CSC358 Intro. to Computer Networks

Lecture 11: VLAN, MPLS, Network Security


ahchinaei@cs.toronto.edu
http://www.cs.toronto.edu/~ahchinaei/

Many slides are (inspired/adapted) from the above source
© all material copyright; all rights reserved for the authors

Office Hours: T 17:00–18:00 R 9:00–10:00 BA4222

TA Office Hours: W 16:00-17:00 BA3201 R 10:00-11:00 BA7172
csc358ta@cdf.toronto.edu
VLANs: motivation

- CS user moves office to EE, but wants to connect to CS switch?
- single broadcast domain:
  - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - security/privacy, efficiency issues
port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch ……

Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple *virtual* LANS over single physical LAN infrastructure.

… operates as *multiple* virtual switches
Port-based VLAN

- **traffic isolation**: frames to/from ports 1-8 can only reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port

- **dynamic membership**: ports can be dynamically assigned among VLANs

- **forwarding between VLANs**: done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers
VLANS spanning multiple switches

- **trunk port**: carries frames between VLANS defined over multiple physical switches
  - Frames forwarded within VLAN between switches can’t be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports
802.1Q VLAN frame format

802.1 frame

802.1Q frame

- preamble
- dest. address
- source address
- data (payload)
- CRC

2-byte Tag Protocol Identifier (value: 81-00)
Recomputed CRC

Tag Control Information (12 bit VLAN ID field, 3 bit priority field like IP TOS)
Multiprotocol label switching (MPLS)

- initial goal: high-speed IP forwarding using fixed length label (instead of IP address)
  - fast lookup using fixed length identifier (rather than longest prefix matching)
  - borrowing ideas from Virtual Circuit (VC) approach
  - but IP datagram still keeps IP address!

![Diagram of MPLS header structure]

- PPP or Ethernet header
- MPLS header
- IP header
- remainder of link-layer frame

<table>
<thead>
<tr>
<th>label</th>
<th>Exp</th>
<th>S</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
MPLS capable routers

- a.k.a. label-switched router
- forward packets to outgoing interface based only on label value (don’t inspect IP address)
  - MPLS forwarding table distinct from IP forwarding tables
- **flexibility**: MPLS forwarding decisions can **differ** from those of IP
- use destination and source addresses to route flows to same destination differently (traffic engineering)
- re-route flows quickly if link fails: pre-computed backup paths (useful for VoIP)
**MPLS versus IP paths**

- **IP routing:** path to destination determined by destination address alone
**MPLS versus IP paths**

- **IP routing**: path to destination determined by destination address alone

- **MPLS routing**: path to destination can be based on source and dest. address
  - **fast reroute**: precompute backup routes in case of link failure

entry router (R4) can use different MPLS routes to A based, e.g., on source address
MPLS signaling

- modify OSPF, IS-IS link-state flooding protocols to carry info used by MPLS routing,
  - e.g., link bandwidth, amount of “reserved” link bandwidth
- entry MPLS router uses RSVP-TE signaling protocol to set up MPLS forwarding at downstream routers
MPLS forwarding tables

```
in label | out label | dest | out interface
---------|-----------|------|----------------
  10      |   A       |  0   |
  12      |   D       |  0   |
   8      |   A       |  1   |

```

```
in label | out label | dest | out interface
---------|-----------|------|----------------
  10      |   6       |   A  |   1
  12      |   9       |   D  |   0
```

```
in label | out label | dest | out interface
---------|-----------|------|----------------
   6      |     -     |   A  |   0
```

Link Layer 5-12
Data center networks

- 10’s to 100’s of thousands of hosts, often closely coupled, in close proximity:
  - e-business (e.g. Amazon)
  - content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
  - search engines, data mining (e.g., Google)

- challenges:
  - multiple applications, each serving massive numbers of clients
  - managing/balancing load, avoiding processing, networking, data bottlenecks

Inside a 40-ft Microsoft container, Chicago data center
Data center networks

load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)
Data center networks

- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy
Link layer, LANs: outline

5.1 introduction, services
5.2 error detection, correction
5.3 multiple access protocols
5.4 LANs
  - addressing, ARP
  - Ethernet
  - switches
  - VLANs
5.5 link virtualization: MPLS
5.6 data center networking
5.7 a day in the life of a web request
Synthesis: a day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - goal: identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - scenario: student attaches laptop to campus network, requests/receives www.google.com
A day in the life: scenario

Comcast network 68.80.0.0/13

Google’s network 64.233.160.0/19

DNS server

school network 68.80.2.0/24

web server 64.233.169.105

web page

Google

browser
A day in the life… connecting to the Internet

- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use **DHCP**

- DHCP request **encapsulated** in **UDP**, encapsulated in **IP**, encapsulated in **802.3 Ethernet**

- Ethernet frame **broadcast** (dest: FFFFFFFF) on LAN, received at router running **DHCP** server

- Ethernet **demuxed** to IP demuxed, UDP demuxed to DHCP
A day in the life… connecting to the Internet

- DHCP server formulates **DHCP ACK** containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

**Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router**
A day in the life… ARP (before DNS, before HTTP)

- before sending HTTP request, need IP address of www.google.com: DNS
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query
A day in the life… using DNS

- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
  - IP datagram forwarded from campus network into comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
  - demux’ed to DNS server
  - DNS server replies to client with IP address of www.google.com
A day in the life...TCP connection carrying HTTP

- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!
A day in the life… HTTP request/reply

- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client
- web page finally (!!!) displayed
Chapter 8: Network Security

Chapter goals:

❖ understand principles of network security:
  ▪ cryptography and its *many* uses beyond “confidentiality”
  ▪ authentication
  ▪ message integrity

❖ security in practice:
  ▪ firewalls and intrusion detection systems
  ▪ security in application, transport, network, link layers
What is network security?

**Confidentiality:** only sender, intended receiver should “understand” message contents
- sender encrypts message
- receiver decrypts message

**Authentication:** sender, receiver want to confirm identity of each other

**Message Integrity:** sender, receiver want to ensure message not altered (in transit, or afterwards) without detection

**Access and Availability:** services must be accessible and available to users
Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate “securely”
- Trudy (intruder) may intercept, delete, add messages

Network Security
Who might Bob, Alice be?

- … well, *real-life* Bobs and Alices!
- Web browser/server for electronic transactions (e.g., on-line purchases)
- on-line banking client/server
- DNS servers
- routers exchanging routing table updates
- other examples?
There are bad guys (and girls) out there!

Q: What can a “bad guy” do?
A: A lot! See section 1.6

- **eavesdrop**: intercept messages
- actively **insert** messages into connection
- **impersonation**: can fake (spoof) source address in packet (or any field in packet)
- **hijacking**: “take over” ongoing connection by removing sender or receiver, inserting himself in place
- **denial of service**: prevent service from being used by others (e.g., by overloading resources)
The language of cryptography

plaintext message

$K_A(m)$ ciphertext, encrypted with key $K_A$

$m = K_B(K_A(m))$
Breaking an encryption scheme

- cipher-text only attack: Trudy has ciphertext she can analyze
- two approaches:
  - brute force: search through all keys
  - statistical analysis
- known-plaintext attack: Trudy has plaintext corresponding to ciphertext
  - e.g., in monoalphabetic cipher, Trudy determines pairings for a,l,i,c,e,b,o,
- chosen-plaintext attack: Trudy can get ciphertext for chosen plaintext
Symmetric key cryptography

**plaintext message, m** → encryption algorithm → **ciphertext** → decryption algorithm → **plaintext**

**Key:** $K_S$
- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

**Q:** how do Bob and Alice agree on key value?
Simple encryption scheme

substitution cipher: substituting one thing for another
- monoalphabetic cipher: substitute one letter for another

plaintext:  abcdefghijklmnopqrstuvwxyz

\[ \downarrow \]
ciphertext:  mnbvcxzasdfghjklpoiuytrewq

e.g.: Plaintext: bob. i love you. alice

ciphertext:  nkn. s gktc wky. mgsbc

Encryption key: mapping from set of 26 letters to set of 26 letters
A more sophisticated encryption approach

- n substitution ciphers, $M_1, M_2, \ldots, M_n$
- cycling pattern:
  - e.g., $n=4$: $M_1, M_3, M_4, M_3, M_2$; $M_1, M_3, M_4, M_3, M_2$; ...
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from $M_1$, o from $M_3$, g from $M_4$

**Encryption key:** n substitution ciphers, and cyclic pattern
- key need not be just n-bit pattern
Symmetric key crypto: DES

DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- block cipher with cipher block chaining
- how secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - no known good analytic attack
- making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys
Symmetric key crypto: DES

**DES operation**

- Initial permutation
- 16 identical “rounds” of function application, each using different 48 bits of key
- Final permutation
AES: Advanced Encryption Standard

- symmetric-key NIST standard, replaced DES (Nov 2001)
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES
Public Key Cryptography

**symmetric key crypto**
- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?

**public key crypto**
- radically different approach [Diffie-Hellman76, RSA78]
- sender, receiver do not share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver
Public key cryptography

plaintext message, m

encryption algorithm

K_B^+(m)

ciphertext

decryption algorithm

m = K_B^-(K_B^+(m))

Bob’s public key

K_B^+

Bob’s private key

K_B^-
Public key encryption algorithms

requirements:

① need \( K_B^+(\cdot) \) and \( K_B^-(\cdot) \) such that
\[
K_B^-(K_B^+(m)) = m
\]

② given public key \( K_B^+ \), it should be impossible to compute private key \( K_B^- \)

**RSA**: Rivest, Shamir, Adelson algorithm
Prerequisite: modular arithmetic

- $x \mod n =$ remainder of $x$ when divide by $n$
- facts:
  
  $[(a \mod n) + (b \mod n)] \mod n = (a+b) \mod n$
  $[(a \mod n) - (b \mod n)] \mod n = (a-b) \mod n$
  $[(a \mod n) \cdot (b \mod n)] \mod n = (a \cdot b) \mod n$
- thus
  
  $(a \mod n)^d \mod n = a^d \mod n$
- example: $x=14$, $n=10$, $d=2$:
  
  $(x \mod n)^d \mod n = 4^2 \mod 10 = 6$
  
  $x^d = 14^2 = 196 \quad x^d \mod 10 = 6$
RSA: getting ready

- message: just a bit pattern
- bit pattern can be uniquely represented by an integer number
- thus, encrypting a message is equivalent to encrypting a number.

**example:**

- \( m = 10010001 \). This message is uniquely represented by the decimal number 145.
- to encrypt \( m \), we encrypt the corresponding number, which gives a new number (the ciphertext).
RSA: Creating public/private key pair

1. choose two large prime numbers $p$, $q$. (e.g., 1024 bits each)

2. compute $n = pq$, $z = (p-1)(q-1)$

3. choose $e$ (with $e < n$) that has no common factors with $z$ ($e$, $z$ are “relatively prime”).

4. choose $d$ such that $ed - 1$ is exactly divisible by $z$. (in other words: $ed \mod z = 1$).

5. public key is $(n, e)$. private key is $(n, d)$.
RSA: encryption, decryption

0. given \((n,e)\) and \((n,d)\) as computed above

1. to encrypt message \(m \ (<n)\), compute
   \[c = m^e \mod n\]

2. to decrypt received bit pattern, \(c\), compute
   \[m = c^d \mod n\]

\[
m = (m^e \mod n)^d \mod n
\]

*magic happens!*
RSA example:


- $e=5$ (so $e$, $z$ relatively prime).
- $d=29$ (so $ed-1$ exactly divisible by $z$).

Encrypting 8-bit messages.

<table>
<thead>
<tr>
<th>bit pattern</th>
<th>$m$</th>
<th>$m^e$</th>
<th>$c = m^e \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001100</td>
<td>12</td>
<td>24832</td>
<td>17</td>
</tr>
</tbody>
</table>

Decrypt:

<table>
<thead>
<tr>
<th>$c$</th>
<th>$c^d$</th>
<th>$m = c^d \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>481968572106750915091411825223071697</td>
<td>12</td>
</tr>
</tbody>
</table>
Why does RSA work?

- must show that $c^d \mod n = m$
  where $c = m^e \mod n$
- fact: for any $x$ and $y$: $x^y \mod n = x^{(y \mod z)} \mod n$
  - where $n = pq$ and $z = (p-1)(q-1)$
- thus,
  
  $c^d \mod n = (m^e \mod n)^d \mod n$
  
  $= m^{ed} \mod n$
  
  $= m^{ed \mod z} \mod n$
  
  $= m^1 \mod n$
  
  $= m$
RSA: another important property

The following property will be very useful later:

\[ K_B^{-1}(K_B^+(m)) = m = K_B^{-1}(K_B^-(m)) \]

use public key first, followed by private key

use private key first, followed by public key

result is the same!
Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

follows directly from modular arithmetic:

$$(m^e \mod n)^d \mod n = m^{ed} \mod n$$

$$= m^{de} \mod n$$

$$= (m^d \mod n)^e \mod n$$
Why is RSA secure?

- suppose you know Bob’s public key \((n,e)\). How hard is it to determine \(d\)?
- essentially need to find factors of \(n\) without knowing the two factors \(p\) and \(q\)
  - fact: factoring a big number is hard
RSA in practice: session keys

- Exponentiation in RSA is computationally intensive
- DES is at least 100 times faster than RSA
- Use public key crypto to establish secure connection, then establish second key — symmetric session key — for encrypting data

**Session key, $K_S$**
- Bob and Alice use RSA to exchange a symmetric key $K_S$
- Once both have $K_S$, they use symmetric key cryptography
Chapter 8 roadmap

8.1 What is network security?
8.2 Principles of cryptography
8.3 Message integrity, *authentication*
8.4 Securing e-mail
8.5 Securing TCP connections: SSL
8.6 Network layer security: IPsec
8.7 Securing wireless LANs
8.8 Operational security: firewalls and IDS
Authentication

**Goal:** Bob wants Alice to “prove” her identity to him

**Protocol ap1.0:** Alice says “I am Alice”

Failure scenario??
Authentication

**Goal:** Bob wants Alice to “prove” her identity to him.

**Protocol ap1.0:** Alice says “I am Alice.”

In a network, Bob cannot “see” Alice, so Trudy simply declares herself to be Alice.
**Authentication: another try**

*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address

---

Network Security
Authentication: another try

*Protocol ap2.0:* Alice says “I am Alice” in an IP packet containing her source IP address

Trudy can create a packet “spoofing” Alice’s address.
Authentication: another try

*Protocol ap3.0:* Alice says “I am Alice” and sends her secret password to “prove” it.

--

Network Security
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.

Playback attack: Trudy records Alice’s packet and later plays it back to Bob.
Authentication: yet another try

Protocol ap3.1: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

Failure scenario??

Network Security
Authentication: yet another try

Protocol ap3.1: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

record and playback still works!
Authentication: yet another try

**Goal:** avoid playback attack

**nonce:** number (R) used only *once-in-a-lifetime*

**ap4.0:** to prove Alice “live”, Bob sends Alice *nonce*, R. Alice must return R, encrypted with shared secret key

“*I am Alice*”

Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?
Authentication: ap5.0

ap4.0 requires shared symmetric key

- can we authenticate using public key techniques?

**ap5.0**: use nonce, public key cryptography

```
“I am Alice”

R

K_{A^-}(R)

“send me your public key”

K_{A^+}

Bob computes

K_{A^+}(K_{A^-}(R)) = R

and knows only Alice could have the private key, that encrypted R such that

K_{A^+}(K_{A^-}(R)) = R
```
**ap5.0: security hole**

*Man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)

\[ m = K_A^- (K_A^+ (m)) \]

\[ K_T^+ (m) \]

\[ T \]

\[ K_T^- (R) \]

\[ R \]

\[ K_A^- (R) \]

\[ K_A^+ \]

\[ K_A^- \]

\[ K_A^+ (m) \]

\[ m = K_A^- (K_A^+ (m)) \]

\[ m \]

**I am Alice**

**Send me your public key**

**Send me your public key**

**I am Alice**

**Trudy gets**

**sends m to Alice**

encrypted with Alice’s public key

Network Security
**ap5.0: security hole**

*man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)

**difficult to detect:**
- Bob receives everything that Alice sends, and vice versa. (e.g., so Bob, Alice can meet one week later and recall conversation!)
- Problem is that Trudy receives all messages as well!
Digital signatures

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- **verifiable, nonforgeable**: recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
simple digital signature for message m:

- Bob signs m by encrypting with his private key $K_B$, creating “signed” message, $K_B(m)$

Bob’s message, m

Dear Alice
Oh, how I have missed you. I think of you all the time! ...(blah blah blah)
Bob

Bob’s message, m, signed (encrypted) with his private key

Public key encryption algorithm

$m, K_B^{-1}(m)$
Digital signatures

- Suppose Alice receives msg m, with signature: m, $K_B^{-}(m)$
- Alice verifies m signed by Bob by applying Bob’s public key $K_B^{+}$ to $K_B^{-}(m)$ then checks $K_B^{+}(K_B^{-}(m)) = m$.
- If $K_B^{+}(K_B^{-}(m)) = m$, whoever signed m must have used Bob’s private key.

**Alice thus verifies that:**
- ✓ Bob signed m
- ✓ no one else signed m
- ✓ Bob signed m and not m′

**non-repudiation:**
- ✓ Alice can take m, and signature $K_B^{-}(m)$ to court and prove that Bob signed m
Message digests

- computationally expensive to public-key-encrypt long messages

**goal:** fixed-length, easy-to-compute digital “fingerprint”

- apply hash function $H$ to $m$, get fixed size message digest, $H(m)$.

Hash function properties:
- many-to-1
- produces fixed-size msg digest (fingerprint)
- given message digest $x$, computationally infeasible to find $m$ such that $x = H(m)$
Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

<table>
<thead>
<tr>
<th>message</th>
<th>ASCII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I O U 1</td>
<td>49 4F 55 31</td>
</tr>
<tr>
<td>0 0 . 9</td>
<td>30 30 2E 39</td>
</tr>
<tr>
<td>9 B O B</td>
<td>39 42 D2 42</td>
</tr>
</tbody>
</table>

B2 C1 D2 AC  different messages but identical checksums!
Digital signature = signed message digest

Bob sends digitally signed message:

1. **large message** m
2. **H: Hash function**
3. **H(m)**
4. **Bob's private key** $K_B^-$ (encrypt)
5. **digital signature** $K_B^-(H(m))$
6. **encrypted msg digest** $K_B^-(H(m))$
7. **Bob's public key** $K_B^+$ (decrypt)
8. **H: Hash function**
9. **H(m)**

Alice verifies signature, integrity of digitally signed message:

1. **encrypted msg digest** $K_B^-(H(m))$
2. **Bob's public key** $K_B^+$ (decrypt)
3. **H: Hash function**
4. **H(m)**
5. **equal?**

Network Security
Hash function algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string $x$, appears difficult to construct msg $m$ whose MD5 hash is equal to $x$
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest
Recall: ap5.0 security hole

*man (or woman) in the middle attack:* Trudy poses as Alice (to Bob) and as Bob (to Alice)

\[ m = K_A^{-1}(K_A^+(m)) \]

- Trudy gets \( m = K_T^-(K_T^+(m)) \)
- sends \( m \) to Alice encrypted with Alice’s public key
Public-key certification

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it’s Bob’s public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn’t even like pepperoni
**Certification authorities**

- **certification authority (CA):** binds public key to particular entity, E.

- E (person, router) registers its public key with CA.
  - E provides “proof of identity” to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E’s public key digitally signed by CA – CA says “this is E’s public key”
Certification authorities

- when Alice wants Bob’s public key:
  - gets Bob’s certificate (Bob or elsewhere).
  - apply CA’s public key to Bob’s certificate, get Bob’s public key