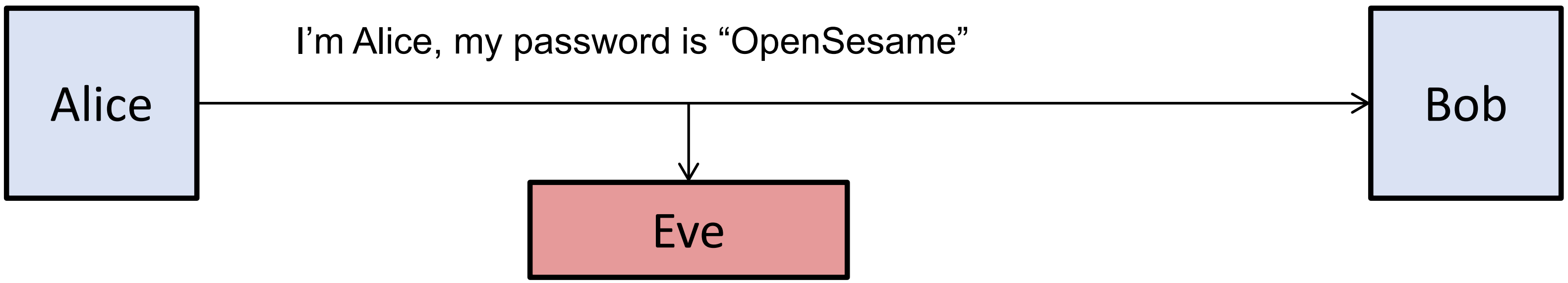
Read http://en.wikipedia.org/wiki/Typosquatting

4.4 Strong Authentication

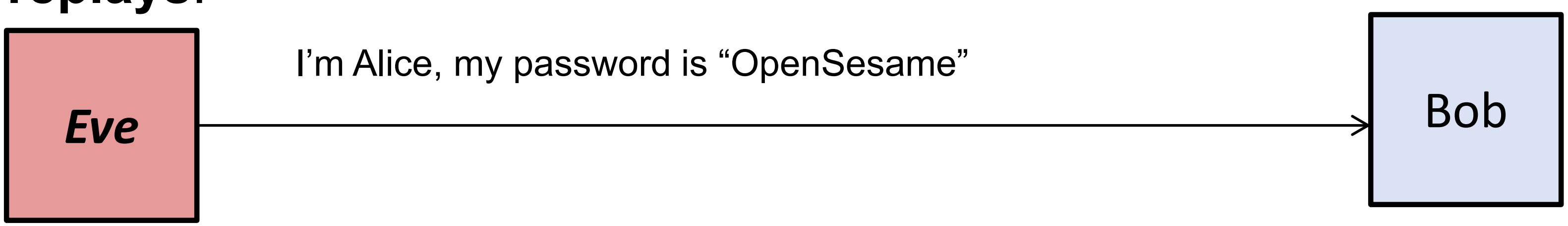
Password and Weak Authentication

We've mentioned that password is a weak authentication system: an eavesdropper can get the password, and **replay** it

Eve eavesdrops:



Eve replays:



The **secret** that is shared between Alice and Bob is the password.
Is it possible to have a mechanism that Alice can "prove" to Bob that she **knows** the secret, **without revealing** the secret?
This seems impossible, but there is an easy way.

Strong Authentication: (SKC-based) Challenge-Response

Suppose Alice and Bob have a shared secret $\boldsymbol{k}$.
Both of them agreed on an encryption scheme, say AES.

(1) Alice sends to Bob a hello message:

“Hi, I’m Alice”

(2) (Challenge) Bob randomly picks a plaintext $\boldsymbol{m}$.
Bob computes:

$$\boldsymbol{y} = \boldsymbol{E_k}(\boldsymbol{m})$$

and sends $\boldsymbol{y}$ to Alice.

(3) (Response) Alice decrypts $\boldsymbol{y}$ to get $\boldsymbol{m}$, and then sends $\boldsymbol{m}$ to Bob.

(4) Bob verifies that the message received is indeed $\boldsymbol{m}$.
If so, accepts; otherwise rejects.

Strong Authentication: Challenge-Response (Analysis)

- Even if Eve can obtain all the communication between Alice and Bob, **Eve still can't get** the secret key *k*
- Eve **can't replay** the response either:
  Because the challenge ***m*** is randomly chosen and likely to be **different** in the next authentication session.
  The ***m*** ensures ***freshness*** of the authentication process.
- The protocol only authenticates Alice:
  Hence it is call ***unilateral authentication.***
  There are also protocols to verify both parties, which are called ***mutual authentication.***

Question*:*
What is "freshness" in the context of authentication protocol?

Unilateral Strong Authentication using PKC

Suppose Alice wants to authenticate Bob.

(1) (Challenge) Alice chooses a **random number *r*** and sends to Bob:

“Bob, here is your challenge”, ***r***

(2) (Response) Bob uses his private key to sign ***r***.
Bob also attaches his certificate:

sign(*r*), Bob’s certificate

(3) Alice verifies Bob’s certificate, extracts Bob’s public key from the certificate, and then verifies that the signature is correct.

An eavesdropper can’t derive Bob’s private key and replay the response because the challenge is likely to be different.

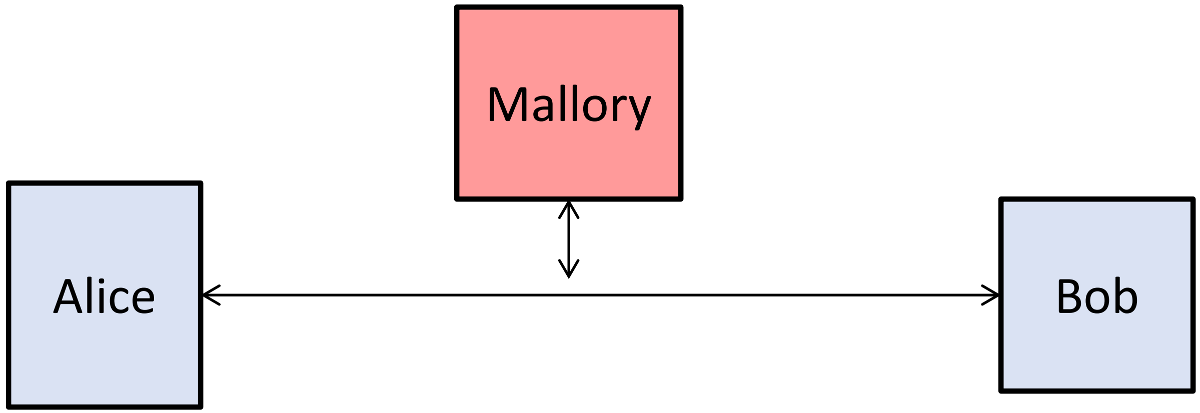
The value ***r*** is also known as the ***cryptographic nonce*** (or simply ***nonce***).

Question*: Which component in the above ensure freshness?*

Remark: the shown protocols have omitted many details.
Designing a secure authentication protocol is **<u>not easy</u>**.

Is Authentication Alone Sufficient?

- You may wonder what come next **after** an authentication
- Consider the typical setting of Alice, Bob and Mallory
- Mallory (who can modify messages) wants to impersonate Alice



- Imagine that Mallory allows Alice and Bob to carry out a strong authentication
- **After** Bob is convinced that he is communicating with Alice, Mallory **interrupts and takes over** the channel.
- Later Mallory pretends to be Alice!

Is Authentication Alone Sufficient?

- Strong authentication, in its basic form, assumes that Mallory is **unable** to interrupt the session
- For applications whereby Mallory **can** interrupt the session, we thus need *something more*!
- The outcome of the authentication process is a new secret ***k*** (a.k.a ***session key***) established by Alice and Bob
- The process of establishing a secret between Alice and Bob is called ***key exchange*** or ***key agreement***

Authenticated Key Exchange

- If the process is incorporated with authentication, then it is called ***authenticated key exchange*** or ***station-to-station protocol*** (if the key-agreement used is Diffie-Hellman):
  - To carry out the protocol, Alice and Bob must have a way to know **each other's public key** (e.g. using PKI)
  - For unilateral authentication, only **one party** needs to have public key
  - After the protocol has completed, **a set of session keys** is thus established: e.g. 1 key for encryption & 1 key for MAC; 1 key for each direction of encryption, etc.

4.5 Putting All Together: Securing a Communication Channel

Secure Channel/Communication Problem

- Consider a communication channel that is subjected to sniffing and spoofing: Does this reminds us of the Internet?
- **Question**: How can we securely communicate over it using cryptographic tools?
- "***A Secure channel***" establishes, between 2 programs, a data channel that has *confidentiality*, *integrity*, *authenticity* against a computationally-bounded "network attacker" (i.e. Mallory)
- Common example:
  Imagine that Alice wants to visit a website Bob.com.
- How to protect the **authenticity** (that Bob.com is authentic), and **confidentiality** & **integrity** of the communication?
- We have discussed some important necessary mechanism:
  authenticated key exchange, PKI, encryption, …
  (How about message-ordering protection?)

Securing Alice's Communication with Bob.com

(Step 1)

Alice and Bob.com carry out a **unilateral authenticated key exchange** using Bob's private/public key

After authentication, both Bob and Alice know two randomly selected ***session keys t*** , ***k***
where: ***t*** is the secret key of a MAC,
    and ***k:*** is the secret key of a symmetric-key encryption,
        e.g. AES

(Details of how t and k can be established are omitted.
See station-to-station protocol for details.)

Securing Alice's Communication with Bob.com

(Step 2)

Subsequent communication between Alice and Bob.com will be protected by ***t***, ***k*** and ***a sequence number (i)***

Suppose m_1, m_2, m_3, … are the sequence of message exchanged, the actual data to be sent for m_{i} will be:

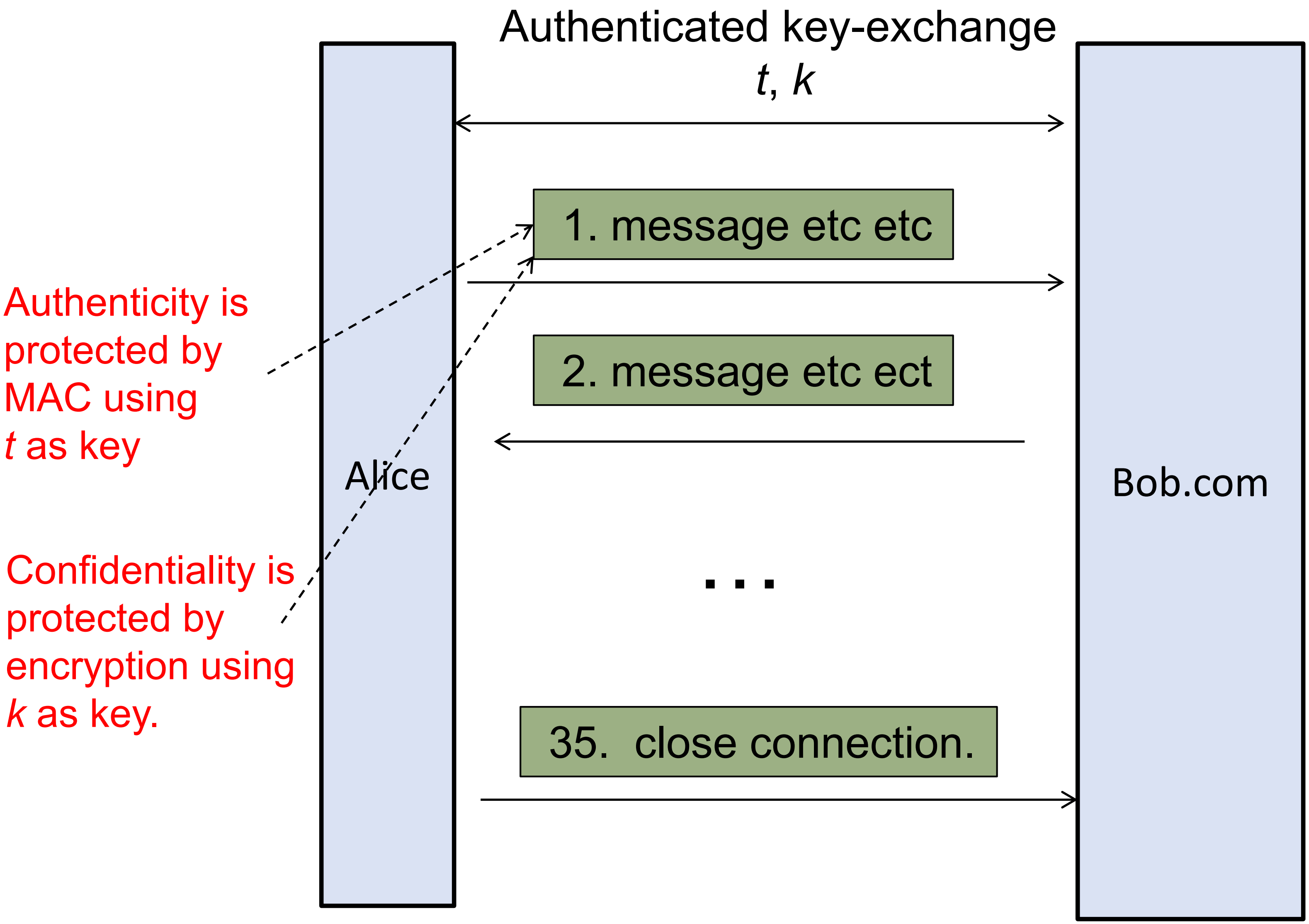
$$E_k\,(\,i\ \ ||\,m_{\mathrm{i}}\,)\ ||\ \ MAC_t\,(\,\mathrm{E}_{\mathrm{k}}\,(\,i\,||\,m_{\mathrm{i}})\,)$$

where: i is the sequence number,

|| refers to concatenation

*(**Optional**) The technique above is known as "encrypt-then-MAC". There are other variants of **authenticated encryption** called "MAC-then-encrypt" and "MAC-and-encrypt".*

Secure Communication between Alice and Bob.com



Question: Why do we need the sequence number?

Secure Communication and PKI

- Recall that in order to carry out an authenticated key-exchange, some mechanism of distributing public keys is required
- Very often, **PKI** is employed to distribute the public key: the authenticated key-exchange is thus likely to involve **certificate**
- Example: suppose **Alice** visits **Bob.com**, and wants to verify that the entity she's communicating is with indeed is **Bob.com**
  - **Alice** then needs to know **Bob.com**'s public key
  - Right in the beginning of the authentication protocol, **Bob.com** directly sends its certificate (which contains his public key) to Alice
- (Note: this is a case of unilateral authentication)

4.6 Putting All Together: HTTPS

HTTPS Protocol

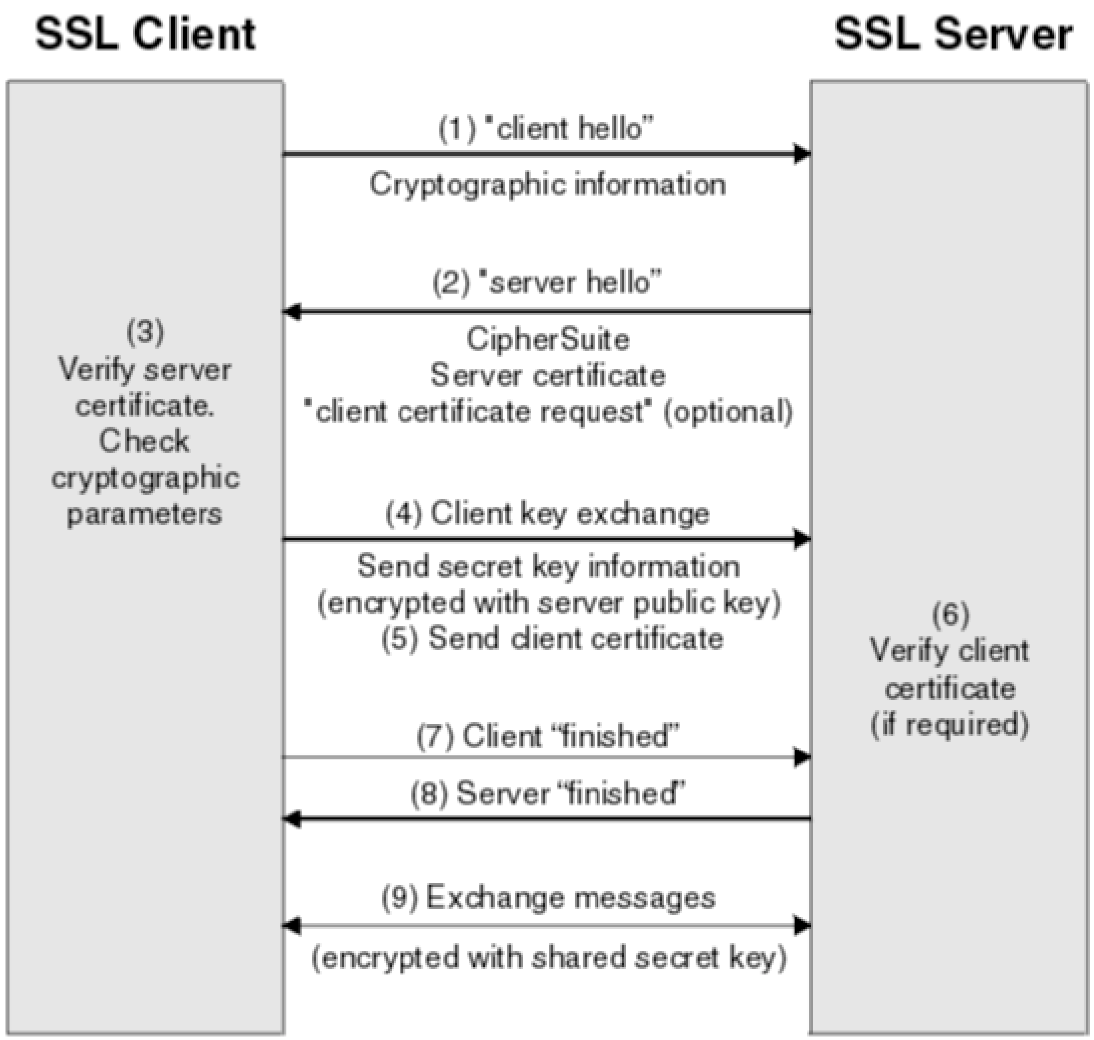
- **HTTPS** (HTTP Secure) is widely used to secure Web traffic
- HTTPS is built on top of SSL/TLS: **HTTPS = HTTP + SSL**
  Hence, HTTPS is also called: HTTP over SSL,
  or HTTP over TLS
- Transport Layer Security (**TLS**) is a protocol to secure communication using cryptographic means:
  TLS 1.2 [2008], TLS 1.3 [Aug 2018]
- **SSL** is predecessor of TLS: Netscape SSL 2.0 [1993]
- TLS/SSL adopts similar framework as in the previous part to secure a communication channel

HTTPS Protocol

- How does HTTPS work **at the high level**?
  - Ciphers negotiation
  - Authenticated Key Exchange (AKE):
    the exchange of session key, which also authenticates the identities of parties involved
  - Symmetric-key based secure communication
  - Re-negotiation (if needed)

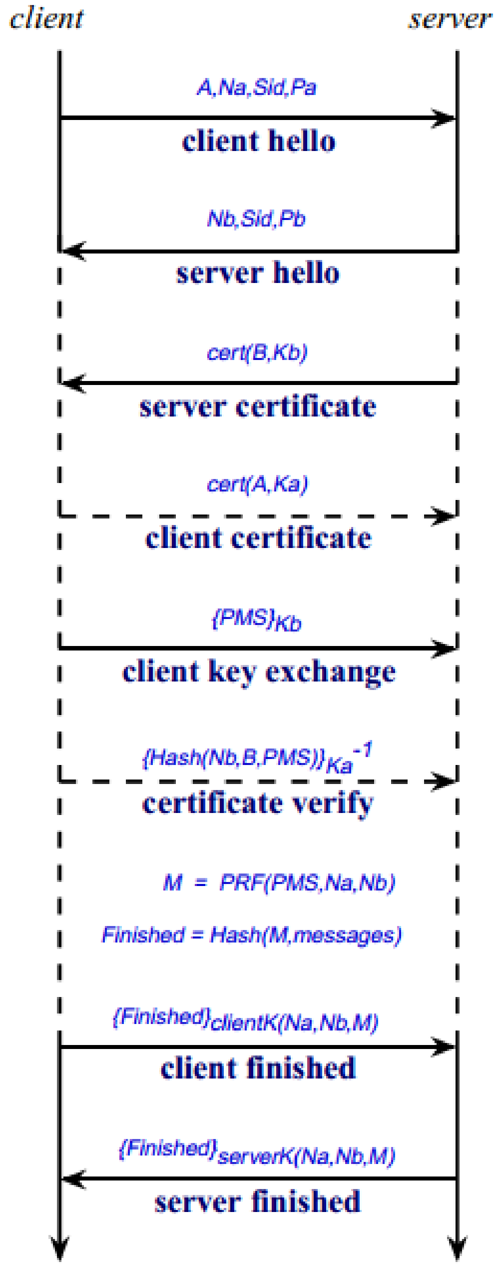
Question: Alice is in a café. She uses the free WiFi to upload her assignment to IVLE (which uses HTTPS). The café owner controls the WiFi router, and thus can inspect every packet going through the network. Can the café owner get Alice's report?

TLS Handshake (Ciphers Negotiation & Authenticated Key Exchange)



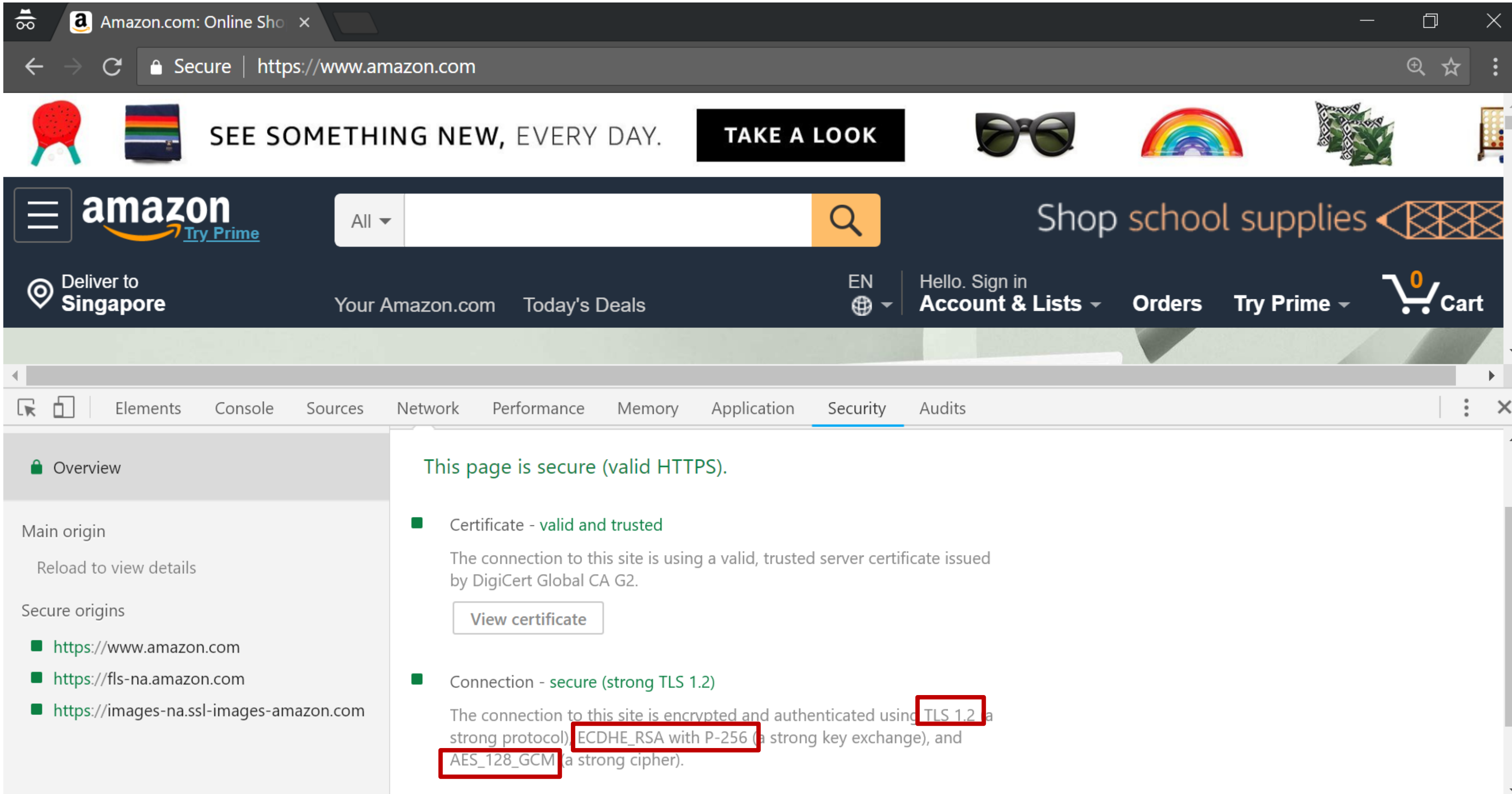
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

TLS Handshake (Ciphers Negotiation & Authenticated Key Exchange)



http://www.cl.cam.ac.uk/~lp15/papers/Auth/tls.pdf

HTTPS Protocol in Action: An Example



See https://tools.ietf.org/html/rfc5246 for the details of TLS Protocol

Side Remark: What is a Protocol?

Protocol Definition and Example

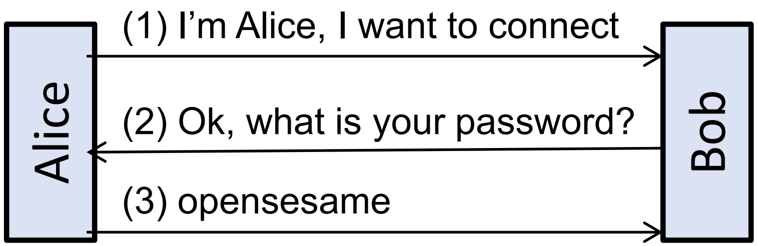
- Slides 60, 62 and 68-69 illustrate a "**protocol**"
- In computer networking, a ***protocol*** is a set of rules for exchanging information between multiple entities
- A protocol is often described as **steps of actions** to be carried by the entities, and the **data to be transmitted**
- For example:

  Alice → Bob: "I'm Alice, I wants to connect"

  Alice ←Bob: "Ok, what is your password?"

  Alice → Bob: "opensesame"

- It can also be visually shown as below:



Assignment 1

- Any questions so far?
- Please ask your TAs, discuss in Luminus Forum
- Available ***consultation sessions on A1*** this week and next week
- Please submit the answers before A1 due date