Change Log (v1):
- Slide 1 (added 4.8 forward secrecy)
- Added section 4.8 as the last few slides.
- Slide 24, 25: Major changes. Switching the role of Alice and Bob Slide 25. Changing notations for consistency.

Topic 4: PKI + Channel Security

Part II: Channel Security

4.4 Authentication Protocols

Main difference of data-origin authentication vs communication authentication: “***Freshness***”

- A piece of authentic data remains authentic over time. In communication, we want to verify authenticity of the interacting entity at that point of time.

3 subtly different threat models with 3 different protocols.

- Authentication
- Key-exchange
- Authenticated key-exchange

Summary & takeaways

Part 2: Channel security

- Protocols: (basic) Authentication, Key Exchange, Authenticated Key Exchange.
  - Authentication. *Adversary extract information and later impersonate*. (Assuming each entity remains the same throughout each session). Unilateral, Mutual.
  - Key exchange: *Adversary sniffs and wants to steal the session key*. No authentication. (PKC, DH)
  - Authenticated key exchange. *Adversary is Mallory* (can sniff, spoof, modify, and thus can take over session) and wants to impersonate and/or steal the session key.

- Putting all together. With crypto primitives, we can obtain a secure *(w.r.t. authenticity & confidentiality, and against Mallory)* channel on top of an underlying unsecure public channel.
  - Method:

    (1) Use ***long-term key*** in ***Authenticated key exchange*** to get a fresh ***session key***

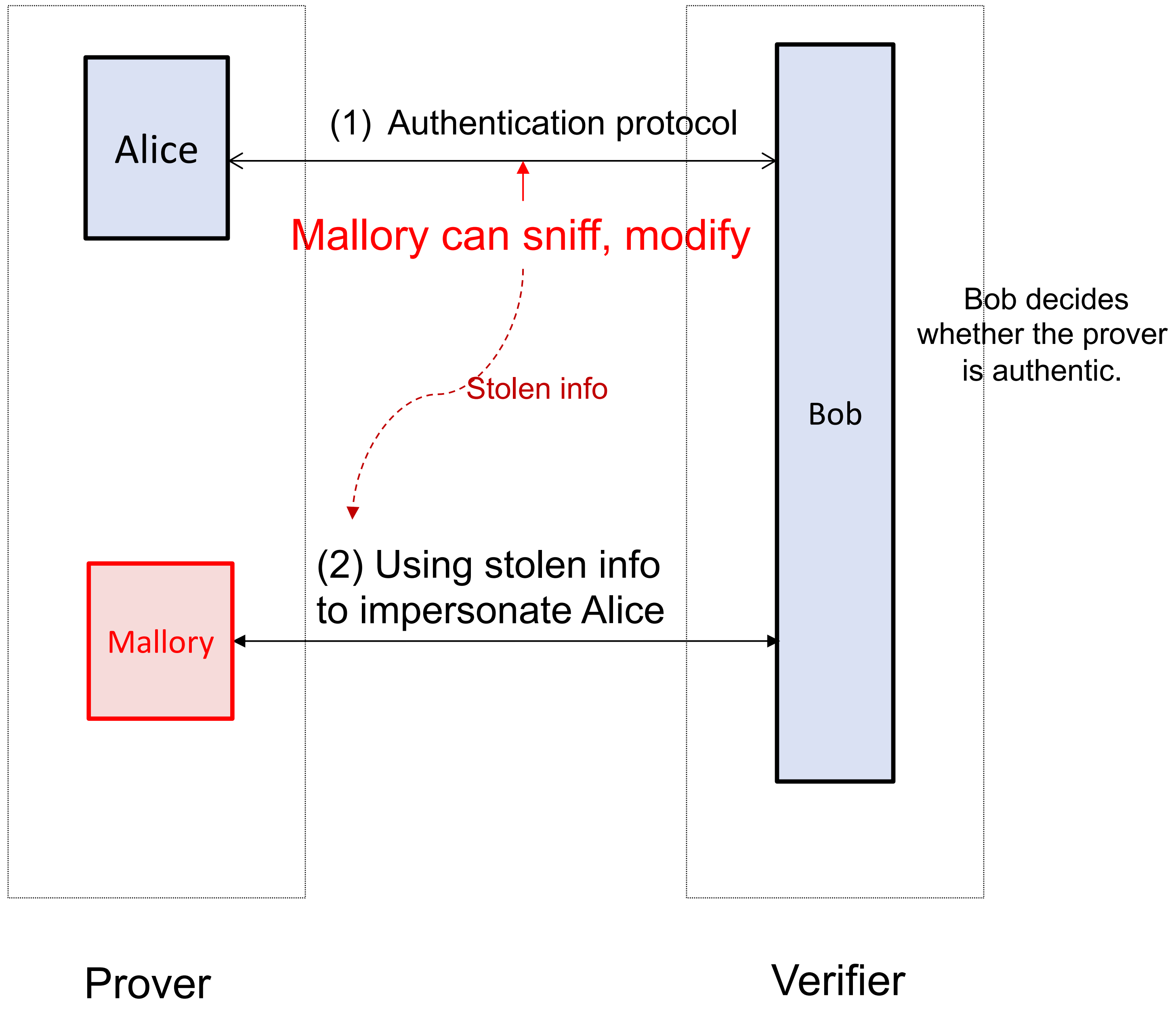
    (2) Use ***session key*** to protect confidentiality (encrypt) & integrity (mac) of subsequent messages via ***authenticated encryption***.

  - Why not just using the long-term key in step (2)?
    - More efficient.
    - Forward secrecy.

Summary: (basic) Authentication

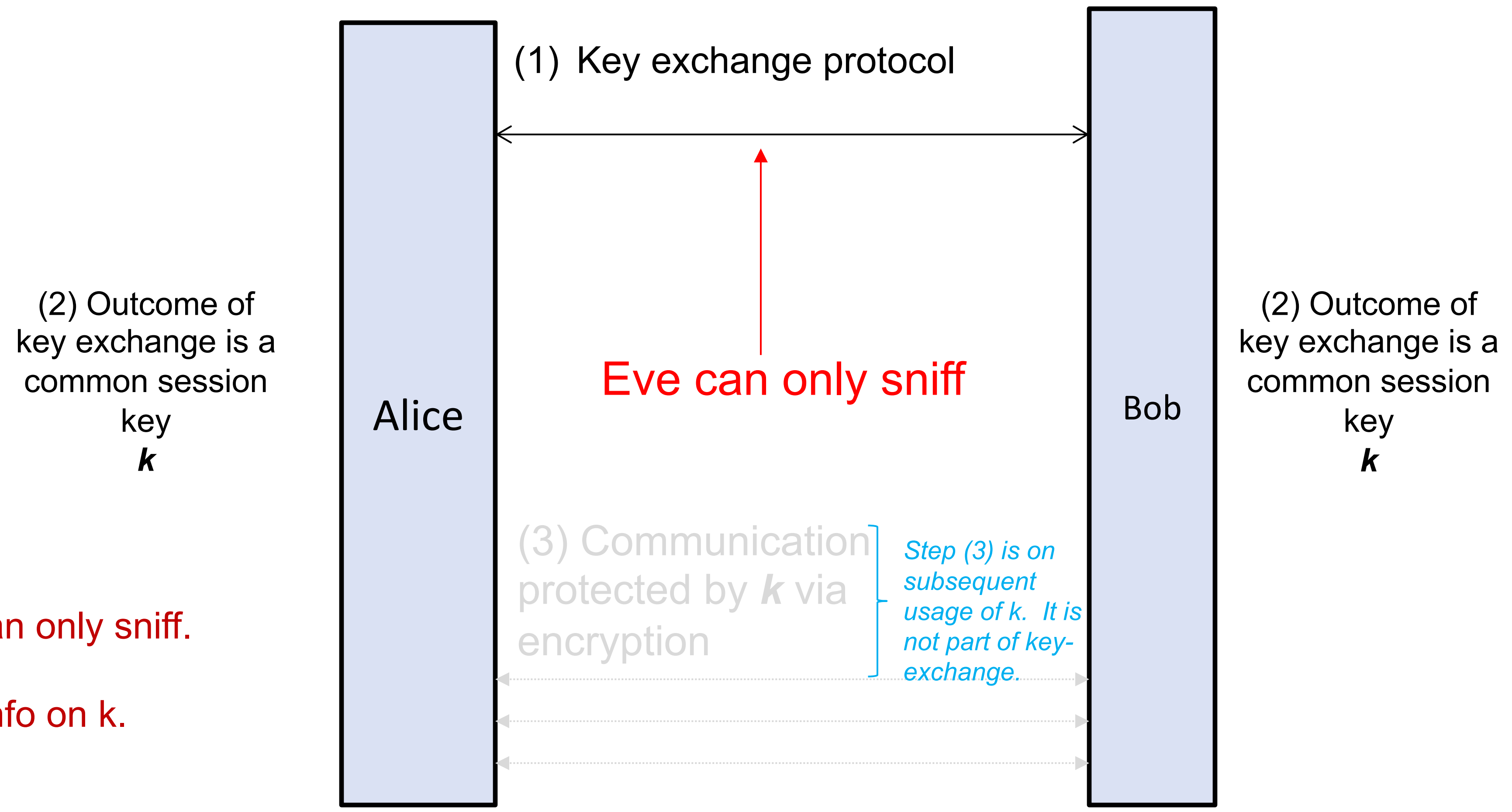
Threat model

- Capability: can be man-in-the-middle during authentication
- Goal: Impersonate Alice in step (2), i.e. trick Bob to accept.


Alice
(1) Authentication protocol
Mallory can sniff, modify
Stolen info
Bob
Bob decides whether the prover is authentic.
(2) Using stolen info to impersonate Alice
Mallory
Prover
Verifier

note: step(1) & (3) cannot be carried out in parallel.

Summary: Key exchange

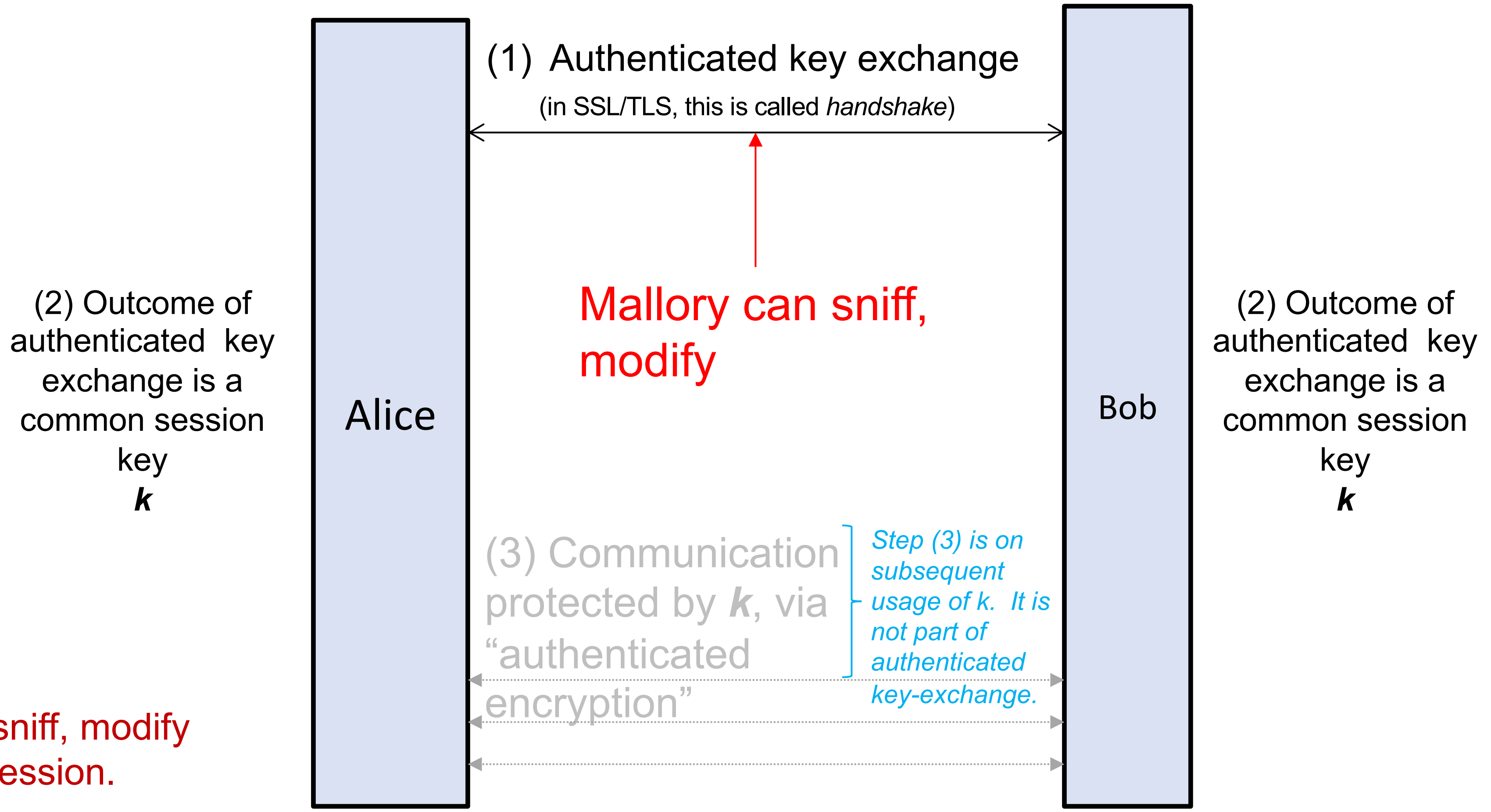

(1) Key exchange protocol
Alice
Bob
Eve can only sniff
(2) Outcome of key exchange is a common session key *k*
(2) Outcome of key exchange is a common session key *k*
(3) Communication protected by *k* via encryption
Step (3) is on subsequent usage of k. It is not part of key-exchange.

Threat model

- Capability: can only sniff.
- Goal: Steal info on k.

It is about confidentiality. Authenticity not considered.

Summary: Authenticated Key-Exchange


(1) Authenticated key exchange
(in SSL/TLS, this is called *handshake*)
Mallory can sniff, modify
(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 "authenticated encryption"
Step (3) is on subsequent usage of k. It is not part of authenticated key-exchange.

Threat model

- Capability: Can sniff, modify throughout the session.
- Goal: Impersonate A/B and/or steal k.

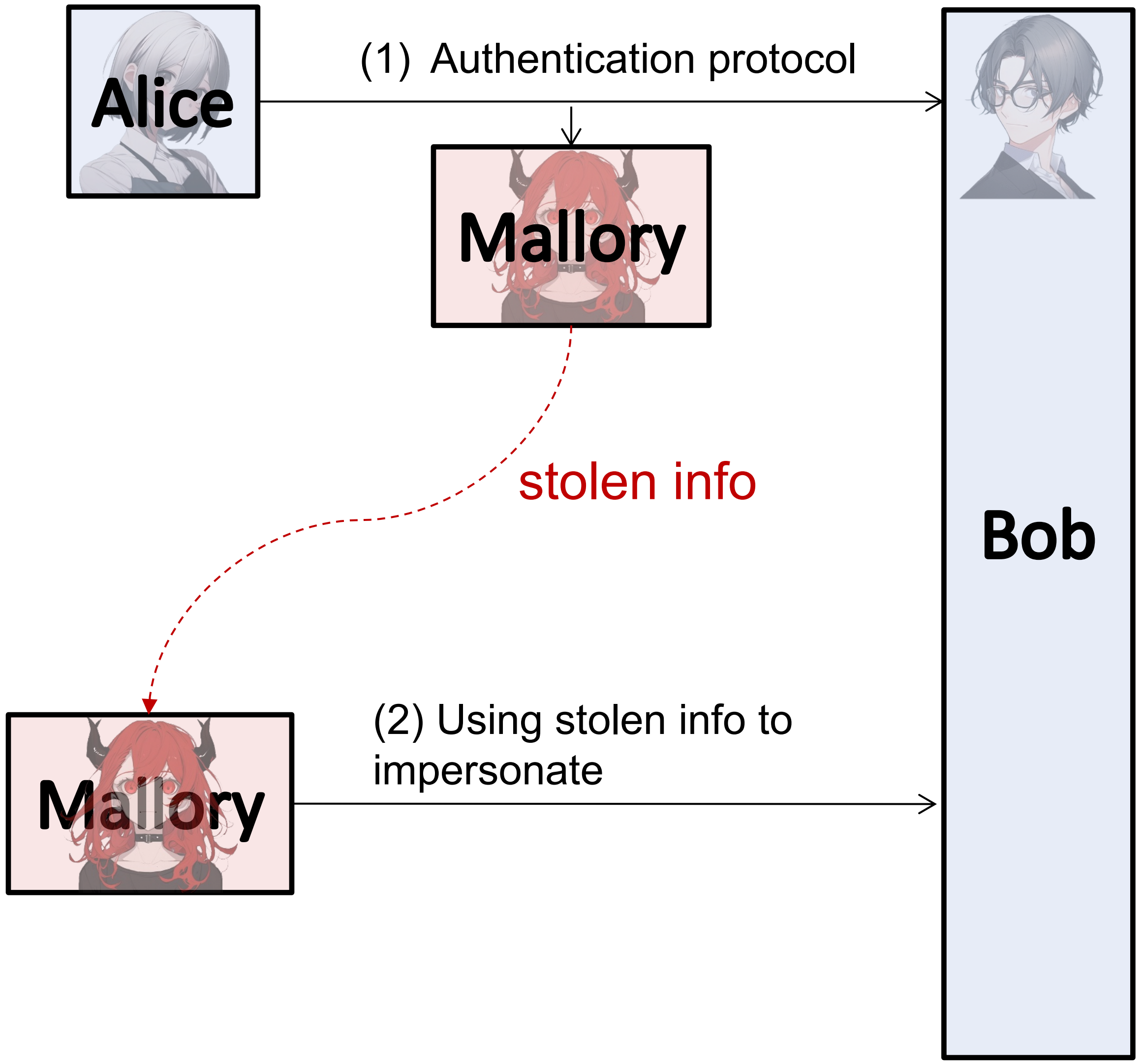
Note: Authenticated Key Exchange is commonly called ***Handshake*** in many standards, e.g. TLS. In this course, to differentiate from non-secure handshake (e.g. TCP handshake), we use the term *authenticated key-exchange.*

(basic) Authentication Protocol

An entity wants to convince Bob that she is indeed Alice. The entity does so by convincing Bob that she knows some “secrets”.

Attack model: attack-then-impersonate

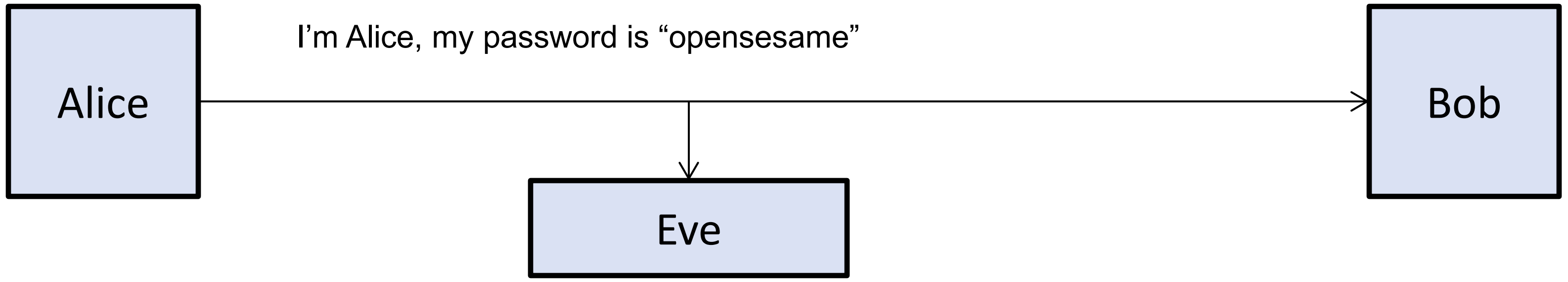
- Mallory can sniff and modify the communication between the authentic Alice and Bob.
- Next, Mallory uses the stolen info to impersonate Alice.


Alice
(1) Authentication protocol
Mallory
stolen info
Bob
Mallory
(2) Using stolen info to impersonate

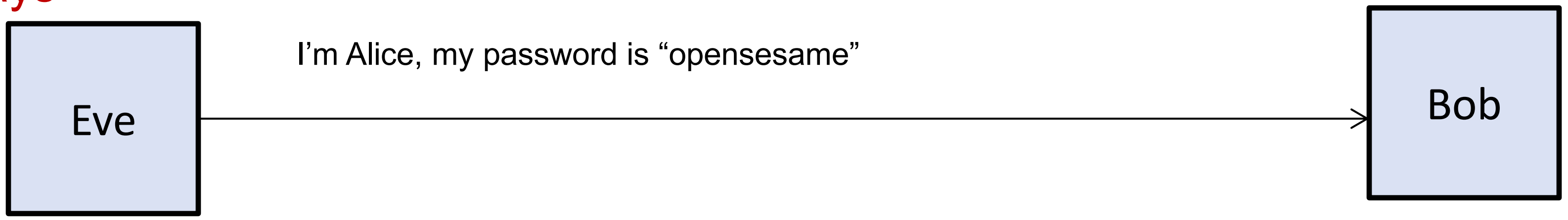
Example of insecure protocol

Sending the authentication credential over is a simple but not secure protocol. Eve can simply “replay” to impersonate.

Eavesdrops



Replays



Note: If the attacker can’t sniff, it is ok to use the above simple method.

To prevent Replay: Challenge-response

Suppose Alice and Bob have a shared secret ***k*****,** and both have agreed on a message authentication code. An entity who knows ***k*** is either Alice or Bob. Now, when an entity **P** wants to convince Bob that she/he is Alice, the following is carried out: (**P** refers to "Prover").

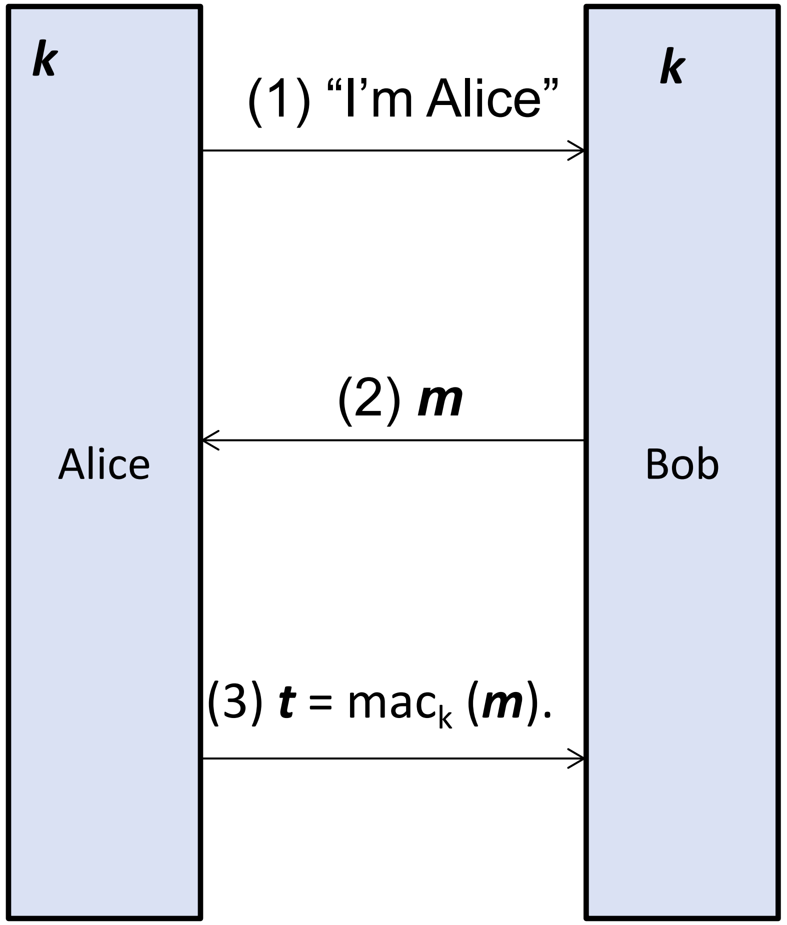
(1) **P** sends to Bob a hello message

"I'm Alice"

(2) (Challenge) Bob randomly picks a message ***m*** and sends ***m*** to **P**.

(3) (Response) **P** computes $t = \text{mac}_k(m)$. **P** sends ***t*** to Bob.

(4) Bob verifies that the tag received is indeed the mac of ***m***. If so, accepts, otherwise rejects.



sketch of proof: Consider an attacker who had sniffed many sessions between Alice and Bob and gathered many pairs of (m,t). After that, the attacker wanted to use the gathered info to convince Bob. Bob would send a challenge m' in step (2). With very high probability, m' was not sent before. Now, by security of mac, the attacker couldn't derive the valid tag of m'.

- By property of mac, even if Eve has sniffed the communication between Alice and Bob, and has obtained multiple pairs of valid ***m***, *t*, Eve still can't forge the mac for messages that Eve has not seen before.

- Eve 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. It is also known as the *cryptographic nonce* (or simply *nonce*).

- This protocol only authenticates Alice. That is, authenticity of Alice is verified. Hence it is called ***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 Bob wants to authenticate an entity **P** who claims to be Alice.

(1) **P** notifies Bob. " I'm Alice". Certificate attached if required.

⟨ "I'm Alice", certificate ⟩

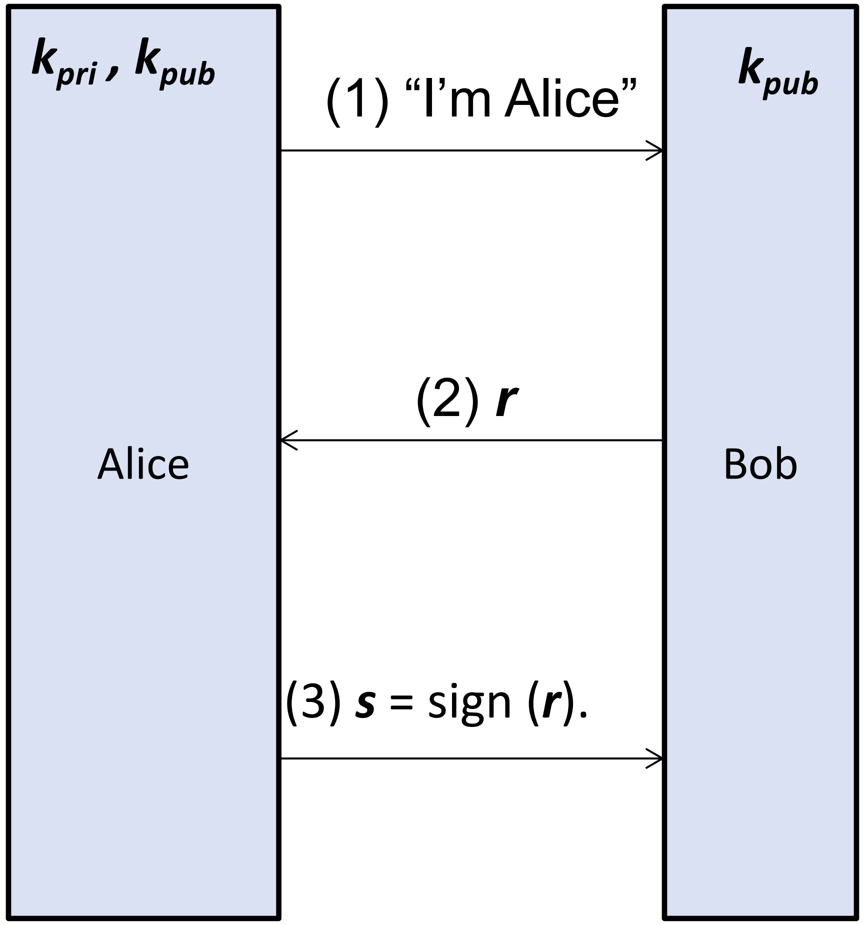
(2) (Challenge) Bob chooses a random message *r* and sends to **P**:

⟨ "here is your challenge", ***r*** ⟩

(3) (Response) **P** uses the private key to sign ***r***.

⟨sign(***r***)⟩

(4) Bob verifies all info (certificate, signature). If valid, accept.

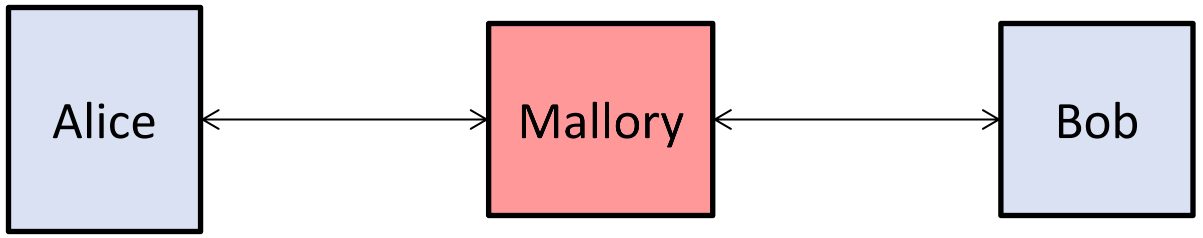


- If Bob already knows Alice's public key, the certificate can be omitted.
- An attacker can observe multiple interactions between the authentic Alice and Bob. By security of signature, the attacker can't forge the response.
- The nonce ***r*** ensure freshness.

We need to handle Mallory and securing subsequent interactions

- The pervious security model assumes the attacker can only attack-and-impersonate. We can use it for applications like door access system: if authentic, door will open. What if in the application, there are follow up communications after authentication?

- For example, consider a Mallory in-between Alice and Bob. That is, Mallory is the MITM. Mallory wants to impersonate Alice. As per assumption, Mallory can sniff, spoof, and modify the message.



Mallory first 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. So, even if we employ the challenge-response in the previous slide, Mallory can still succeed in this setting.

- To protect subsequent interactions, we can use ***authenticated key-exchange.*** The communication consists of two phases:
  - First phase is authentication key-exchange. The outcome is a new shared secret ***k*** known as ***session key.***
  - Subsequently, all communication will be protected (encrypted + mac) using ***k.***

- To understand authentication key-exchange, we first look at another protocol known as Key-exchange (or Key-agreement). In Key exchange, there are two communicating entities Alice and Bob. Alice and Bob want to establish a shared key between them.

- Attack Model:
  - Eve can only sniff.
  - Eve's goal is to steal the established key.

4.5 Key-exchange

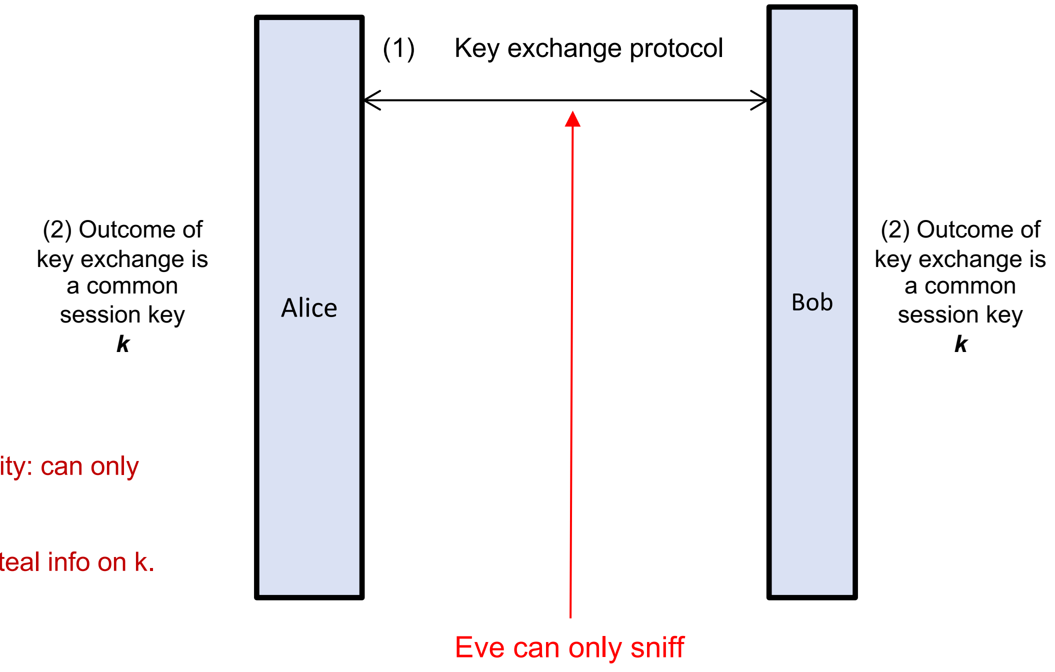
Attack Model

- Capability: Eve.
- Goal: Steal session key.

Two Methods: PKC-based, DH

Threat model

- Capability: can only sniff.
- Goal: Steal info on k.


(1) Key exchange protocol
(2) Outcome of key exchange is a common session key *k*
Alice
Bob
(2) Outcome of key exchange is a common session key *k*
Eve can only sniff

Key-exchange (Eve. No authentication)

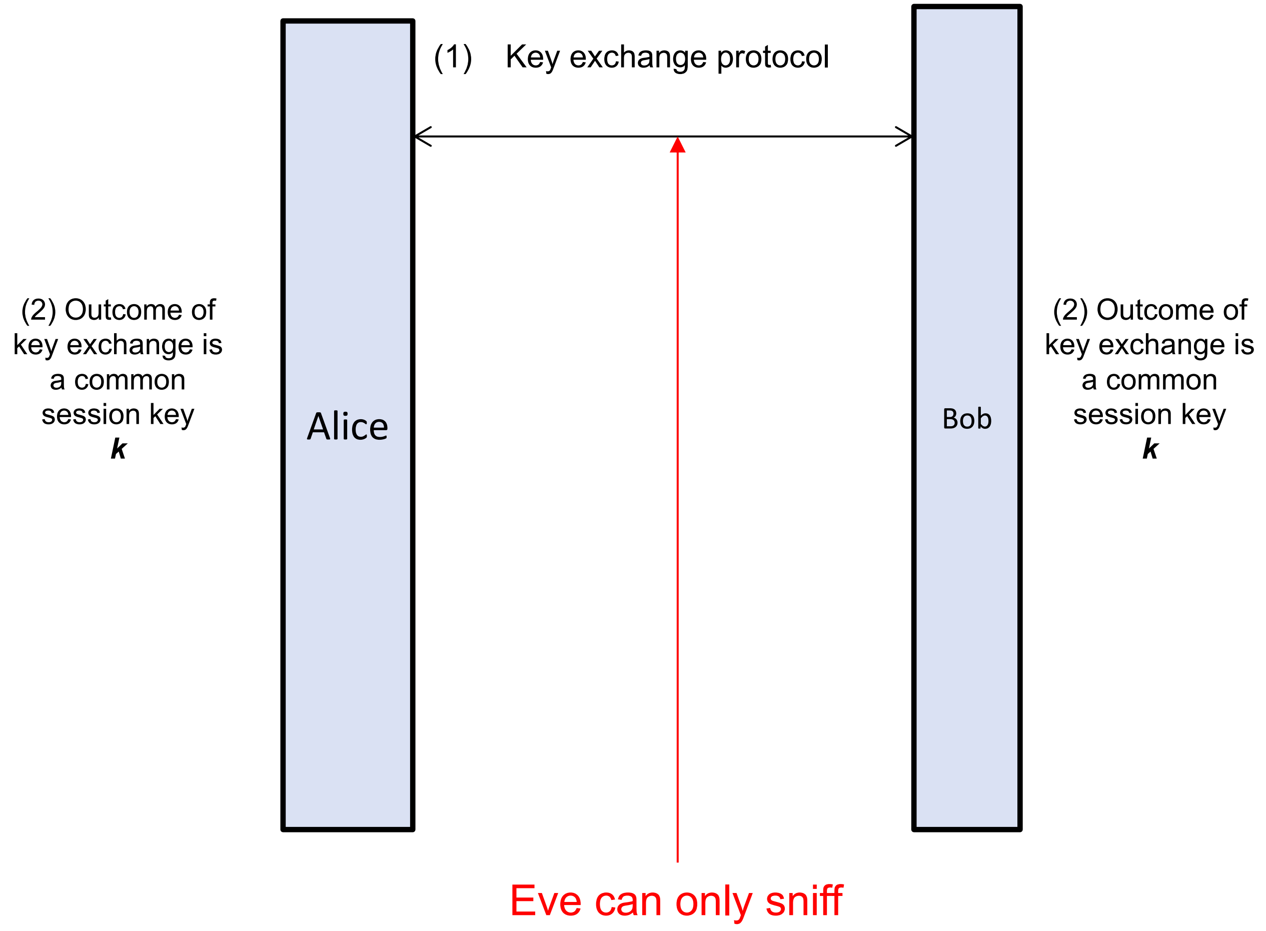
- Alice and Bob want to establish a common key. The established ***session key*** can be used to protect (e.g. via cipher, mac) subsequent communication between Alice and Bob.

- Threat Model. We consider Eve.

Capability: Eve can only sniff the communication.

Goal: Even want to steal info of the session key k.

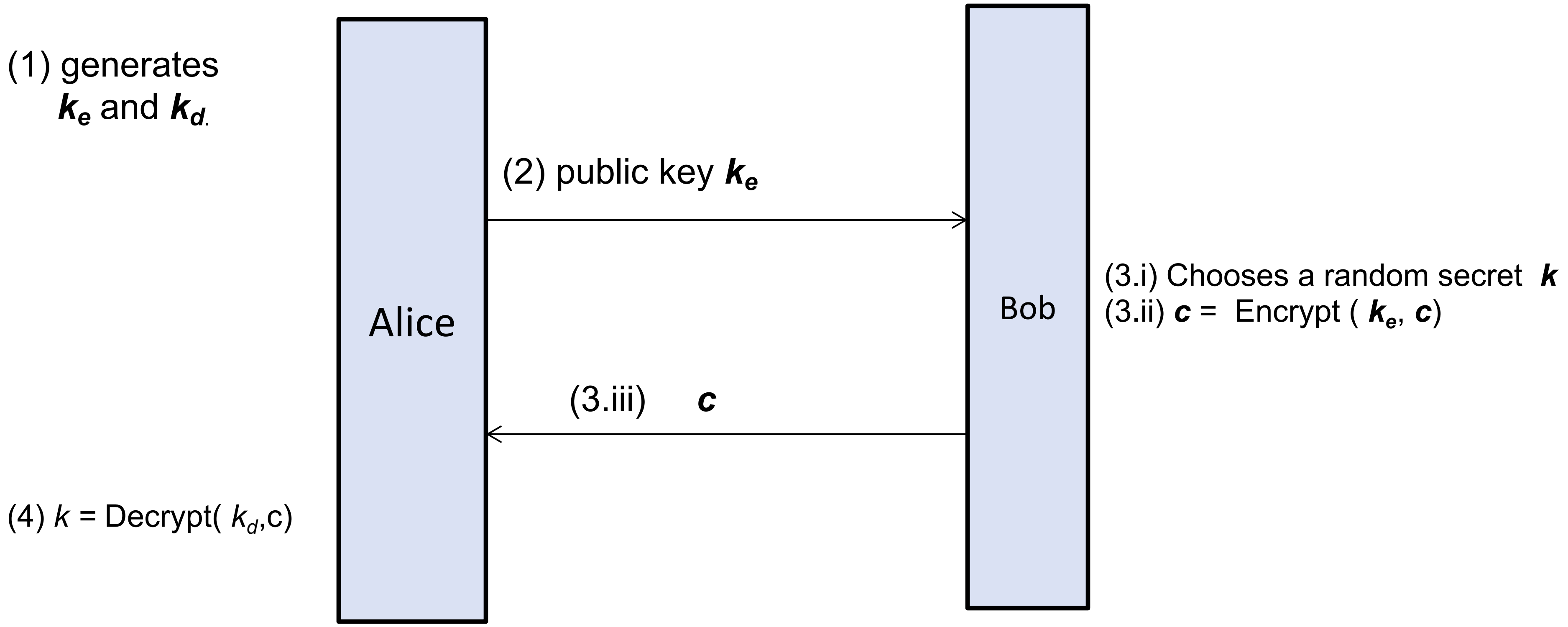
- Two methods: PKC-based, DH.


(1) Key exchange protocol
(2) Outcome of key exchange is a common session key *k*
Alice
Bob
(2) Outcome of key exchange is a common session key *k*
Eve can only sniff

PKC-based Key-exchange

Here is a key-exchange that uses a PKC

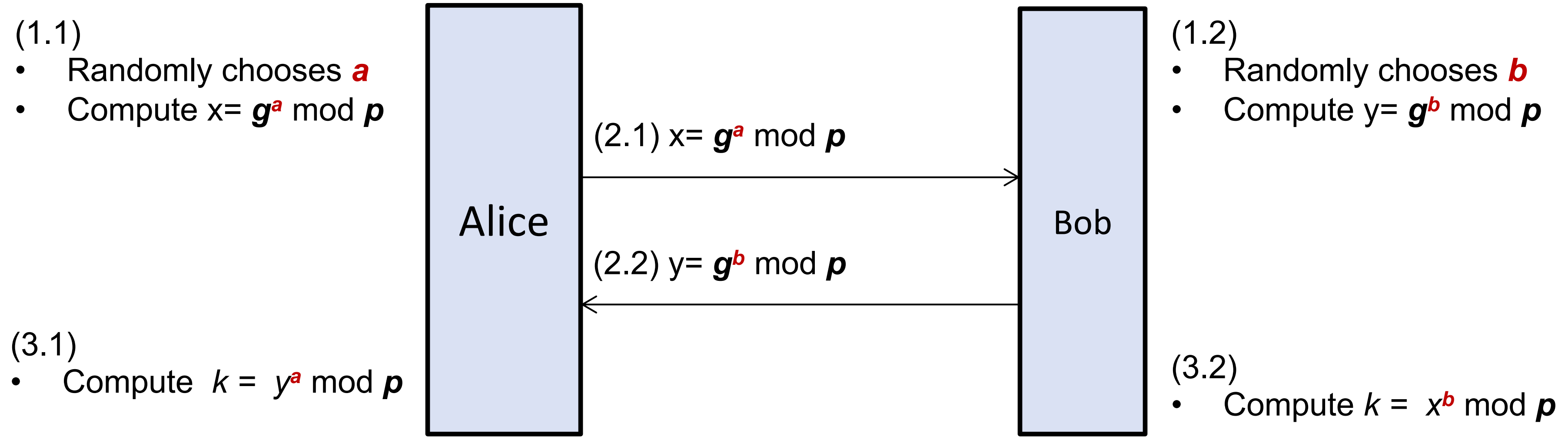
1. Alice generates a pair of private/public key.
2. Alice sends the public key $\boldsymbol{k_e}$ to Bob.
3. Bob carries out the following
   i. Randomly chooses a secret $\boldsymbol{k}$,
   ii. Encrypts $\boldsymbol{k}$ using $\boldsymbol{k_e}$.
   iii. Sends the ciphertext $\boldsymbol{c}$ to Alice.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


(1) generates $\boldsymbol{k_e}$ and $\boldsymbol{k_d}$.
Alice
(2) public key $\boldsymbol{k_e}$
Bob
(3.i) Chooses a random secret $\boldsymbol{k}$
(3.ii) $\boldsymbol{c}$ = Encrypt ($\boldsymbol{k_e}$, $\boldsymbol{c}$)
(3.iii) $\boldsymbol{c}$
(4) k = Decrypt(k_d,c)

- Attacker (Eve) can obtain the public key $\boldsymbol{k}_e$ and the ciphertext $\boldsymbol{c}$.
- By security of PKC, from the public key and ciphertext, attacker can’t get any information of the plaintext, which is the key $\boldsymbol{k}$.

Diffie-Hellman key-exchange

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 $\boldsymbol{p}$
Alice
(2.1) x= $\boldsymbol{g}^{a}$ mod $\boldsymbol{p}$
(2.2) y= $\boldsymbol{g}^{b}$ mod $\boldsymbol{p}$
Bob
(1.2)
• Randomly chooses $\boldsymbol{b}$
• Compute 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.

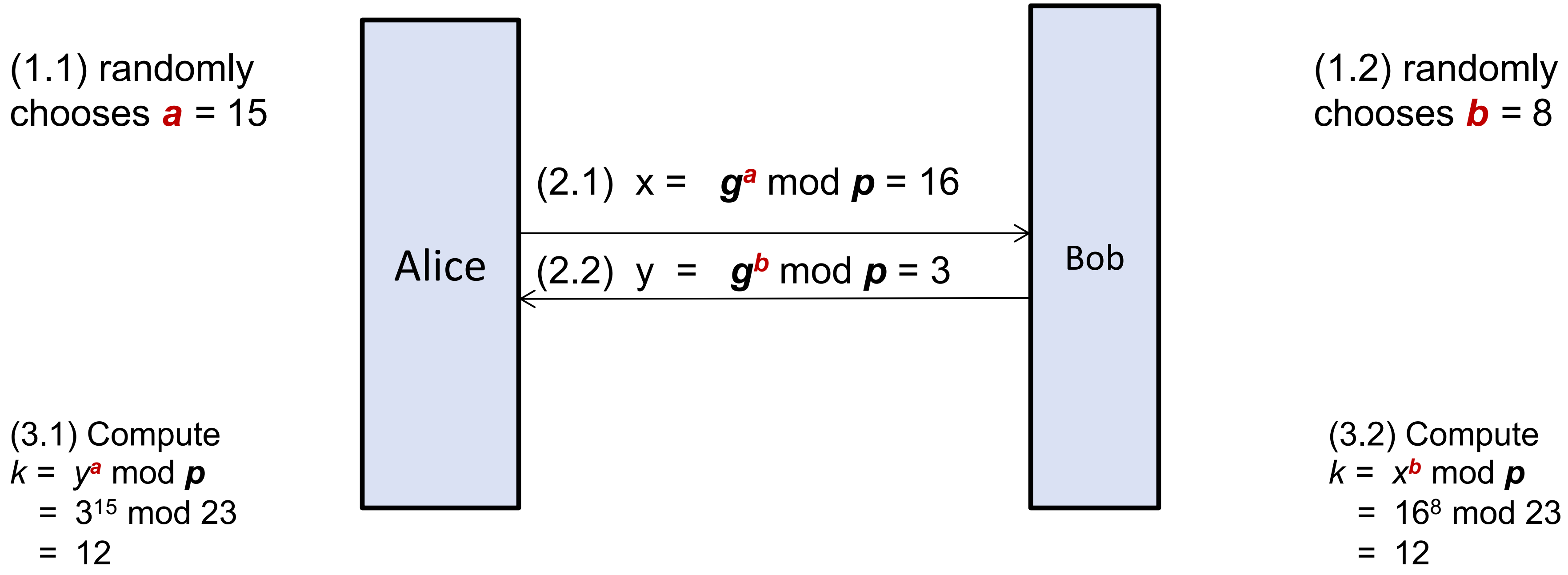
Computational Diffie-Hellman CDH assumption:
Given g, p, x= $\boldsymbol{g}^{a}$ mod $\boldsymbol{p}$, y= $\boldsymbol{g}^{b}$ mod $\boldsymbol{p}$, it is computationally infeasible to find k=$\boldsymbol{g}^{ab}$ mod $\boldsymbol{p}$.

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.

$g = 2$, $p = 23$



(1.1) randomly chooses a = 15
Alice
(2.1) $x = g^a \bmod p = 16$
(2.2) $y = g^b \bmod p = 3$
Bob
(1.2) randomly chooses b = 8
(3.1) Compute $k = y^a \bmod p = 3^{15} \bmod 23 = 12$
(3.2) Compute $k = x^b \bmod p = 16^8 \bmod 23 = 12$

From 16, and 3, it is very difficult to get 12.
(here, we are referring to very large, say 2000-bit integer)

Optional remark on DH: The DH in slide 18 leaks some info on k. There is a fast method that, given g, p, and $g^c \bmod p$, determine whether c is odd or even. So, adversary can infer whether (ab) is odd or even. If it is odd, then both a and b must be odd. If it is even, at least one of them must be even. Thus, in practice, always choose a and b that are even to avoid this leakage. *(several crypto CTFs exploit this fact in their challenges).*

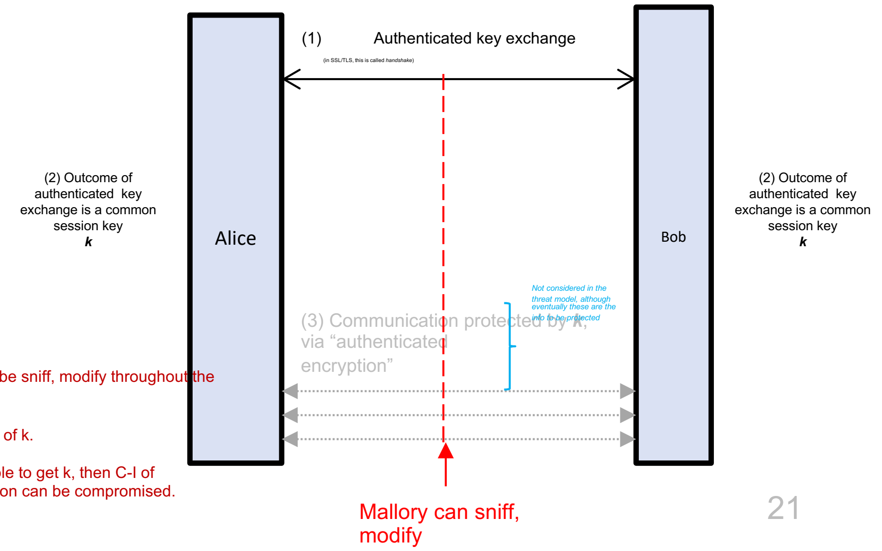
Forward Secrecy

- DH based method meets the ***Forward Secrecy*** requirement. PKC based method couldn’t.
- Forward secrecy is an important requirement, and we will cover it in Tutorial.
- TLS 1.3 mandates forward secrecy.

4.6 Authenticated Key-exchange

Attack model:

- Capability: Mallory,
- Goal: get session key and/or impersonate Alice/Bob


(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***
Not considered in the threat model, although eventually these are the info to be protected
(3) Communication protected by ***k***, via "authenticated encryption"
Mallory can sniff, modify

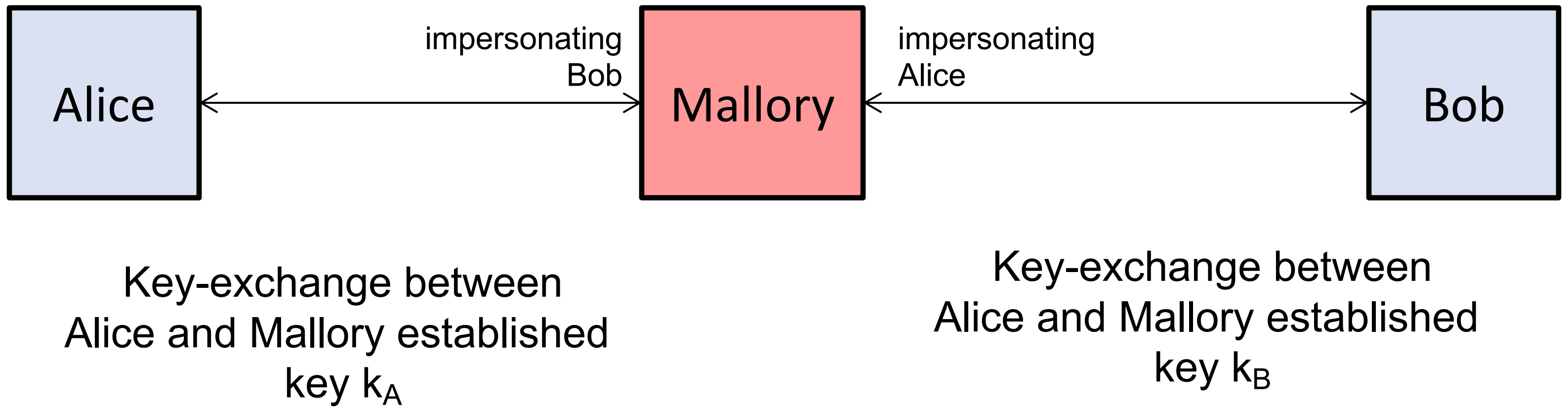
Threat model

- Capability: Can be sniff, modify throughout the session.
- Goal: Steal info of k.

  If attacker able to get k, then C-I of communication can be compromised.

Key-exchange alone can't guard against Mallory.

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


Alice
impersonating Bob
Mallory
impersonating Alice
Bob
Key-exchange between Alice and Mallory established key k_A
Key-exchange between Alice and Mallory established key k_B

In this case, Bob mistaken that Mallory is Alice and vice versa . 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.

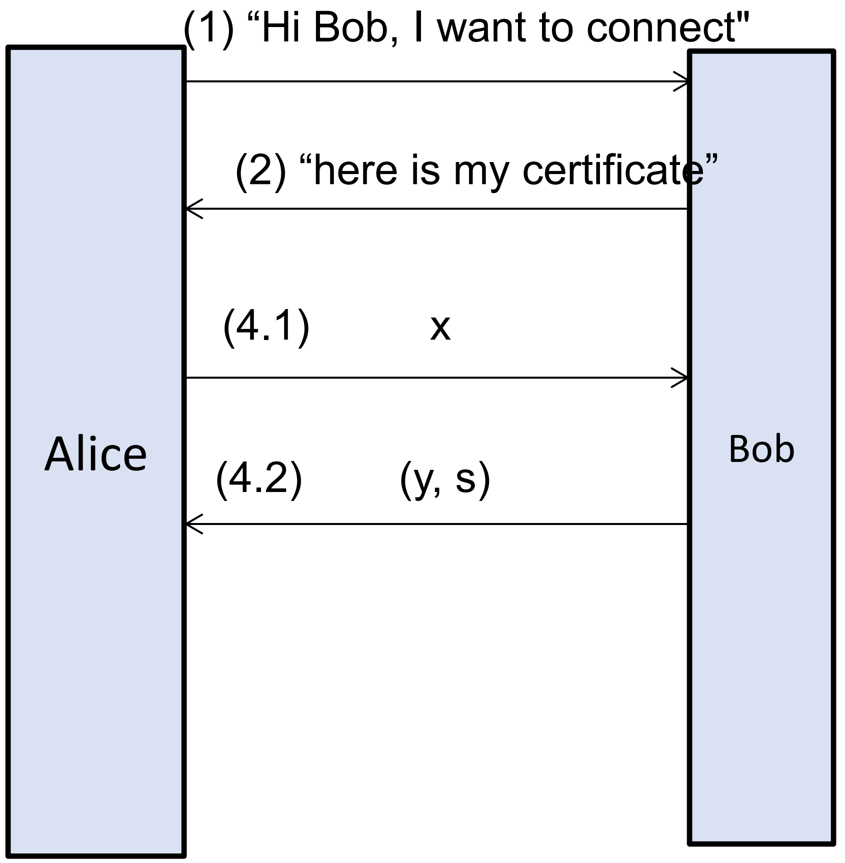
Authenticated key-exchange

- A key-exchange protocol assumes that the adversary can only sniff, but not malicious.
- To prevent malicious Mallory, we need ***authenticated key-exchange.***
- It turns out that authenticated key-exchange can be easily obtained from existing key-exchange.
- For DH based, this can be done by signing all communication using the private key. We have a special name for authenticated key-exchange that uses DH: it is known as ***Station-To-Station Protocol (STS).***
- For PKC-based, step (1) can be omitted and simply use Alice's existing public/private key. Because only Alice has the private key, so, the entity that can correctly decrypt it must be Alice.

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. Can be easily extended to mutual authentication.



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

From certificate, Alice get Bob’spublic key $\text{Bob}_{\text{public}}$

(3.1)
- Randomly chooses ***a***
- Compute x= $\boldsymbol{g}^{\boldsymbol{a}}$ mod ***p***

(3.2)
- Randomly chooses ***b***
- Compute y= $\boldsymbol{g}^{\boldsymbol{b}}$ mod ***p***
- Sign y to obtain signature ***s***

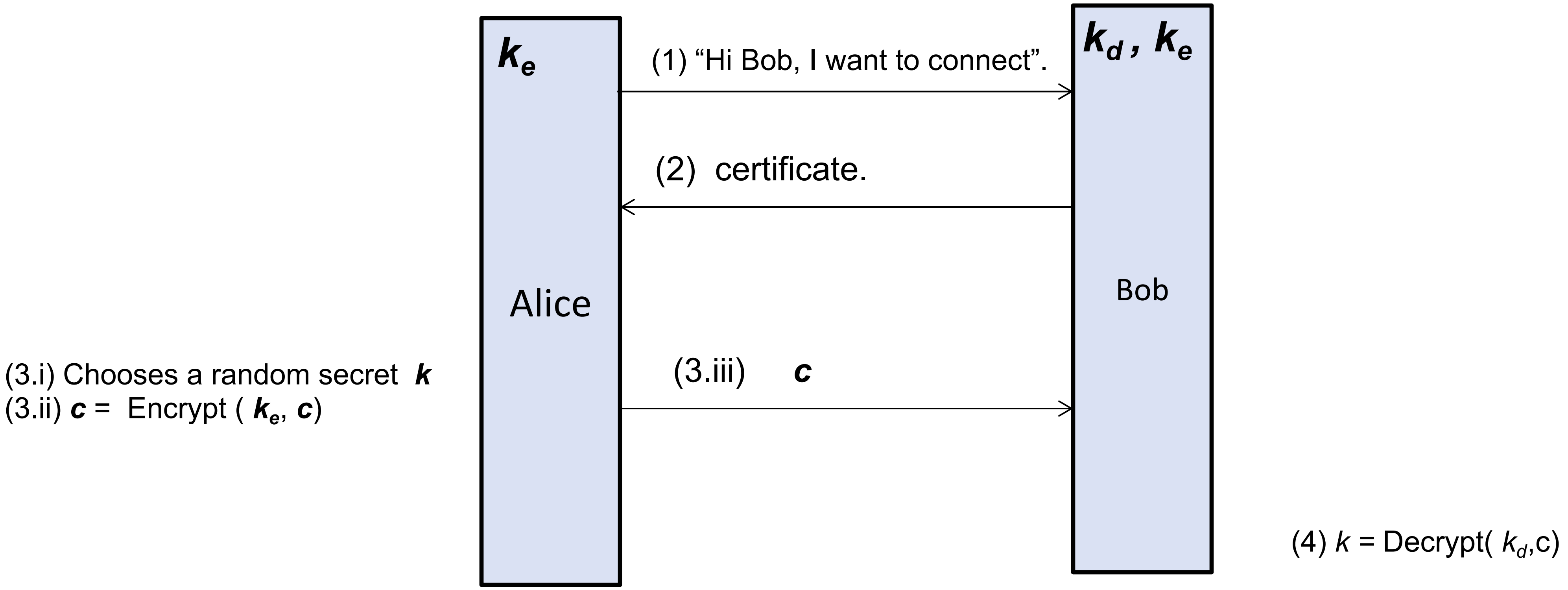
(3.1)
- Verify signature s
- Compute $k = y^{a}$ mod ***p***

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

Remark: This is unilateral authentication. Can extend it to mutual by making Alice signs her messages in step (4.1).

PKC-based authenticated Key-exchange often called RSA-based

1. Alice indicates that she wants to connect.
2. Bob sends his public key ***k**_e* to Alice.
3. Alice carries out the following
   i. Randomly chooses a secret ***k**,*
   ii. Encrypts ***k*** using $\boldsymbol{k_e}$.
   iii. Sends the ciphertext ***c*** to Alice.
4. Bob uses his private key $\boldsymbol{k_d}$ to decrypt and obtain ***k.***



k_e
Alice
k_d , k_e
Bob
(1) "Hi Bob, I want to connect".
(2) certificate.
(3.iii) c
(3.i) Chooses a random secret k
(3.ii) c = Encrypt (k_e, c)
(4) k = Decrypt(k_d,c)

Password based Authenticated Key Exchange

- (Asymmetric) Previous authenticated key-exchange protocols such as Station-to-station are based on public key. That is, an entity is considered authentic if it can convince the other that it know the private key of the associated public key.

- (Symmetric) There are also symmetric key version, i.e both entities share a symmetric key. An entity is authentic if it can prove to the other that it knows the key.

- (Password) A special case of symmetric key is when the key is a password. Password usually has very low entropy, and thus potentially could be vulnerable to "offline" dictionary attack (see tutorial). There are secure protocols, known as called "*Password-Authenticated Key agreement (PAKE)*", that prevent offline guessing and thus force the attacker to carry out online dictionary attack.

https://en.wikipedia.org/wiki/Password-authenticated_key_agreement

Remark: In tutorial, we note that offline dictionary can be carried out on the popular WPA-personal for home wifi, and there are also tools to do that (aircrack) . Strange that WPA standard doesn't use PAKE. Maybe due to higher computation in PAKE.

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.

4.7 Putting all together: Securing Communication Channel

E.g. TLS

Channel Security

Suppose we have a “public channel”. The public channel facilitates communication, however, there are presences of Mallory.

Now, using the above unsecure public communication as the carrier, can we add a layer of crypto primitives on the messages, so that the channel is as secure as a “private channel”?   Yes.

(By definition: A public channel has a Mallory; A private channel is free from Eve and Mallory)

Bob wants to visit a website `alice.com.` Alice is using the free wifi in a café called Mallory. Everyone can access the free wifi and thus it is a public channel.

We want to secure the public channel using crypto.

TLS/SSL secure the public channel in this way: (https uses TLS)

(1) Using ***long-term keys (i.e. Alice's public and private key),*** carry out authenticated key-exchange (aka handshake in TLS). Outcomes are:
   - Bob is convinced that she is interacting with Alice.
   - Both Alice and Bob have a shared ***session key.***

(2) Subsequent communication protected by the session key.

More details in next slide:

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

(Step 0) Alice obtains `bob.com`'s public key securely. 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 a shared key ***k***, which could come in the form of ***k*** =⟨***k1***,***k2***⟩ where ***k1*** is the secret key of the MAC, and ***k2*** is the secret key of the symmetric-key encryption, or a single key ***k*** when authenticated encryption (e.g. GCM) is in used. These keys are called the ***session keys***. By property of the protocol, Alice is convinced that only Bob and herself know the session key. Here (unilateral authentication), Bob doesn't care about Alice's authenticity.

(Step 2) Subsequent interactions between Alice and `Bob.com` will be protected by the session keys 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

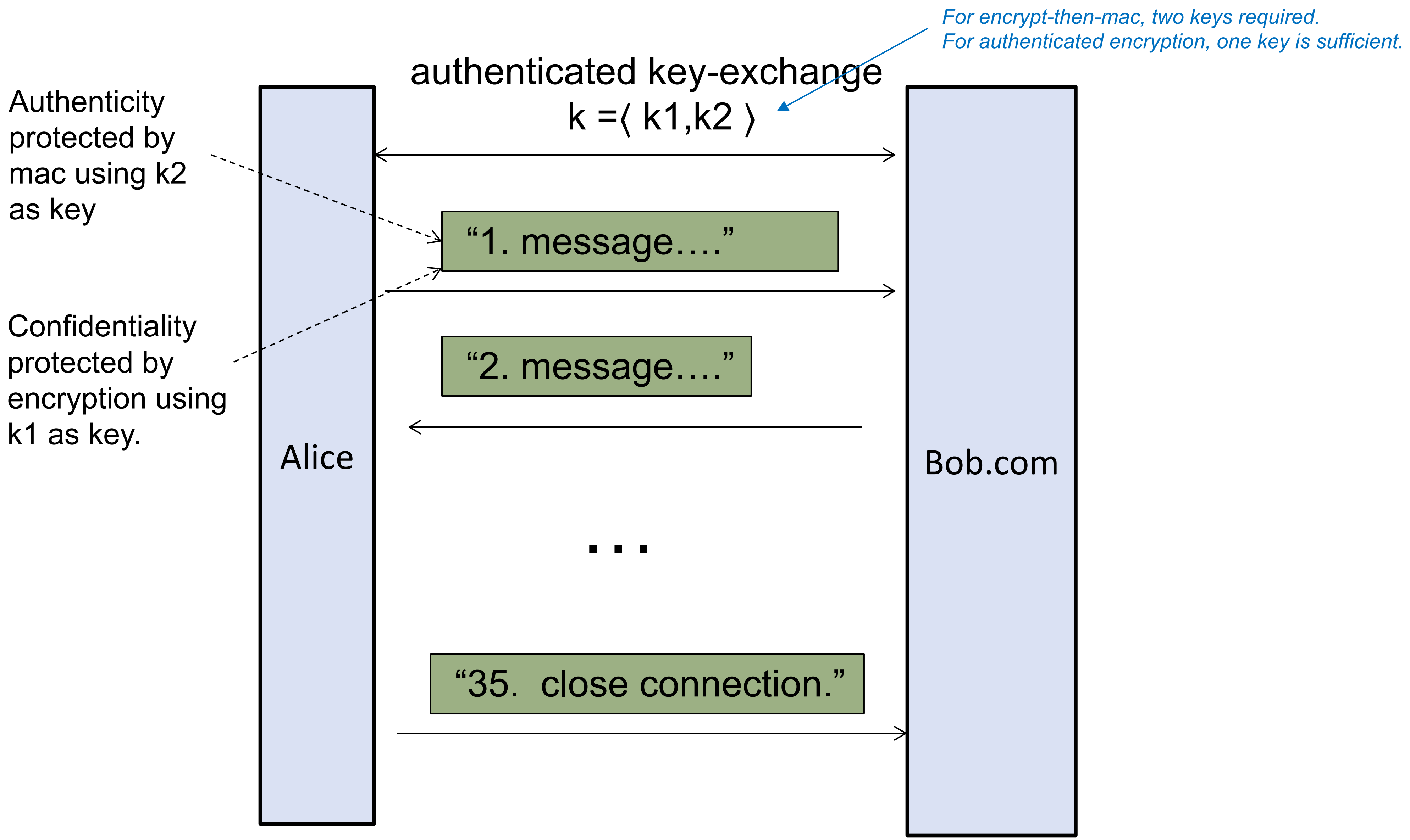
$$E_{k1}(\ i \ \| \ m\ ) \ \| \ mac_{k2}(\ E_{k1}(\ i \| m)\ ) \longleftarrow$$ ***Still in use but not recommended..***

where i is the sequence number.

For GCM mode or other authenticated encryptions, the message to be sent is simply

$$E_k(\ i \ \| \ m\ ) \longleftarrow$$ ***Recommended.***

- ‖ refer to string concatenation.
- The above is known as "encrypt-then-mac". There are other variants: "mac-then-encrypt" and "mac-and-encrypt". Using the wrong variant might leak info. It is recommended to use "authenticated encryption" such as AES GCM mode.



For encrypt-then-mac, two keys required.
For authenticated encryption, one key is sufficient.
authenticated key-exchange
k =⟨ k1,k2 ⟩
Authenticity protected by mac using k2 as key
Confidentiality protected by encryption using k1 as key.
Alice
Bob.com
"1. message…."
"2. message…."
. . .
"35. close connection."

The data eventually sent is ***E_{k1} ("1. message etc") ‖ mac_{k2} (E_{k1} ("1. message etc"))*** ***or using authenticated encryption***

Question: What is the role of the sequence number 1,2,3, … and the last message "close connection"?

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.

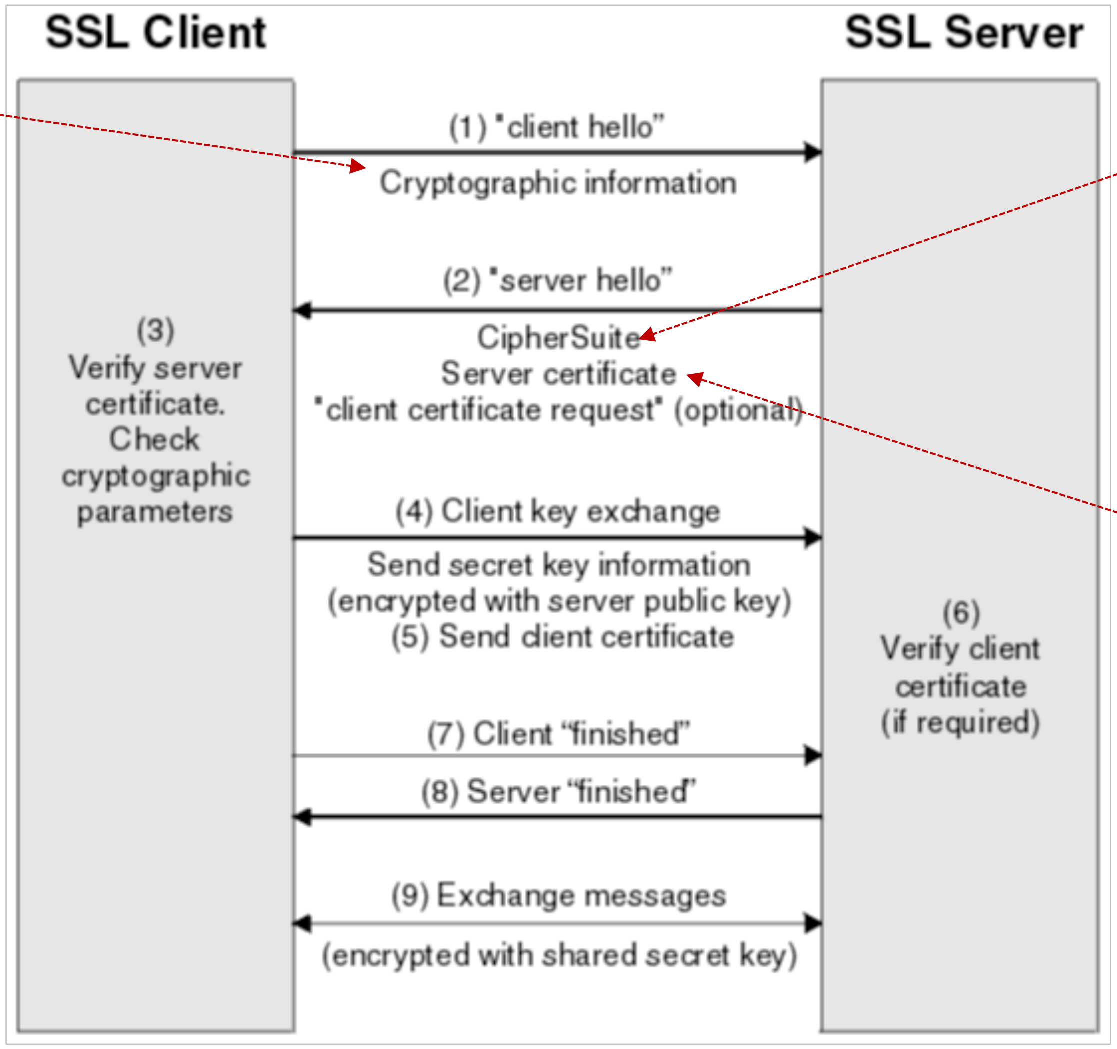
Remarks:

- SSL 3.0 has a few 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 (aka downgrading attack), and thus TLS 1.0 is also vulnerable. So, to be secure, at least TLS 2.0. See https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.0

TLS handshake (authenticated key exchange)

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

Authenticated key-exchange


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages (encrypted with shared secret key)

Server confirms or counter-proposes the crypto suite.

Server's public key

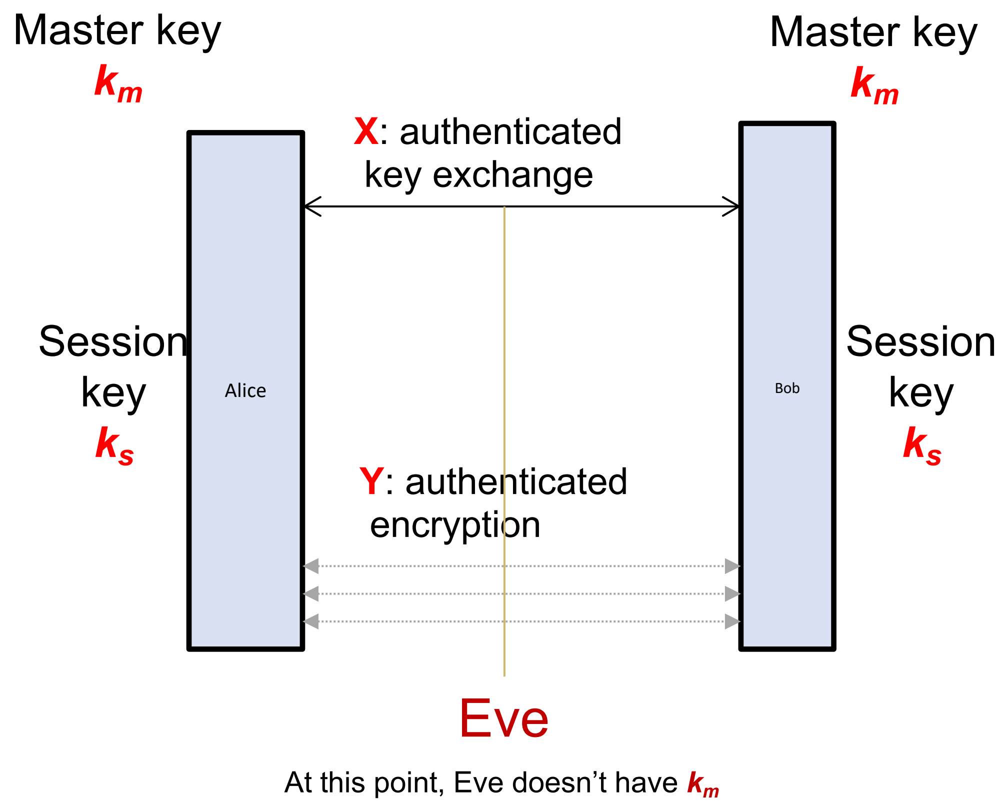
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

4.8 Forward Secrecy

Forward Secrecy is a security requirement on authenticated key exchange. Forward Secrecy considers this attack scenario.

1. Eve has logged the communication between two entities
   a) ***X***: Authenticated key exchange based on the master key, k_m
   b) ***Y***: Subsequence messages protected by the session key, k_s
2. Eve obtains the master k_m. (Eve somehow obtains it through other means, e.g. Eve successfully hacks into the server).
3. From k_m , ***X***, and ***Y***, Eve wants to recover the plaintext in ***Y***. (if Eve can obtain k_s, then Eve can recover the plaintext)

If Eve cannot succeed, then we say that the authenticated key exchange achieve ***Forward Secrecy***.


Master key k_m
Master key k_m
X: authenticated key exchange
Session key k_s
Alice
Bob
Session key k_s
Y: authenticated encryption
Eve
At this point, Eve doesn't have k_m

$$X,\ Y,\ k_m \longrightarrow \text{Eve} \longrightarrow \text{Plaintext of } Y \text{ (or } k_s\text{)}$$

- PKC-based authenticated key-exchange does not achieve forward secrecy.

  (Tutorial 6)

- Station-to-Station key exchange achieves forward secrecy
  - (proof omitted)
  - If an attacker can solve CDH (slid 18), then the attacker can compromise forward secrecy of station-to-station.