Lecture 4: PKI + Channel Security

4.1 Distribution of public key

4.2 PKI

- 4.2.1 Certificate
- 4.2.2 CA

4.3 Limitations/attacks on PKI

4.4 Strong authentication

4.5 Key-exchange

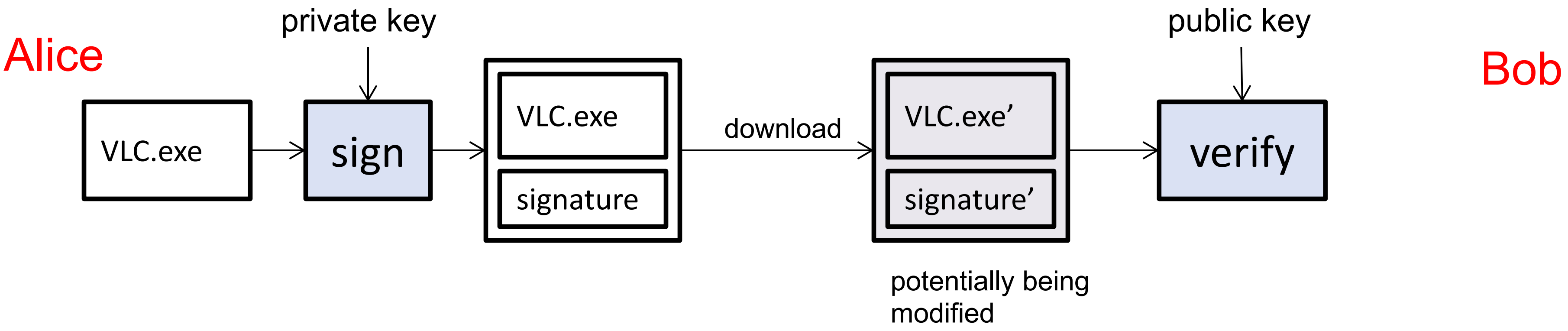
4.6 Putting all together: Securing Communication Channel

4.7 Remarks on security reduction

4.1 Public key Distribution

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, say Alice, "signed" the file VLC.exe using the the developer's private key.
    2. A user, say Bob, 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 Alice's public key.



Secure Channel:
Compare to the symmetric key setting (mac), here, we don't need to have a secret key between Alice to Bob.

However, we still need a secure channel for Alice to send her public key to Bob.

Example of “signed” email using PGP public key

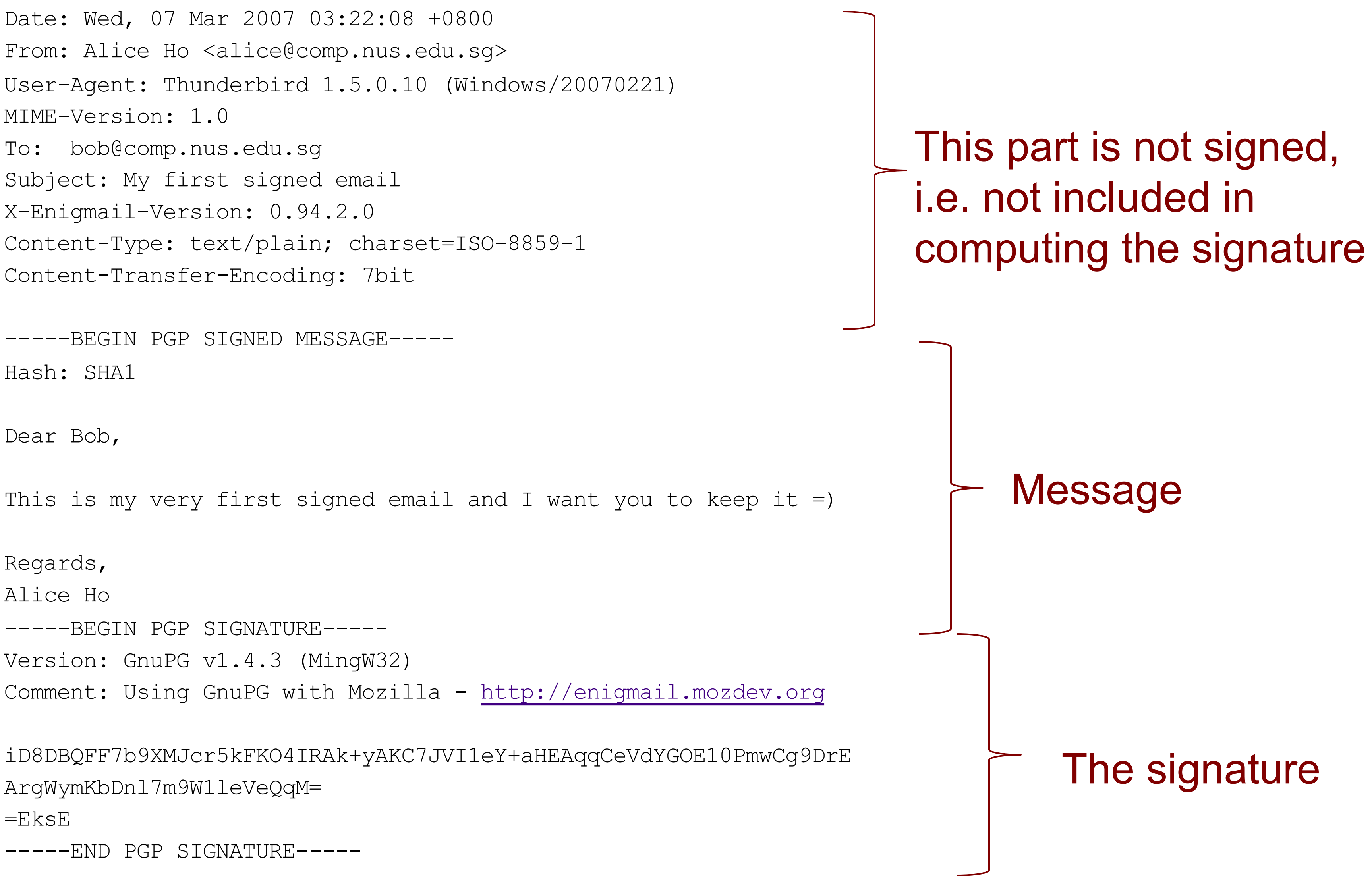
- Alice (with email account alice@comp.nus.edu.sg) sent an email to Bob. Alice has a pair of “PGP” public-private key. Alice’s email is signed using her private key. (see next slide for the actual email sent).

- 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. ☹

- We are now back to square one: we need some secure channel for Alice to send her public key to Bob.

Example of “signed” email using PGP public key



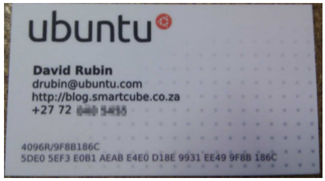
```
Date: Wed, 07 Mar 2007 03:22:08 +0800
From: Alice Ho <alice@comp.nus.edu.sg>
User-Agent: Thunderbird 1.5.0.10 (Windows/20070221)
MIME-Version: 1.0
To:  bob@comp.nus.edu.sg
Subject: My first signed email
X-Enigmail-Version: 0.94.2.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: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

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

Message

The signature

Motivation: What if public key is spoofed?



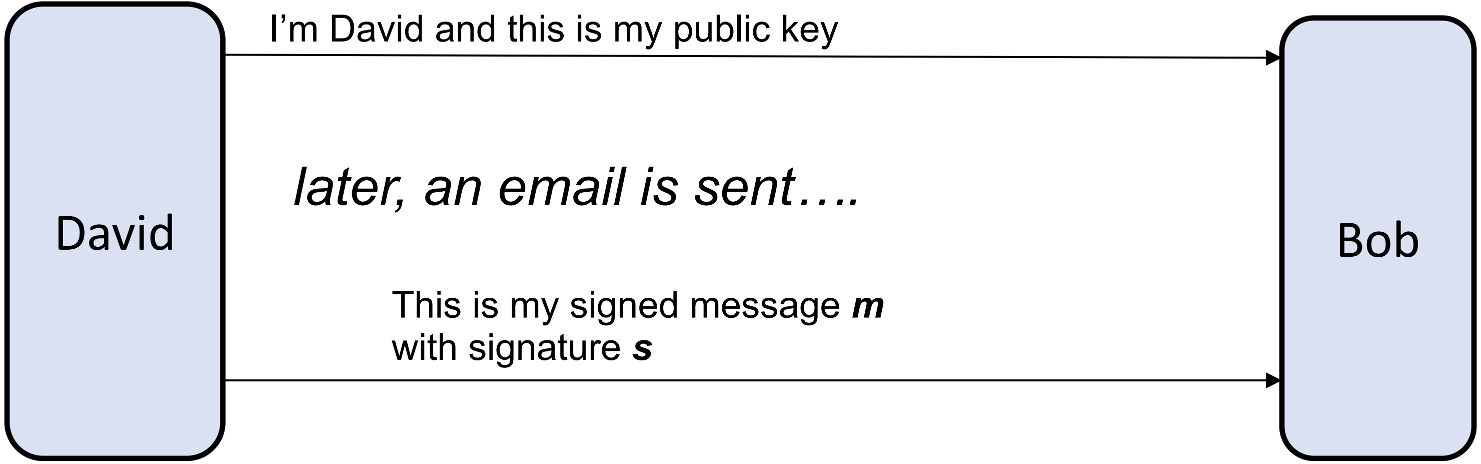
ubuntu

David Rubin
drubin@ubuntu.com
http://blog.smartcube.co.za
+27 72 [redacted]

4096R/9F8B186C
5DE0 5EF3 E0B1 AEAB E4E0 D18E 9931 EE49 9F8B 186C

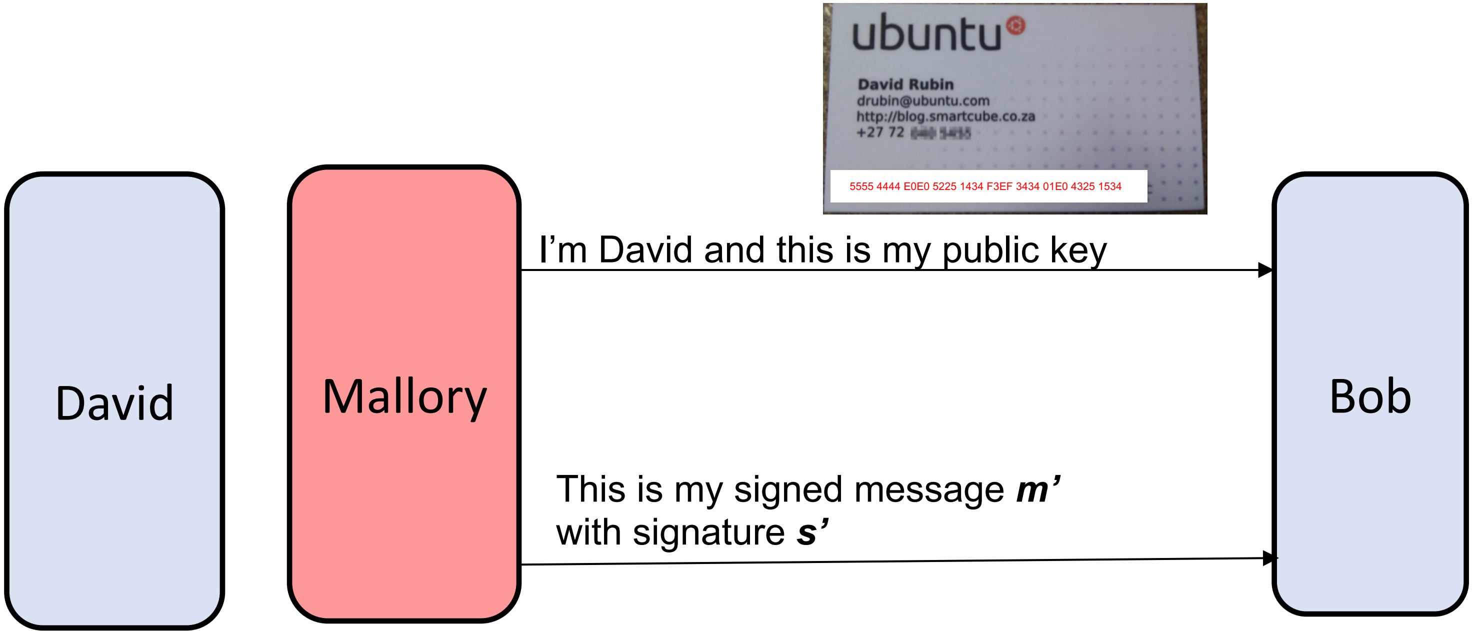
Image from
https://wordtothewise.com/2014/09/alice-and-bob-and-pgp-keys/

Intended situation



David
I'm David and this is my public key
later, an email is sent….
This is my signed message ***m***
with signature ***s***
Bob

Under attack



ubuntu

David Rubin
drubin@ubuntu.com
http://blog.smartcube.co.za
+27 72 [redacted]

5555 4444 E0E0 5225 1434 F3EF 3434 01E0 4325 1534

David
Mallory
I'm David and this is my public key
This is my signed message ***m'***
with signature ***s'***
Bob

Public key distribution vs Symmetric key distribution

Nevertheless, it is easier to securely distribute public compare to symmetric key.

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

- ***Public key***: An entity doesn't need to know the existence of the receiving end before broadcasting its public key. (In the previous slide, David doesn't need to know Bob nor talk to Bob)
- ***Symmetric key:*** Both entities need to interact to establish the key,

Key distribution

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

3 methods:

- Public Announcement
- Publishing publicly available directory
- Public Key Infrastructure

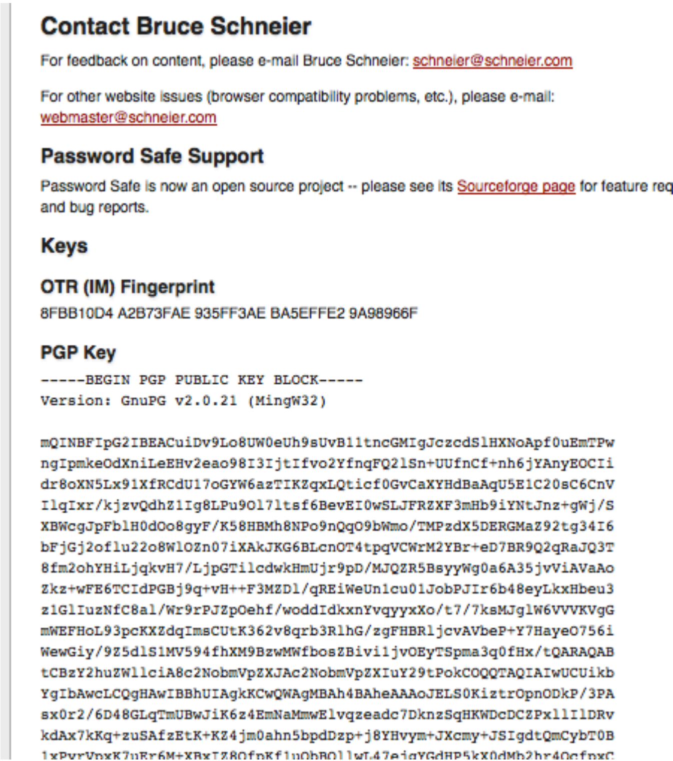
Key distribution: Public Announcement

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

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

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

Of course, we might get back to square one: How to authenticate info listed in blog.? Apparently, we need a trusted starting point.



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
1xPyrVpxK7uEr6M+XBxIZ8OfpKf1uObBOllwL47ejqYGdHP5kX0dMb2hr4OcfpxC
```

- Limitations: not standardized, and thus there isn’t a systematic way to find/verify the public key when needed.

- Eventually, we still need some “trusted” starting points.

  In the previous example on Schneier’s PGP key, we trust the web-site.

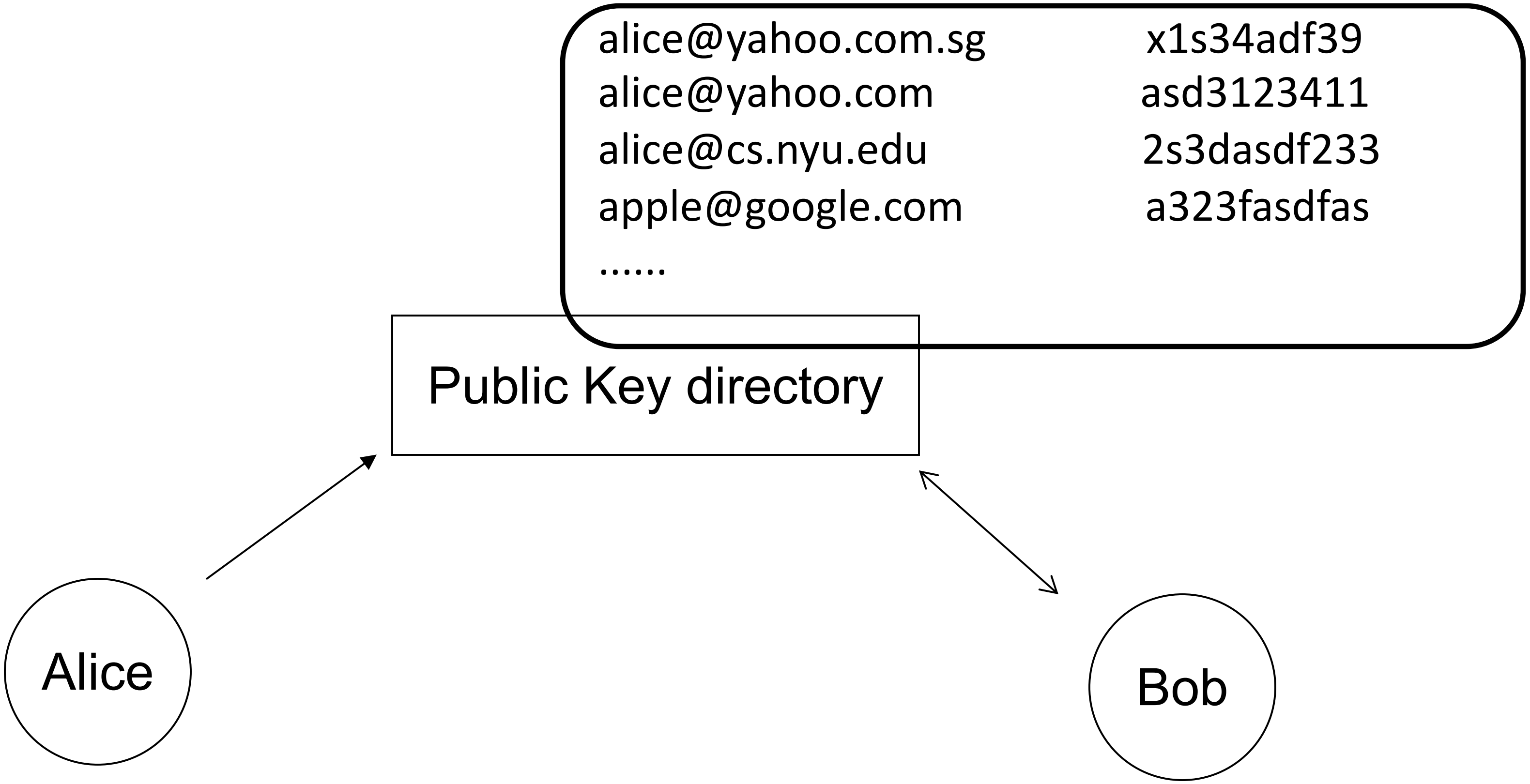
Get a public key and send a signed email.   (try PGP)

Key distribution: 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.



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

Potential Issues:

- Anyone can post their public keys in the “trusted” server.

For e.g. https://pgp.mit.edu/

- Suppose the server receives a request to post a public key, how does the server verify that the information is authentic?

- Eventually, some entity needed to be trusted. Eg https://pgp.mit.edu/

PKI

PKI is a standardized system that distribute 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.

PKI Mechanism

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

4.2 Public Key Infrastructure

4.2.1 CA & Certificate

Certificate Authority

- The CA issues and signs digital *certificates.* (next few slides)

- We can view CA as a trusted authority that manage a directory of public keys. An entity can request adding its public key to the public directory.   In addition, anyone can send queries to search the directory.

- The CA also has its own public-private key. We first assume that the CA's public key has been securely distributed to all entities involved. (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.

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 get Alice’s public key. One method is for Bob to query the CA.

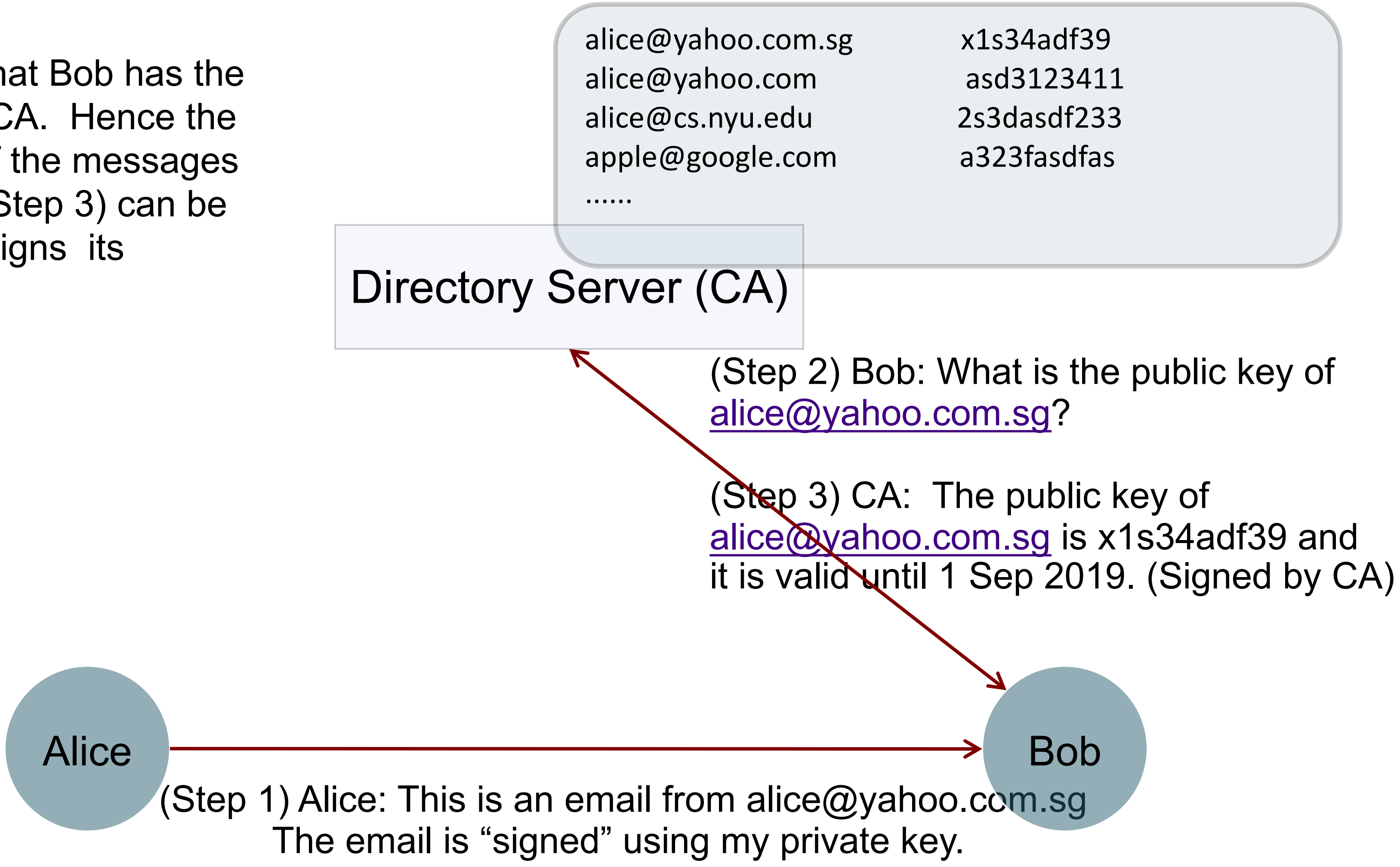
There are two limitations of retrieving the public key from the CA as-and-when needed:

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

Using certificates is a “smart” way to avoid the above limitations.

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.



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 2019. (Signed by CA)

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

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

signature: xsdewsdesd

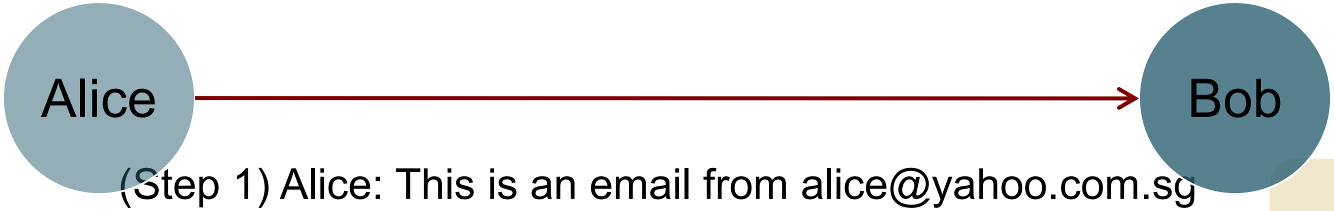
With Certificate

Note that in the previous slide, data in step 3 is always the same. So, a "lazy" CA would "sign" the message before hand and pass it to Alice. Such 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 indeeds 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.

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

CA's public key

Certificate

- A ***certificate*** is a digital document that contains ***at least*** the following 4 main items

  1) The identity of an owner, for e.g. “alice@yahoo.com”

  2) The public key of the owner.

  3) The time window that this certificate is valid.

  4) The signature of the CA.

It also has additional information like the purpose of the public key. e.g.

- Whether the owner has a single (e.g. www.comp.nus.edu.sg) or a group of identities (e.g. *.comp.nus.edu.sg).
- Usage of the certificate, (is the owner a CA, or a webserver, or …)
- …

Identity of the owner may change, so the certificate should expire

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

- (Chain of trust) Bob trusts the CA. Hence, information signed/certified by the CA is also trusted.

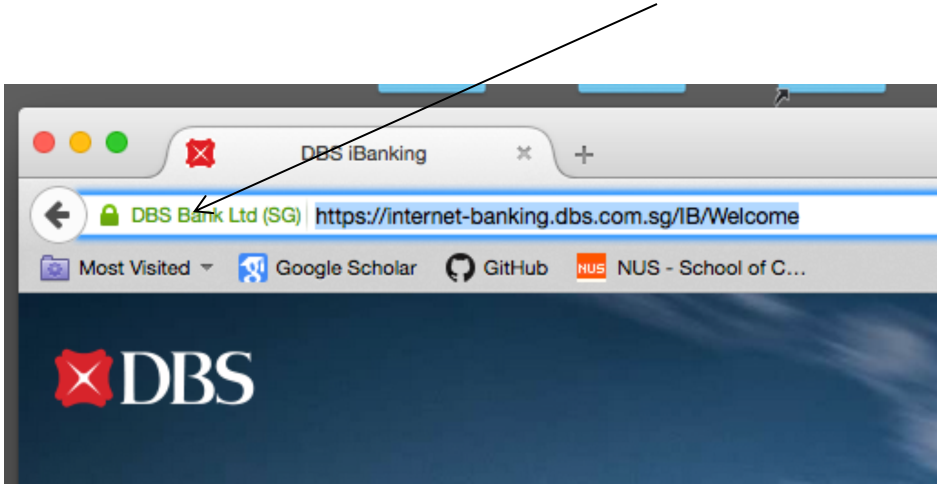
- In other words: We first assume that Bob has the authentic CA’s public key. Thus, information verified using CA’s public key is authentic. Hence, the public key extracted from the certificate is indeed Alice’s. From Alice’s public key, Bob can verify authenticity of the email.

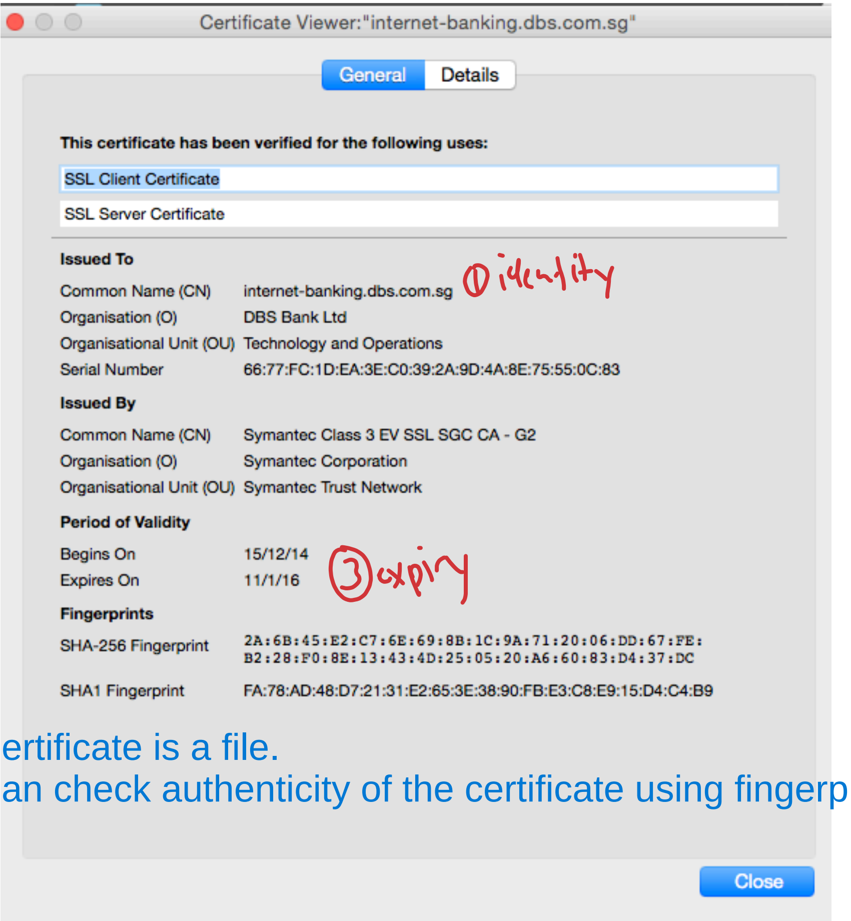
CA’s public key → Certificate → Alice’s public key → Alice’s email.

Example of Certificate

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

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





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

① identity

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

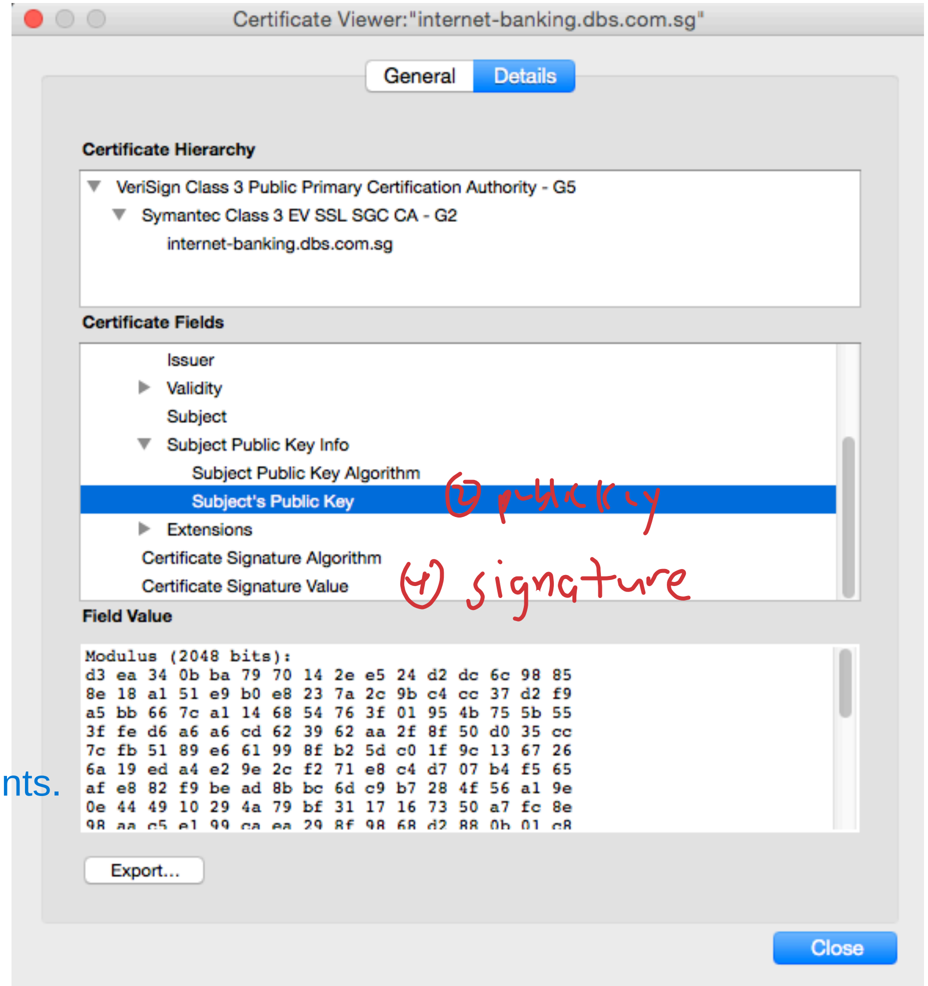
Begins On	15/12/14
Expires On	11/1/16

③ expiry

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

Certificate is a file.
Can check authenticity of the certificate using fingerprints.



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

② public key

④ signature

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
```

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 *(Note Signature Algorithm below too)*
    - Issuer Name
    - Validity period: Not Before, Not After
    - 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

How Do I Get a Certificate?

- Goto 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 was predicted that it will reach **90%** by the end of 2019

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 “certifies” that the public key is indeed belonged to the claimed identity.

- We assume that Bob already has CA’s public key 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 CA is not free. In fact, it is expensive! (lookup the price list)

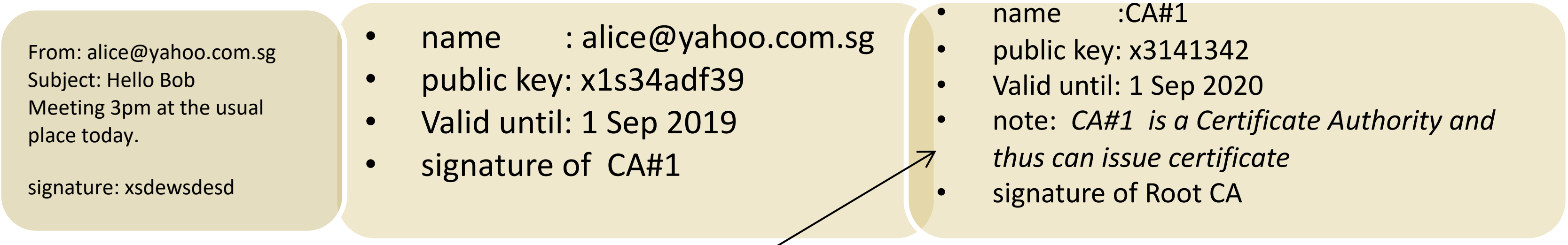
Certificate Chain

- 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.



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 CA#1

- name :CA#1
- public key: x3141342
- Valid until: 1 Sep 2020
- note: *CA#1 is a Certificate Authority and thus can issue certificate*
- signature of Root CA

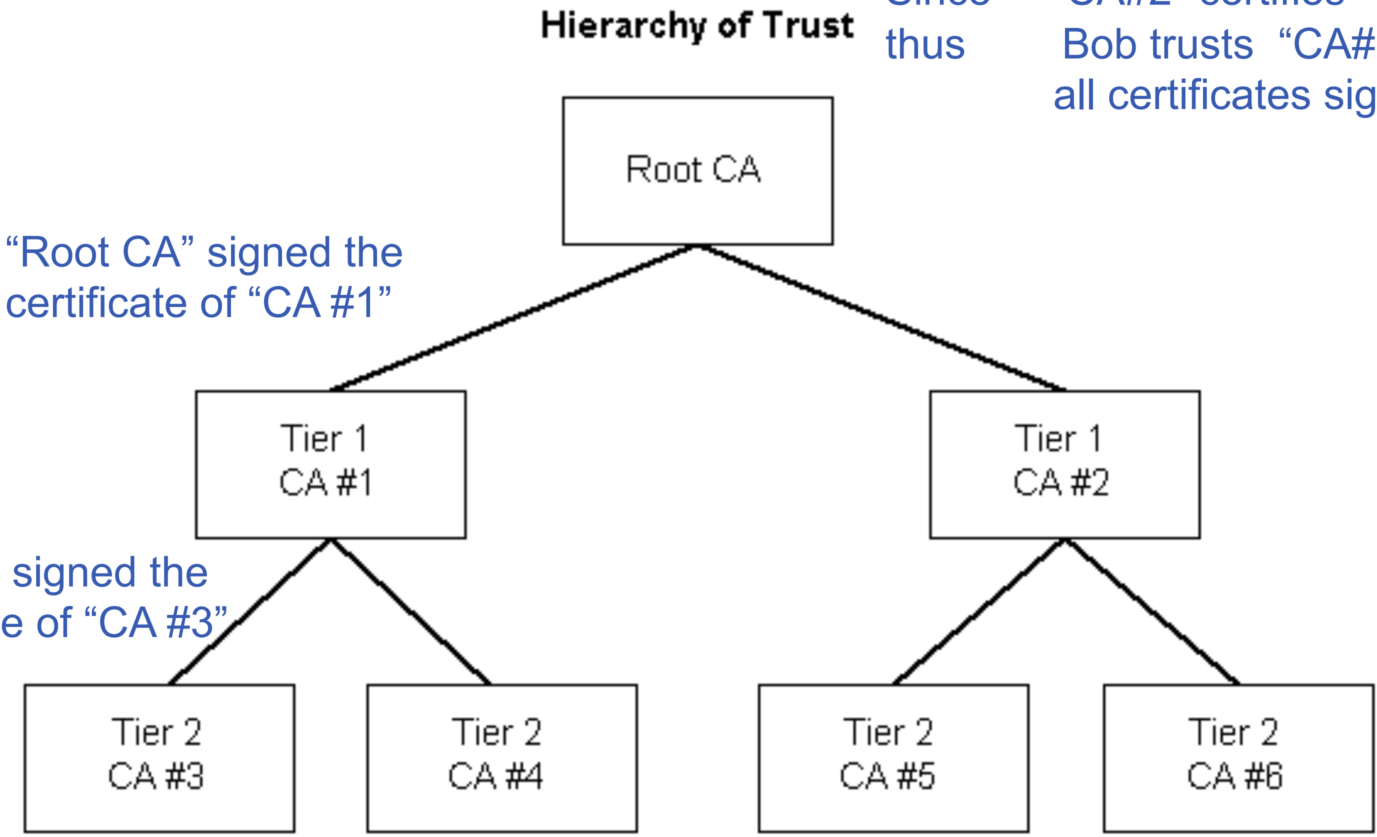
In our example, the certificate issued to CA#1 clearly indicate that CA#1 is a certificate authority and can issue certificate. Without that “note”, the owner can’t issue other certificates.

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

Hierarchy of Trust

Since Bob trusts the “Root CA”,
and “Root CA” certifies “CA#1”,
thus Bob trusts “CA#1”.

Since “CA#2” certifies “CA#3”,
thus Bob trusts “CA#3”, and
all certificates signed by “CA#3”.



“Root CA” signed the
certificate of “CA #1”

“CA1#1” signed the
certificate of “CA #3”

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

See [PF] page 117 to 121 for detailed explanation of ***Trust***.

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 certificate expired, no certificate, wrong signature, etc)

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
- Two 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
- Either way, 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

Implementation Bugs

the null character is displayed as the string "\0"

- There are quite a number of well-known implementation bugs leading to severe vulnerability. Here is one 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 which is used in verifying the certificate:

  "www.comp.nus.edu.sg**\0**hacker.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/

Question: Terminologies. What is CVE?

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/screwup

(reserved for class presentation)

Social Engineering

Malicious hackers may carry out “typosquatting”.

E.g.

1. A hacker registers for a domain name

   ivle.nvs.edu.sg

and obtained a valid certificate of the above name.

2. The hacker employs “phishing attack”, tricking the victim to click on the above link, which is a spoofed site of

   ivle.nus.edu.sg

3. The address bar of the victim’s browser correctly displayed

   https://ivle.nvs.edu.sg

but the victim doesn’t notice that, and log in using the victim’s password.

It is also possible that the hacker doesn’t carry out step 2. He just wait and hope that some students accidentally type the wrong address ivle.nvs.edu.sg

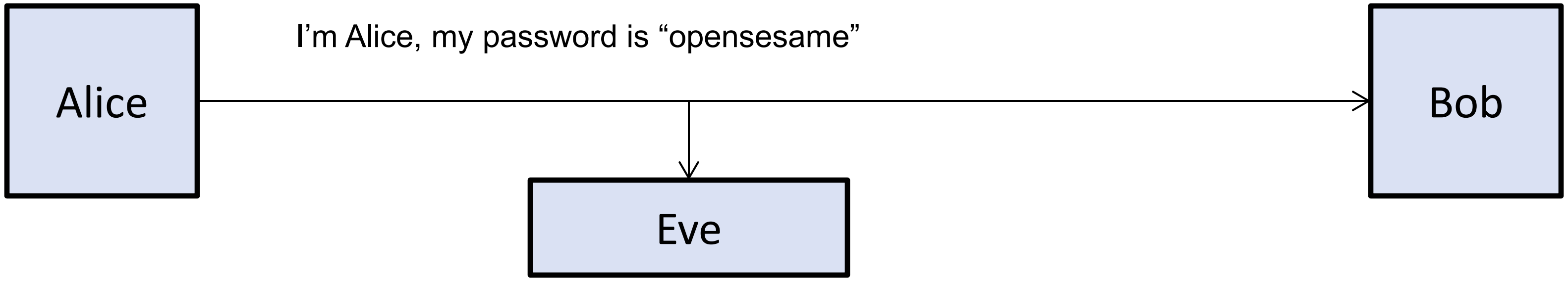
read http://en.wikipedia.org/wiki/Typosquatting

4.4 Strong Authentication

Weak Authentication

Sending password in clear is a weak authentication. An eavesdropper can get the password and replay it.

Eavesdrops



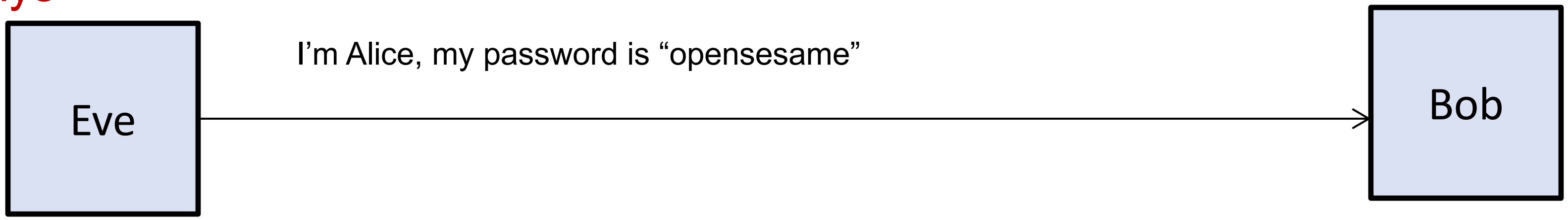
Alice

I’m Alice, my password is “opensesame”

Bob

Eve

Replays



Eve

I’m Alice, my password is “opensesame”

Bob

Interestingly, it is possible to have a mechanism that Alice can “prove” to Bob that she knows the secret, without revealing the secret.

Strong Authentication: Challenge-response

Suppose Alice and Bob have a shared secret ***k*,** and both have agreed on a message authentication code.

(1) Alice sends to Bob a hello message

"Hi, I'm Alice"

(2) (Challenge) Bob randomly picks a plaintext ***m*** and sends ***m*** to Alice.

(3) (Response) Alice computes $\boldsymbol{t}$ = mac_k (***m***). Alice sends ***t*** to Bob.

(4) Bob verifies that the tag received is indeed the mac of ***m***. If so, accepts, otherwise rejects. (Only the entity who knows ***k*** can produce the mac, and hence must be Alice or Bob.)

- By property of mac, even if Eve can sniff the communication between Alice and Bob, Eve still can't get the secret key ***k***, and can't forge the mac for messages that Eve has not seen before.

- Eve also can't replay the response. This is because the challenge is randomly chosen and likely to be different in the next authentication session. The challenge ***m*** ensures ***freshness*** of the authentication process.

- This protocol only authenticates Alice. That is, authenticity of Alice is verified. Hence it is call ***unilateral authentication.*** There are also protocols to verify both parties, which are called ***mutual authentication.***

Unilateral authentication using PKC

We can also have a public key version using signature. 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 verifies that the signature is correct.

By security property of signature, the eavesdropper can't derive Bob's private key and replay the response. If Alice already knows Bob's public key, the certificate can be omitted.

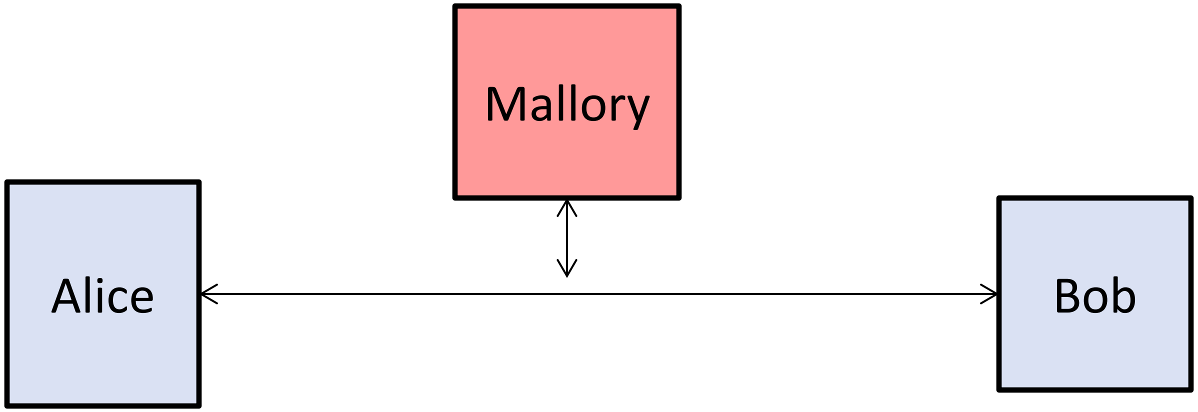
The value ***r*** is also known as the *cryptographic nonce* (or simply *nonce*).

Question: What is the purpose of the nonce? *To achieve freshness & prevent replay attack.*

Terminologies: "Nonce" is similar to "challenges. We can view "nonce" as a type of challenges. Nonce is typically short and randomly chosen, whereas some challenges can be long and derived from some information.

Key-exchange and authenticated key-exchange

- You may wonder what come next after authentication. Consider the setting of Alice, Bob and the malicious Mallory. Mallory wants to impersonate Alice. Recall that Mallory is malicious and can sniff, spoof, and modify the message.



Imagine that Mallory allows Alice and Bob to carry out the 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!   (Here, Mallory is the man-in-the-middle).

Remarks: (sniff==listen, spoof==forge & insert, modify== replace/delete)

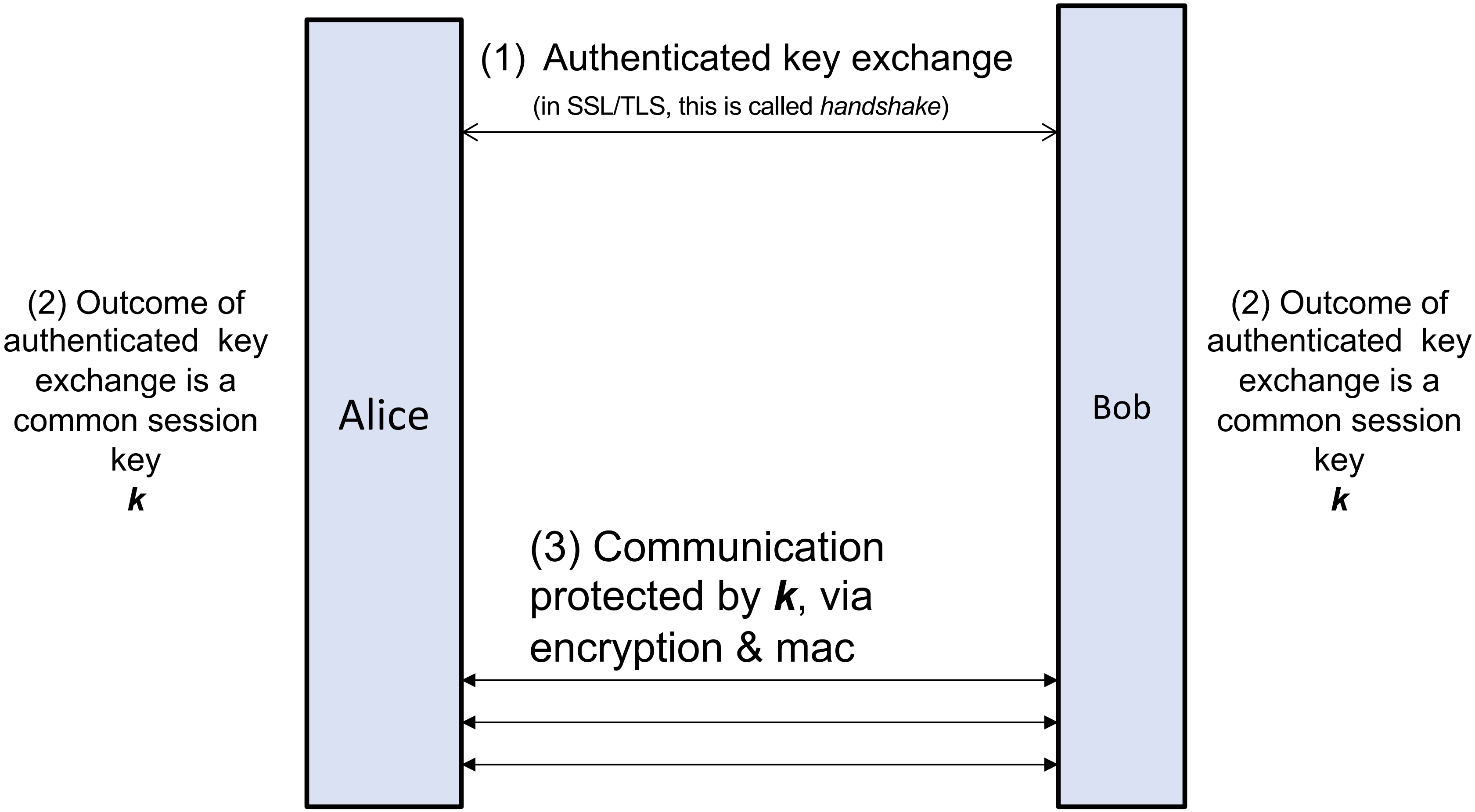
- Authentication protocols we had studied so far assume that the adversary is unable to interrupt the session.

- For applications whereby Mallory can interrupt the session, we need something more: The outcome of the authentication process must somehow pass to the next phase. This is done by establishing a new shared secret ***k.*** In other words, the outcome of the authentication protocol is a shared secret ***k***. This shared key is aka ***session key.*** Subsequent phase will be protected using ***k.***

- The process of establishing a secret (with or without authentication) between Alice and Bob is called

  key-exchange, or key-agreement.

- If the process is incorporated with authentication, then it is called ***Authenticated key-exchange***. A well-known scheme is called ***station-to-station*** protocol.



(1) Authenticated key exchange

(in SSL/TLS, this is called *handshake*)

(2) Outcome of authenticated key exchange is a common session key ***k***

Alice

Bob

(2) Outcome of authenticated key exchange is a common session key ***k***

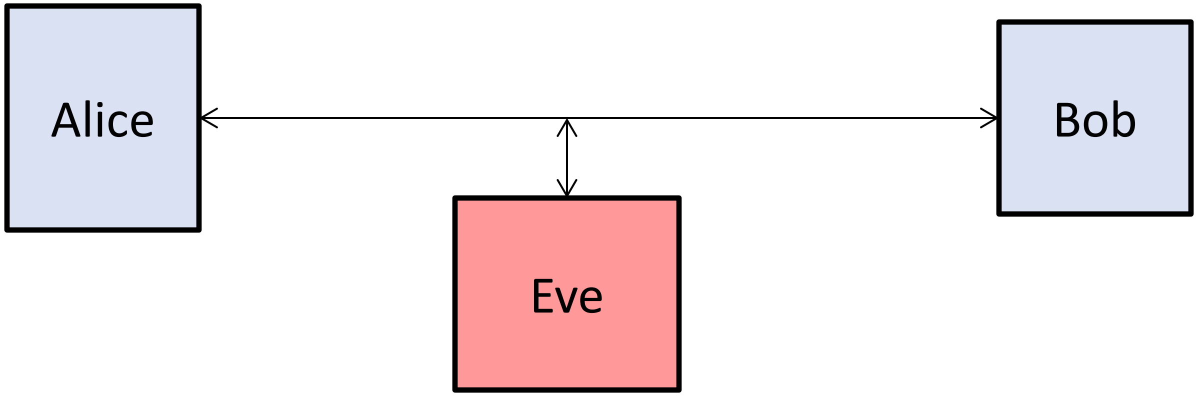
(3) Communication protected by ***k***, via encryption & mac

4.5 Key-exchange, authenticated key-exchange

Key-exchange (Secure against Eve, but not Mallory. No authentication)

- Alice and Bob want to establish a common key. The channel could be eavesdropped by Eve. We assume that both Alice and Bob are authentic. They want a protocol such that Eve is unable to extract any information of the established key.

- The established key can be used to protect (e.g. via cipher, mac) subsequent communication between Alice and Bob.



- The next slide gives a method based on PKC. Another well-known method is Diffie-Hellman Key exchange. DH key exchange achieves additional security (optional: *forward secrecy*).

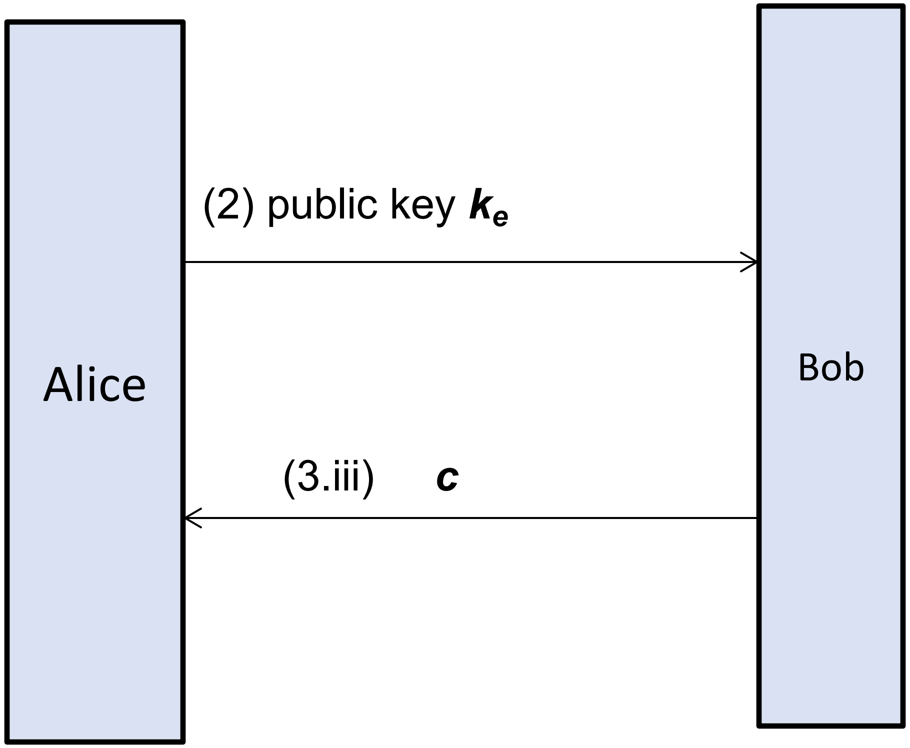
Here, we only consider Eve who can sniff, not the malicious Mallory who can modify the communication.

PKC-based Key-exchange (no authentication)

Here is a key-exchange that uses a PKC

1. Alice generates a pair of private/public key.
2. Alice sends the public key $\boldsymbol{k_e}$ to Bob.
3. Bob carries out the following
   1. Randomly chooses a secret $\boldsymbol{k}$,
   2. Encrypts $\boldsymbol{k}$ using $\boldsymbol{k_e}$.
   3. Sends the ciphertext $\boldsymbol{c}$ to Alice.
4. Alice uses her private key $\boldsymbol{k_d}$ to decrypt and obtain $\boldsymbol{k}$.

(1) generates $\boldsymbol{k_e}$ and $\boldsymbol{k_d}$.



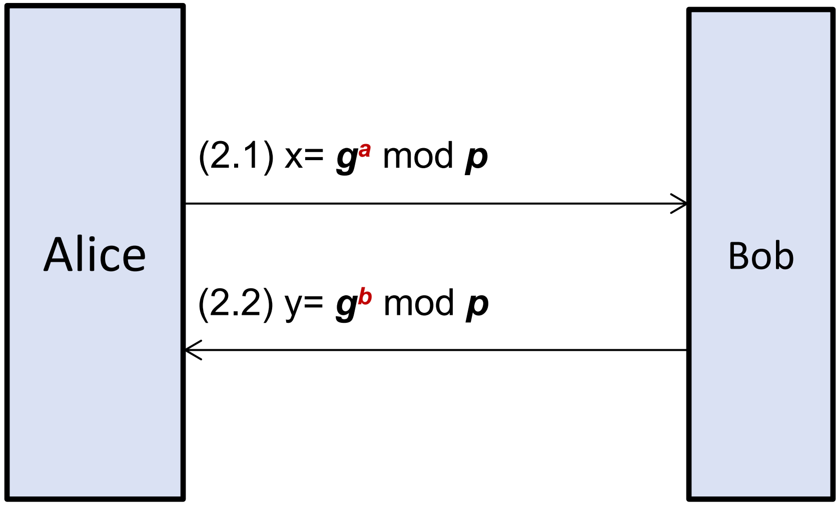
(3.i) Chooses a random secret $\boldsymbol{k}$
(3.ii) $\boldsymbol{c}$ = Encrypt ($\boldsymbol{k_e}$, $\cancel{c}$ k)

(4) k = Decrypt(k_d,c)

DH has forward secrecy but PKC doesn't

Dillie-Hellman key-exchange (no authentication)

We assume both Alice and Bob have agreed on two *public* parameters, a *generator* g and a large (e.g. 1000 bits) prime p. Both g and p are not secret and known to the public.



(1.1)
- Randomly chooses $\boldsymbol{a}$
- Compute x= $\boldsymbol{g^{a}}$ mod $\boldsymbol{p}$

(1.2)
- Randomly chooses $\boldsymbol{b}$
- Compute y= $\boldsymbol{g^{b}}$ mod $\boldsymbol{p}$

Alice

Bob

(2.1) x= $\boldsymbol{g^{a}}$ mod $\boldsymbol{p}$

(2.2) y= $\boldsymbol{g^{b}}$ mod $\boldsymbol{p}$

(3.1)
- Compute k = y^{a} mod $\boldsymbol{p}$

(3.2)
- Compute k = x^{b} mod $\boldsymbol{p}$

Security relies on the CDH assumption.

Computational Diffie-Hellman CDH assumption:
Given g, p, x= $\boldsymbol{g^{a}}$ mod $\boldsymbol{p}$, y= $\boldsymbol{g^{b}}$ mod $\boldsymbol{p}$, it is computationally infeasible to find k=$\boldsymbol{g}^{ab}$ mod $\boldsymbol{p}$.

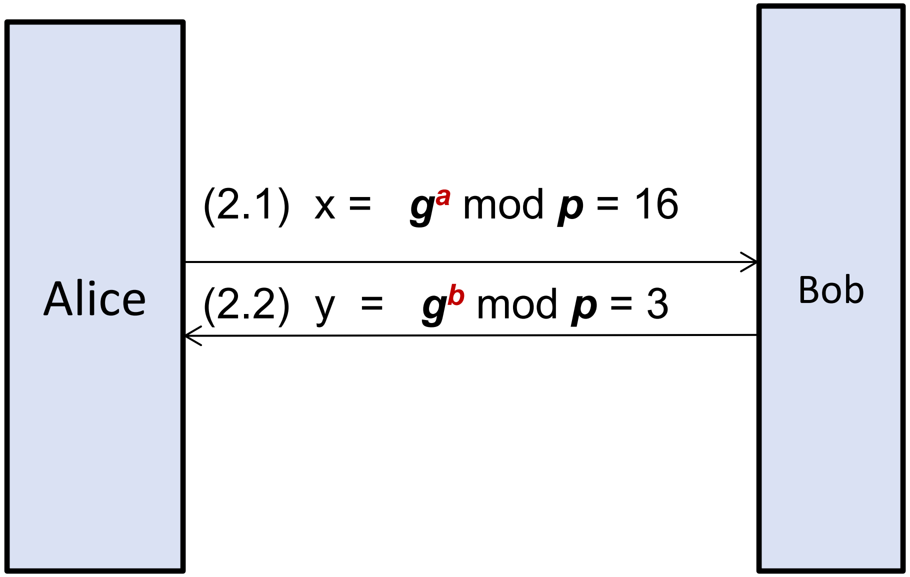
Remarks:
1. Step (1.1)&(1.2), (2.1)&(2.2), (3.1)&(3.2) can be carried out in parallel.
2. The assumption seems self-fulfilling. Nonetheless, there are many evidences that it holds.
3. The operation of "exponentiation" can be applied to any algebraic group, i.e. not necessary integers. CDH doesn't hold in certain groups. The crypto community actively searches for groups that CDH holds. E.g. Elliptic Curve Cryptography ECC is based on elliptic curve group where CDH believed to hold.

Eg.

$g = 2, \qquad p = 23$

(1.1) randomly
chooses $\boldsymbol{a} = 15$

(1.2) randomly
chooses $\boldsymbol{b} = 8$



Alice

Bob

(2.1) $x = \boldsymbol{g}^{a} \bmod \boldsymbol{p} = 16$

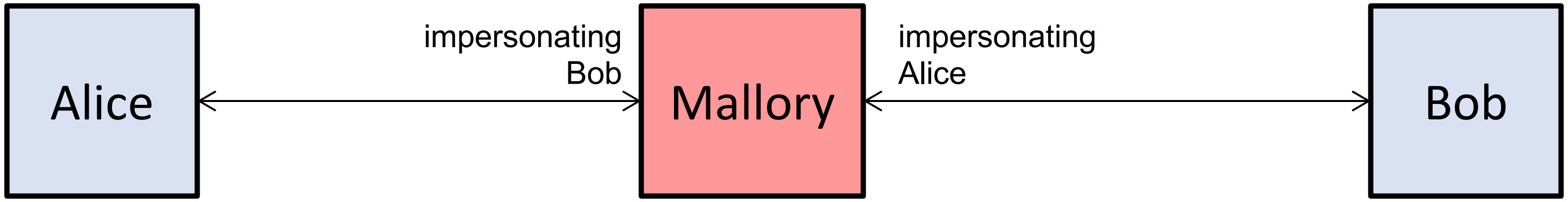
(2.2) $y = \boldsymbol{g}^{b} \bmod \boldsymbol{p} = 3$

(3.1) Compute
$k = y^{a} \bmod \boldsymbol{p}$
$= 3^{15} \bmod 23$
$= 12$

(3.2) Compute
$k = x^{b} \bmod \boldsymbol{p}$
$= 16^{8} \bmod 23$
$= 12$

The basic key-exchange can't guard against Mallory.

- What if the adversary is malicious? Example, a man-in-the-middle?



Key-exchange between Alice and Mallory established key k_A

Key-exchange between Alice and Mallory established key k_B

In this case, Alice mistaken that Mallory is Bob. Communication from Alice is encrypted using k_A. Mallory can decrypt using k_A and re-encrypt using k_B. Hence, Mallory can see and modify the message.

Authenticated key-exchange

- A key-exchange protocol assume that the adversary is curious (i.e. can only sniff), but not malicious. Hence, it might be vulnerable to Mallory.

- For authenticity, we need ***authenticated key-exchange.*** The authentication can be achieved using PKC or shared password. In public key version, Alice and Bob need to know each other public key (this can be achieved using PKI and certificate). In unilateral authentication, only one party need to have public key. After the protocol has completed, a common key (i.e the ***Session key***) is established.

- When using DH key exchange with PKC, this special case of authenticated key-exchange is also known as the ***Station-To-Station Protocol (STS).***

- It turns out that PKC-based authenticated key-exchange can be easily obtained from existing key-exchange. This is done by simply signing all communication.

Station-to-station protocol (Authenticated key-exchange based on DH)

We assume both Alice and Bob have agreed on two *public* parameters, a *generator* g and a large (e.g. 1000 bits) prime p. Both g and p are not secret and known to the public.

Here, we consider unilateral authentication. Alice want to authenticate Bob.

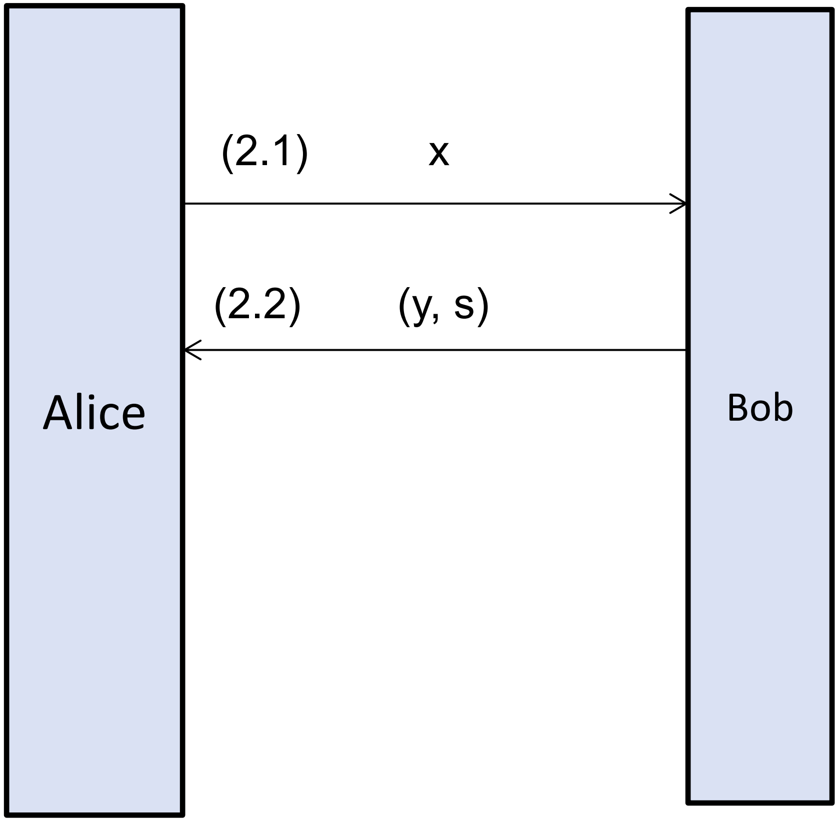
Alice knows Bob's public key $\text{Bob}_{\text{public}}$

(1.1)

- Randomly chooses $\boldsymbol{a}$
- Compute x= $\boldsymbol{g}^{a}$ mod $\boldsymbol{p}$

(3.1)

- Verify signature s
- Compute $k = y^{a}$ mod $\boldsymbol{p}$



Alice

Bob

(2.1) x

(2.2) (y, s)

($\text{Bob}_{\text{public}}$, $\text{Bob}_{\text{private}}$)

(1.2)

- Randomly chooses $\boldsymbol{b}$
- Compute y= $\boldsymbol{g}^{b}$ mod $\boldsymbol{p}$
- Sign y to obtain signature $\boldsymbol{s}$

(3.2) Compute
$k = x^{b}$ mod $\boldsymbol{p}$

Remark: This is unilateral authentication. Can extend it to mutual by making Alice signs her messages in step (2.1).

Summary: Mutual Authenticated key exchange

- *Before the protocol*:
  1. Alice has a pair of public, private key (A_{public} , $A_{private}$).
  2. Bob has a pair of public, private key (B_{public} , $B_{private}$).
  3. Alice knows Bob public key and vice versa. These two sets of keys are known as the ***Long-term key*** or ***Master key.***

- They carry out Authenticated key exchange protocol (e.g. STS). If an entity is not authentic, the other will halt.

- *After the protocol*:
  1. Both A and B obtain a shared key **k,** known as the ***Session key.***

- Security Requirement.
  1. (Authenticity) Alice is assured that she is communicating with an entity who knows $B_{private.}$
  2. (Authenticity) Bob is assured that he is communicating with an entity who knows $A_{private.}$
  3. (confidentiality) Attacker unable to get the session key.

Remarks

Remark: Mutual Authentication vs Unilateral Authentication

- In Unilateral authentication, only one party get authenticated: Alice wants to be assured of Bob's identity, but Bob does not care about Alice's identity.

- Example. Alice is surfing a website https://www.bbc.com Here, Alice wants to be assured that indeed she is visiting the authenticatic website, whereas the website doesn't care.

Password authenticated key exchange

- The authenticated key exchange can also be used in symmetric key setting. In symmetric key setting, both Alice and Bob share a common secret, typically a password. They conduct authentication + key exchange based on this secret.
- When the shared secret is a password, it is often called "Password authenticated key exchange". See https://en.wikipedia.org/wiki/Password-authenticated_key_agreement
  - Crucial difference between human generated password and machine generated key: passwords typically are shorter, and vulnerable under dictionary attack. (Of course, if the password is strong, e.g. 20 characters that are randomly chosen, then dictionary attack is infeasible).
  - Offline dictionary attacks: After the attacker logged the traffic, it carries out offline exhaustive/dictionary attack by trying all possible passwords. The offline process could take longer time, but doesn't need to remain connected to the victim.

- A secure password authenticated key exchange ensure that even if the passwords are short, the adversary can't carry out offline dictionary attack. The adversary could be Eve, or Mallory who pretend to be one of the authenticating party.

- Note that simply adding mac signed using passwords cannot prevent offline dictionary attack. Since mac is deterministic, Eve, after captured the mac, can test it on the dictionary in offline. So, design of password authenticated key exchange is more complicated. Details omitted.

- Example:
  - Encrypted Key Exchange (EKE)
  - PEAP (Protected Extensible Authentication Protocol)

- Some protocols like LEAP (**Lightweight Extensible Authentication Protocol)** are vulnerable to offline dictionary attacks. In contrast, PEAP is secure against offline dictionary attacks.

- Question: Which protocol NUS wifi employs? In our previous example of an attacker spoofing a fake NUS hotspot in NUH bus stop, could the attacker successfully steal password?

  see NUS wifi setup instruction http://www.nus.edu.sg/comcen/gethelp/guide/itcare/wireless/NUS-WPA2%20Network%20Configuration%20Guide%20for%20Android%207.0.pdf

4.6 Putting all together: Securing Communication Channel

E.g. TLS

Consider a communication channel that is subjected to man-in-the-middle (sniffing spoofing, modifying), how to secure it using cryptographic tools?

E.g. Alice wants to visit a website `Bob.com`, how to achieve authenticity (i.e. Alice being assured that `Bob.com` is authentic) and confidentiality (i.e. no info leakage of the communication) ?

In TLS/SSL: (https uses TLS)

(1) Using ***long-term keys,*** carryout handshake protocol (i.e. the authenticated key-exchange) to establish ***session keys.***

(2) Subsequent communication protected by the session keys.

See more details of TLS handshake in later slides.

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

Alice wants to visit `Bob.com`: How TLS does it.

(Step 0) Alice obtains `Bob.com`'s public key. This is done by having Bob sending his certificate to Alice.

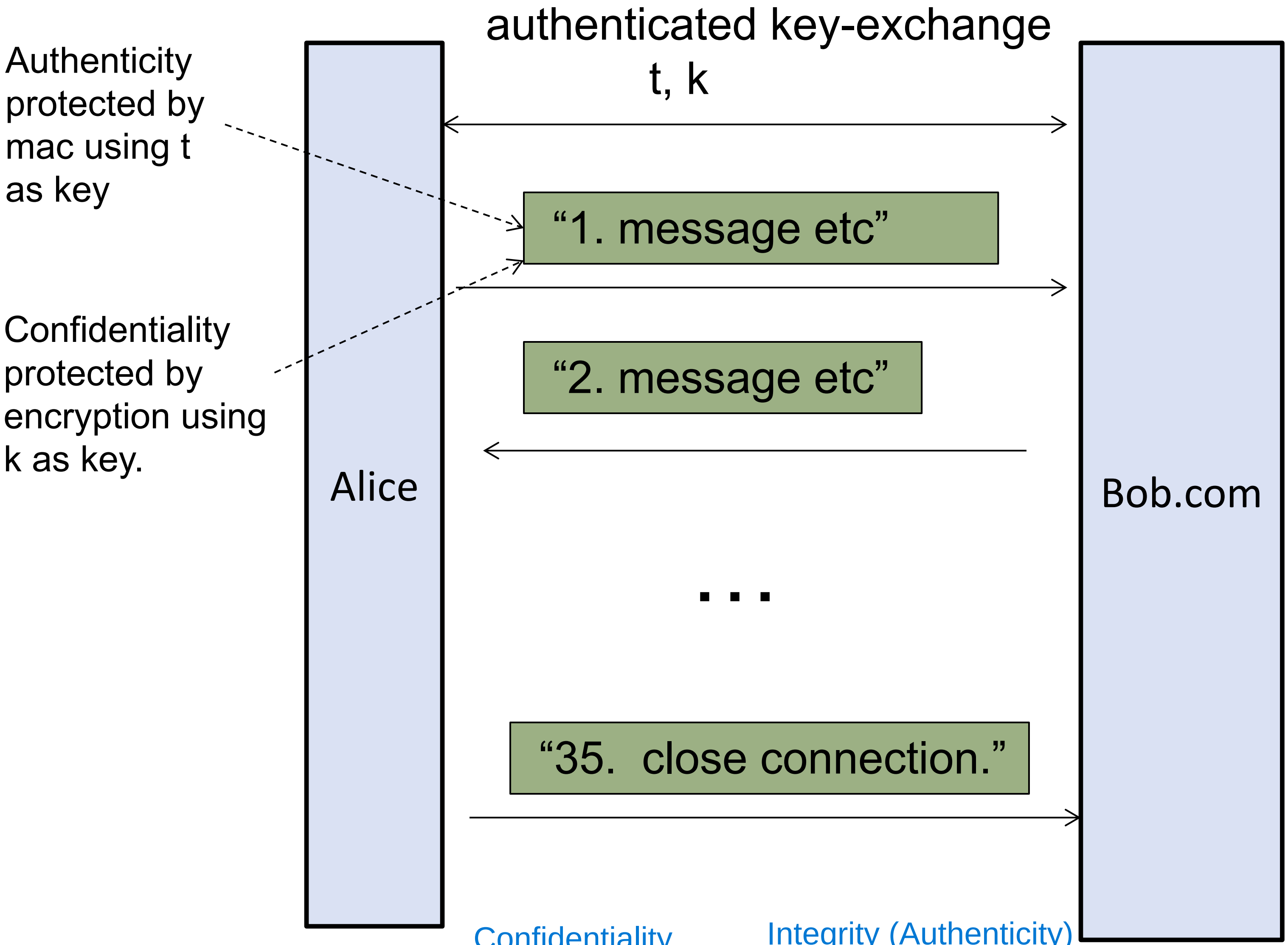
(Step 1) Alice and `Bob.com` carry out ***unilateral authenticated key exchange*** protocol with Bob's private/public key. After the protocol, both Bob and Alice obtain two shared keys ***t*** , ***k*** where ***t*** is the secret key of the MAC, and ***k*** is the secret key of the symmetric-key encryption, say AES. They are called the ***session keys***. From Alice's point of view, the protocol is secure in the sense that, only an entity who knows Bob's private key is able to complete the protocol. So, Alice is convinced that the entity who now holds ***t***, ***k*** is Bob. Here, Bob doesn't care about Alice's authenticity.

(Step 2) Subsequent interactions between Alice and `Bob.com` will be protected by ***t***, ***k*** and a sequence number. Suppose m_1, m_2, m_3, … are the sequence of message exchanged, the actual data to be sent for m_i is

$$E_k (\, i \, \| \, m) \quad \| \quad mac_t (\, E_k (i \, \| \, m) \,)$$

where *i* is the sequence number.

- || refer to string concatenation.
- (optional) The above is known as "encrypt-then-mac". There are other variants of "mac-then-encrypt", "mac-and-encrypt", "authenticated encryption".



Authenticity protected by mac using t as key

Confidentiality protected by encryption using k as key.

Confidentiality

Integrity (Authenticity)

The data eventually sent is ***E_k (”1. message etc”) || mac_t (E_k (”1. message etc”))***

Question: What is the role of the sequence number?

Messages can be dropped or replayed

Remarks

- Recall that to carry out the authenticated key-exchange, some mechanism of distributing the public key still required.

- TLS uses PKI to distribute the public key.

- TLS also supports mutual authentication.

Relationship among TLS/SSL/https

- SSL and Transport Layer Security (**TLS**) are protocols that secure communication using cryptographic mean.
- SSL is the predecessor of TLS.
- Https is built on top of TLS.

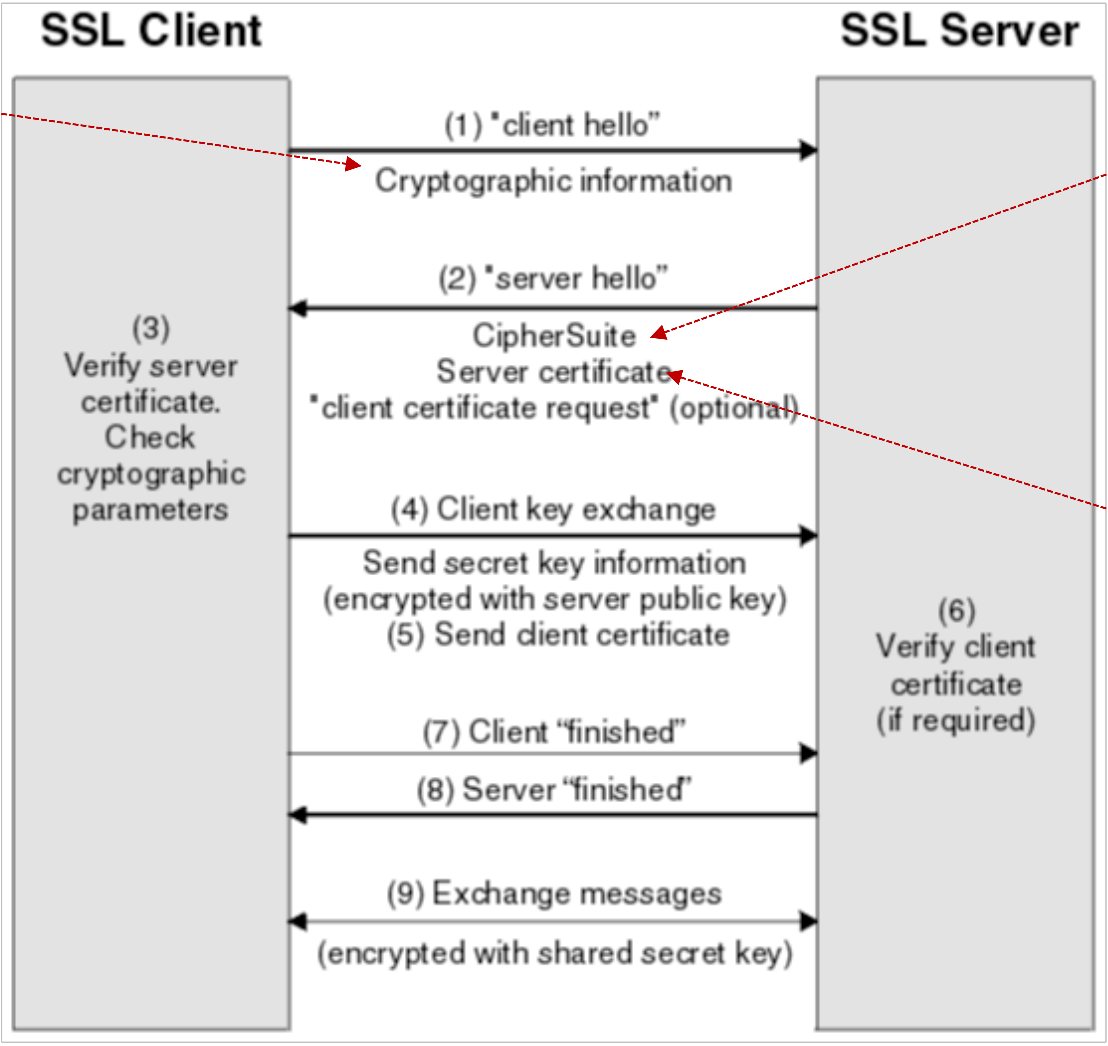
Question: Alice is in a café. She uses the free wifi to upload her assignment to IVLE (which uses https). The café owner controls the wifi router, and thus can inspect every packet going through the network. Can the café owner see Alice's report? ***No***.

Remarks:

- SSL 3.0 has a number of known vulnerabilities in its crypto implementation. Vulnerable to padding oracle attack, etc. (e.g *CVE*-2014-8730). TLS 1.0 is an upgraded version of SSL3.0 (in 1999). But TLS 1.0 allows fallback to SSL3.0, and thus TLS 1.0 is also vulnerable. See https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.0

TLS handshake (authenticated key exchange)

Negotiation of the crypto "suite", e.g. whether it is STS, RSA, and the key length. Here, Client propose the suite to be used.



Server confirms or counter-proposes the crypto suite.

Server's public key

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

4.7 Remark on security reduction

Comparing the security of two systems/schemes.

- Tutorial 3 illustrates the notion of security reduction. In a security reduction, there are two security requirements, say A (one way in T3Q8) and B (collision-resistant in T3Q3).
- In the analysis of reduction, we want to prove the following 3 equivalent statements.

For any system S,

There exists an attack on S that breaks A ⇒ There exists an attack on S that breaks B

For any system S,

There does not exists attack on S that breaks A ⇐ There does not exists attack on S that breaks B

For any system S,

The system S achieves requirement A ⇐ The system S achieves requirement B

Comparing the security of two systems/schemes.

Invert: Given y, find x s.t. H(x) = y
Collision: Find x1,x2 s.t. H(x1) = H(x2)

1. Randomly pick x
2. y = H(x)
3. x* = invert(y)
4. Check x* == x?

For any hash function H,

There exists an attack that invert H ⇒ There exists an attack that finds collision of H

For any hash function H,

There does not exists an efficient way to invert ⇐ There does not exists an efficient way to find collision

For any hash function H,

H is one-way ⇐ H is collision resistant

Existence of One-way function

- In this module, I always use the term “we believe”, “there is no known attacks”, and conditional statement. E.g.
  1. We believe that SHA3 is collision resistant.
  2. There is no known efficient attack on AES.
  3. Finding the private key from public key is difficult if factorization is difficult.

- Why we don’t commit ourselves and give a definite statement? Like
  - SHA3 is collision resistant!!!
  - It is difficult to find the private key!!!!

- Security of many crypto primitives relies on the assumption that one-way function exists. Unfortunately, there is no known mathematical proof on the existence of one-way functions. Nevertheless, many people believe that indeed one-way function exists. So, instead of claiming

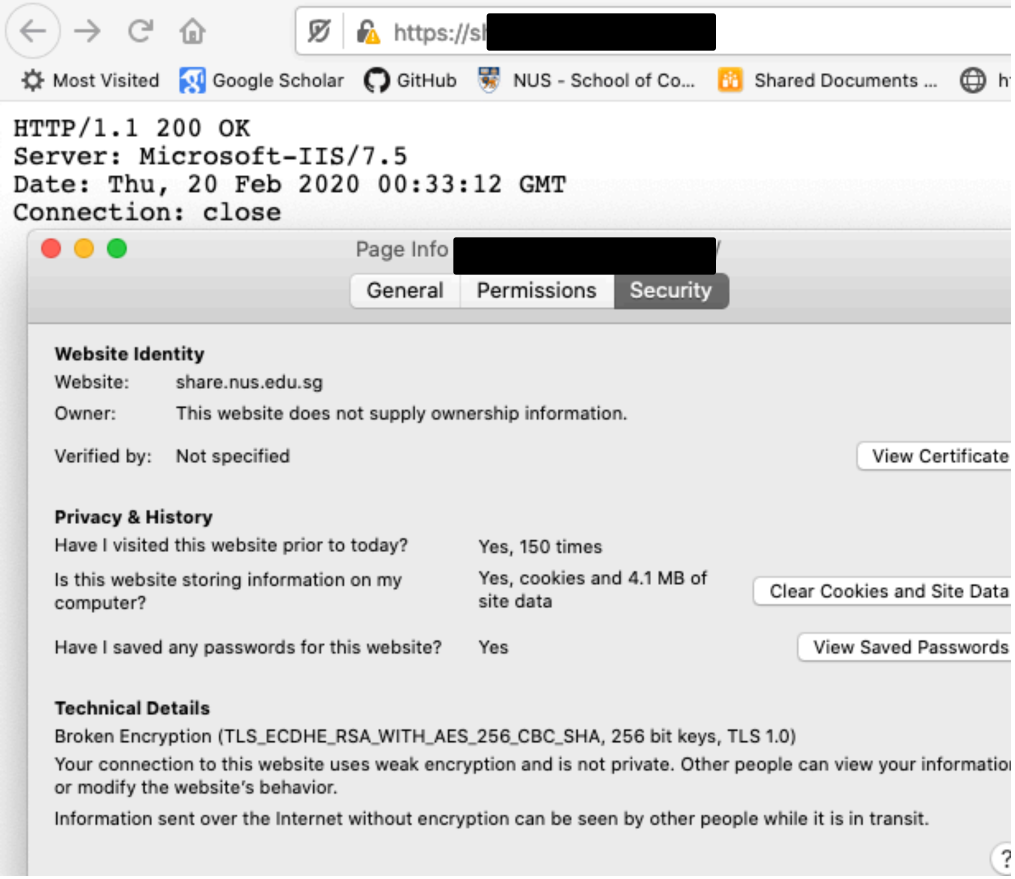
  “SHA3 is collision resistant”,

  a mathematically rigorous claim is

  “SHA3 is believed to be collision resistant”.

Backup slides

TLS 1.0



```
HTTP/1.1 200 OK
Server: Microsoft-IIS/7.5
Date: Thu, 20 Feb 2020 00:33:12 GMT
Connection: close
```

Website Identity

Website: share.nus.edu.sg

Owner: This website does not supply ownership information.

Verified by: Not specified

Privacy & History

Have I visited this website prior to today? Yes, 150 times

Is this website storing information on my computer? Yes, cookies and 4.1 MB of site data

Have I saved any passwords for this website? Yes

Technical Details

Broken Encryption (TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, 256 bit keys, TLS 1.0)

Your connection to this website uses weak encryption and is not private. Other people can view your informatio or modify the website's behavior.

Information sent over the Internet without encryption can be seen by other people while it is in transit.