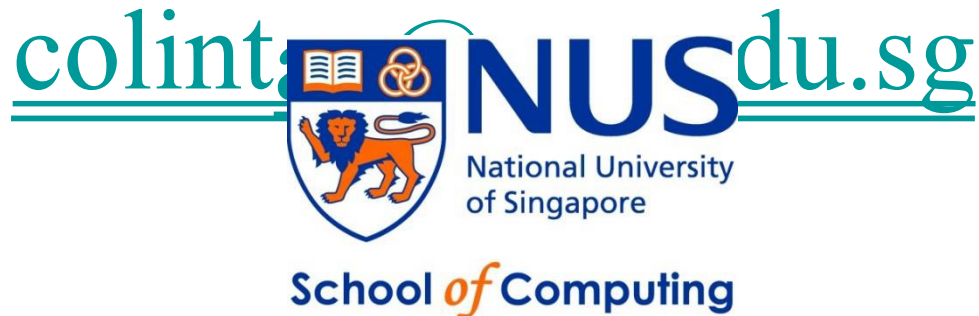


# CG1112

## Engineering Principles and Practices II for CEG

### Week 9 Studio 2

### Secure Networking



# Motivation

- **At the moment, Alex can talk to the Arduino to tell the motors how far to go, how to turn, etc.**
- **But to control Alex, the user has to:**
  - ssh
  - Use VNC Viewer.
- **This is not a very good way to do things.**
- **What we want is to write a specialized client program that lets you control Alex from your laptop WITHOUT using ssh or VNC Viewer.**
  - Much more flexible. Can even write web pages to control Alex!.
- **We also want to do this securely.**
  - Don't want people to intercept and hijack poor Alex!
  - Must use TLS programming!
- **We will cover the basic concepts in this lecture, and the more practical aspects in the studio:**
  - Certificate generation, TLS programming, etc.

## Some Notes

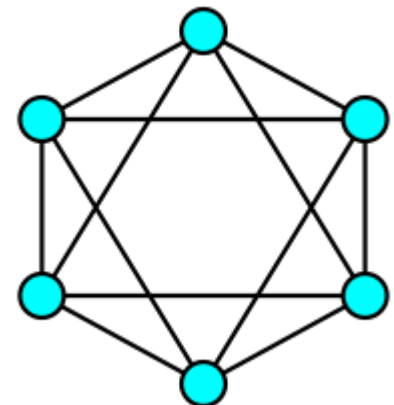
- **TCP/IP programming:**
  - How to write programs that can communicate over the Internet.
  - Uses “sockets”.
- **TLS programming:**
  - How to write programs that use “transport layer security” to secure data over the internet.
- **Due to the very involved way of writing TCP/IP and TLS programming, we have provided libraries that automatically create TLS clients and servers for you.**
- **Please see the optional “Under the Hood” documents for details on how to write your own TCP/IP and TLS programmes.**

## Secure Networking

# HOW THE INTERNET WORKS: TCP/IP

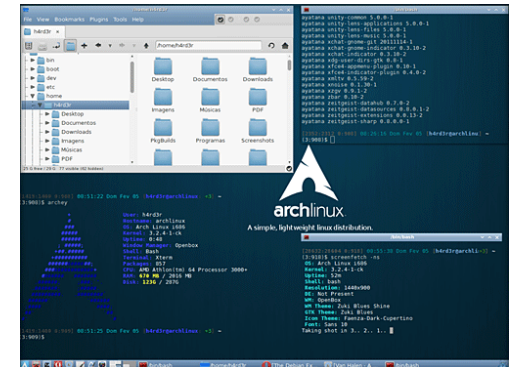
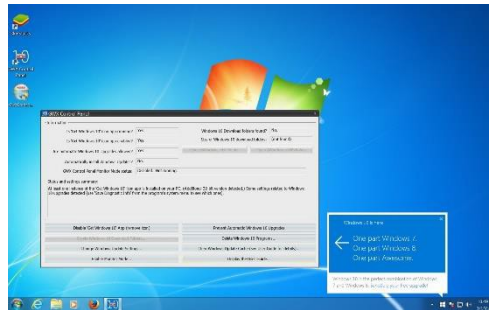
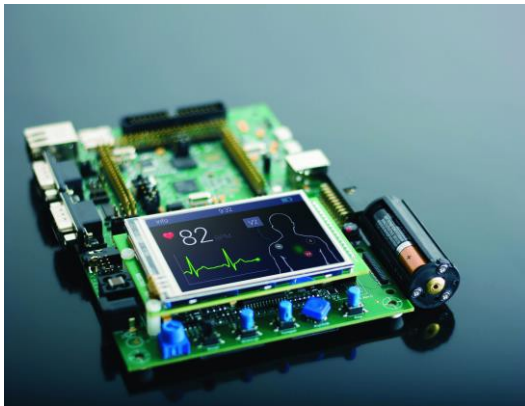
# Introduction

- **The “Internet Problem”:**
  - How do we connect two computers on opposite ends of the world together?
- **Issues:**
  - **Distance:** We can’t possibly run a wire from your computer to the Google servers in the U.S.
  - **Complexity:** We can’t possibly run wires from every computer in the world to every other computer in the world. We will have  $O(n^2)$  wires, where  $n$  is in the billions.



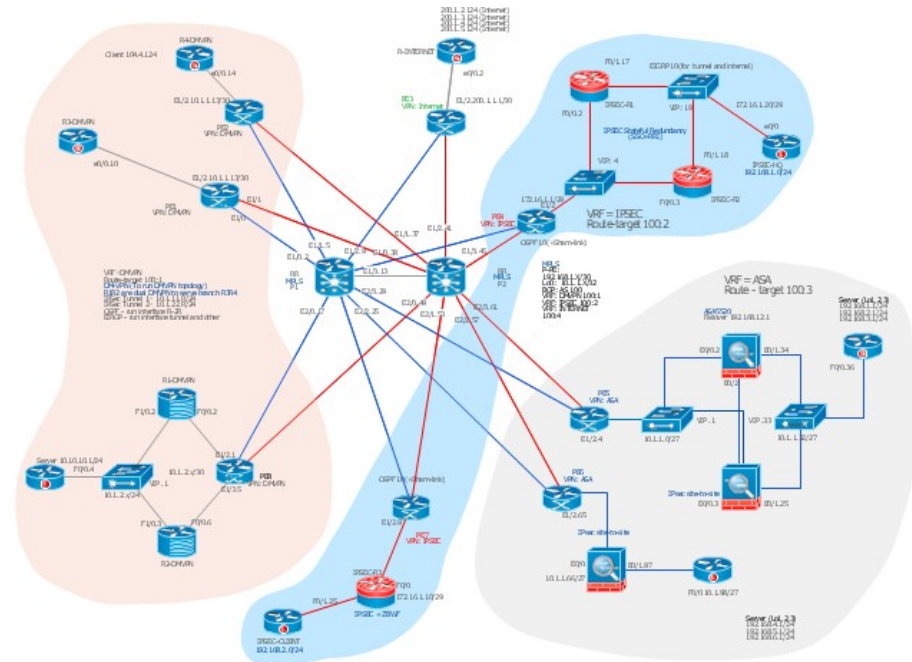
# Introduction

- **Issues:**
  - Disparity: Noob computers running Windows, proper computers running variants of Unix (OSX, Linux, iOS, Android, etc.). Little-endian ARM-based embedded systems, big-endian Intel-based systems, fast computers, slow computers, etc.
  - Etc., etc., etc.



# The Internet

- **The internet is designed to solve these problems, and most of all is designed to be resilient:**
  - Originally “ARPANET”, designed to link U.S. universities, government organizations and the military together.
  - Designed to withstand severe damage to infrastructure, e.g. in a nuclear attack.
- **Achieved through massive network of “routers”:**
  - Specially designed computers that route packets of data around the world.





# The Internet

- **Traffic is carried across the world through undersea cables, overland cables and satellite.**

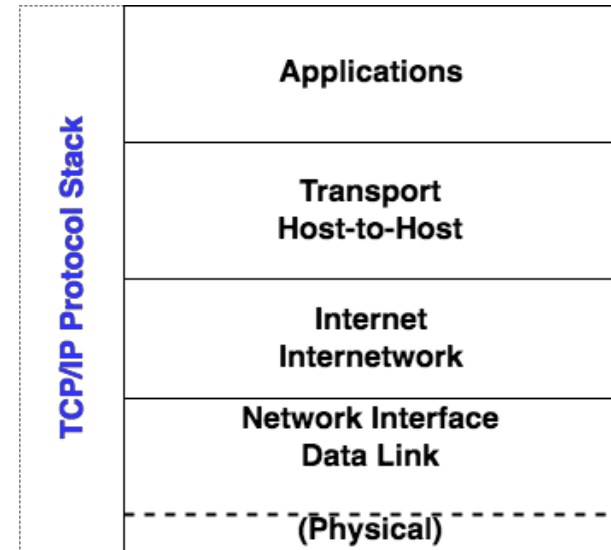


- **September 4, 2017**
  - Severe storms off Hong Kong and Macau severe the SEA-ME-WE3 undersea cable, that carries Internet traffic from Singapore and many other countries in the region.
  - Traffic re-routed through other cables, and satellite links.
  - Main effect: Slower access to servers in the U.S. and Europe, but still have access!



## The Internet: TCP/IP

- **TCP/IP is the main protocol “suite” (i.e. collection of protocols) for the internet.**
  - TCP – Transport Layer Protocol.
  - IP – Internet Protocol.
- **The TCP/IP “stack” consists of four main layers:**
  - **Physical:** Defines voltage levels, types of wire to use (e.g. UTP, STP, co-axial, fibre, etc.). Also includes the link layer that defines how to communicate from point to point.
  - **Network:** Defines how to send a packet across switches.
  - **Transport:** Defines how to get a packet from one host to another.
  - **Application:** Defines how an application exchanges information with its server, etc.



## Secure Networking

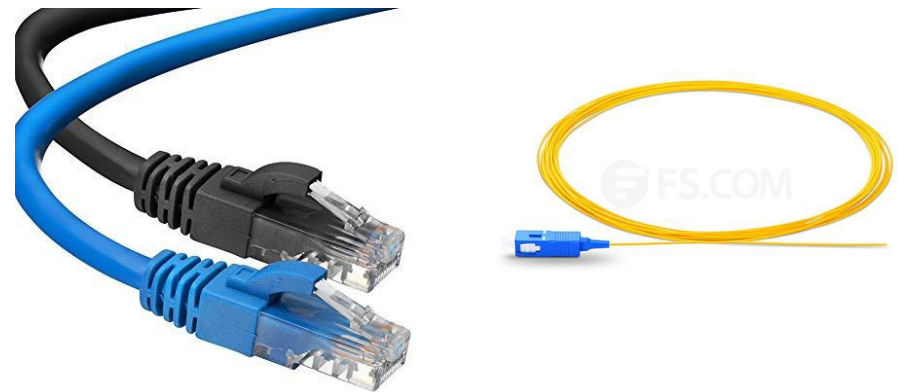
# TCP/IP: THE PHYSICAL LAYER

# The Physical Layer

- **The “physical layer” actually covers both the hardware specifications and the link layer specifications:**
  - This is very similar to the USART protocol that we looked at earlier.
- **TCP/IP however runs across many different kinds of hardware:**
  - Ethernet / WiFi – Connects computers within an organization or home.
  - Fibre broadband – Connects most homes (in Singapore) to the ISP.
  - High Capacity (e.g. T-Carrier) Links – Connects ISPs and large organizations to the rest of the internet.
  - Etc.

# The Physical Layer

- **All have different physical and link layer characteristics:**
  - **Ethernet:**
    - ✓ **Physical:** Unshielded twisted pair, +/- 2.5 volts, data bits encoded using Manchester Coding.
    - ✓ **Link Layer:** Carrier Sense Multiple Access with Collision Detection and Exponential Back-off (CSMA/CD).
  - **Fibre Broadband:**
    - ✓ **Physical:** Uses glass cables and lasers.
    - ✓ **Link Layer:** Various (FTTX)

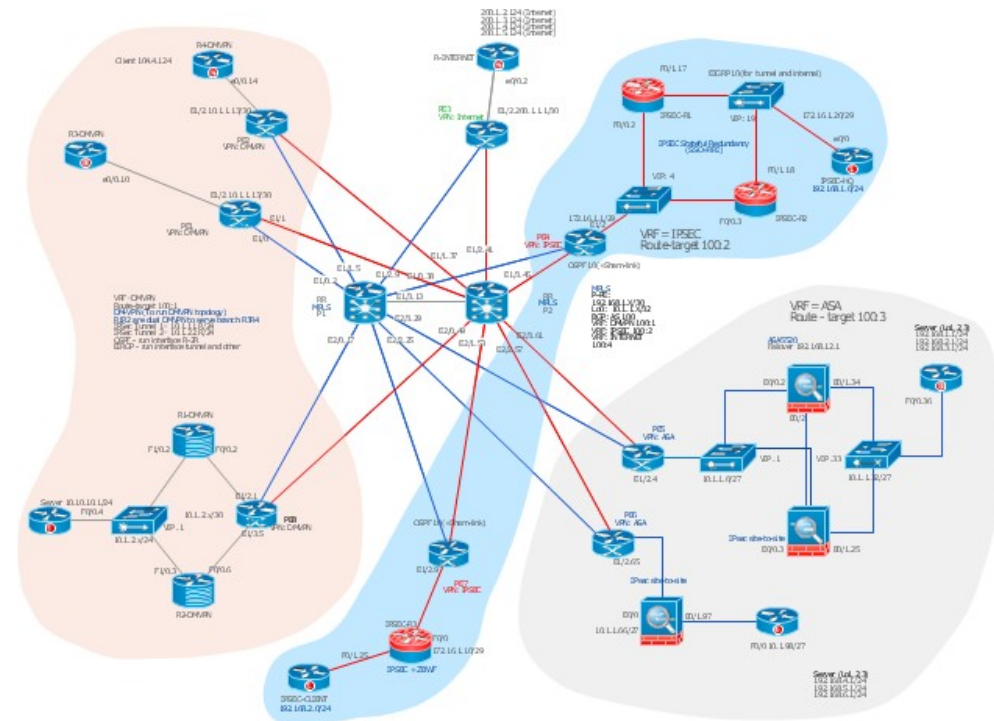


## Secure Networking

# TCP/IP: THE NETWORK (IP) LAYER

# The Network Layer

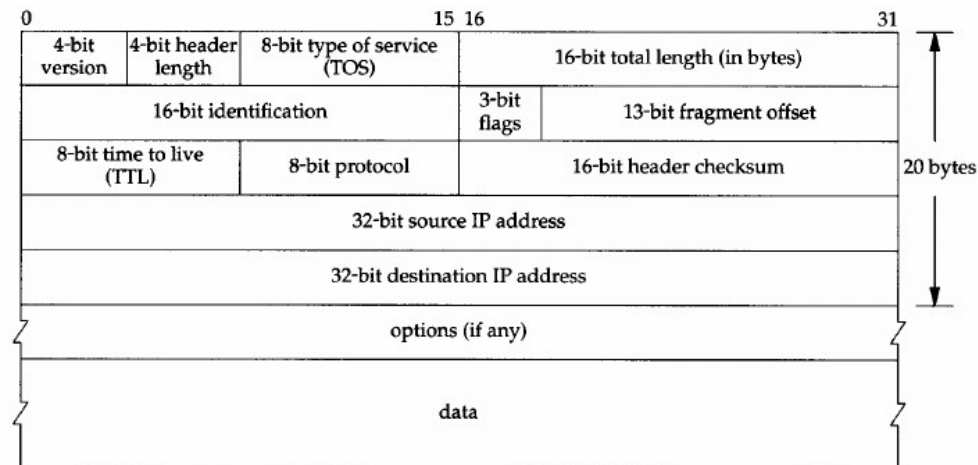
- The physical layer connects computers to the ISP, the ISP to other routers, the routers to each other, etc.
- But we need to figure out how to send a packet from one end to another.
- Note:
  - There are many routes!



## The Network Layer: IP Protocol

- The IP Protocol is a standard specification of algorithms and packet formats on how to get data from one point to another (e.g. from a computer to a router, from one router to another, etc.)
- Conceptually similar to the Alex Protocol that we saw for communication between the Pi and the Arduino (except that the Alex Protocol doesn't do routing).
- The “data” portion of the packet carries user data. E.g. web pages, etc.

IP Header





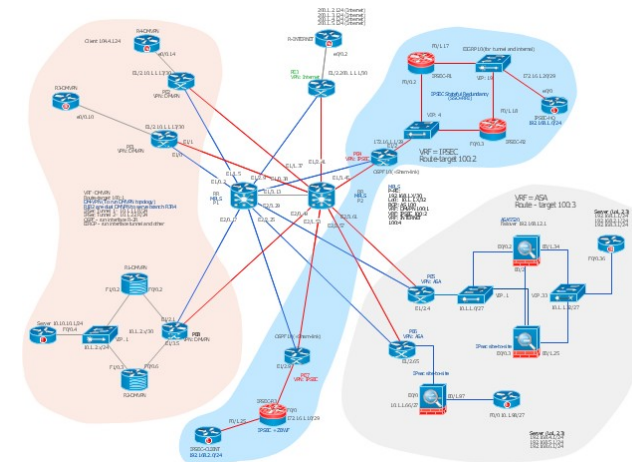
# The Network Layer: IP Protocol

- Routers look at the destination address field, and consult internal "routing tables" to decide which router to forward to.**

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	71.46.14.1	0.0.0.0	UG	0	0	0	ppp0
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	eth0
71.46.14.1	0.0.0.0	255.255.255.255	UH	0	0	0	ppp0
169.254.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
172.16.0.0	0.0.0.0	255.240.0.0	U	0	0	0	eth0
192.168.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0
192.168.1.0	192.168.96.1	255.255.255.0	UG	0	0	0	eth0
192.168.96.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

- This is repeated until the packet reaches its destination.**



## Secure Networking

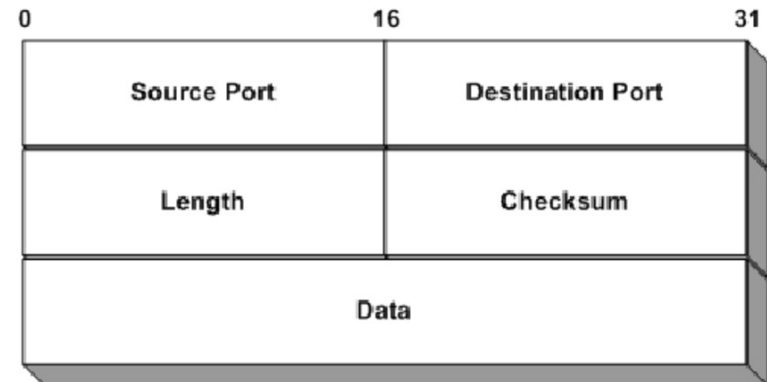
# TCP/IP: THE TRANSPORT (TCP / UDP) LAYER

## Transport Layer

- **The IP Layer routes packets from one end to another, but is mostly concerned with how to get a packet from one router to another until it reaches the end.**
- **The IP Layer DOES NOT guarantee delivery!**
  - Packets can be dropped any time in between due to:
    - ✓ **TTL exceeded (prevents packets from being circulated indefinitely).**
    - ✓ **Lack of space in the router's buffers.**
- **The IP layer also does not deal with “ports”:**
  - A port is a unique integer identifier that identifies which application on the host is to receive the packet.
  - Otherwise:
    - ✓ **If you have ssh and Chrome running, which app should we send a data packet to?**
- **The transport layer deals with these problems.**

## Transport Layer: User Datagram Protocol (UDP)

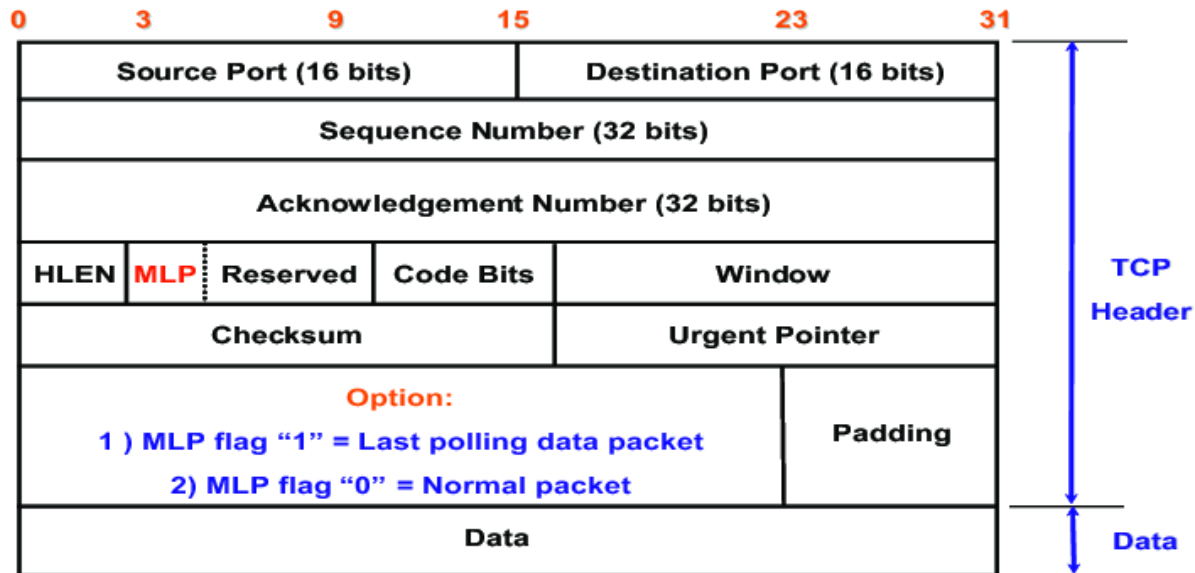
- **This is the simplest transport layer protocol:**
  - Just provides source and destination port numbers to allow data to be sent to the correct applications.
  - Also has a checksum field to detect errors in the data.
- **Best-effort delivery:**
  - Doesn't know whether a packet has been dropped.
  - Also quietly drops packets with bad checksums.
- **Fast, low overhead, useful in video streaming where losing data is not that big a deal.**
- **User data is put into the “Data” part of the UDP packet, which is in turn put into the “Data” part of the IP packet.**



## Transport Layer: Transport Control Protocol (TCP)

- **UDP is satisfactory for streaming:**
  - Most streamed data has enough redundancy to tolerate dropped packets.
- **UDP is terrible for most other things:**
  - Imagine having parts of your web-page (or commands to Alex) being randomly dropped!
- **The TCP protocol adds in “flow control” that:**
  - Adds “acknowledgement” (ACK) and “negative acknowledgement” (NAK) information to packets, to acknowledge previous packets, or request re-sends of previous packets (NAK).
  - Computes checksums and requests for the packet to be re-sent if the packet is corrupted.
  - Detects dropped packets through time-outs.
- **Collectively this is known as “ARQ” – Acknowledgement Request.**

## Transport Layer: Transport Control Protocol (TCP)



- The user data goes into the **Data** part of the TCP packet.
- As with UDP, the TCP packet is in turn stuffed into the “data” portion of the IP packet.

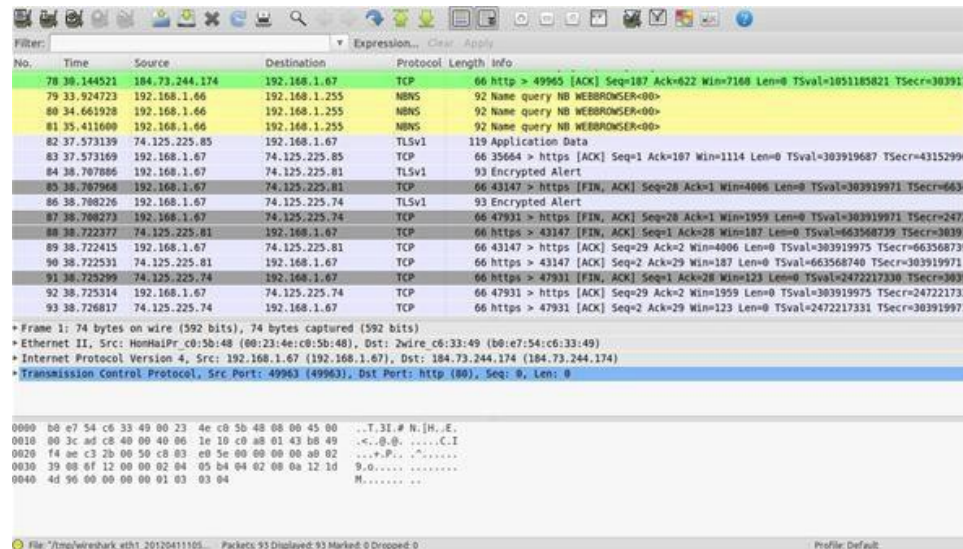
## Secure Networking

# TCP/IP: LACK OF SECURITY



## Securing Alex's Data

- Now we can see how TCP/IP transports data across the world, and gives us some potential to control Alex from any part of the world.
  - May need to use reverse-tunnels, etc. But let's leave that to another battle.
- The problem is that the data is not secure:
  - By using “packet sniffers” you can see what is being transmitted.
  - By doing “DNS poisoning” or hijacking IP addresses, we can launch “man-in-the-middle” attacks.



No.	Time	Source	Destination	Protocol	Length	Info
78	30.144521	184.73.244.174	192.168.1.67	TCP	66	http > 49965 [ACK] Seq=187 Ack=622 Win=7168 Len=0 TSval=1051185821 TSecr=30391971
79	33.924723	192.168.1.66	192.168.1.255	NBNS	92	Name query NB WEBBROWSER=00>
80	34.661928	192.168.1.66	192.168.1.255	NBNS	92	Name query NB WEBBROWSER=00>
81	35.411600	192.168.1.66	192.168.1.255	NBNS	92	Name query NB WEBBROWSER=00>
82	37.573139	74.125.225.85	192.168.1.67	TLSv1	119	Application Data
83	37.573169	192.168.1.67	74.125.225.85	TCP	66	35664 > https [ACK] Seq=1 Ack=187 Win=1114 Len=0 TSval=303919687 TSecr=43152996
84	38.707866	192.168.1.67	74.125.225.81	TLSv1	93	Encrypted Alert
85	38.707868	192.168.1.67	74.125.225.81	TCP	66	43147 > https [FIN, ACK] Seq=28 Ack=1 Win=4006 Len=0 TSval=303919971 TSecr=6634
86	38.708226	192.168.1.67	74.125.225.74	TLSv1	93	Encrypted Alert
87	38.708273	192.168.1.67	74.125.225.74	TCP	66	47931 > https [FIN, ACK] Seq=28 Ack=1 Win=1959 Len=0 TSval=303919971 TSecr=2472
88	38.722377	74.125.225.81	192.168.1.67	TCP	66	https > 43147 [FIN, ACK] Seq=1 Ack=28 Win=187 Len=0 TSval=663568739 TSecr=30391
89	38.722415	192.168.1.67	74.125.225.81	TCP	66	43147 > https [ACK] Seq=29 Ack=2 Win=4006 Len=0 TSval=303919975 TSecr=663568735
90	38.722531	74.125.225.81	192.168.1.67	TCP	66	https > 43147 [ACK] Seq=2 Ack=29 Win=187 Len=0 TSval=663568740 TSecr=303919971
91	38.725299	74.125.225.74	192.168.1.67	TCP	66	https > 47931 [FIN, ACK] Seq=1 Ack=28 Win=123 Len=0 TSval=2472217330 TSecr=3035
92	38.725314	192.168.1.67	74.125.225.74	TCP	66	47931 > https [ACK] Seq=29 Ack=2 Win=1959 Len=0 TSval=303919975 TSecr=247221733
93	38.726817	74.125.225.74	192.168.1.67	TCP	66	https > 47931 [ACK] Seq=2 Ack=29 Win=123 Len=0 TSval=2472217331 TSecr=303919971

\* Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)  
 \* Ethernet II, Src: HonHaiPr.c0:5b:48 (00:23:4e:c0:5b:48), Dst: Zwire.c6:33:49 (b6:e7:54:c6:33:49)  
 \* Internet Protocol Version 4, Src: 192.168.1.67 (192.168.1.67), Dst: 184.73.244.174 (184.73.244.174)  
 \* Transmission Control Protocol, Src Port: 49965 (49963), Dst Port: http (80), Seq: 0, Len: 0

```

0000  b0 e7 54 c6 33 49 00 23 4e c0 5b 48 00 00 45 00  ..T.31.#N.[H..E.
0010  00 3c ad c8 40 00 00 06 1e 10 c0 a0 01 43 b8 49  .<..B. ....C.I
0020  f4 ae c3 2b 00 50 c8 03 e0 5e 00 00 00 00 a0 02  ...P. ....
0030  39 08 8f 12 00 00 02 04 05 b4 04 02 00 0a 12 1d  9. ....
0040  4d 56 00 00 00 00 01 03 04  ..M.....
  
```

File: "http://wireless\_ath1\_20120411105..." Packets: 93 Displayed: 93 Marked: 0 Dropped: 0 Profile: Default

## Secure Networking

# CRYPTOGRAPHY: BASIC TERMS

# Basic Concepts: Terms

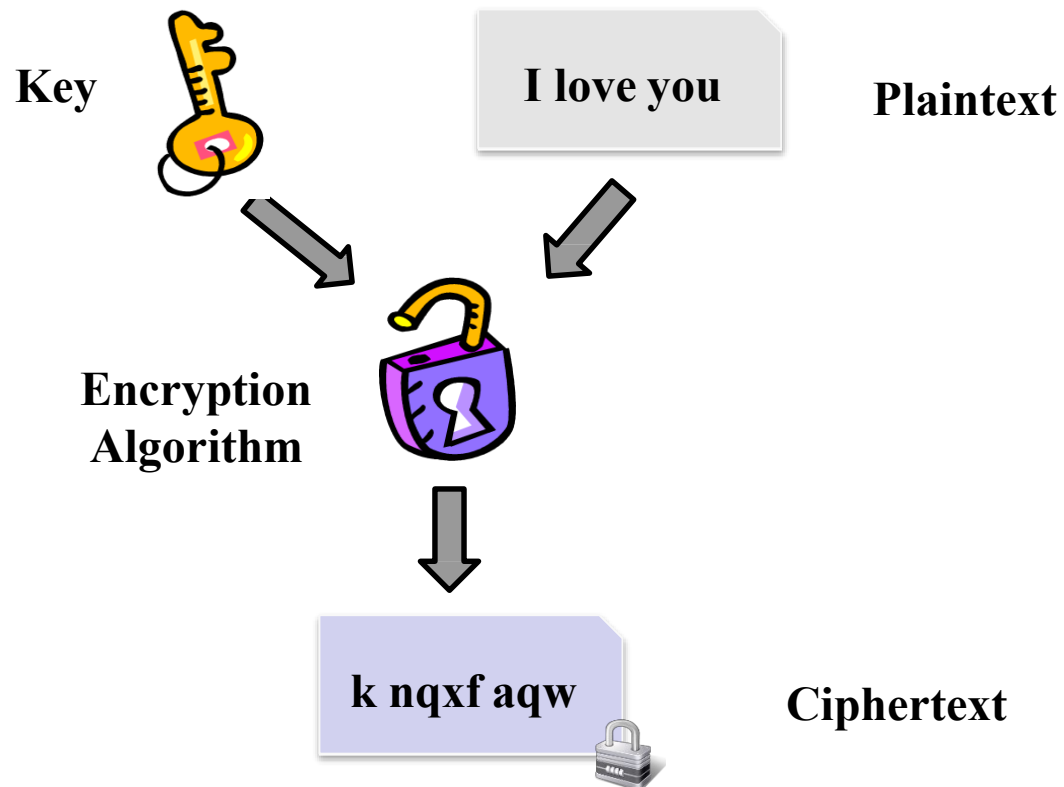
- **Basic terms:**
  - **Plaintext:** "Clear" data that can be intercepted and understood by anyone.
  - **Ciphertext:** "Scrambled" data that is generated from "encryption" algorithms to prevent plaintext from being understood.
  - **Encryption:** Converting from plaintext to ciphertext.
  - **Decryption:** Converting from ciphertext to plaintext.
  - **Key:** A particular value or string that is used to encrypt/decrypt messages.
  - **Cipher, cryptosystems:** Other names for encryption/decryption algorithms.
  - **Symmetric Cipher:** The same key is used to encrypt and decrypt. Also known as private key cryptosystems.
  - **Asymmetric Cipher:** A different key is used for encryption (the "public" key) than for decryption (the "private" key).

## Secure Networking

# CRYPTOGRAPHY: SYMMETRIC (PRIVATE KEY) CRYPTOSYSTEMS

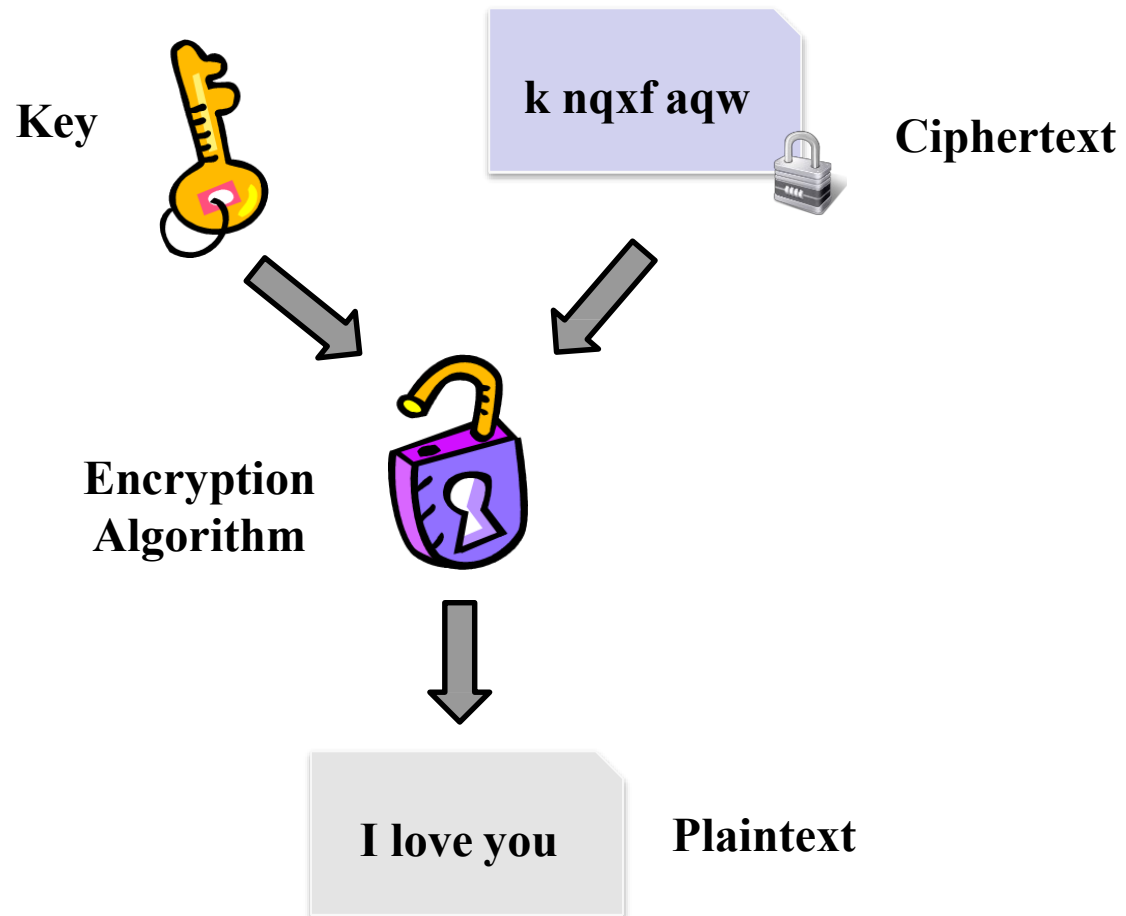
# Private Key Cryptosystems

- Encryption:**



# Private Key Cryptosystems

- Decryption:**



## Example

- **We will look at an example symmetric cipher – “substitution cipher”**
  - Step 1: Decide on a key. We will use “THIS IS THE COOLEST COURSE!”
  - Step 2: Remove all duplicate letters to get the shortened key “THISECOLUR”
  - Step 3: Tell your friend your shortened key.
  - Step 4: Both of you create the following substitution table:

A	B	C	D	E	F	G	H	I	J
T	H	I	S	E	C	O	L	U	R

K	L	M	N	O	P	Q	R	S	T
A	B	D	F	G	J	K	M	N	P

U	V	W	X	Y	Z				
Q	V	W	X	Y	Z				



## Example

- **Send a message “I LOVE TLS” to your friend, using the earlier table to do substitutions.**
  - We look up plaintext letters in the top row, and substitute with ciphertext letters in the bottom row.
  - ✓ **Substitute U for I, B for L, etc, to get U BGVE PBN.**

A	B	C	D	E	F	G	H	I	J
T	H	I	S	E	C	O	L	U	R

K	L	M	N	O	P	Q	R	S	T
A	B	D	F	G	J	K	M	N	P

U	V	W	X	Y	Z				
Q	V	W	X	Y	Z				

## Secure Networking

# CRYPTOGRAPHY: ASYMMETRIC (PUBLIC KEY) CRYPTOSYSTEMS

# Public Key Cryptosystems

- **A public key cryptosystem (PKC) is one where:**
  - Both parties (let's call them Alice and Bob) have two pairs of keys.
  - Each pair consists of:
    - ✓ A “public key” that is well known and publicized to everyone. This key is use for encrypting messages.
    - ✓ A “private key” that is, well, private and known only to the owner. This key is used for decrypting messages.
  - This solves our key distribution problem!
    - ✓ We can just post our public key on our website, or email it freely to anyone who wants to send us secure messages.
    - ✓ Transport Layer Security does all of this for you very elegantly (see later).
  - Let's look at an example of Alice and Bob communicating, with Eve eavesdropping (haha).



## Public Key Cryptosystems: Example

- **The most common PKC is the Rivest-Shamir-Addleman (RSA) algorithm, and we will look at that.**
- **Another common example is Elliptic Curve Cryptography (ECC) which is harder to understand so we will not look at it.**
- **It is based on:**
  - **Choosing two VERY LARGE prime numbers  $p$  and  $q$ .**
    - ✓  **$p$  and  $q$  are your private keys.**
    - ✓ **In most secure systems,  $p$  and  $q$  are between 1024 and 8192 bits long.**
    - ✓ **This means that  $p$  and  $q$  are in the order of at least  $2^{1024}$ , to as big as  $2^{8192}$ .**
  - **Compute  $N = p * q$ .**
    - ✓ **This is the public key.**
  - **Because  $N$ ,  $p$  and  $q$  are very large, deriving  $p$  and  $q$  from  $N$  is extremely difficult.**

## Public Key Cryptosystems: Example

- You can try the following if you aren't convinced. 😊

- ▶  $N = 143$      $p = \underline{\hspace{2cm}}$      $q = \underline{\hspace{2cm}}$
- ▶  $N = 8633$      $p = \underline{\hspace{2cm}}$      $q = \underline{\hspace{2cm}}$
- ▶  $N = 175,828,273$      $p = \underline{\hspace{2cm}}$      $q = \underline{\hspace{2cm}}$

- PKC is thus considered to be “practically unbreakable”
  - You can brute-force search for  $p$  and  $q$ , but it will take too long to be useful.
  - Again this could change with the dawn of quantum computing.

## Public Key Cryptosystems: Example

- **Now let's look at a “real” example. Let's suppose Alice (who has a balding problem) wants to send the number of strands left on her head to Bob, but does not want Eve to know (the horror!). Now let's suppose she has 12 strands of hair left. For that Bob must send Alice his public key.**
  - **Step 1: Choose two very large numbers  $p$  and  $q$ , and compute the “modulus”  $n$ .**
    - ✓ **Here we choose 3 and 5, giving  $n = 15$ . (I know this is not large, but we also want to keep our sanity while working out an example.)**
  - **Step 2: Compute our “totient”,  $t(n) = (p - 1) * (q - 1)$** 
    - ✓  **$t(15) = (3 - 1) * (5 - 1) = 2 * 4 = 8$**
  - **Step 3: Choose an  $e > 1$ , such that  $\gcd(e, t(n)) = 1$ . Our public key is  $(e, n)$ .**
    - ✓ **We choose  $e = 7$ . We can verify that  $\gcd(7, 8) = 1$ . This gives our public key  $(7, 15)$ .**
    - ✓ **Bob sends over  $(7, 15)$  to Alice and is intercepted by Eve. Now both Alice and Eve know that Bob's public key is  $(7, 15)$ .**



## Public Key Cryptosystems: Example

▪ Step 4: Choose a  $d$  such that  $d * e = 1 + k * t(n)$ , where  $k$  is arbitrary, chosen so that  $d$  is integral. Your private key is now  $(d, n)$ .

✓ We choose  $k$  to be 20. This gives us  $7d = 1 + 20 * t(n)$ , which gives us  $7d = 161$ .

✓ Solving for  $d$  gives us  $d = 23$ . Bob's private key is  $(23, 15)$ .

▪ Step 5: Taking our plaintext  $t$ , we compute the ciphertext  $c = t^e \bmod n$ , where  $e$  and  $n$  come from the public key  $(e, n)$ . Send  $c$  over.

✓ Bob's public key as sent earlier was  $(7, 15)$ .

✓ Alice computes  $c = 12^7 \bmod 15 = 3$ .

✓ Alice sends 3 over to Bob.

▪ Step 6: On the receiving end, compute  $t = c^d \bmod n$ , where  $d$  and  $n$  are from the recipient's private key  $(d, n)$ :

✓ Bob's private key is  $(23, 15)$ .

✓ Bob computes  $t = 3^{23} \bmod 15 = 12$ , giving him the number of strands of hair that Alice has left.

## Secure Networking

# CRYPTOGRAPHY: SIGNING MESSAGES

# Public Key Cryptography: Digital Signatures

- **There is a neat feature of PKC:**
  - We normally encrypt using the public key and decrypt using the private key.
  - BUT:
    - ✓ If we encrypt using the private key, we can ALSO decrypt using the public key!
- **We can try this out:**
  - Bob starts with his key-pair from earlier on: Private key = (23, 15), public key = (7, 15).
  - Bob encrypts  $m=3$  with his private key:
    - ✓  $c = 3^{23} \bmod 15 = 2$
  - Bob sends  $(m, c) = (3, 2)$  over to Alice.
  - Alice computes  $m'$  using Bob's public key (7, 15):
    - ✓  $m' = 2^7 \bmod 15 = 3$
  - Alice sees that  $m' == m$ , and thus knows that  $m$  was encrypted using Bob's private key (otherwise she can't decrypt with his public key).

## Public Key Cryptography: Digital Signatures

- **Consider what this means:**
  - If  $m == m'$ , this means that Alice successfully decrypted  $c$  using Bob's public key.
  - This means that Bob must have encrypted using his private key.
  - Since only Bob knows the private key, then the message  $(m, c)$  **MUST** come from Bob, and not from someone pretending to be Bob!
- **This is called a digital signature!**
  - Note: The message  $(m, c)$  can be encrypted with Alice's public key to keep it secure.

## Public Key Cryptography: Digital Signatures

- **Note that generally  $m$  can be quite large (e.g. a public key), and thus generating  $c$  can be very inefficient.**
  - Bob uses a “hash function” like SHA256 to generate  $h(m)$ , where  $h(m)$  is a much shorter summary of  $m$ .
  - Bob encrypts  $h(m)$  to get  $c$ . He sends  $(m, c)$  as usual over to Alice.
  - Alice receives  $(m, c)$ , then computes  $h(m)$ . She decrypts  $c$ , and if the result is equal to  $h(m)$ , she knows that Bob sent her the message.
- **Some important properties of hash functions (also called digest functions):**
  - Given  $h(m)$ , it must be impossible to derive  $m$ .
  - Given even the tiniest change to  $m$ ,  $h(m)$  must change drastically.
  - It must be non-trivial to produce  $h(m)$  from  $m$ .
  - No matter how big  $m$  is,  $h(m)$  always produces a hash of exactly the same size.
  - The “gold standard” in hash functions is SHA256.

## Public Key Cryptography: Digital Signatures

- **Digital signatures are used primarily to distribute public keys.**
  - Given Bob's public key  $PUB$ , he computes  $h(PUB)$ .
  - Bob encrypts  $h(PUB)$  using his private key  $PRV$ , giving  $c$ .
  - Bob distributes  $(PUB, h(PUB), c)$  to everyone (using TCP/IP, for example).
  - Alice can:
    - ✓ **Compute  $h(PUB)$**
    - ✓ **Decrypt  $c$  using  $PUB$**
    - ✓ **If the result is  $h(PUB)$ , Alice knows that  $c$  was generated using  $PRV$ , and since only Bob knows  $PRV$ ,  $(PUB, h(PUB), c)$  must come from Bob!**
- **The message  $(PUB, h(PUB), c)$  is called a “certificate”.**

## Secure Networking

# CRYPTOGRAPHY: GETTING TRUSTED PEOPLE TO VERIFY SIGNATURES

## Public Key Cryptography: Certificate Authorities

- **We still have one more problem:**
  - Alice has never met Bob, does not know what he looks like or what he sounds like. So she can't video-conference or call Bob to verify that he really did send the certificate.
  - Is this a problem? Consider:
    - ✓ Eve intercepts Bob's certificate, and sends her own fake certificate to Alice.
    - ✓ Now Alice will use Eve's fake certificate to encrypt her message. Eve can then decrypt Alice's message, read it, before re-encrypting using Bob's real certificate and sending it to him. Note that Eve can also change the message before sending!
    - ✓ This is called “(wo)man in the middle attack”.



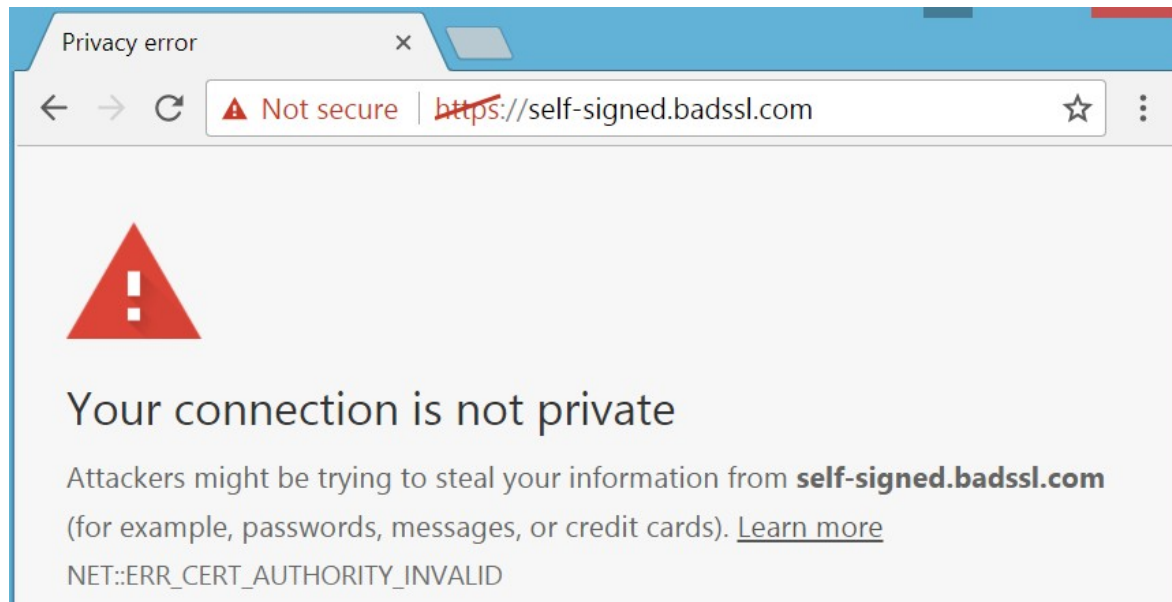
## Public Key Cryptography: Certificate Authorities

- **Fortunately both Alice and Bob have a common friend Charlie, whom they trust with their lives.**
  - Now Bob can send his certificate to Charlie.
  - Charlie knows it is Bob's certificate, and he will sign the certificate with his (Charlie's) private key before sending it to Alice.
  - Alice will then use Charlie's public key to verify that Bob's certificate was indeed signed by Charlie.
    - ✓ **This means that Alice must have Charlie's certificate.**
  - Since Alice trusts Charlie, she is confident that the certificate is from Bob.

# Public Key Cryptography: Certificate Authorities

- **Charlie is called a “Certificate Authority” or CA.**
- **Very famous Internet CAs include Globalsign, Verizon and Digicert.**
- **In practice a CA will require identification documents from you before signing your certificates.**
  - They also require plenty of \$\$\$\$. Typical price is about \$1,000 to \$5,000 a year.
- **All modern browsers hold certificates from various CAs.**
  - They extract the CA’s public key from these certificates and use these to check the certificates from other websites.
  - If the website certificate is not signed by a recognized CA, the browser will refuse to load the pages (see next page).
- **In the studio you will be generating “self-signed certificates”.**
  - Mostly because we don’t have the budget to get Verizon to certify 40 certificates (will cost between \$40,000 and \$200,000.)
  - But you will also learn how to become a CA! :D

# Public Key Cryptography: Certificate Authorities



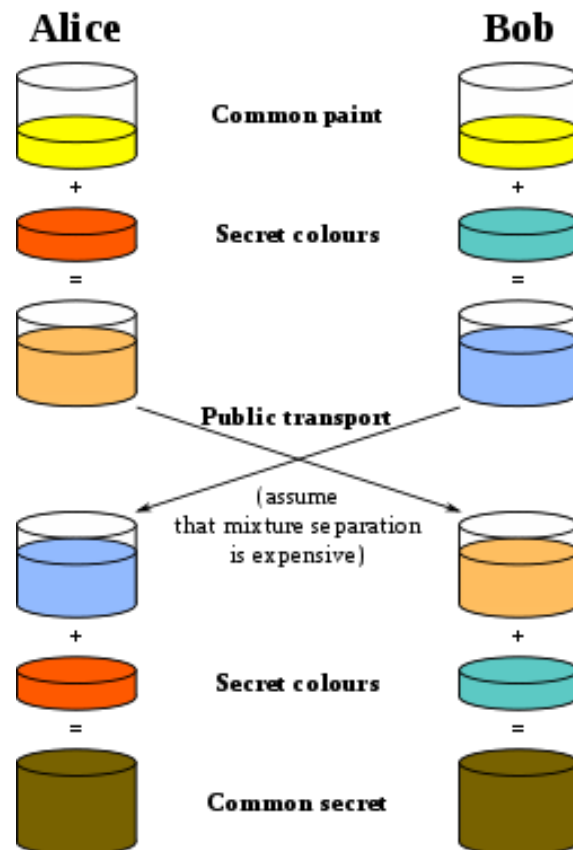
## Secure Networking

# CRYPTOGRAPHY: KEY EXCHANGE

## Public Key Cryptosystems: Issues

- **The problem with PKC is that it generates VERY LARGE ciphers.**
  - If we use a 2048 bit modulo (considered to be the minimum for secure systems), then every byte of plaintext will produce 256 bytes of ciphertext! (2048 bits = 256 bytes).
- **We use a hybrid system!**
  - We generate a key  $k$  for a symmetric cipher (efficient), then use PKC to transmit  $k$ .
  - Once both sides have  $k$ , use a symmetric cipher for the rest of the communications!
- **Computer Scientists have come up with ways for both Alice and Bob to derive the key  $k$ , without actually having to send  $k$  over!!**
  - We will look at an example called “Diffie-Hellman Key Exchange”. Sometimes called “Diffie-Hellman-Merckle Key Exchange”.

# Public Key Cryptosystems – Key Exchange



# Public Key Cryptosystems – Diffie-Hellman Key Exchange

- **Suppose Alice and Bob want to exchange information using a symmetric cipher (because asymmetric ciphers are too large), but have no way to exchange keys securely.**
- **They use the Diffie-Hellman key exchange algorithm:**
  - Alice and Bob agree on two numbers  $g$  and  $p$ :
    - ✓  $g$  is a prime base.
    - ✓  $p$  is a prime modulus.
    - ✓  $g$  must be a primitive root modulo  $p$ .
    - ✓ Here Alice and Bob agree on  $g = 5$  and  $p = 23$ .
  - Alice and Bob each maintain their own secret numbers:
    - ✓ Alice's secret number  $a = 6$ .
    - ✓ Bob's secret number  $b = 15$ .
    - ✓ It should be obvious that  $a$  is known only to Alice and  $b$  only to Bob.

# Public Key Cryptosystems – Diffie-Hellman Key Exchange

- Alice computes  $A = g^a \bmod p$  and sends it publicly to Bob.  
✓  $A = 5^6 \bmod 23 = 8$
- Bob computes  $B = g^b \bmod p$  and sends it publicly to Alice.  
✓  $B = 5^{15} \bmod 23 = 19$
- Now it's time for Alice to get the shared key from the number B shared by Bob! She computes  $S(\text{Alice}) = B^a \bmod p$ :  
✓  $S(\text{Alice}) = 19^6 \bmod 23 = 2$
- Bob will do the same with the number A shared by Alice. He computes  $S(\text{Bob}) = A^b \bmod p$ :  
✓  $S(\text{Bob}) = 8^{15} \bmod 23 = 2$
- Notice how, just by agreeing on p and g, and maintaining their own respective secret numbers, Alice and Bob are able to derive the exact same shared key! :D
  - In Computer Science, this is called “magic”. :D



## Secure Networking

# CRYPTOGRAPHY: TRANSPORT LAYER SECURITY

## Putting it All Together – Transport Layer Security

- **Transport Layer Security (TLS – Formerly known as Secure Socket Layer or SSL) is a Presentation Layer protocol that combines:**
  - Public Key Cryptography.
  - Private Key Cryptography.
  - Digital Signatures, Certificates and Certificate Authorities.
  - Key Exchange Mechanisms.
  - Entirely over TCP/IP!
- **Familiar “applications” like SSH and HTTPS all use TLS.**
- **Let’s look at how a secure session is set up using TLS:**
  - Note: SSL v2.0 and v3.0 are insecure and have been broken. You should only use TLS v1.0 upwards.

# Transport Layer Protocol

## The Handshake

- **Suppose Alice and Bob communicate using TLS. This is what TLS does in the “handshake” where they exchange session keys for use in symmetric ciphers:**
  - Bob sends Alice a HELLO message, listing all the symmetric ciphers he understands. Some examples include AES (used in WiFi), DES, Blowfish, etc.
  - Alice sends back her own HELLO message, telling Bob with symmetric cipher to use.
  - Alice presents her certificate to Bob. Bob uses Charlie’s public key to check the digital signature portion of Alice’s certificate to check that it is valid.
  - Bob generates a random string, and encrypts it using Alice’s public key, which he got from her certificate.
  - He sends the encrypted string to Alice, and they both use this to generate a common session key  $S$ . An algorithm like Diffie-Hellman may be used here. Bob uses  $S$  and the selected cipher

# Transport Layer Protocol

## The Handshake

- Bob uses  $S$  and the selected symmetric cipher to send Alice a FINISHED message. If Alice can successfully decrypt the FINISHED message (it is a known message) using  $S$ , she considers the handshaking to be completed.
- Alice also uses  $S$  and the selected symmetric cipher to send Bob a FINISHED message. If Bob can successfully decrypt the FINISHED message, he considers the handshaking to be completed.
- **From here on, Alice and Bob exchange information securely using  $S$  and the symmetric cipher**
- **Note:**
  - This handshaking is done once per session. In the case of ssh this is when you connect to the server. In the case of https this can be as frequent as the number of images, CSS scripts, javascript code, etc. to transfer. Can be very inefficient!
  - Here only Alice presents her certificate to Bob. Bob can also be made to present his certificate for Alice to verify, using Charlie's public key.