

Lecture 5: Secure Channel, TLS/SSL & Miscellaneous Cryptography Topics

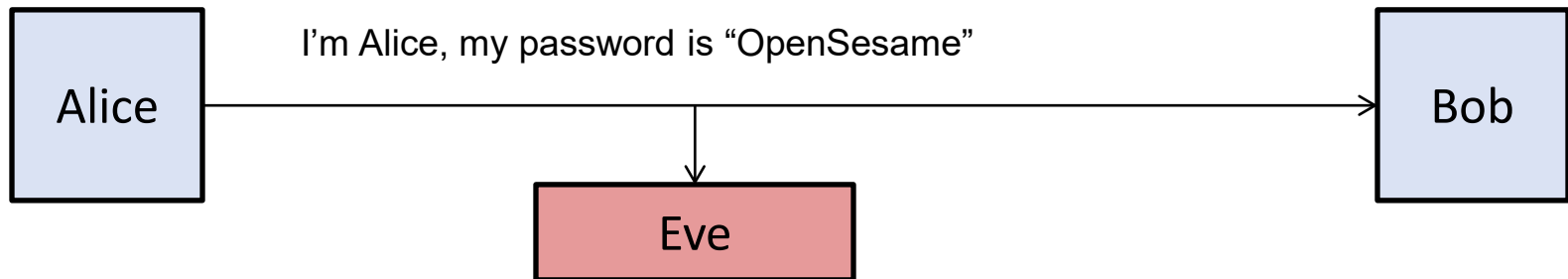
- 5.1 Strong authentication
- 5.2 Key exchange & authenticated key exchange
- 5.3 Putting all together: Securing a communication channel
- 5.4 Putting all together: TLS/SSL
- 5.5 Authenticated encryption
- 5.6 Time-memory tradeoff for dictionary attack
- 5.7 Birthday attack variant
- 5.8 Other interesting cryptography topics
- 5.9 Summary of cryptography

5.1 Strong Authentication

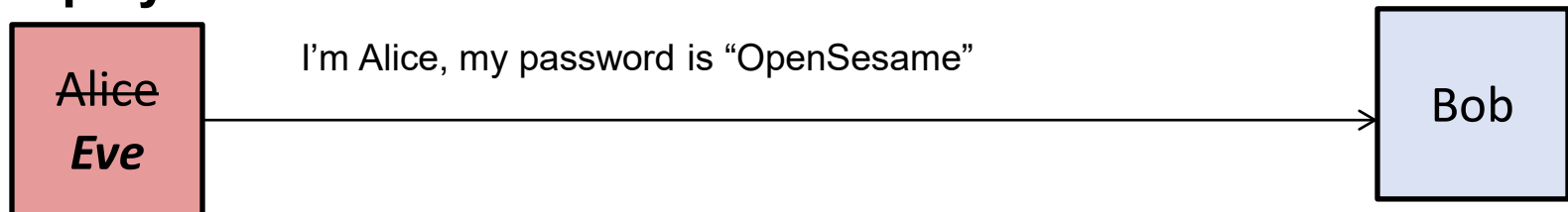
Password and Weak Authentication

- Password is a **weak** authentication system:
an eavesdropper can get the password, and *replay* it

Eve **eavesdrops**:



Eve **replays**:



- The password: the secret **shared** and **exchanged** between Alice & Bob
- 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 key k , and both of them agree on an **encryption scheme**, say AES

(1) Alice sends to Bob a hello message:

“Hi, I’m Alice”

(2) (**Challenge**) Bob randomly picks m , and sends to Alice:

$$y = E_k(m)$$

(3) (**Response**) Alice decrypts y to get m , and then sends m to Bob

(4) Bob verifies that the message received is indeed m
If so, accepts;
otherwise rejects

Strong Authentication: SKC-based 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?

Strong Authentication: PKC-based Challenge-Response

Suppose Alice wants to authenticate herself to Bob using PKC

(1) Alice sends to Bob a hello message:

“Hi, I’m Alice”

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

“Alice, here is your challenge”, r

(2) (**Response**) Alice uses her **private key** to sign r , and sends to Bob:
sign(r), Alice’s certificate

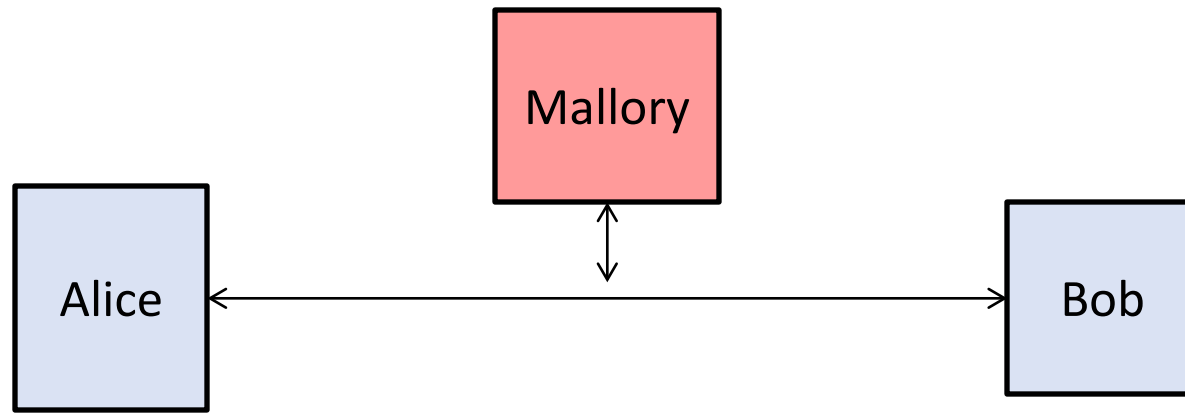
(3) Bob verifies Alice’s certificate, extracts Alice’s public key from the certificate, and then verifies that the **signature is correct**

Strong Authentication: PKC-based Challenge-Response (Analysis)

- An eavesdropper can't know/derive **Alice's private key** and replay the response because the challenge r is likely to be different
- The value r is also known as the ***cryptographic nonce*** (or simply ***nonce***)
- ***Question:*** Which component in the above ensures freshness?
- **Remarks:**
 - The shown protocols have omitted many details
 - Designing a secure authentication protocol is **not easy**

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
- Subsequently Mallory pretends to be Alice! (*Duh*)

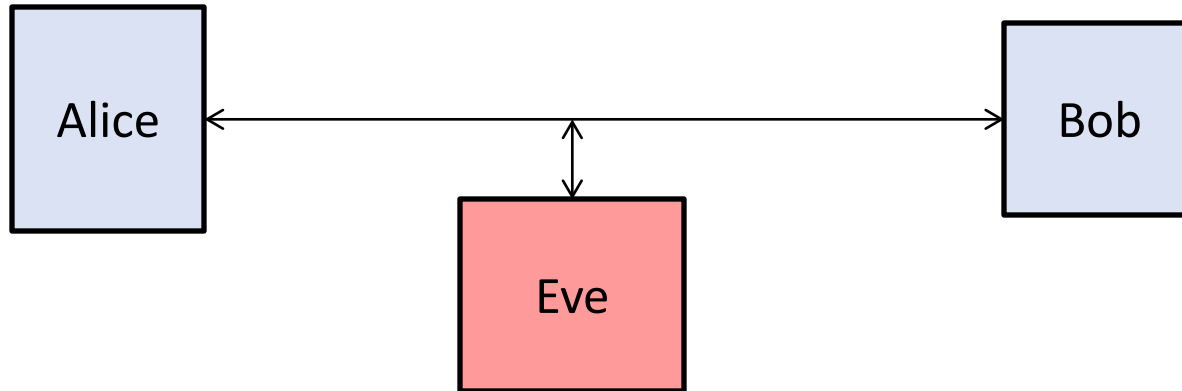
Is Authentication Alone Sufficient?

- 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 must be:
a new secret ***k*** (a.k.a. ***session key***) established by Alice & Bob
- The process of establishing a secret between Alice & Bob is called ***key exchange***, ***key agreement*** or ***key establishment***
- Subsequent communication between Alice & Bob must be secure using the session key

5.2 Key Exchange & Authenticated Key Exchange

Basic/Unauthenticated Key Exchange (No Authentication)

- Alice & Bob want to establish a common key: both are assumed to be *authentic*
- The channel could be eavesdropped by *Eve*

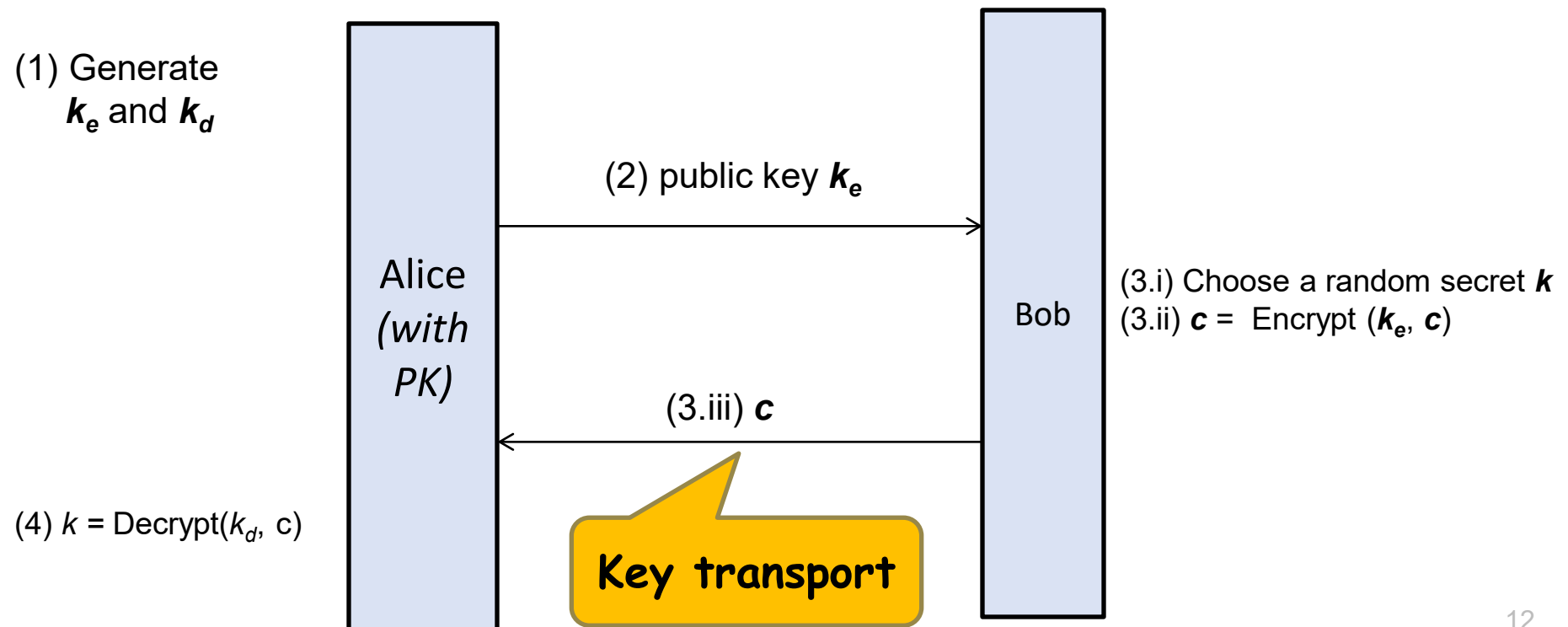


- They want a **key exchange protocol**, such that Eve is unable to **extract any information** of the established key:
the established key can be used to protect (e.g. via cipher, MAC)
subsequent communication between Alice & Bob

*Note: Here we only consider **Eve** who can sniff,
and not the malicious Mallory who can modify the communication!*

PKC-based *Unauthenticated* Key Exchange

1. Alice generates a pair of private/public key
2. Alice sends the public key k_e to Bob
3. Bob carries out the following:
 - i. Randomly choose a secret k
 - ii. Encrypt k using k_e
 - iii. Send the ciphertext c to Alice
4. Alice uses her private key k_d to decrypt and obtain k .



Basic/Unauthenticated Diffie-Hellman Key Exchange

- We assume both Alice and Bob have agreed on two **publicly-known** parameters: a **generator** g , and a large (e.g. 1,000-bit) prime p

Step 1-a:

- Randomly choose a
- Compute $x = g^a \pmod{p}$

Step 1-b:

- Randomly choose b
- Compute $y = g^b \pmod{p}$

Step 2-a: $x = g^a \pmod{p}$

Step 2-b: $y = g^b \pmod{p}$

Step 3-a:

- Compute $k = y^a \pmod{p}$

Step 3-b:

- Compute $k = x^b \pmod{p}$

$$k = g^{ab} \pmod{p}$$

$$k = g^{ab} \pmod{p}$$

**Key
agreement**

Remark:

Steps 1-a & 1-b, Steps 2-a & 2-b, and Steps 3-a & 3-b can be carried out **in parallel**

Diffie-Hellman Key Exchange: Computational Analysis

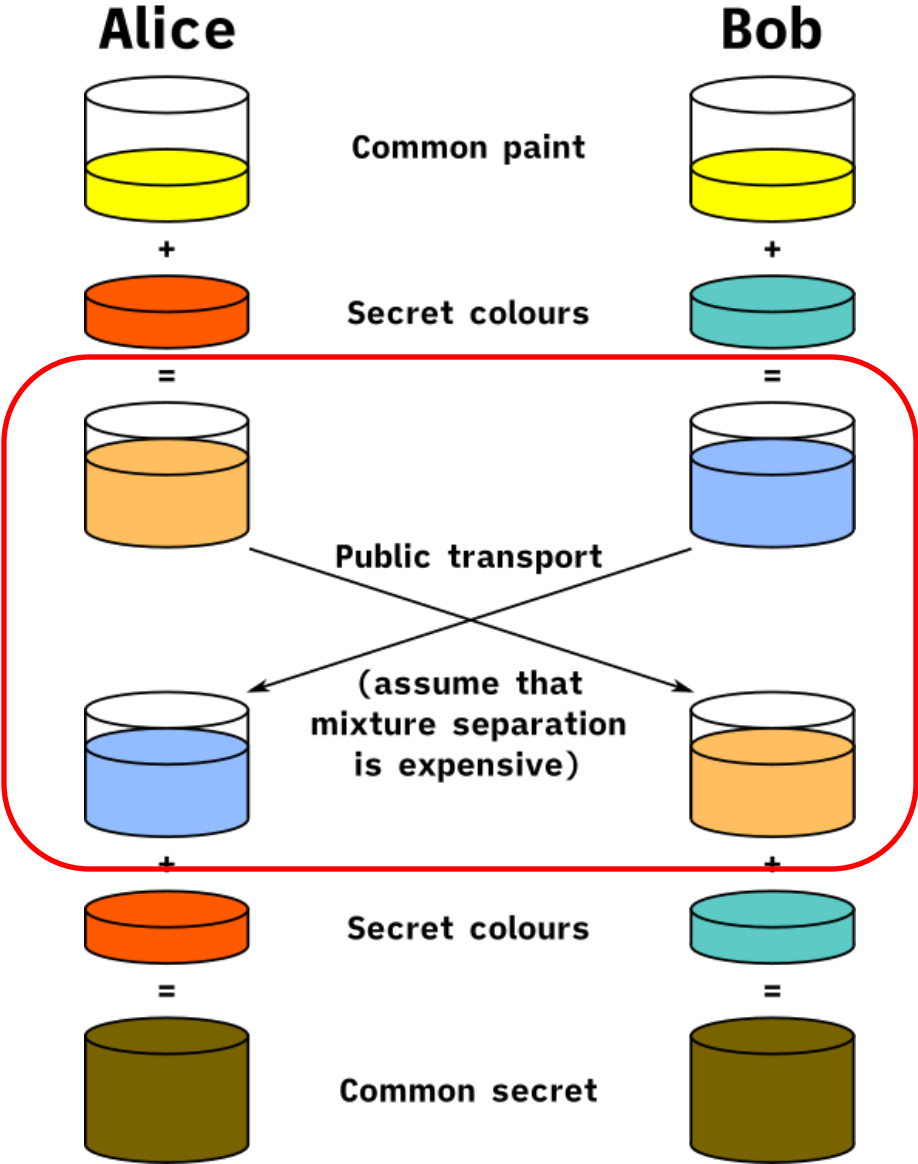
Optional

- Hard problems in **RSA**:
 - **Factoring** problem: given $n=p.q$; find p and q
 - **RSA** problem: given n , e , and $c = m^e \pmod{n}$; find m
- **Discrete Log (DL)** problem:
 - A **cyclic group**: (optional https://en.wikipedia.org/wiki/Cyclic_group)
 - Let g be the **generator** of the cyclic group:
 g can derive all other elements in the set, e.g. by using exponentiation
 - Given n , g , and $x = g^e \pmod{p}$; find e
- **Computational Diffie-Hellman (CDH)** problem:
 - Given p , g , $x=g^a \pmod{p}$, $y=g^b \pmod{p}$; find $k=g^{ab} \pmod{p}$
- Diffie-Hellman key exchange works by assuming **Discrete Log** and **Computational Diffie-Hellman** are computationally hard

Remarks:

1. The assumption seems self-fulfilling. Nonetheless, there are much evidence that it holds.
2. The “exponentiation” operation can be applied to any algebraic group, i.e. not necessary integers.
Note that 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 is believed to hold.

Diffie-Hellman Key Exchange: Analogy/Visualization



Source: Wikipedia

Diffie-Hellman Key Exchange: Example

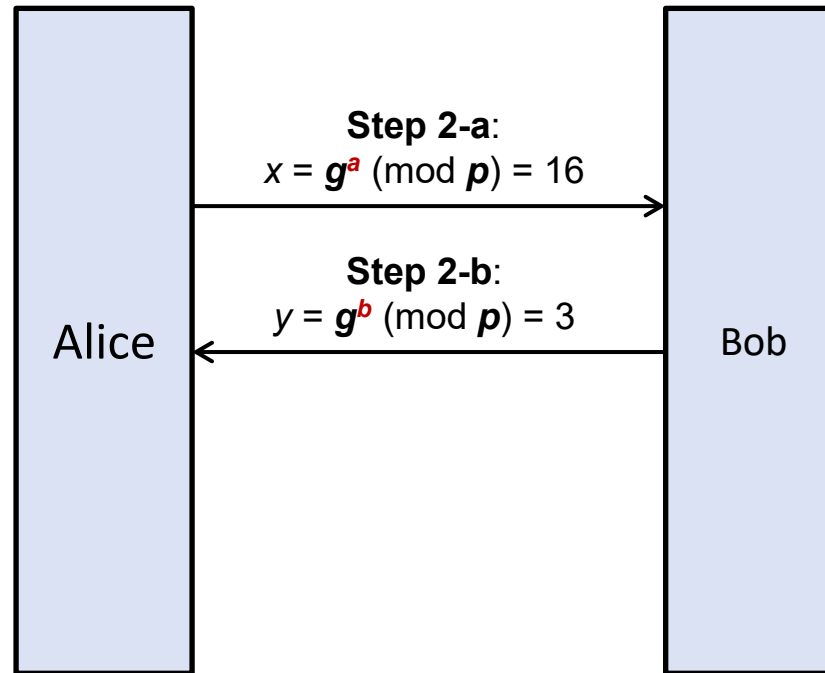
- Suppose $g = 2$, and $p = 23$

Step 1-a:

- Randomly choose $a=15$

Step 3-a:

- Compute
$$k = y^a \pmod{p}$$
$$= 3^{15} \pmod{23}$$
$$= 12$$



Step 1-b:

- Randomly choose $b=8$

Step 3-b:

- Compute
$$k = x^b \pmod{p}$$
$$= 16^8 \pmod{23}$$
$$= 12$$

Remark (optional):

DH key exchange achieves additional security property called **forward secrecy (FS)**:

https://en.wikipedia.org/wiki/Forward_secretcy

Diffie-Hellman Key Exchange: Another Example

- From Wikipedia:
 $g = 5$, $p = 23$, $a=6$, $b=15$, and the key $s=2$

| Alice | | Bob | | Eve | |
|-------------------------|---------|----------------------------|---------|-----------------|---------|
| Known | Unknown | Known | Unknown | Known | Unknown |
| $p = 23$ | | $p = 23$ | | $p = 23$ | |
| $g = 5$ | | $g = 5$ | | $g = 5$ | |
| $a = 6$ | b | $b = 15$ | a | | a, b |
| $A = 5^a \bmod 23$ | | $B = 5^b \bmod 23$ | | | |
| $A = 5^6 \bmod 23 = 8$ | | $B = 5^{15} \bmod 23 = 19$ | | | |
| $B = 19$ | | $A = 8$ | | $A = 8, B = 19$ | |
| $s = B^a \bmod 23$ | | $s = A^b \bmod 23$ | | | |
| $s = 19^6 \bmod 23 = 2$ | | $s = 8^{15} \bmod 23 = 2$ | | | s |

Source: Wikipedia

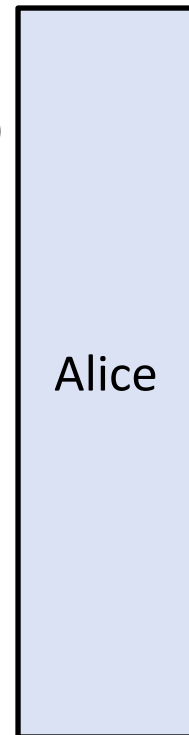
Is *Basic/Unauthenticated* DH Key Exchange Secure?

- Now, assume the presence of **Mallory**



Step 1-a:

- Randomly choose ***a***
- Compute $x = g^a \pmod{p}$



Step 2-a: $x = g^a \pmod{p}$

Step 2-a':
 $x' = g^c \pmod{p}$

Step 2-b:
 $y = g^b \pmod{p} = 3$

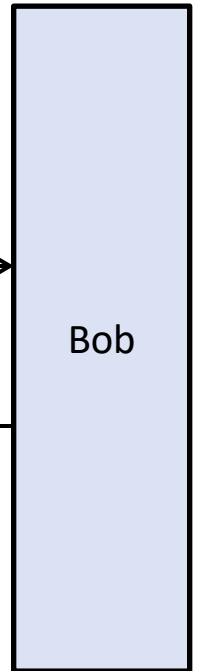
Step 2-b':
 $y' = g^d \pmod{p}$

Step 3-a:

- Compute
 $k_1 = y'^a \pmod{p}$
 $= g^{da} \pmod{p}$

Step 3-b:

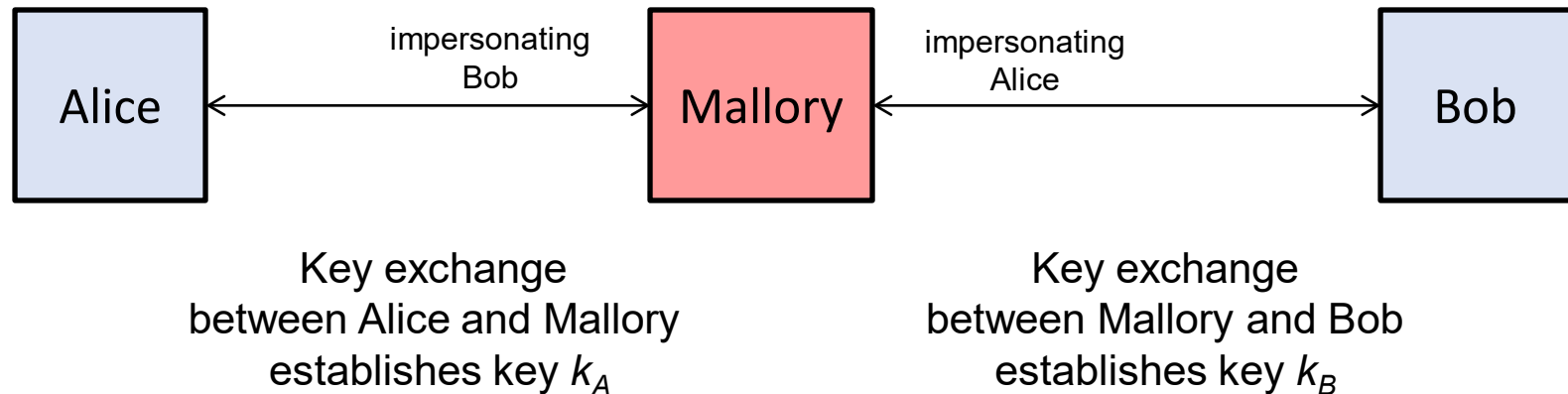
- Compute
 $k_2 = x'^b \pmod{p}$
 $= g^{cb} \pmod{p}$



From the attack: DH does ***not*** achieve entity authentication

Is *Basic/Unauthenticated* Key Exchange Secure?

- With **Mallory** as a man-in-the-middle



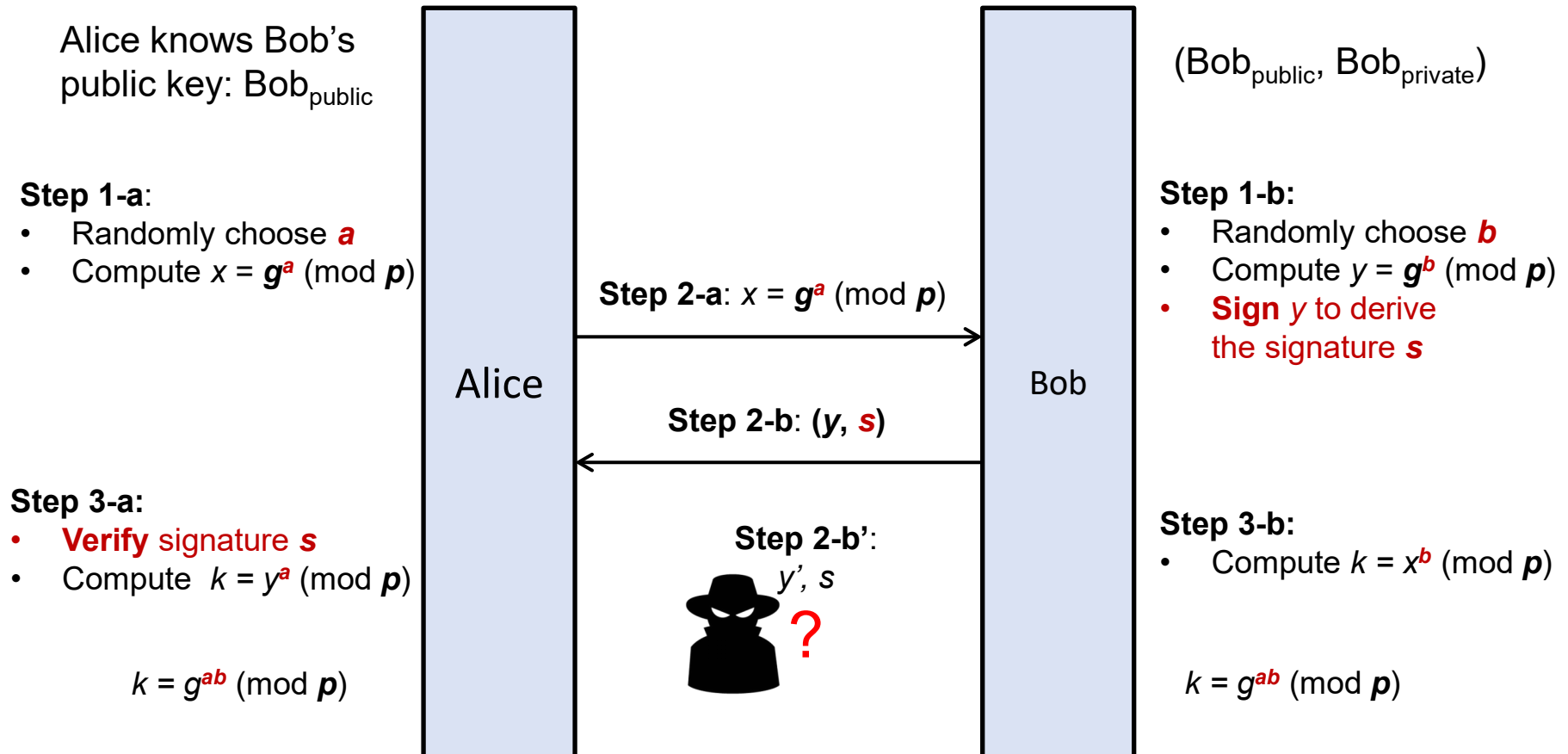
- In this case, Alice mistakes Mallory for Bob
- Since communication from Alice is encrypted using k_A , Mallory **can decrypt** it using k_A and re-encrypt using k_B
- Hence, Mallory can **see and modify** the message
- The same attack technique applies to communication from Bob
- (It turns out that **PKC-based authenticated key-exchange** can be easily obtained from existing key-exchange: simply **sign** all communications)

Why is That so?

- *So, what's still wrong really?*
- Think about these 2 **different** goals/ requirements for an entity A in communicating with B :
 - ***Key Secrecy (Confidentiality):***
Is k a “good key” to talk to B ?
 - ***Entity Authenticity:***
Can A be sure that it's really talking to B ?
- **Solution:**
Incorporate the key exchange process with authentication → ***authenticated key exchange (AKE)***

Station-to-Station (STS) Protocol

- Adding **signatures** to DH:
authenticated key-exchange based on DH
- Here, we consider unilateral authentication: Alice want to authenticate Bob



Mutual Authenticated Key Exchange

- The unilateral authentication protocol earlier can be extended to a **mutual authentication**:
make Alice sign her messages in Step 2-a
- **Requirements** for carrying out an authenticated key exchange:
 - For a mutual authentication, Alice and Bob must have a way to know **each other's public key** (e.g. using PKI)
 - For a unilateral authentication, only **one party** needs to have public key
 - After the protocol has completed, **a set of session keys** is established using a **Key Derivation Function (KDF)** like HKDF (based on HMAC) https://en.wikipedia.org/wiki/Key_derivation_function, e.g.:
 - 1 key for encryption & 1 key for MAC;
 - Additionally, 1 key for each direction of encryption/MAC

Summary: Mutual Authenticated Key Exchange

- *Before the protocol:*
 - A has a pair of public/private key pair (A_{public} , A_{private})
 - B has a pair of public/private key pair (B_{public} , B_{private})
 - A knows B public key and vice versa:
these two keys are known as the ***long-term/master key***
- Carry out the authenticated key exchange protocol:
if an entity is not authentic, the other will halt
- *After the protocol:*
 - Both A and B obtain a shared key k , known as the ***session key***
 - Other secret key can be derived from k
- Security requirements:
 - Authenticity-1:
A is assured that it is communicating with an entity who knows B_{private}
 - Authenticity-2:
B is assured that it is communicating with an entity who knows A_{private}
 - Key confidentiality: the attacker is unable to get the session key k

Additional Remarks (Optional)

Password Authenticated Key Exchange (PAKE) *Optional*

- The authenticated key exchange can also be used in **symmetric-key setting**.
In symmetric-key setting, both *A* and *B* share a common secret, typically a **password**. Both conduct authentication + key exchange based on this secret.
- When the shared secret is a password, it is often called ***Password Authenticated Key Exchange*** (PAKE):
 - See https://en.wikipedia.org/wiki/Password-authenticated_key_agreement
 - A crucial difference between human-generated password and machine-generated key: passwords are typically 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 process could take longer time, but it doesn't need to remain connected to the victim.
- A **secure password authenticated key exchange** ensures that even if the passwords are short, the adversary (Eve or Mallory who pretends to be one of the authenticating party) can't carry out offline dictionary attack.
- Note that simply adding **MAC** using passwords cannot prevent offline dictionary attack. Since MAC is deterministic, Eve, after capturing the MAC, can test it on the dictionary in offline. So, the design of password authenticated key exchange is more complicated. Details are omitted.

Password Authenticated Key Exchange (PAKE)

Optional

- Some examples:
 - 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 does NUS WiFi employ?*

In our previous example of an attacker spoofing a fake NUS hotspot in NUH bus stop, can the attacker successfully steal password?

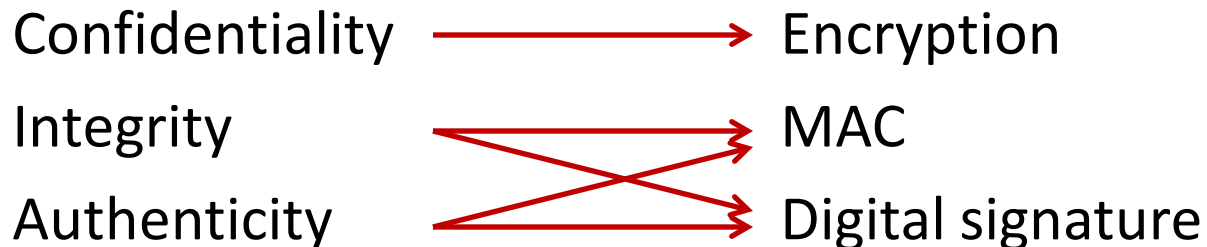
(See NUS WiFi setup instruction at:

<http://www.nus.edu.sg/comcen/gethelp/guide/itcare/wireless/NUS-WPA2%20Network%20Configuration%20Guide%20for%20Android%207.0.pdf>)

5.3 Putting All Together: Securing a Communication Channel

Secure Communication Channel Problem

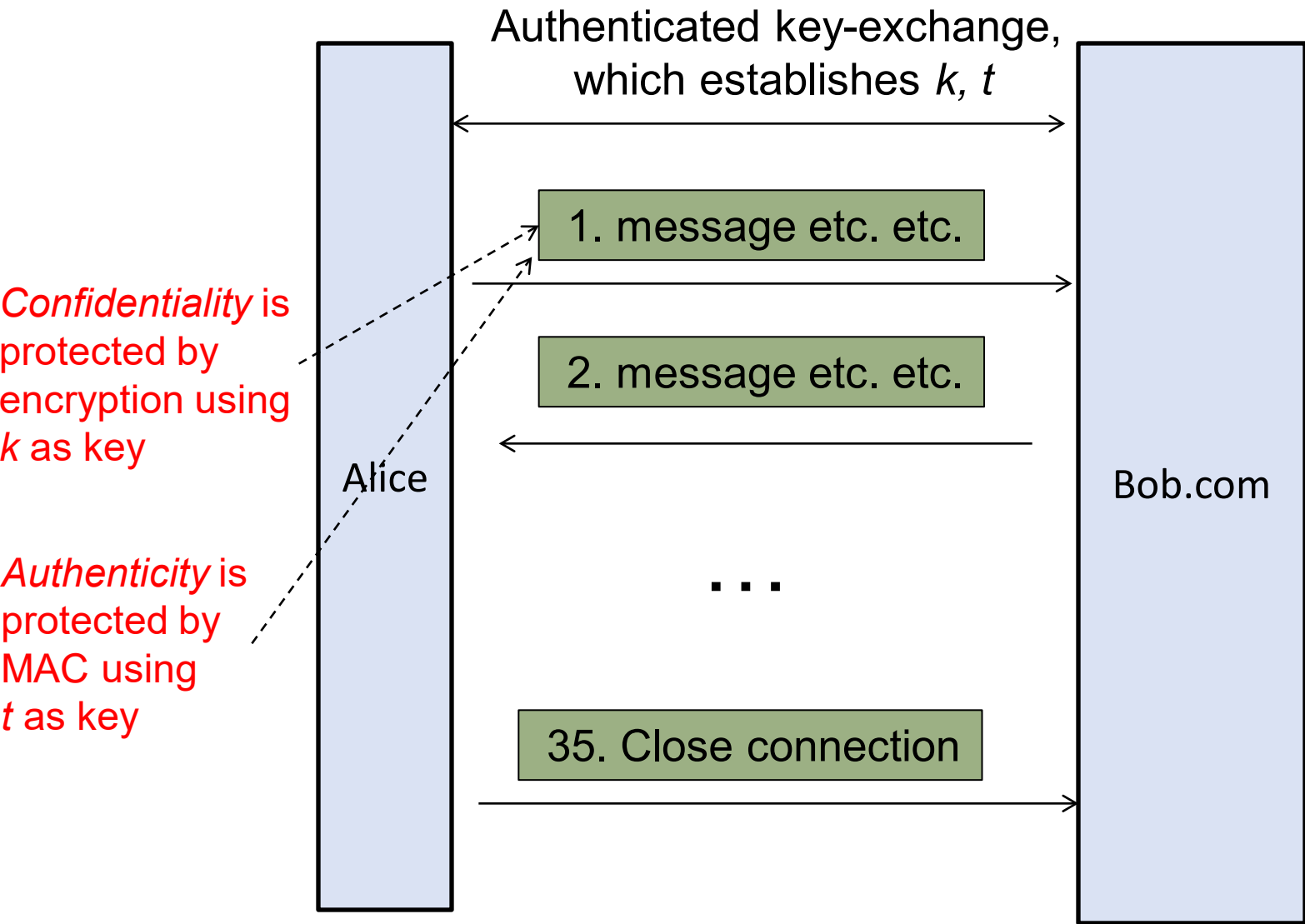
- Consider a communication channel that is subjected to **sniffing and spoofing**: does this reminds us of the Internet?
- **Question**: How can we securely communicate over it using cryptographic primitives?
- A ***“secure channel”***:
establishes, between 2 programs, a data channel that has **confidentiality, integrity, authenticity** against a computationally-bounded network attacker (i.e. Mallory)
- *Question: what cryptographic primitives to use?*



Secure Communication Channel Problem

- A common **example**:
Imagine that Alice wants to visit a website Bob.com.
- **Question**: How to protect the **authenticity** (that Bob.com is authentic), and **confidentiality** & **integrity** of the communication?
- **Answer**: we need to establish a **secure channel** between Alice (i.e. her browser) and Bob.com (i.e. its web server)
- We have discussed some important necessary mechanisms: authenticated key exchange, PKI, encryption, ... (*message-ordering protection?*)
- Let's now see how we can establish a secure channel

Secure Channel between Alice and Bob.com



Question: Why do we need the sequence number?

Secure Channel between Alice and Bob.com: The Steps

(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*** k , t

where: k is the secret key of a symmetric-key encryption,
e.g. AES

t is the secret key of a MAC

*(Details of how k and t can be established are omitted here:
see, for instance, station-to-station protocol for details)*

Secure Channel between Alice and Bob.com: The Steps

(Step 2)

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

Suppose m_1, m_2, m_3, \dots are the sequence of message exchanged, the **actual data** to be sent for m_i will be:

$$E_k (i || m_i) || MAC_t (E_k (i || m_i))$$

where: i is the sequence number,

$||$ refers to concatenation

Note: The technique above is known as “**encrypt-then-MAC**”.

There are other variants of **authenticated encryption** called “MAC-then-encrypt” and “MAC-and-encrypt”: *see later in this lecture*

Secure Channel and PKI Usage

- 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 (a case of unilateral authentication) :
 - 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

5.4 Putting All Together: TLS/SSL

HTTPS & TLS/SSL Protocols

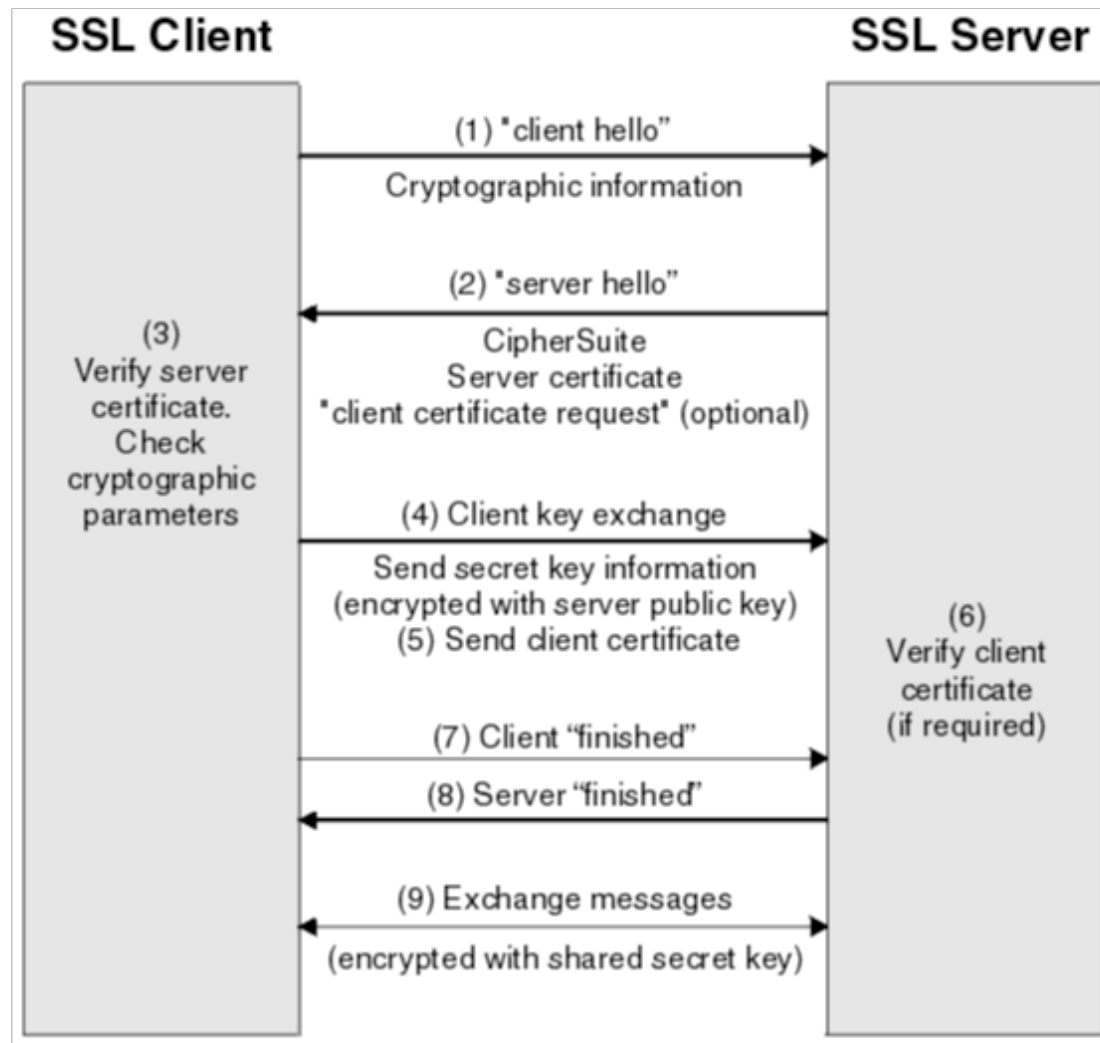
- **HTTPS** (HTTP Secure) is widely used to secure Web traffic
- HTTPS is built on top of SSL/TLS: **HTTPS = HTTP + SSL**
Hence, HTTPS is also called: HTTP over SSL,
or HTTP over TLS
- Transport Layer Security (**TLS**) is a protocol to secure communication using cryptographic means:
TLS 1.2 [2008], 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 **establish a secure communication channel**
- TLS/SSL sits in between TCP Transport & Application layers

TLS Protocol (Suite)

- How does TLS work **at the high level**?
 1. **Ciphers negotiation**
 2. **Authenticated Key Exchange**: the exchange of session key, which also authenticates the identities of parties involved
 3. Symmetric-key based **secure communication**
 4. **Re-negotiation** (if needed)
- Two (sub) protocols (see <https://docs.microsoft.com/en-us/windows/win32/secauthn/transport-layer-security-protocol>):
 - **Handshake Protocol**: for 1, 2, 4 above; also uses the Record Protocol
 - **Record Protocol**: for 3; a lower-layered protocol

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)

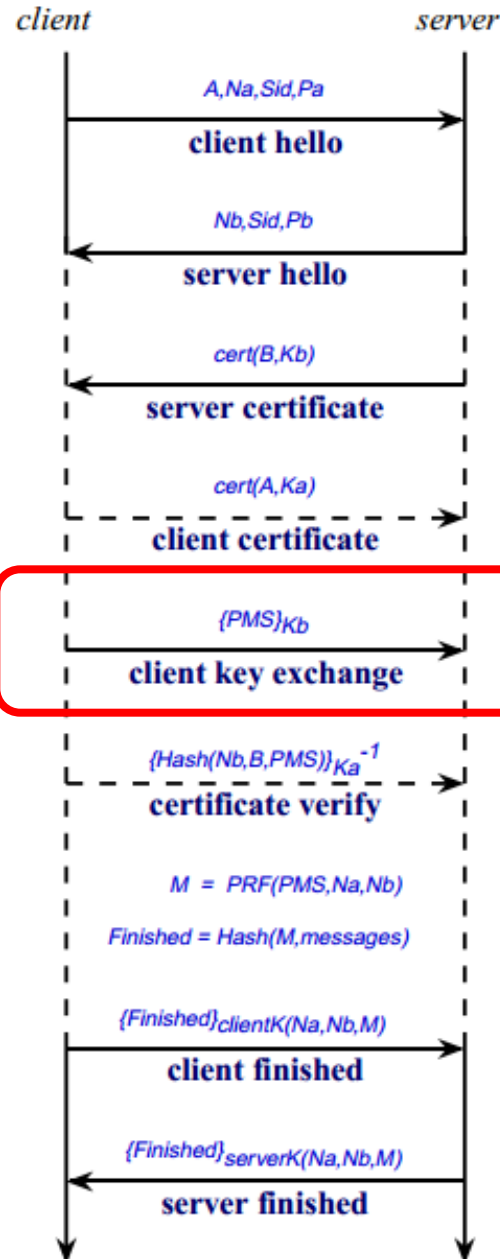


https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

TLS Handshake (Ciphers Negotiation & Authenticated Key Exchange)

Optional

PMS can be derived using DH KE too



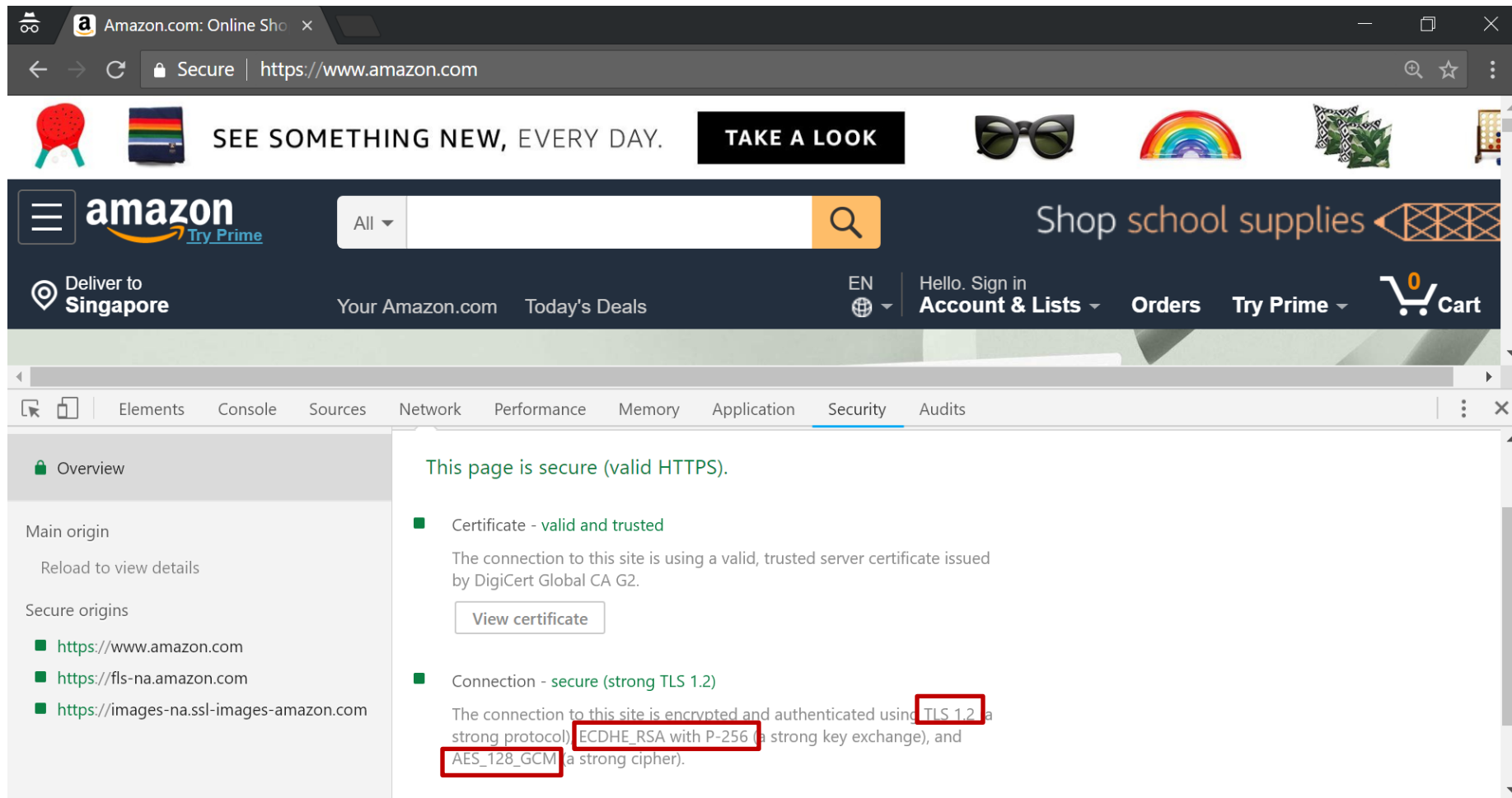
Notes on notation:

- Na, Nb : nonces from A & B, respectively
- Sid : session ID
- Pa, Pb : a set of preferences from A & B, respectively
- **PMS: Pre-Master Secret**
- PRF: Pseudo Random Function
- **M: Master secret**

See:

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

HTTPS Protocol in Action: An Example



The screenshot shows a web browser displaying the Amazon.com homepage. The address bar indicates a secure connection to <https://www.amazon.com>. The browser's developer tools are open, with the 'Security' tab selected. The security overview shows that the page is secure (valid HTTPS) and provides details about the certificate and connection.

Security Overview:

- This page is secure (valid HTTPS).**
- Certificate - valid and trusted**
The connection to this site is using a valid, trusted server certificate issued by DigiCert Global CA G2.
[View certificate](#)
- Connection - secure (strong TLS 1.2)**
The connection to this site is encrypted and authenticated using **TLS 1.2** (a strong protocol), **ECDHE_RSA with P-256** (a strong key exchange), and **AES_128_GCM** (a strong cipher).

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

Additional Remarks on TLS (Optional)

Optional

- Previous **attacks** on vulnerable TLS implementations:
 - TLS 1.2 supports AES in CBC mode: vulnerable to padding oracle attack
 - Some attacks: Heartbleed (“buffer over-read”), BEAST, CRIME, POODLE
- The **latest TLS 1.3** version:
 - Ditches **insecure** schemes:
e.g. MD5, SHA-1, RC4, AES in CBC mode
 - Ditches **unnecessary** (legacy) features:
e.g. optional data compression, which enables the CRIME attack
 - Uses state-of-the-art **ciphers**:
AES-GCM, AES-CCM, ChaCha20+Poly1305 authenticated ciphers
 - Adds useful protection mechanisms:
e.g. downgrade protection
 - Performance improvement:
e.g. session resumption

Side Remark: What is a Protocol?

Protocol Definition and Example

- Slides 4, 6, 12, 13, and 21 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:

