Topic 4: PKI + Channel Security

Part 1: PKI (seems easy but lots of implementation issues)

- 4.1 Distribution (broadcasting) of public keys
- 4.2 PKI
  - 4.2.1 Certificate
  - 4.2.2 CA
- 4.3 Limitations/attacks on PKI

Part 2: Channel Security

- 4.4 Protocol 1: Authentication
- 4.5 Protocol 2: Key Exchange
- 4.6 Protocol 3: Authenticated Key Exchange
- 4.7 Putting all together: Securing Communication Channel

Part 1: PKI

4.1 Distribution (broadcasting) of public keys

Summary & takeaways

Part 1: PKI

- PKC requires a "secure broadcast channel" to distribute the public keys: PKI.
  (consequence of having the wrong public key).

  - Certificate: a piece of document that binds a "name" to a "public key" & certified by an authority, called CA. A Certificate contains:

    - name, public key, ***expiry date***,

    - meta info: usages, type of crypto, name of CA, etc,

      (meta info often omitted in textbooks but are crucial info, especially "usage")

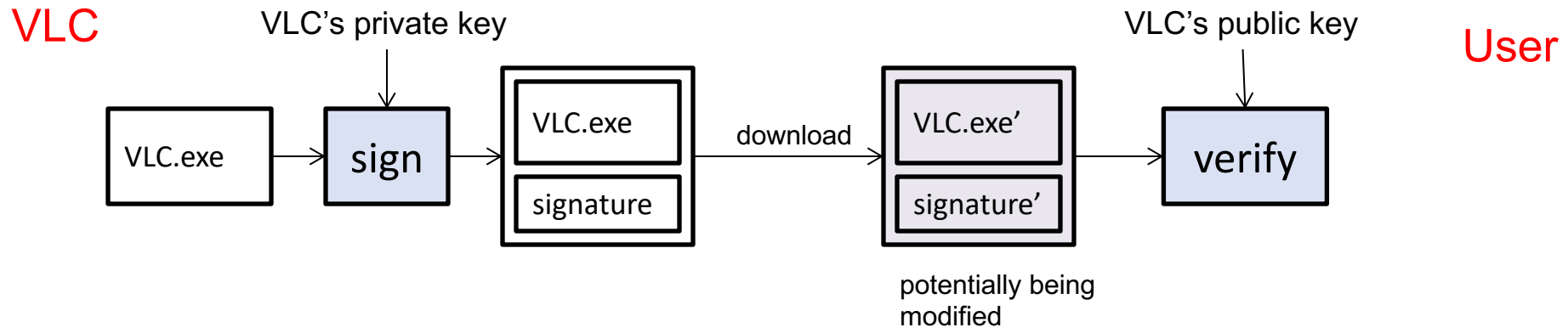
    - CA's signature

  - PKI: infrastructure to broadcast the key. Comprise of
    - Certificate Authority (CA)
    - The processes on issuing, verification, revocation of certificates.
    - The mechanism of chain-of-trust. (A root CA can certificate other CA. Root CA's public keys are pre-installed or "manually" installed.)

  - "Public PKI" usually refers to the one for Internet (often simply called "PKI"). Public PKI's limitations: Too many root CA's. A "private PKI" uses a separate group of private CA, forming another chain-of-trust.

Overview: examples of signature application

- Consider the previous example on VLC.exe.
- To overcome the need of a secure channel to send the unkeyed digest or the symmetric key, we can use signature (public key) in this way:
  1. The developer, "signed" the file VLC.exe using the the developer's private key.
  2. A user who has downloaded the file VLC.exe (together with the signature) from an unverified source (for e.g. from the CNET download site) , can verify the authenticity of the file using VLC's public key.



Distribution of public key:
Compared to the symmetric key setting (mac), here, we don't need to have a secret key between VLC and User.

However, we still need a secure way for User to obtain VLC's public key. If the User obtained the wrong public key from an attacker, the attacker could trick the User to accept the wrong file.

Example of “signed” email using PGP public key

- 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 private key. (see next slide for the actual email sent).

- After Bob has received the email, with Alice’s public key, he can check the authenticity by verifying the signature.

- To carry out the authenticity, Bob needs to know Alice’s public key.

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-----
```

This part is not signed, i.e. not included in computing the signature

Message

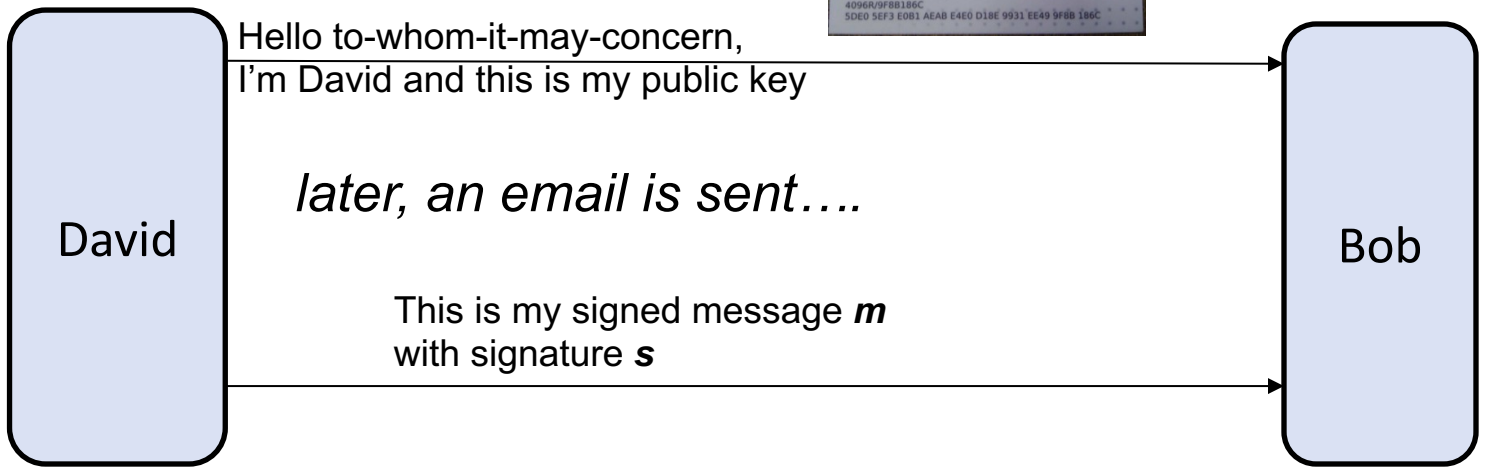
The signature

What if Mallory change the public key?

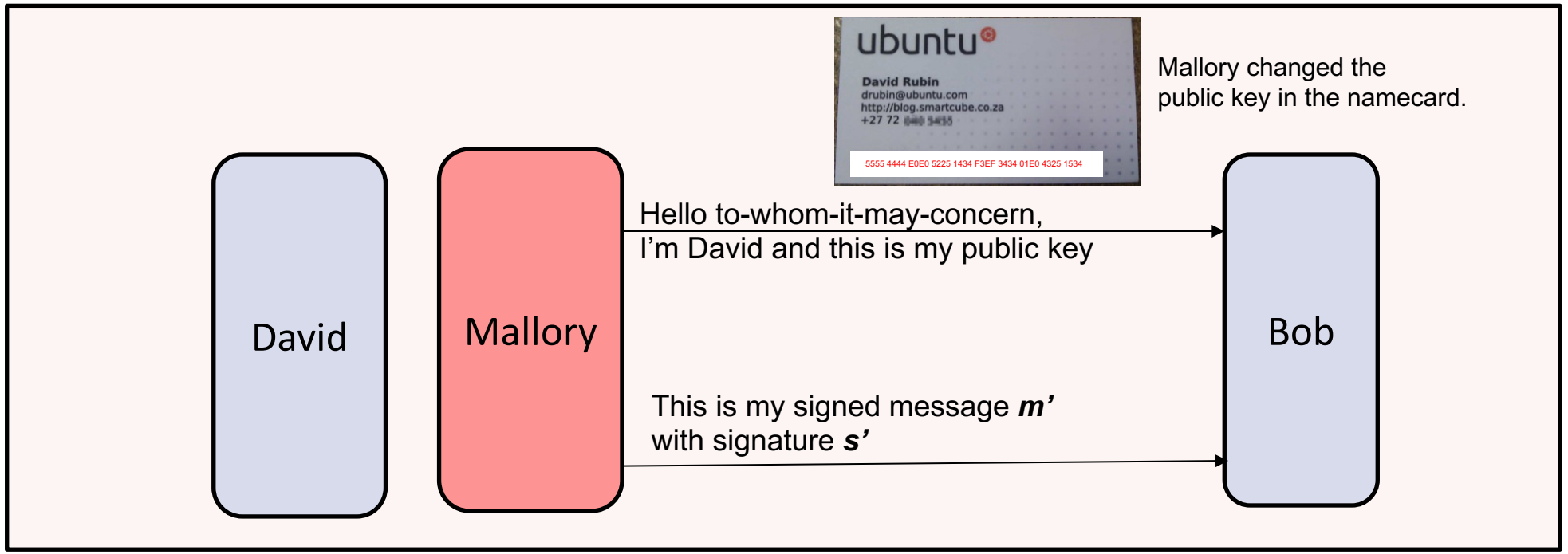
David pinned his namecard on signboard

Image from
https://wordtothewise.com/2014/09/alice-and-bob-and-pgp-keys/

Intended situation



Under attack



Public key distribution vs Symmetric key distribution

Both need secure channel to distribute keys. Nevertheless, it is easier to securely "broadcast" public key compared to "establish" a different symmetric key for every pair.

- ***Public key***: An entity distribute/broadcast its public key to some public billboard only once. (constant)
- ***Symmetric key***: An entity needs to securely establish a different symmetric key with each of the rest. (linear)

- ***Public key***: An entity doesn't need to know the existence of the receiver while distributing its public key. (In the previous slide, David could place his name card in canteen's table, stick in signboard and doesn't need to interact with Bob)
- ***Symmetric key:*** Both entities interact to establish the key.

Key distribution*

The previous example illustrates the need of a mechanism to distribute public keys. With the public key securely distributed, we can use it for encryption *(confidentiality)* and signature verification *(authenticity)*.

Methods:

- Public Announcement on existing mechanism (Social media, name-card), hard coded in verification programs.
- Publish in a publicly available directory.
- Public Key Infrastructure.

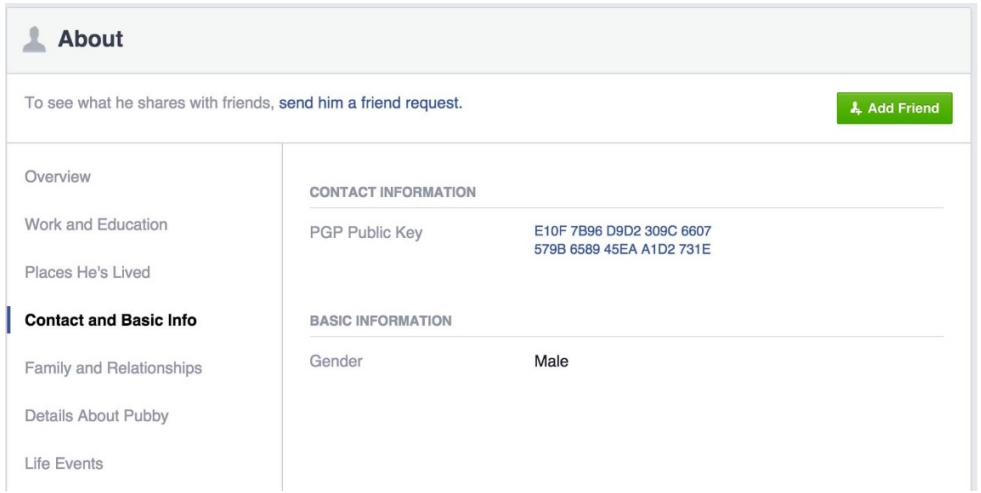
*: while I often uses the term “broadcast” to describe the process, a commonly used term is “distribution”)

Key distribution: Public Announcement/hardcoded

- The owner broadcasts her public key. For example, by sending it to friends via email, publishing in social media, or simply the physical namecard.

Many owners listed their “PGP public key” in blog, personal webpage, etc.

Facebook. Could be outdated. I am unable to find the feature.



from https://www.expressvpn.com/blog/encrypt-facebook-notifications/

https://wordtothewise.com/2014/09/alice-and-bob-and-pgp-keys/

- Limitations:
  - Not standardized.
  - No systematic way to search/verify the public key.

(optional) Get a PGP public key for yourself and send a signed email.

Key distribution: Hard coded

Some apps/software have the public key hardcoded in the programs. For e.g. a game might have the developer's public key hardcoded. The public key is to be used to verify data sent by the developer.

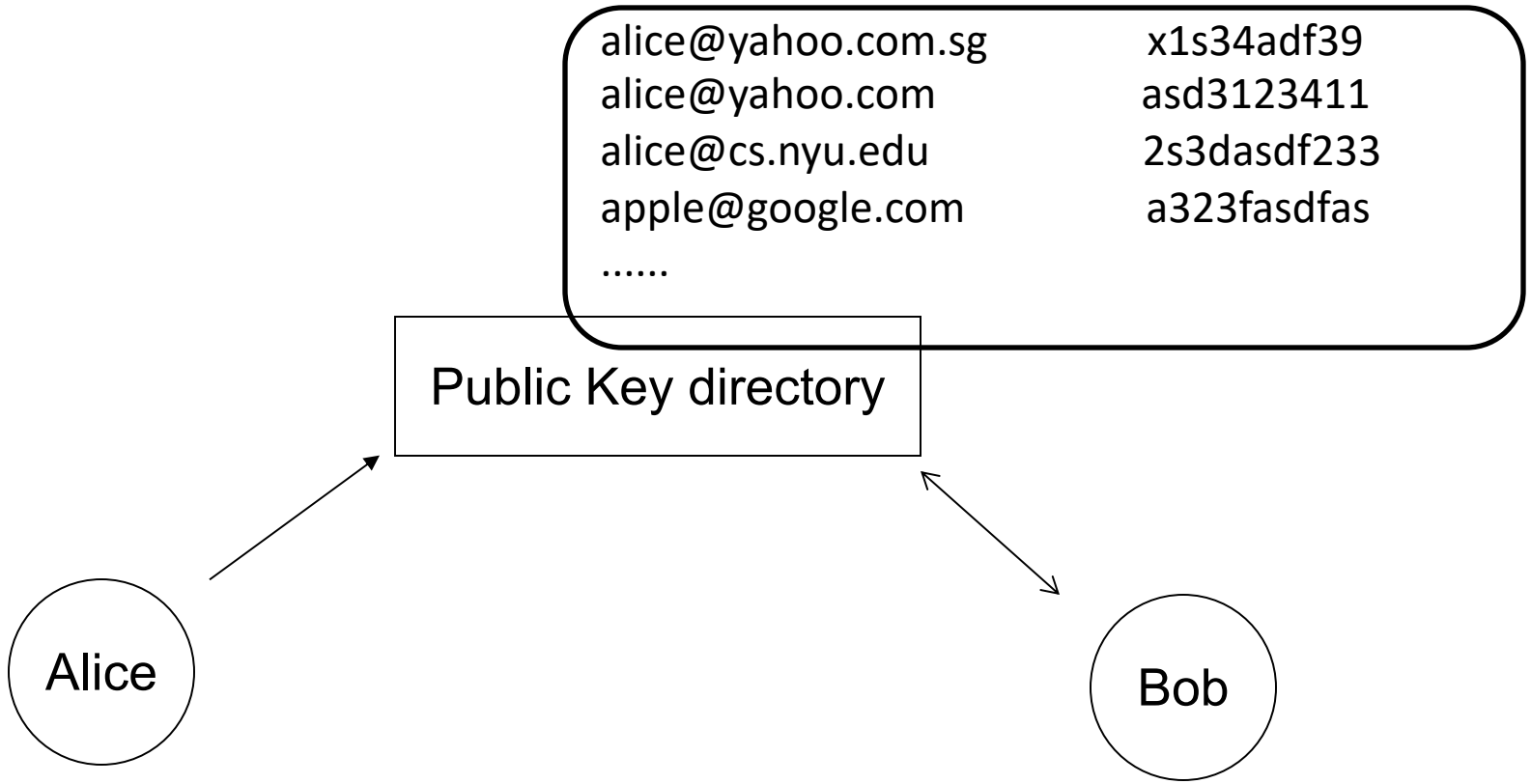
Key distribution: Publicly Available Directory

If Bob wants to find the "***public key***" associated to a "***name***", say

alice@yahoo.com.sg

he can search the public directory by querying the directory server.

For Example: https://pgp.mit.edu/



Potential Issues:

- Anyone can post their public keys in the server. Not clear how to verify the info. Suppose the server receives a request to post a public key, how does the server verify that the information is authentic? For example, if an entity request to post a public key for a particular passport number, how to verify? (for email, the server could verify by sending a confirmation email/OTP to the address. What about other types of name, such as web domain name, passport number?)

- Not everyone trust the server. E.g. Certain entities might not trust mit.edu to host the server. Some server might be more diligent in verifying information of the posting request, while some might be very lax and simply accept any posting request.

- The server could be overwhelmed and can't handle the load.

- The framework doesn't scale. For instance, different platforms may setup different directories, using different formats and request protocols. If so, a requester need to query different platforms using different protocols, and that would be very confusing and difficult to implement. For scalability, it would be good to have a "standard" and a way to delegate trust.

In view of the above, the community agreed on a standard and established Public Key Infrastructure (PKI).

PKI

- PKI is a standardized system that distribute public keys. A main objective is to be **deployable** on a large scale.

- Centered around two components:
  - ***Certificate***
  - ***Chain of trust of Certificate Authority (CA)***

- "Public PKI" refer to *the* PKI we adopted in Internet for domain name, emails address etc. In short, we simply call it "PKI".
- "Private PKI" are systems specific applications, and it has its own set of "CA". For e.g. it is possible for a large organization to set up a PKI for its own usage only.

4.2 Public Key Infrastructure

4.2.1. Certificate & CA

4.2.2. CA’s chain-of-trust

4.2.3. Revocation

4.2.1 Certificate & CA

Certificate Authority (CA)

- "***Certificates***" are useful in distributing public keys. An CA issues certificates.

- Essentially, CA is a trusted authority that manages a directory of public keys. An entity can request adding its public key to the public directory. Anyone can send queries to search the directory. (Certificates facilitate checking/querying to be done in an offline manner.)

- The CA also has its own public-private key. We first assume that at least some CA's public keys have been securely distributed via other means. (so, we still need a secure channel to distribute the CA's public key).

- Most OSes and browsers have a few pre-loaded CAs' public keys: they are known as the "***root***" CAs. Not all CAs' public keys are preloaded. Other CA's public keys can be added through the chain-of-trust to be described later.

- When Bob's machine has a CA's public key, we sometime say that the Bob has "installed" the CA's public key/root-certificate.
- While a browser and system providers can pre-load any root CA's public key of their choices, there are well-accepted stringent requirements. For e.g, it must pass **WebTrust audit** (http://www.webtrust.org/homepage-documents/item76002.pdf)

Certificate

Consider the situation where Bob wants to obtain Alice's public key. Bob can directly send a query to the CA.

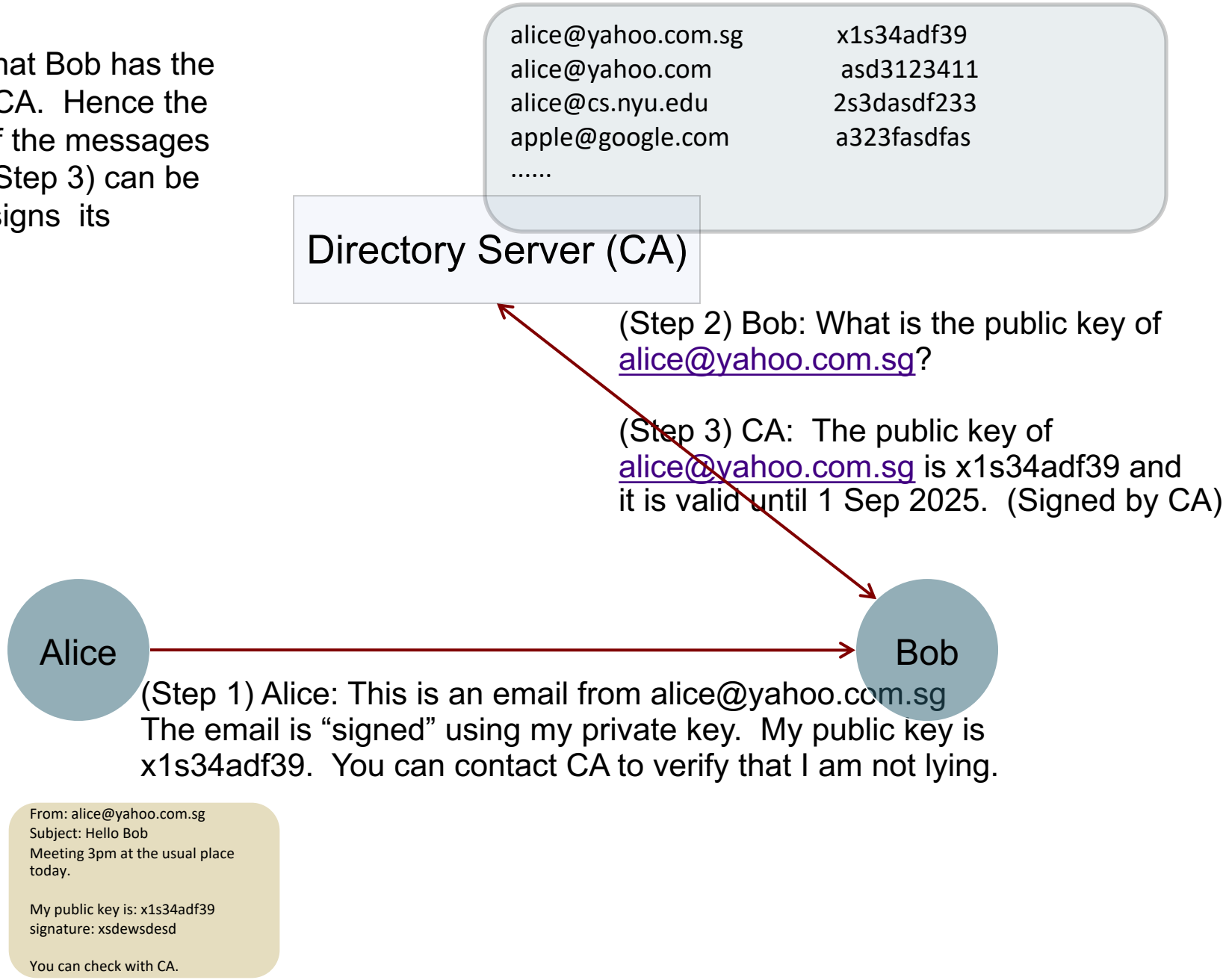
There are two limitations of querying the CA as-and-when needed:

- The CA became a bottleneck
- Bob needs to have online access to the CA at the point of verification

Certificate is a "smart" workaround to get the public key in an offline manner.

Bob getting Alice's public key: Without Certificate

We assume that Bob has the public key of CA. Hence the authenticity of the messages from CA (i.e. Step 3) can be verified: CA signs its message.



With Certificate

Note that in the previous slide, data in step 3 is always the same. So, a “lazy” CA would “sign” the message beforehand and pass it to Alice. Alice can directly send the signed message to bob. That signed message is called 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 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.

(Step 1) Alice: This is an email from alice@yahoo.com.sg
The email is “signed” using my private key.
My public key is listed in 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 2019
- signature of the CA

Certificate

- A ***certificate*** is a digital document that contains ***at least*** the following 4 main items
  1) The name(s), for e.g. *alice@yahoo.com* or *bbc.com* or **.bbc.com* *(note that *.bbc.com is a set of names)*
  2) The public key of the owner.
  3) The time window that this certificate is valid.
  4) The signature of the CA.

- Other important information.
  - Usage of certification: (1) the type of "name", whether it is email address or domain name. (2) whether the "name" can take the role of a CA (chain-of-trust).
  - Digest (Fingerprint). For verification without using CA's public key (e.g. the VLC.exe setting in earlier slides)
  - Type of algorithms used to verify the signature (e.g. ECC or RSA, key length, SHA3)
  - etc

- Now, from the certificate, Bob can obtain Alice's public key, and verifies its authenticity even without connection to the CA.

- We assume that Bob trusts the CA. Hence, information signed/certified by the CA is also trusted. Since CA signed Alice's certificate, the public key extracted from the certificate is indeed Alice's.

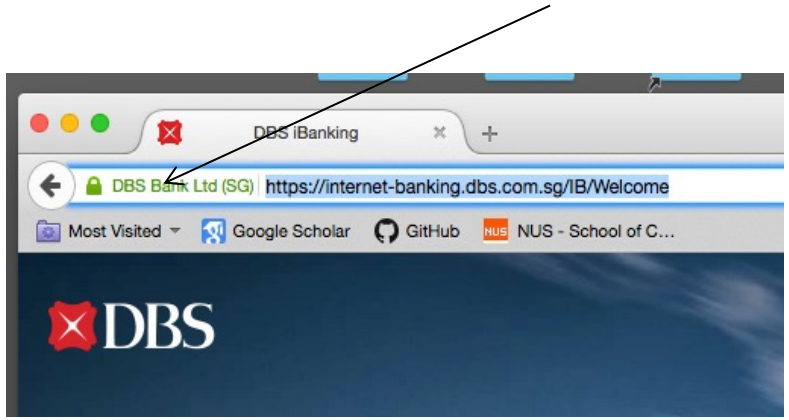
Bob still must start by having the CA public key. The CA's public key could be pre-loaded into Bob's device.

Example of Certificate

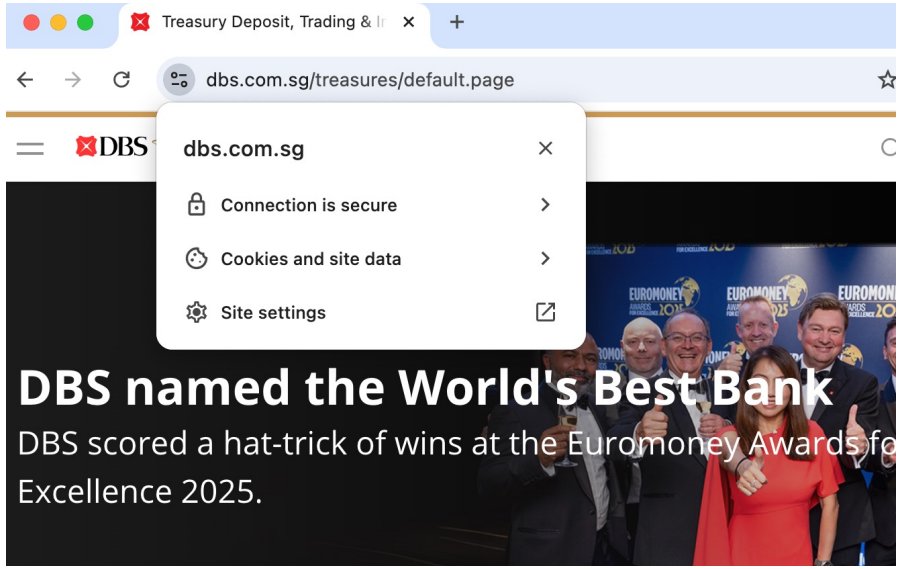
browser design 2,3 years ago

Visit https://internet-banking.dbs.com.sg/IB/Welcome

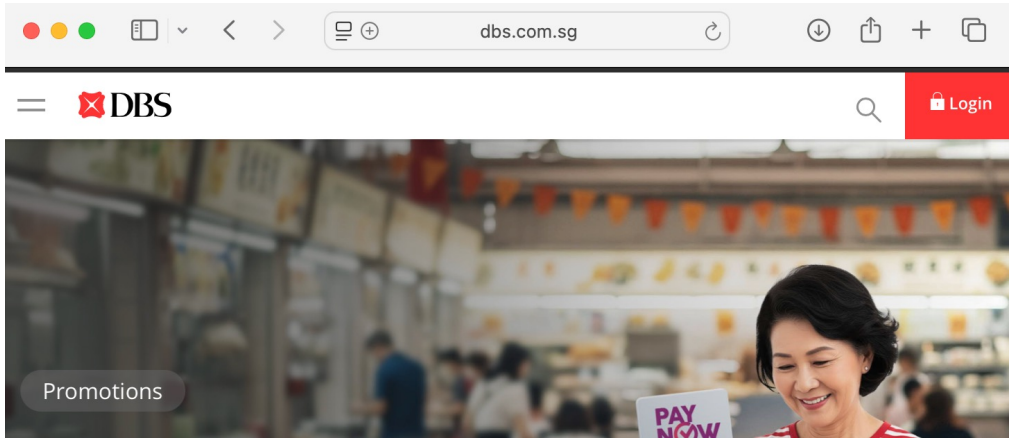
See the certificate's detail. (click the address bar)

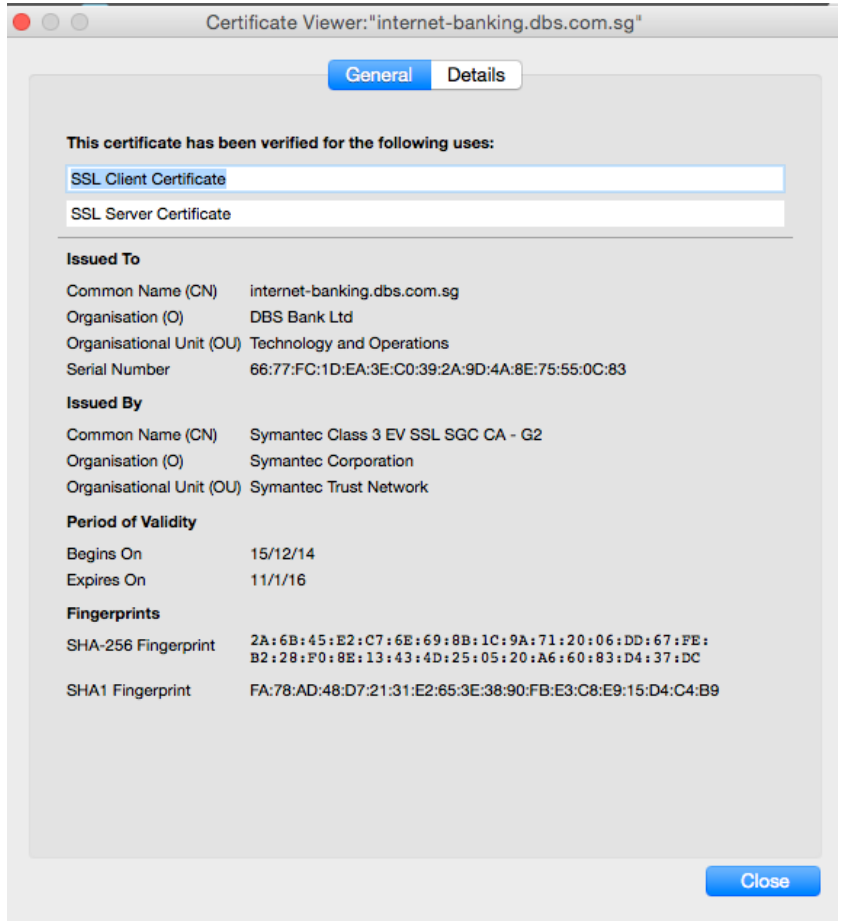


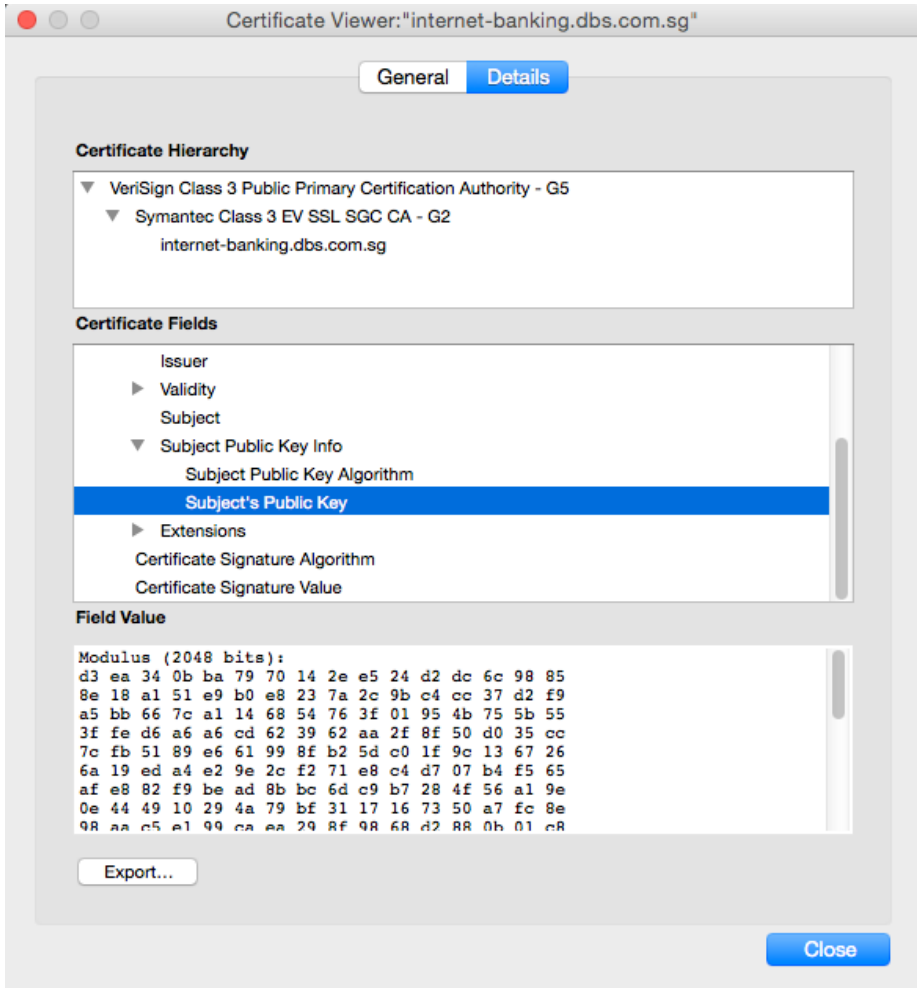
recent design (chrome)



recent design (Safari) note: to view the certificate, goto the menu to search for it.




Certificate Viewer:"internet-banking.dbs.com.sg"
General
Details
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 Number 66:77: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 On 15/12/14
Expires On 11/1/16
Fingerprints
SHA-256 Fingerprint 2A: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 Fingerprint FA: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"
General
Details
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
Export...
Close

Standard: X509

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

What the standard standardize?

Eg:

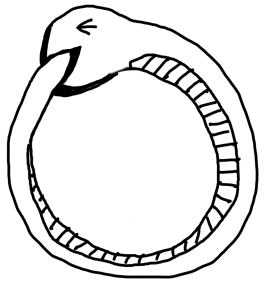
- **Structure** of an X.509 v3 digital certificate:
  - Certificate:
    - Version Number
    - Serial Number
    - Signature Algorithm ID
    - Issuer Name
    - Subject Name
    - Subject Public Key Info: Public Key Algorithm, Subject Public Key
    - Validity period: Not Before, Not After
    - Issuer Unique Identifier (optional)
    - Subject Unique Identifier (optional)
    - Extensions (optional)
  - Certificate Signature Algorithm
  - Certificate Signature

How Do I Get a Certificate?

- Get a certificate from a **CA.** ~$10 - $50 per year
- "*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
  - Number of issued certs: **1M** (March 2016), **450M** (Sept 2024)

https://letsencrypt.org/

Self-signed certificate



- See tutorial.
- A self-signed certificate is signed by its owner (i.e the “name” in the certificate). It is to be verified using the “public key” listed in the certificate. Self-fulfilling!
- Self-signed certificate, although sounds contradicting and useless, is convenient in manual installation of public key. Suppose a user wants to include a binding of name & public key of an entity, say D, but D doesn’t have a certificate signed by CA. To handle this, D can “self-sign” a certificate and pass to the user. Next, the user manually accepts the certificate.
- In other words, by accepting the self-signed certificate, the user instructs the machine to accept the binding of the D’s “name” and the “public key”, and the user takes the responsibility in ensuring that info in the certificate is correct.
- A self-signed certificate of a CA is also called “root-certificate”.

Summary

- A certificate is simply a document signed by a CA.

  (1) An identity.
  (2) The associated public key
  (3) The time window that this certificate is valid.

  (4) The signature of the CA.

- The document binds (2) to (1). It is certified by (4).

- We assume that Bob already has CA's public key "pre-installed" in his machine.

4.2.2 Certificate Authority & Trust relationship

Responsibility of 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 the identity

  www.nus.edu.sg

the CA should check that the applicant indeed own the above domain name. This may involve manual checking and thus it could be costly.

Getting a certificate signed by a reputable CA is not free.

Certificate Chain-of-trust

- There are many CA's.
- Most OS, browsers already have a few CA's public key pre-loaded. These are the "***root CA***".
- Suppose Alice's certificate is ***issued*** (i.e. signed) by CA#1, but Bob doesn't have the public key of CA#1, why should he do?
- In the first place, Alice, anticipating the Bob might not have the public key of CA#1, can send her email, her certificate, and CA#1 certificate (issued by the root CA) to Bob. Bob can now

  - verify CA#1's certificate using root CA's public key.

  - verify Alice's certificate using CA#1's public key.

  - verify Alice's email using Alice's public key.

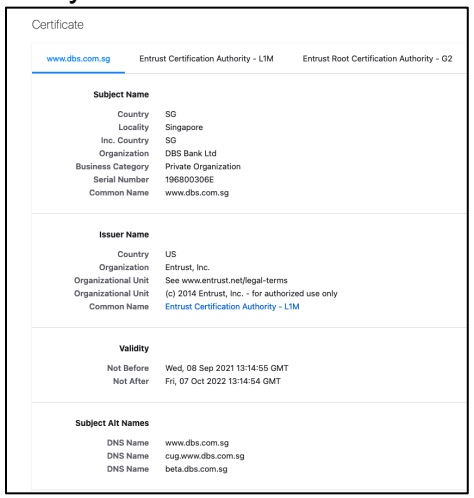
In our example, the certificate issued to CA#1 clearly indicate that CA#1 is a certificate authority and can issue certificate. Without that "note", the owner can't issue other certificates.

- If Alice doesn’t attach CA#1’s certificate, then Bob can obtain it from other sources.

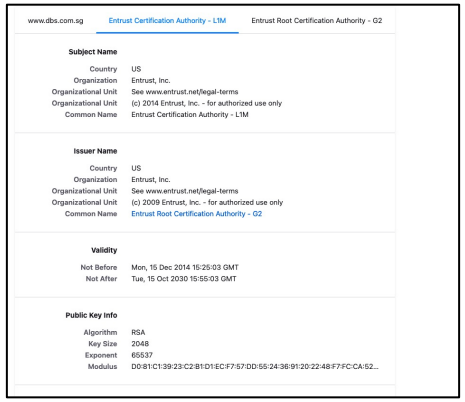
Example

Check the certificate of www.dbs.com.sg. Most browsers would show the chain. (below obtained from Firefox during 2022)

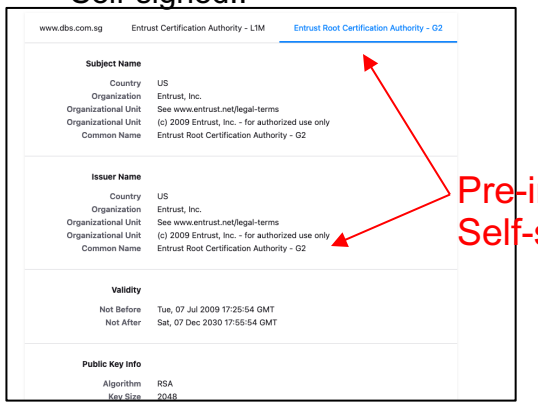
DBS's certificate, signed by ***Entrust CA-L1M***



Entrust CA-L1M's certificate, signed by ***Entrust Root CA-G2***

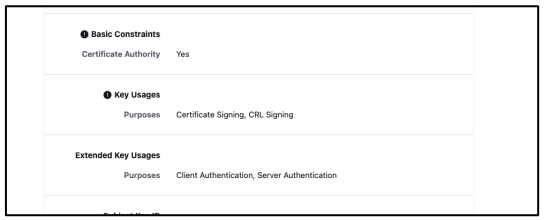


Entrust Root CA-G2 certificate. Self-signed!!



Pre-installed, Self-signed!

Look at the field "Basic Constraint" and "key usage". It indicates whether the "name" can take the role of CA.



Screenshot/info obtained during 2022.

Question

- 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? (The "issuer" here probably refers to the CA. So, the browser doesn't have the CA's public key.)

- While installing a new package using 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? (error message indicate failure to verify the certificate but does not give sufficient info on which part fails. It could be certificate expired, no certificate, wrong signature, etc)

4.2.3 Revocation

Certificate Revocation

- Non-expired certificates to be revoked due to different reasons:
  - Private key was compromised, due to breaches, insider attack, vulnerability in choosing private keys (very rare but did happen) etc.
  - System admin left an organization
  - Business entity closed
  - Issuing CA was compromised (extremely rare but did happen)

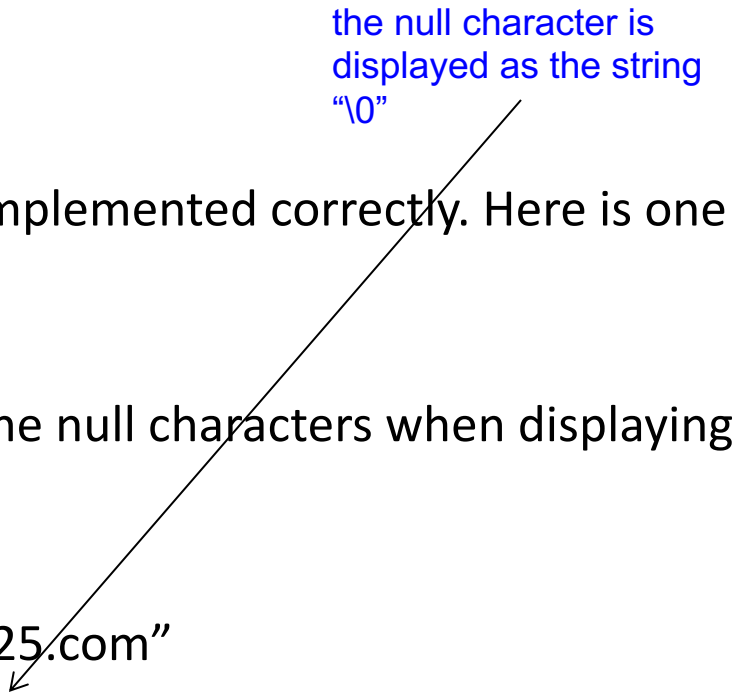
- A verifier needs to check whether a certificate in question is *still valid*, although the certificate is not expired yet. (conflicting requirement. Main purpose of certificate is to facilitate offline checking, and yet need to online verify whether it is being revoked).

- Two different approaches to certificate revocation:
  - Certificate Revocation List (CRL) :
    CA periodically signs and publishes a revocation list. Need an *online* CRL Distribution Point.
  - Online Certificate Status Protocol (OCSP):
    OCSP Responder validates a cert in question. Need an online OCSP Responder.

- Recommendation is for a user (e.g. browser) to periodically (say weekly) update its local cache of revocation list. The user does not need to online check whenever the user wants to verify a certificate.

4.3 Limitations/Attacks on PKI

Implementation Bugs (revisit in software security)



- Same as many other secure designs, it is vulnerable if not implemented correctly. Here is one interesting example:

- Some browsers ignore substrings in the “name” field after the null characters when displaying it in the address bar but include them when verifying the certificate.

  (a) Name appeared in the certificate:

  “www.comp.nus.edu.sg**\0.**hacker.13525.com”

  or

  “*.hacker.13525.com”

  (b) The browser displays it as

  “www.comp.nus.edu.sg”

- As a result, the viewers thought that they are connecting via https to (b), but in fact is connecting to (a).
- See

www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/

Abuse by CA

- There are so many CA's. One of them could be malicious. A rogue CA can practically forge any certificate. Here is a well-known incident.

- Trustwave issued a "subordinate root certificate" (i.e. the receipt can now issue certificate) 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.

see

http://www.computerworld.com/article/2501291/internet/trustwave-admits-issuing-man-in-the-middle-digital-certificate--mozilla-debates-punishment.html

Another famous case of abuse (or ignorant?)

- Lenovo's SuperFish scandal.

(previously reserved as class presentation. In this semester, we would not cover it. This is a good example, but we have too many materials to cover.)

Social Engineering

PKI ”certifies” the association (name, public key). In the first place, how the verifiers know that the “name” is correct? Social engineering exploits confusing naming convention, so that the verifiers are tricked into using a wrong name. E.g, an attacker first rightfully registered for a domain name that resembles a targeted domain name. Next, used the registered domain to confuse the victim in phishing attack. A few techniques to confuse the verifier: “Domain typosquatting”, “Homograph attack”, using sub-domain name. These attack are aka Domain spoofing, URL spoofing, Fake URL, etc.

Method 1 (typosquatting)

1. An attacker registered for a domain name
   luminus.nus.edv.sg
   and obtained a valid certificate of the above name. No one has registered edv.sg and thus the attacker is about to get it.
2. The attacker employed “phishing attack”, tricking the victim to click on the above link, which was a spoofed site of
   luminus.nus.edv.sg
3. The address bar of the victim’s browser correctly displayed
   https://luminus.nus.edv.sg
   but the victim didn’t notice that and logged in using the victim’s password.

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

See https://en.wikipedia.org/wiki/IDN_homograph_attack

Method 2 (sub-domain)

A more commonly deployed method uses sub domain. E.g.

Attacker is the rightful owner of 134566.com. Attacker creates a sub.domain

luminus.nus.edu.sg.134566.com.

Since attacker is the owner of 134566.com, it can get a valid certificate of luminus.nus.edu.sg.134566.com or *.134566.com

Question

- How browsers such as Safari, Firefox, Chrome display the url "`https://www.nus.edu.sg/admissions`" in the address bar? Why they choose to highlight nus.edu.sg?