

Lecture 1: Encryption

(First step towards security)

- 1.1: Definitions: Encryption/decryption/keys
- 1.2: Classical ciphers + illustration of attacks
- 1.3: Modern ciphers + recommended key length
- 1.4: Attacks on cryptosystem implementations
- 1.5: Kerckhoffs' principle vs security through obscurity
- 1.6: Interesting historical facts

How Important is Encryption?

Hadi Partovi, co-founder of Code.org:

“Encryption is at least as foundational as photosynthesis”



“We don’t teach biology or chemistry to kids because they’re going to become surgeons or chemists.

We teach them about photosynthesis and that water is H₂O, or how lightbulbs work, just to understand the world around us.

You don’t use any of it, but you **do on a day-to-day basis use public-key encryption**”

Increasing Data Protection Need

The Economist

Topics ▾ Current edition More ▾

Subscribe

Regulating the internet giants

The world's most valuable resource is no longer oil, but data

The data economy demands a new approach to antitrust rules

Ref:

<https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data>



Print edition | Leaders >

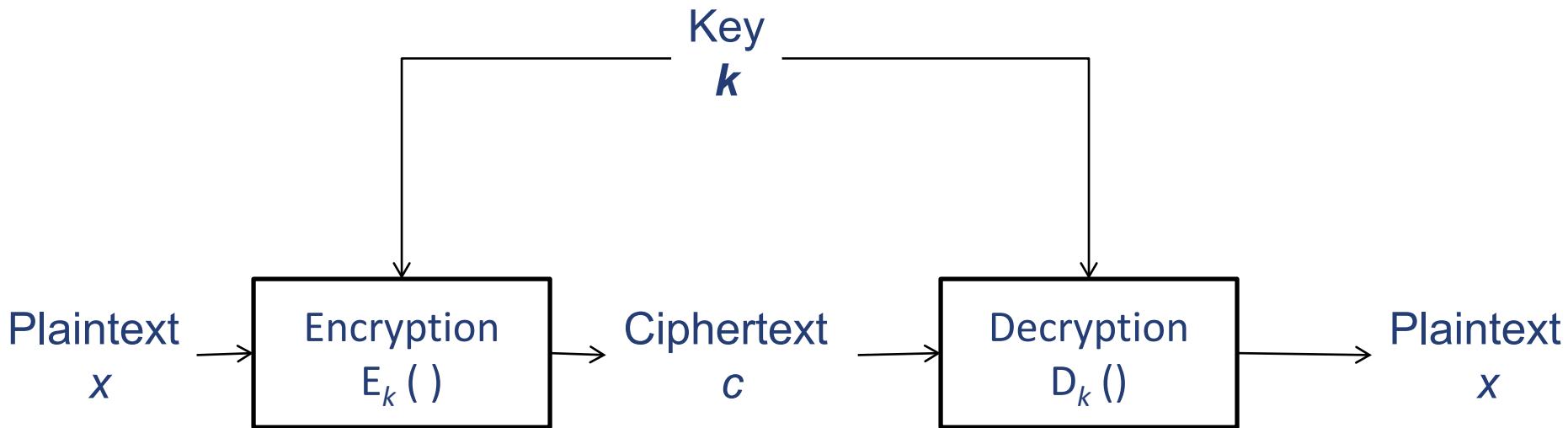
May 6th 2017



1.1 Definitions

Encryption

An **encryption scheme** (also known as **cipher**) consists of two algorithms:
encryption and decryption



Two requirements:

Correctness: For any plaintext x and key k , $D_k(E_k(x)) = x$

Security: Given the ciphertext, it is “difficult” to derive useful information of the key k and the plaintext x . The ciphertext should resemble a sequence of random bytes.

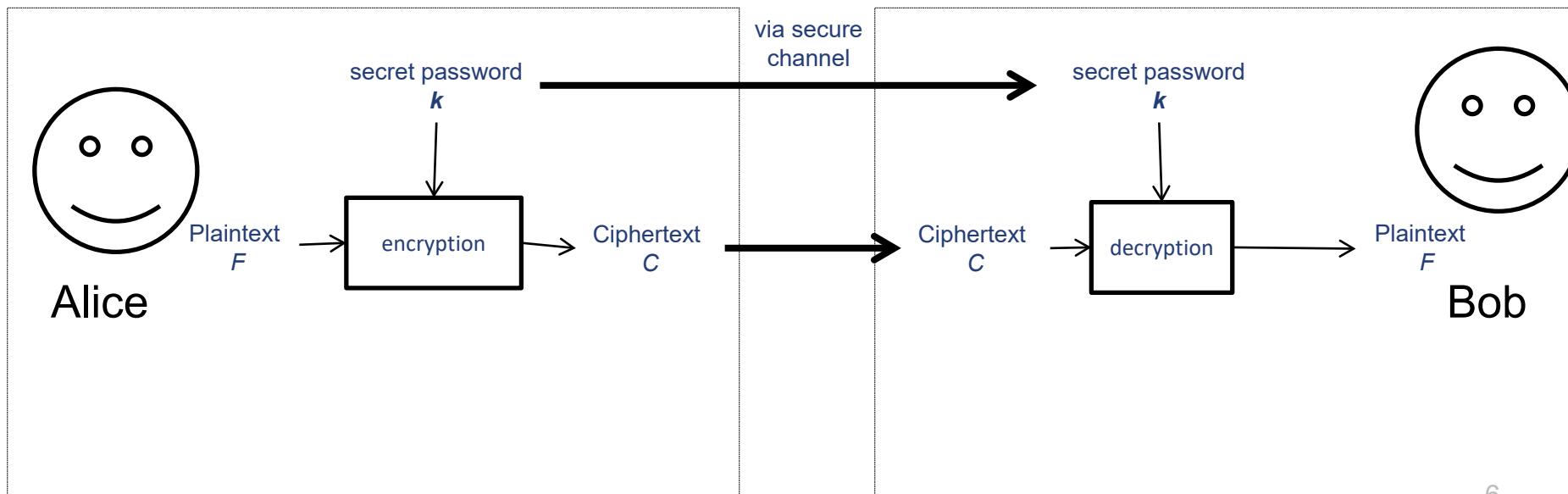
(There are many refined formulations of security requirements, e.g. *semantic security*.

In this module, we will not go into details.

There's also a performance requirement: the encryption and decryption processes can be efficiently computed.)

A Simple Application Scenario

- Alice had a large file F (say an Excel file containing information of her bank accounts and financial transactions).
- She “encrypted” the file F using Winzip with a password “13j8d7wjnd”, and obtained the ciphertext C .
- Next, she called Bob to tell him the password, and subsequently sent the ciphertext to Bob via email attachment.
- Later, Bob received C , and decrypted the ciphertext with the password to recover the plaintext F .



A Simple Application Scenario

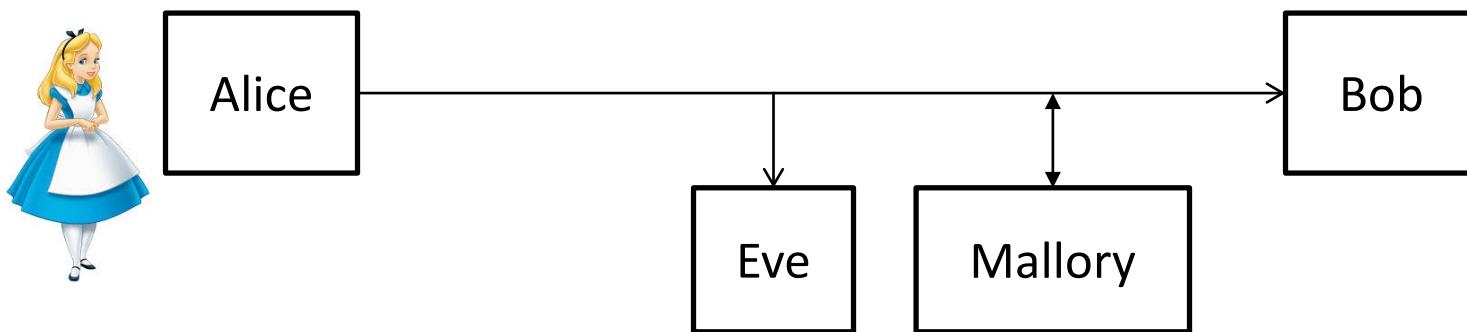
- Anyone who has obtained C , without knowing the password, is unable to get any information on F
- Although C indeed contains information of F , the information is “hidden”
- To someone who doesn’t know the secret, C is just a **sequence of random bits**
- *Remark:*
Winzip is **not** an encryption scheme.
It is an application that employs standard encryption schemes, such as AES.

Cryptography (vs Cryptology?)

- **Cryptography** is the study of techniques in securing communication in the present of *adversaries* who have access to the communication
- Although cryptography is commonly associated with encryption, encryption is *just one* of the primitives in cryptography
- Others include cryptographic hash, digital signature, etc.
- *How about cryptology?*

Characters in Cryptography

- (Terminology) Common placeholders used in cryptography:
 - Alice: usually the originator of message
 - Bob: usually the recipient
 - Eve: eavesdropper, can only listen to sent messages
 - Mallory: malicious, can also modify sent messages



- See the interesting list of crypto characters in:
https://en.wikipedia.org/wiki/Alice_and_Bob
- Depending on context, Alice may not be a human:
she could be the machine that encrypts the message

In this module,

“read”: Part of the teaching materials. Read it.

“see”: Info that is good to know.

“optional”: Optional information.

1.2 Classical Ciphers

For illustration, we will look into a few classical ciphers.

Classical ciphers are **not** secure in the computer era.

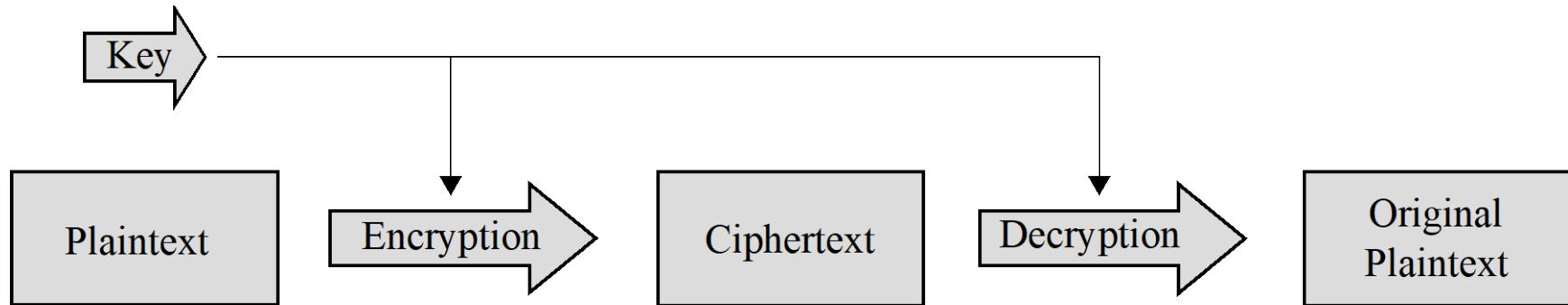
(The exception: the “unbreakable” one-time-pad).

(See <http://ciphermachines.com/index>

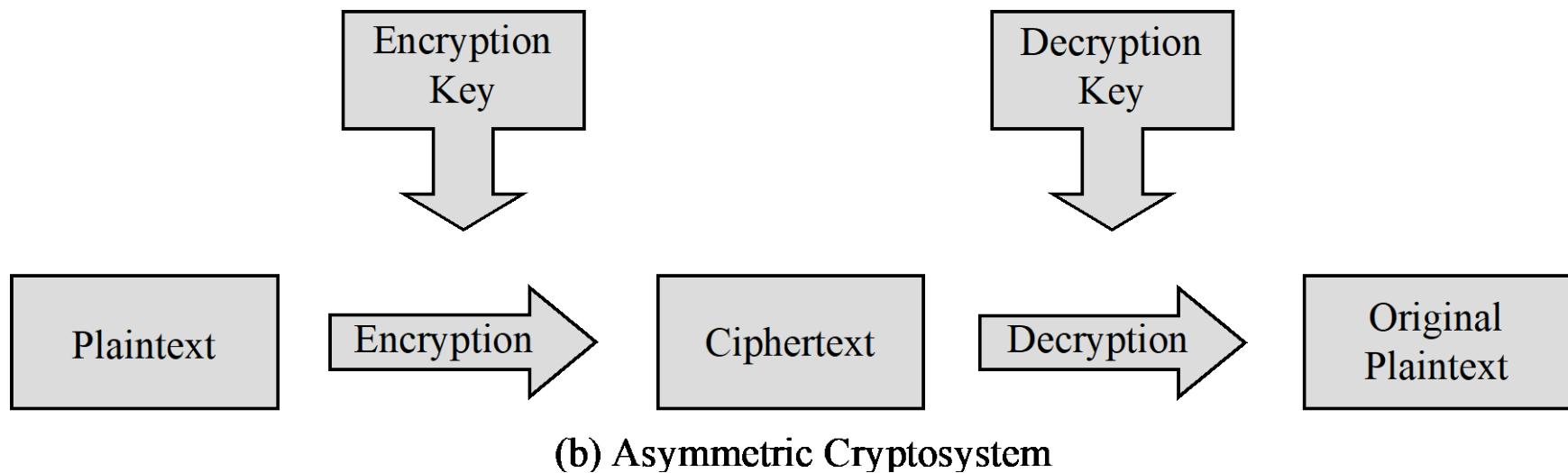
for a good listing of classical ciphers and cipher machines used during WWII.)

- 1.2.1. Substitution Cipher
- 1.2.2. Permutation Cipher
- 1.2.3. One Time Pad

Note: Symmetric vs. Asymmetric Cryptosystems



(a) Symmetric Cryptosystem



(b) Asymmetric Cryptosystem

From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

1.2.1 Substitution Cipher

Substitution Cipher

- **Plaintext** and **ciphertext**: a string over a set of symbols U

E.g.

Let $U=\{\text{“a”, “b”, “c”, …, “z”, “_”}\}$.

Example of plaintext: “hello_world”

- **Key**: a substitution table S ,
representing an 1-1 onto function from U to U

E.g.

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | _ |
| g | v | w | b | n | e | f | h | d | a | t | l | u | c | q | m | z | i | r | s | j | x | o | y | k | _ | p |

$$S(a) = g, \quad S(b) = v, \quad \dots$$

The inverse of S :

$$S^{-1}(g)=a, \quad S^{-1}(v) = b$$

Substitution Cipher

Some terms:

- The ***key space***: the set of all possible keys
- The ***key space size*** or ***size of key space***:
the total number of possible keys
- The ***key size*** or ***key length***: the number of bits required to represent a particular key
- For substitution cipher:
 - The key space?
 - The key space size: $27!$
 - The key size: at least $\log_2(27!) \approx 94$ bits

Substitution Cipher: Encryption/Decryption

Encryption: Given a plaintext of length n , which is a string $X = x_1 x_2 x_3 \dots x_n$, and the key S , output the ciphertext

$$E_S(X) = S(x_1) S(x_2) S(x_3) \dots S(x_n)$$

Example:

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | _ |
| g | v | w | b | n | e | f | h | d | a | t | l | u | c | q | m | z | i | r | s | j | x | o | y | k | - | p |

plaintext: h e l l o _ w o r l d

ciphertext: h n l l q p o q i l b

Decryption: Given a string of ciphertext of length n $C = c_1 c_2 c_3 \dots c_n$ and the key S , output the plaintext

$$D_S(C) = S^{-1}(c_1) S^{-1}(c_2) S^{-1}(c_3) \dots S^{-1}(c_n)$$

Attacks on a Cipher

- In general, the attacker's goal is:
 - To find the key: if the key can be found, then the plaintext can be obtained
(How about the converse?)
 - To obtain some information of the plaintext
- Before commencing an attack, the attacker needs access to some information, such as:
 - A large number of ciphertexts that are all encrypted using the same key → “**ciphertext only**”; or
 - Pairs of ciphertext and the corresponding plaintext → “**known plaintext**”

Exhaustive Search (aka Brute-Force Search) Attack

- A *simple (brute-force) attack* is to exhaustively search the keys:
 - i.e. examine all possible keys one by one
- For most schemes, exhaustive search is *infeasible*
- Surprisingly, for some modern ciphers e.g. DES, it is feasible to break it using exhaustive search
- More sophisticated attacks exploit the properties of the encryption scheme to speedup the process

Exhaustive Search (aka Brute-Force Search) Attack

- Consider the substitution cipher (with table size of 27)
- Suppose the attacker somehow has access to a ciphertext C and a plaintext X ,
how difficult for him to find the key using exhaustive search?
- Let S be the set of all possible substitution table
- Given X, C :
 1. For each S in S
 2. Compute $X' = D_S(C)$;
 3. If ($X' == X$) then break;
 4. end-for
 5. Display (“The key is ”, S);

Exhaustive Search (aka Brute-Force Search) Attack

- The running time depend on the size of the key space S
- Since a key can be represented by a sequence of 27 symbols, the size is key space is $27!$

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | _ |
| g | v | w | b | n | e | f | h | d | a | t | l | u | c | q | m | z | i | r | s | j | x | o | y | k | - | p |

- Eventually, exhaustive search will find the key
- However, in the worst case, the exhaustive search needs to carry out $27! \approx 2^{94}$ loops, and on average $\approx 2^{93}$ loops. This is infeasible using current computing power (see Tutorial 1).
- *Can we attack substitution cipher more efficiently?*

Attack on Substitution Cipher: Known-Plaintext Attack

- “*Known plaintext attack scenario*”: when an adversary has access to pairs of ciphertexts and their corresponding plaintexts, and try to guess the key
- The attacker doesn’t need to carry out exhaustive search.
Given a plaintext and ciphertext, e.g.

plaintext: h e l l o _ w o r l d

ciphertext: h n l l q p o q i l b

The attacker can figure out the entries in the key

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | _ |
| | | | b | n | | | h | | | l | | | q | | | i | | | | | o | | | | p | |

- For a sufficiently long ciphertext, the full table can be determined

Attack on Substitution Cipher: Known-Plaintext Attack

- If the adversary can successfully derive the key, we say that the scheme is:
“insecure under known plaintext attack” or
“broken under known plaintext attack”
- Hence, substitution cipher is *not secure* under known plaintext attack!

Some Remarks on Known-Plaintext Attack

- To carry out a known-plaintext attack, the attacker needs to obtain at least a pair of ciphertext and its corresponding plaintext
- *Is this requirement reasonable??*
- In many cases, the attacker *doesn't need* to know the full plaintext: only *the first few bytes* of the plaintext are sufficient
- These first few bytes of the plaintext can sometimes be guessed:
 - Email data: certain words in its header are fixed, such as “From”, “Subject”, etc.
 - Many network protocols have fixed headers, or only a few choices in their first few bytes of data packets
 - During WWII, cryptologists exploited commonly-used words like “weather” and “nothing to report” as the known plaintext to guess the secret keys
- (Optional: Read more about the “Enigma Machine”.)

Attack on Substitution Cipher: Ciphertext-Only Attack

- Suppose the attackers have access to ciphertext only (i.e. without the corresponding plaintext), and knows that the plaintext are English sentences.
- Can he successfully find the key using exhaustive search?
- Yes!
- Let S be the set of all possible substitution table
Given C :
 1. For each S in S
 2. Compute $X = D_S(C)$;
 3. If X contains words in the English dictionary, then break;
 4. end-for
 5. Display (“The key is ”, S);

Attack on Substitution Cipher: Ciphertext-Only Attack

- Eventually, the exhaustive search will find the key
- However, the attack is also infeasible due to the large key space size
- Note: There is a very small probability that the above algorithm finds a wrong key.
Yet, for a sufficiently long ciphertext, say 50 characters long, the probability that a wrong key will give a meaningful English sentence is very low (treated as “*negligible*”).
- Is there an effective ciphertext-only attack technique on substitution cipher?
- Yes!

Frequency Analysis

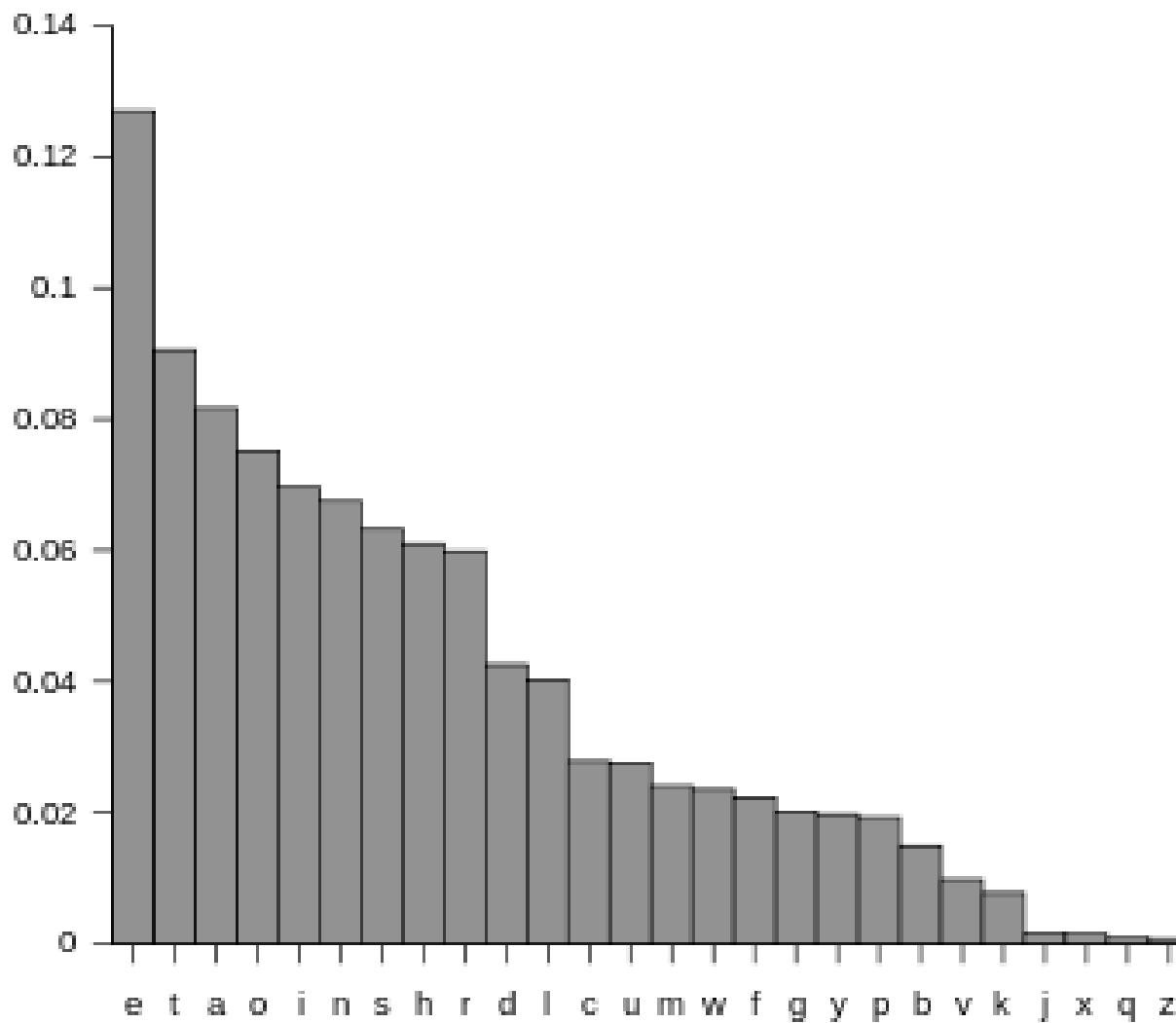
- Substitution cipher is vulnerable to *frequency analysis*
- Note that in the hello_world example below, symbol “o” appears 2 times in the plaintext, whereas the corresponding “q” also appears 2 times in the ciphertext

plaintext: h e l l o _ w o r l d

ciphertext: h n l l q p o q i l b

- Suppose the plaintexts are English sentences.
The **letter frequency distribution** in English text is **not** uniform, for e.g. “e” is more commonly used than “z”.
- How can an adversary apply frequency analysis and break substitution cipher?

Letter Frequency Distribution in English Text



From http://en.wikipedia.org/wiki/Letter_frequency

Frequency Analysis on Substitution Cipher

- If adversary is aware that the plaintexts are English sentences, given a sufficiently long ciphertext (say around 50 characters), then an adversary may be able to guess the plaintext by:
 - Mapping the *frequently-occurring letters in the ciphertext* to the *frequently-occurring letters of English*.
- Frequency analysis can be successfully carried out!
- Hence, substitution cipher is ***not secure under ciphertext-only attack*** either, when the plaintexts are English sentences
- In fact, the attack is effective on any language

LumiNUS Forum Challenge

- “**Breaking substitution cipher**” challenge in LumiNUS forum
- You’ll be given a ciphertext
- Do break the substitution cipher by finding the correct corresponding plaintext
- The **first person** who can post the correct plaintext will get **3 (three) extra marks** for assignment!
- The challenge will be posted this evening at **~8pm**

Ongoing NUS Bug Bounty Challenge!

- <https://nusit.nus.edu.sg/its/announcements/nus-bug-bounty-challenge/>



Ongoing NUS Bug Bounty Challenge!

- I'll give **10 (ten) extra marks** for CS2107 assignment to a Bounty Winner: no double claim in AY19/20

OTHER REWARDS

| Modules | Description |
|---------|-------------------------------|
| CS2107 | Intro to Information Security |
| CS3235 | Computer Security |
| CS4238 | Computer Security Practice |
| CS4239 | Software security |
| CS5321 | Network Security |
| CS5331 | Web Security |

IVLE's Cryptanalysis Challenge on Substitution Cipher

- The substitution table used:

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | _ |
| p | x | c | o | l | g | y | b | h | _ | q | d | w | m | s | e | f | k | v | a | j | z | r | i | t | n | u |

- Mapping from plaintext to ciphertext:

i_can_break_this_substitution_cipher_to_show_that_
hucpmuxklpquabhvuvjxvahajahsmucheblkuasuvbsruabpau

it_is_insecure_and_i_can_get_three_extra_marks
hauhvuhmvlcjkklupmouhucpmuylauabklluliakpuwpkqv

Some Useful Heuristics

- What's the most-frequently occurring character?
 - e, _ (space)?
- Spaces must break up the sentence reasonably well
- Single-letter words:
 - a: an indefinite article
 - i: the first (singular) person
- Digraphs:
 - **to, it, is, do, on, in, at, of, or, an, he, ...**
- Trigraphs:
 - **can, and, get, not, for, the, has, one, man, men, ...**
- You can also utilize some available online tools!

Quiz

- Suppose a substitution cipher works over a set of symbols U with $|U| = 50$
- What is:
 - Its key space size?
 - (The lower bound of) its key size/length?

Quiz

Caesar cipher

- A type of substitution cipher
- Each letter in the plaintext is “shifted”
a certain number of places down the alphabet
- Example: with a shift of **1**, A would be replaced by B,
B would become C,, Z would be replaced by ?
- How about its:
 - Key space size?
 - Key size (lower bound)?

1.2.2 Permutation Cipher

Permutation Cipher

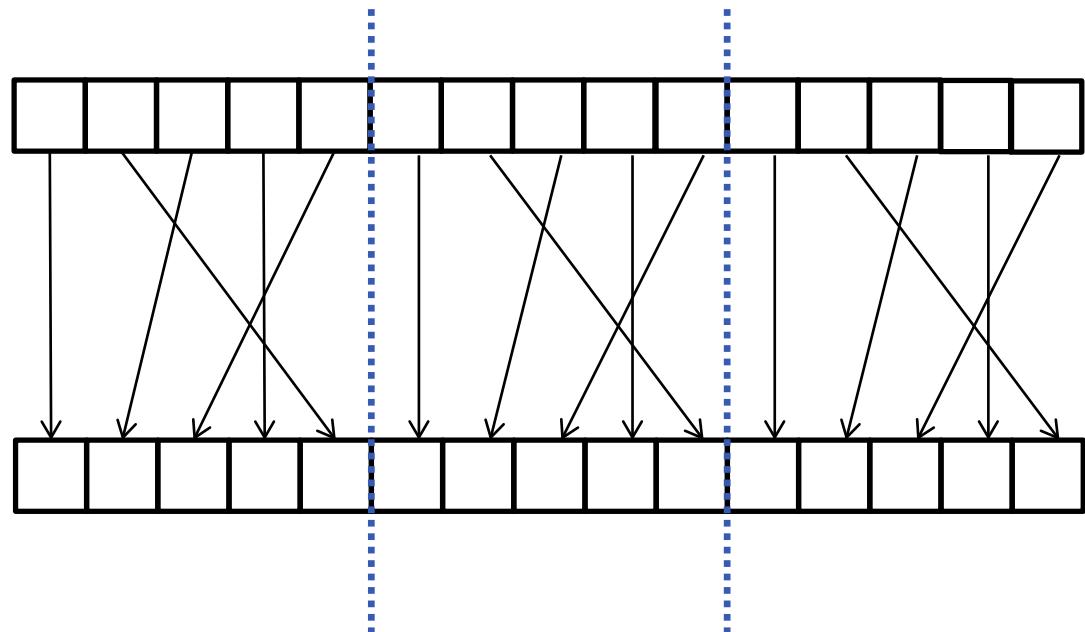
- Also known as **transposition** cipher
- It first groups the plaintext into blocks of t characters, and then applied a secret “permutation” to the characters in **each** block by shuffling the characters
- The key is the secret “permutation”: an 1-1 onto function e from $\{1,2,\dots,t\}$ to $\{1,2,\dots,t\}$
- We can write the permutation p as a sequence
$$p = (p_1, p_2, p_3, \dots, p_t),$$
which shifts the character at position/index i within the block to the position p_i ;
- The block size t could be part of the key:
 t is also kept secret

Permutation Cipher

- Example:

Given the plaintext and the key $t=5$, $p=(1,5,2,4,3)$:

plaintext



ciphertext

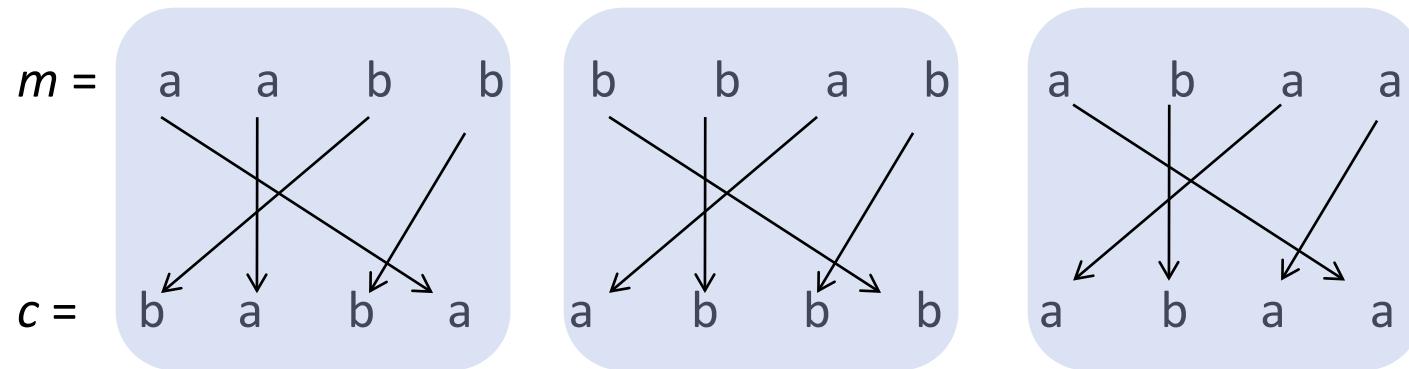
Cryptanalysis (Known-Plaintext Attack)

- Permutation cipher fails miserably under known-plaintext attack.
- Given a plaintext and a ciphertext, it is very easy to determine the key.
- Example:

$$m = \begin{matrix} a & a & b & b & b & b & a & b & a & b & a & a \end{matrix}$$
$$c = \begin{matrix} b & a & b & a & a & b & b & b & a & b & a & a \end{matrix}$$

- Question:
 - In the above, what is the block size t ?
 - What is the permutation?

Solution



- Permutation cipher can be easily broken under ciphertext-only attack if the plaintext is an English text

1.2.3 One Time Pad

One-Time-Pad

- Encryption:
Given an n -bit plaintext: $x_1x_2\dots x_n$ and n -bit key: $k_1k_2\dots k_n$,
output the ciphertext:
$$C = (x_1 \text{ xor } k_1) (x_2 \text{ xor } k_2) (x_3 \text{ xor } k_3) \dots (x_n \text{ xor } k_n)$$
- Decryption:
Given an n -bit ciphertext: $c_1c_2\dots c_n$ and n -bit key: $k_1k_2\dots k_n$,
output the plaintext:
$$X = (c_1 \text{ xor } k_1) (c_2 \text{ xor } k_2) (c_3 \text{ xor } k_3) \dots (c_n \text{ xor } k_n)$$
- In short:
Encryption: plaintext \oplus key \rightarrow ciphertext
Decryption: ciphertext \oplus key \rightarrow plaintext

Xor (Exclusive Or) Operation

- Xor operation: $A \oplus B = (A+B) \bmod 2$

- Xor table:

| A | B | $A \oplus B$ |
|-----|-----|--------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- Some interesting properties:
 - Commutative: $A \oplus B = B \oplus A$
 - Associative: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
 - Identity element: $A \oplus 0 = A$
 - Self-inverse: $A \oplus A = 0$

One-Time-Pad Example

| | | | | | | | |
|------------|---|---|---|---|---|---|---|
| PlainText | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| Key | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| Ciphertext | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

decryption

encryption

- Why does its decryption process works?
- For any x, k :
$$\begin{aligned} (x \oplus k) \oplus k &= x \oplus (k \oplus k) && [\text{by associativity}] \\ &= (x \oplus 0) && [\text{by self-inverse}] \\ &= x && [\text{by identity element}] \end{aligned}$$

Security of One-Time-Pad

- From a pair of ciphertext and plaintext, yes, the attacker can derive the key.
- However, such a key is useless, since it *will not* be used any more!
- Note that it is not clear how to apply exhaustive search on one-time pad
- In fact, It can be shown that one-time-pad leaks **no information** of the plaintext, even if the attacker has an arbitrary running time (unlimited computing power)
- Hence, it is sometime called “*unbreakable*” (provided that a good “random” key is used)

Some Remarks

- Yes, one-time-pad can be shown to be “unbreakable”
- However, the length of the key is the same as plaintext’s, which is useless in many applications
- Nevertheless, it is practical in some scenarios
(See <http://ciphermachines.com/otp>)



<http://ciphermachines.com/otp>

- Yet, also check “*The Venona Story*”, where one-time-pads fails

(Optional: https://www.nsa.gov/about/_files/cryptologic_heritage/publications/coldwar/venona_story.pdf)



1.3 Modern Ciphers

Modern ciphers generally refer to schemes that
use computer to encrypt/decrypt.

Examples are: RC4, DES, A5, AES, RSA

Modern Ciphers

- Designs of modern ciphers take into considerations of known-plaintext-attack, frequency analysis, and other known attacks
- Examples: DES (Data Encryption Standard, 1977)
RC4 (Rivest's Cipher 4, 1987)
A5/1 (used in GSM, 1987)
AES (Advanced Encryption Standard, 2001)
- They are supposed to be “secure”, so that any successful attack does not perform noticeably better than exhaustive search

(Optional: Nevertheless, RC4 is broken in some adoptions, A5/1 is vulnerable, and DES's key length is too short. Wiki pages on RC4, A5/1 and DES give quite good descriptions. AES is believed to be secure, and classified as “Type 1” by NSA
https://en.wikipedia.org/wiki/NSA_cryptography#Type_1_Product.)

Exhaustive Search and Key Length (See Work factor [PF2.3page91])

- If the key length is 32 bits, there are 2^{32} possible keys.
Hence, the exhaustive search has to loop for:
 - 2^{32} times in the worst case
 - 2^{31} times on average
- We can quantify the security of an encryption scheme by the *length of the key*
- Example:
 - Comparing a scheme *A* with 64-bit keys and a scheme *B* with 54-bit keys,
then scheme *A* is more secure w.r.t. exhaustive search

Exhaustive Search and Key Length (See Work factor [PF2.3page91])

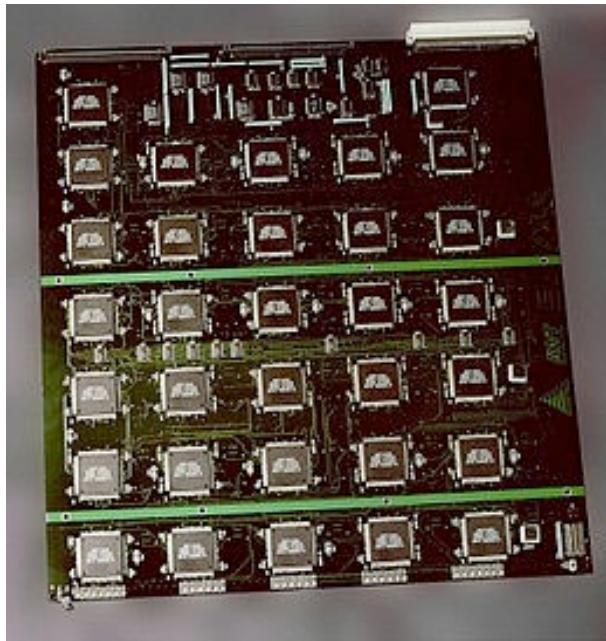
- **Additional Note:** Some schemes, e.g. RSA, have known attacks that are more efficient than exhaustively searching all the keys
- In those cases, we still want to quantify the security by the **equivalent exhaustive search**
- For example, in the best known attack on a 2048-bit RSA, roughly 2^{112} searches are required:
 - Hence, we treat its security as equivalent to 112 bits
 - So, we say that the 2048-bit RSA has key strength of 112 bits
- Question: How **many bits** are considered “secure”?
- Answer: See Tutorial 1
- Also *read* NIST Recommended key length for AES
<http://www.keylength.com/en/4/>

Exhaustive Search on DES

- Key length of DES is 56 bits
- While exhaustive search on 56 bits seemed infeasible in the 70s, very soon, it was possible using distributed computing or specialized chip
- *RSA Security* hosted a few challenges on DES:
 - **DES Challenge II-1:** "*The secret message is: Many hands make light work.*"
(Found in 39 days using distributed computing, early 1998)
 - **DES Challenge II-2:** "*The secret message is: It's time for those 128-, 192-, and 256-bit keys.*"
(Found in 56 hours using a specialized hardware, 1998)
- (Note: *RSA* is an encryption scheme, whereas *RSA Security* is a company)

Exhaustive Search on DES

- EFF's DES cracking machine ("*Deep Crack*"):



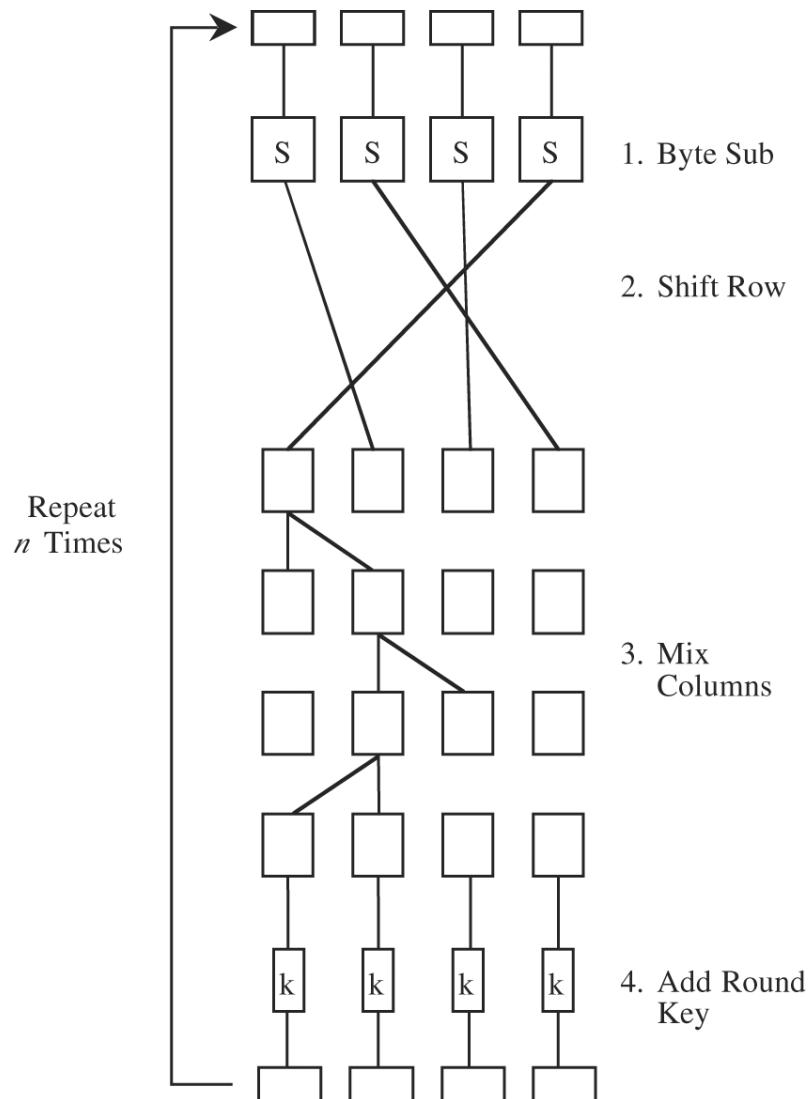
Optional:

[https://en.wikipedia.org/wik/
i/ EFF_DES_cracker](https://en.wikipedia.org/wiki/EFF_DES_cracker)

- A question is: why would an agency designed a scheme that could be broken in the near future?
- Many believed that it was intentional

AES: Advanced Encryption System

- Symmetric block cipher
- Developed in 1999 by independent Dutch cryptographers
- Still in common use



DES vs AES

| | DES | AES |
|---------------------------------|---|---|
| Date designed | 1976 | 1999 |
| Block size | 64 bits | 128 bits |
| Key length | 56 bits (effective length); up to 112 bits with multiple keys | 128, 192, 256 (and possibly more) bits |
| Operations | 16 rounds | 10, 12, 14 (depending on key length); can be increased |
| Encryption primitives | Substitution, permutation | Substitution, shift, bit mixing |
| Cryptographic primitives | Confusion, diffusion | Confusion, diffusion |
| Design | Open | Open |
| Design rationale | Closed | Open |
| Selection process | Secret | Secret, but open public comments and criticisms invited |
| Source | IBM, enhanced by NSA | Independent Dutch cryptographers |

From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

Properties of Ciphers: Confusion and Diffusion

- Two properties of an encryption system:
confusion and diffusion
- ***Confusion***: an attacker should *not* be able to predict what will happen to the ciphertext when **one character** in the plaintext changes
- This means:
 - The input (i.e. plaintext and key pair) undergoes *complex* transformations during encryption
 - A cipher with **good confusion**:
it has a “*complex functional relationship*” between the plaintext/key pair and the ciphertext

Properties of Ciphers: Confusion and Diffusion

- **Diffusion:** a change in the plaintext will affect *many parts* of the ciphertext
- This means:
 - Information from the plaintext is *spread over* the entire ciphertext
 - The transformations *depends equally* on all bits of the input
- A cipher with **good diffusion:**
it requires an attacker to access *much of* the ciphertext in order to infer the encryption algorithm
- Block cipher: high diffusion
- Stream cipher: low diffusion

1.3.2 Stream Cipher and IVs

Stream Cipher

- Stream cipher is “inspired” by one-time-pad
- Suppose the plaintext is 2^{20} bits,
but the secret key is only 256 bits
- Steam cipher generates a 2^{20} -bit *sequence* from the key,
and takes the generated sequence as the “*secret key*” in
one-time-pad
- The generator has to be carefully designed, so that it gives
(cryptographically-secure) pseudorandom sequence

Stream Cipher

- Visually:

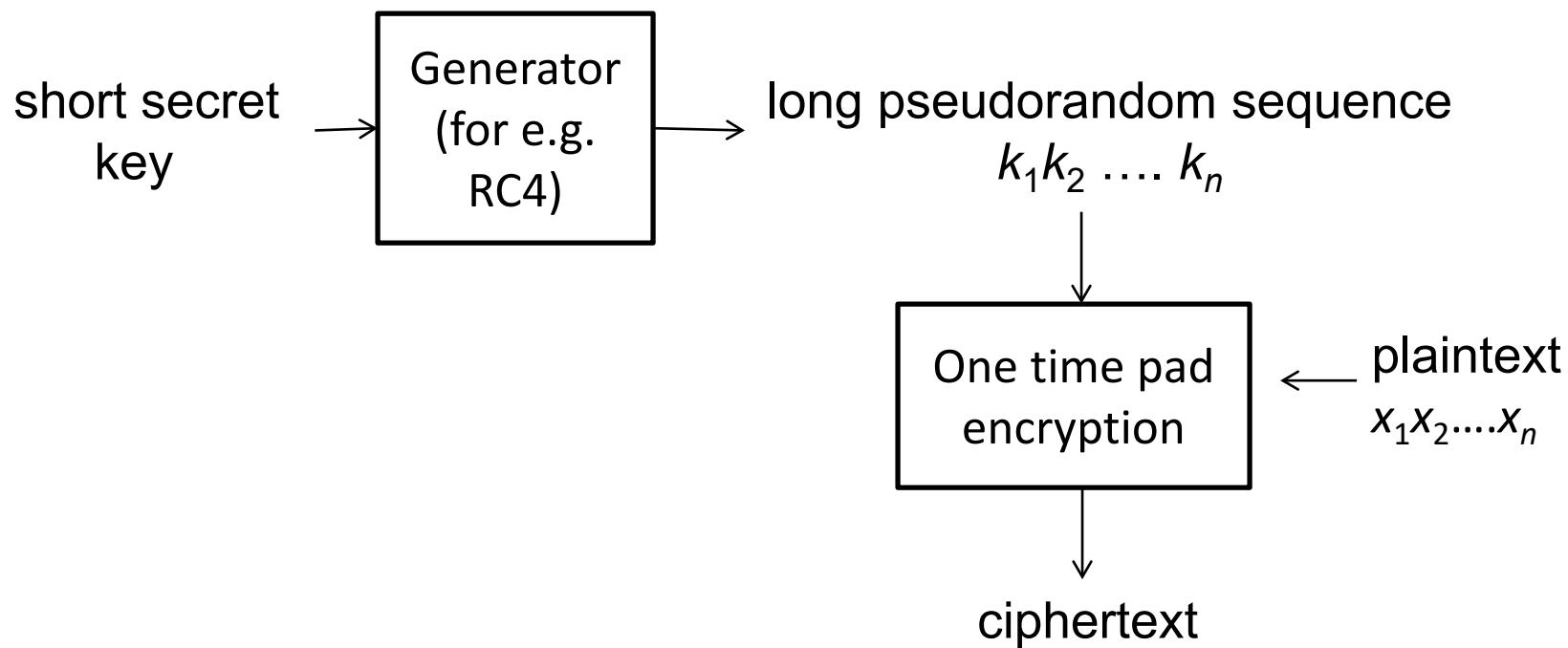
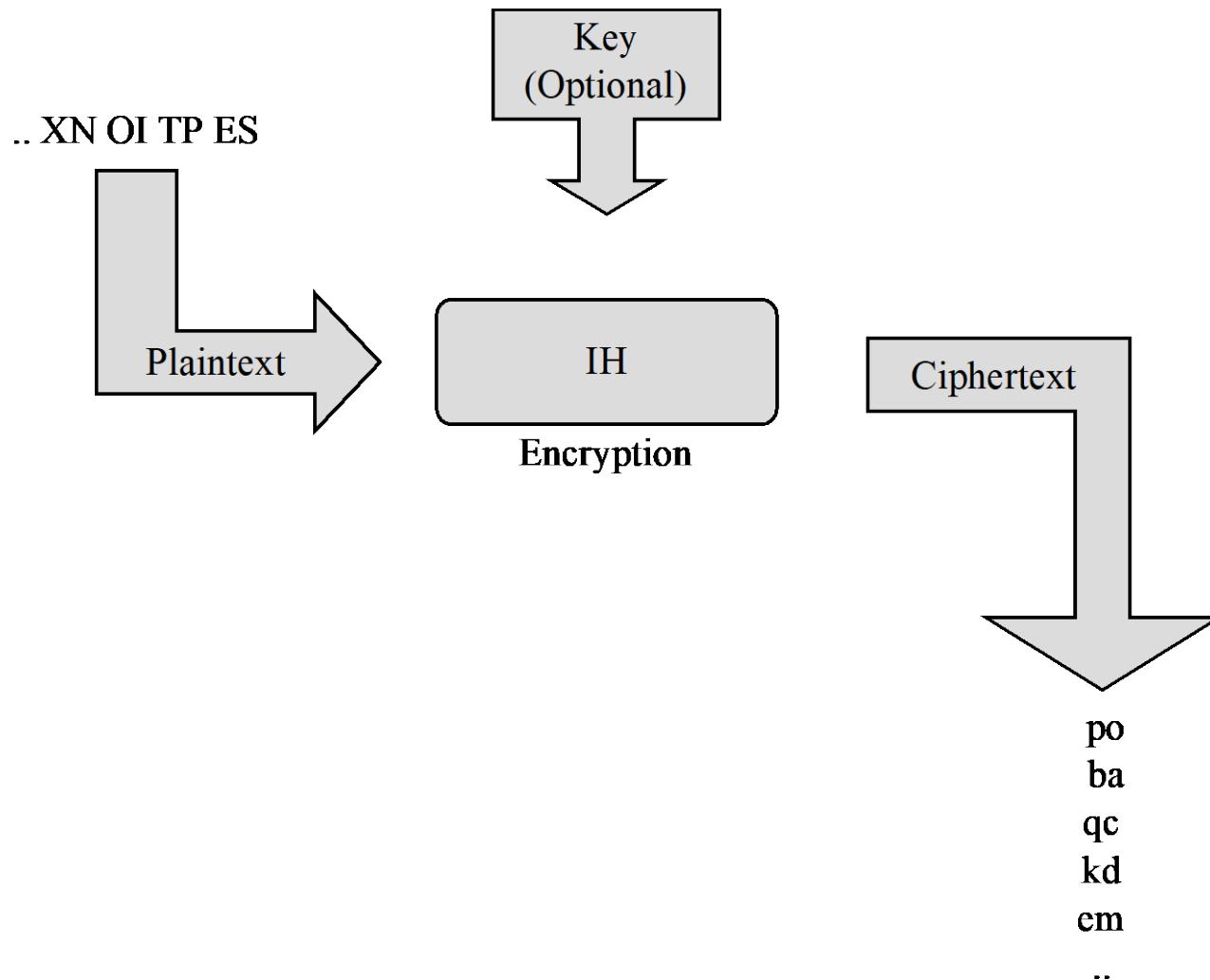
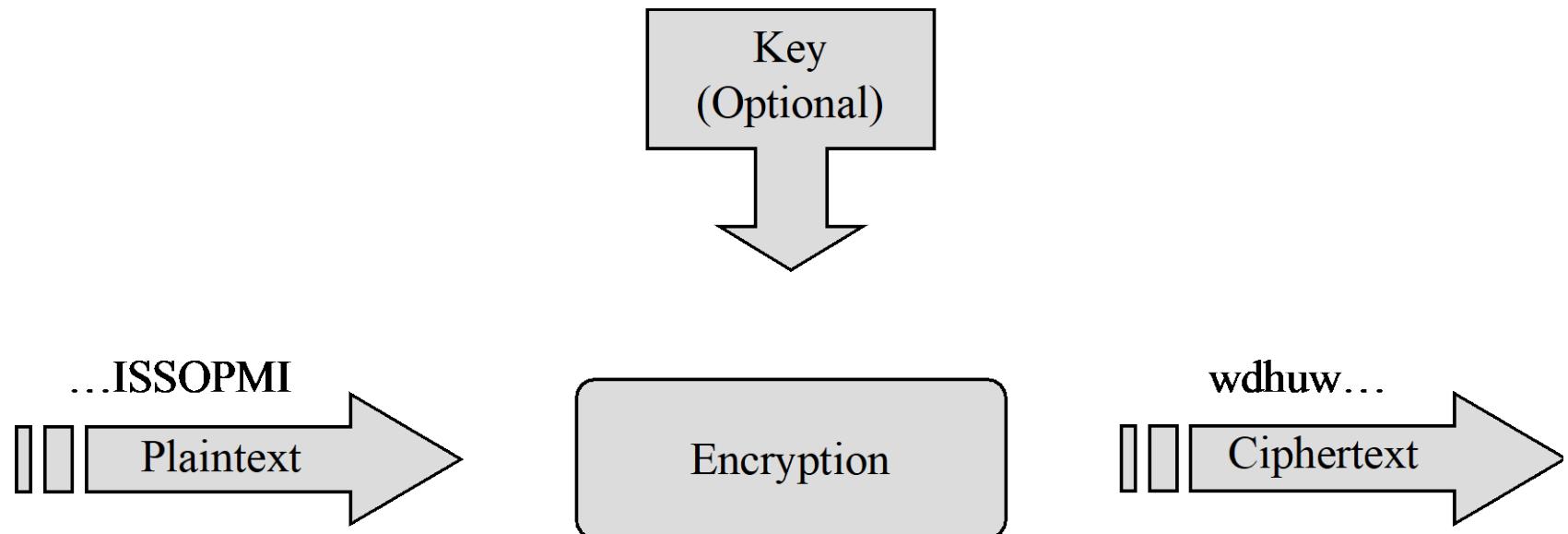


Illustration of a Block Cipher



From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

Illustration of a Stream Cipher



From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

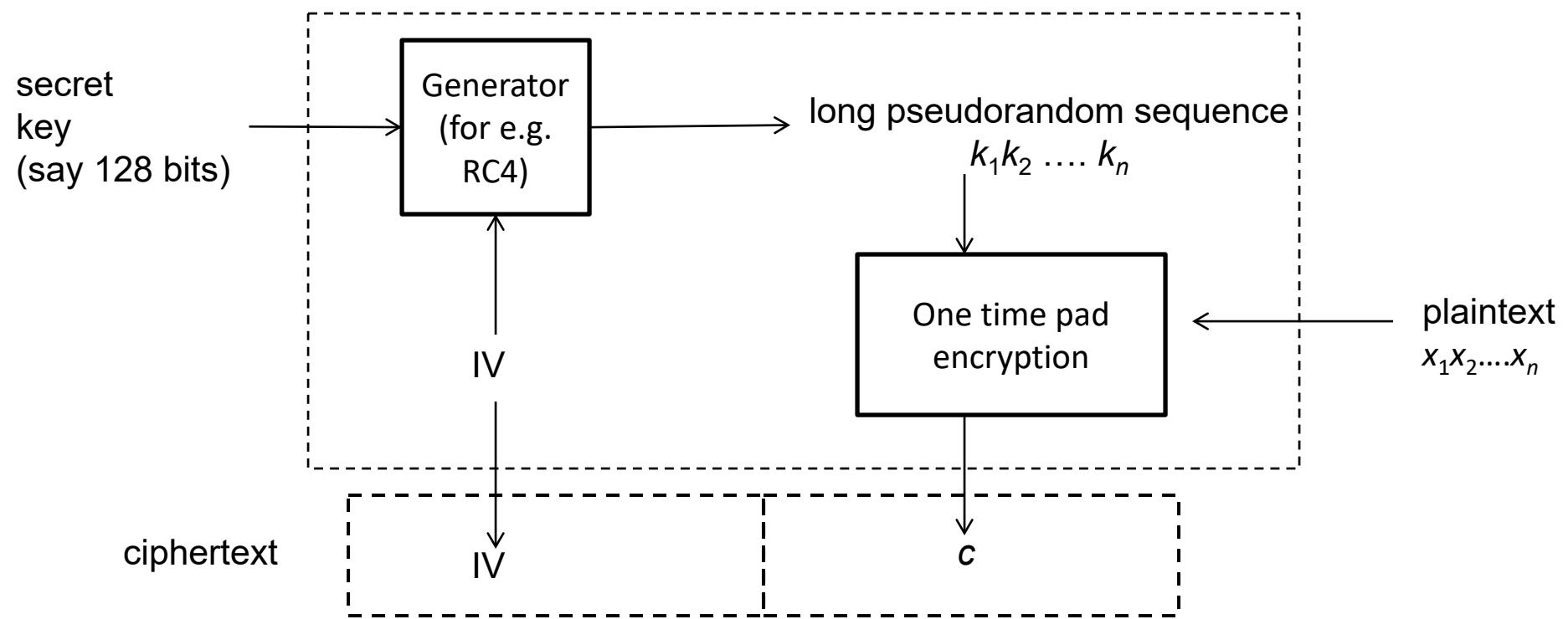
Stream vs Block Ciphers

| | Stream | Block |
|---------------|--|--|
| Advantages | <ul style="list-style-type: none">• Speed of transformation• Low error propagation | <ul style="list-style-type: none">• High diffusion• Immunity to insertion of symbol |
| Disadvantages | <ul style="list-style-type: none">• Low diffusion• Susceptibility to malicious insertions and modifications | <ul style="list-style-type: none">• Slowness of encryption• Padding• Error propagation |

Stream Cipher with Initial Value (IV)

- Stream cipher typically has an *Initial Value* or *Initialization Vector* (IV)
- The IV can be randomly chosen or from a counter
- A long pseudorandom sequence is generated from the secret key together with the IV
- The final ciphertext contains the IV, followed by the output of the one-time-pad encryption
- *Note:* In some documents, the term “ciphertext” does not include the IV. In any case, the IV **must** appear in clear, and every entity can see it.
- For decryption:
 - The IV is extracted from the ciphertext
 - From the IV and the key, the same pseudorandom sequence can be generated and thus obtain the plaintext

Stream Cipher with Initial Value (IV)



Important question: What if IV is omitted in the ciphertext?

Example

Encryption:

Given:

15-bit plaintext $X = 0000011110000$
short key = 0101

Step 1: Randomly generate an IV, say: $IV = 0001$

Step 2: From the short key and IV, generate a 15-bit sequence

$$K = 011010100100110$$

Step 3: Output IV, followed by $K \text{ xor } X$



Decryption:

Given the short key and the ciphertext with the IV

Step 1: Extract the IV from the ciphertext

Step 2: From the short key and IV, generate the long sequence K

Step 3: Perform xor to get the plaintext

Why IV?

- Suppose there isn't an IV
(or set the IV to be always a string of 0's)
- Consider the situation where the same key is used to encrypt two different plaintexts:

$$X = x_1 x_2 x_3 x_4 x_5 \quad \text{and}$$

$$Y = y_1 y_2 y_3 y_4 y_5$$

- Further, suppose that an attacker eavesdrops and obtains the two corresponding ciphertexts U, V

Why IV?

- The attacker can now compute:

$$\mathbf{U} \oplus \mathbf{V} = (\mathbf{X} \oplus \mathbf{K}) \oplus (\mathbf{Y} \oplus \mathbf{K})$$

- By associative and commutative properties of xor:

$$(\mathbf{X} \oplus \mathbf{Y}) \oplus (\mathbf{K} \oplus \mathbf{K}) = \mathbf{X} \oplus \mathbf{Y}$$

- So, from ciphertexts \mathbf{U} and \mathbf{V} , the attackers can obtain information about $\mathbf{X} \oplus \mathbf{Y}$, i.e. the following sequence:

$$(x_1 \oplus y_1) (x_2 \oplus y_2) (x_3 \oplus y_3) (x_4 \oplus y_4) (x_5 \oplus y_5)$$

- **Question:** What's the big deal about revealing $\mathbf{X} \oplus \mathbf{Y}$?

What's the big deal about revealing $X \oplus Y$?

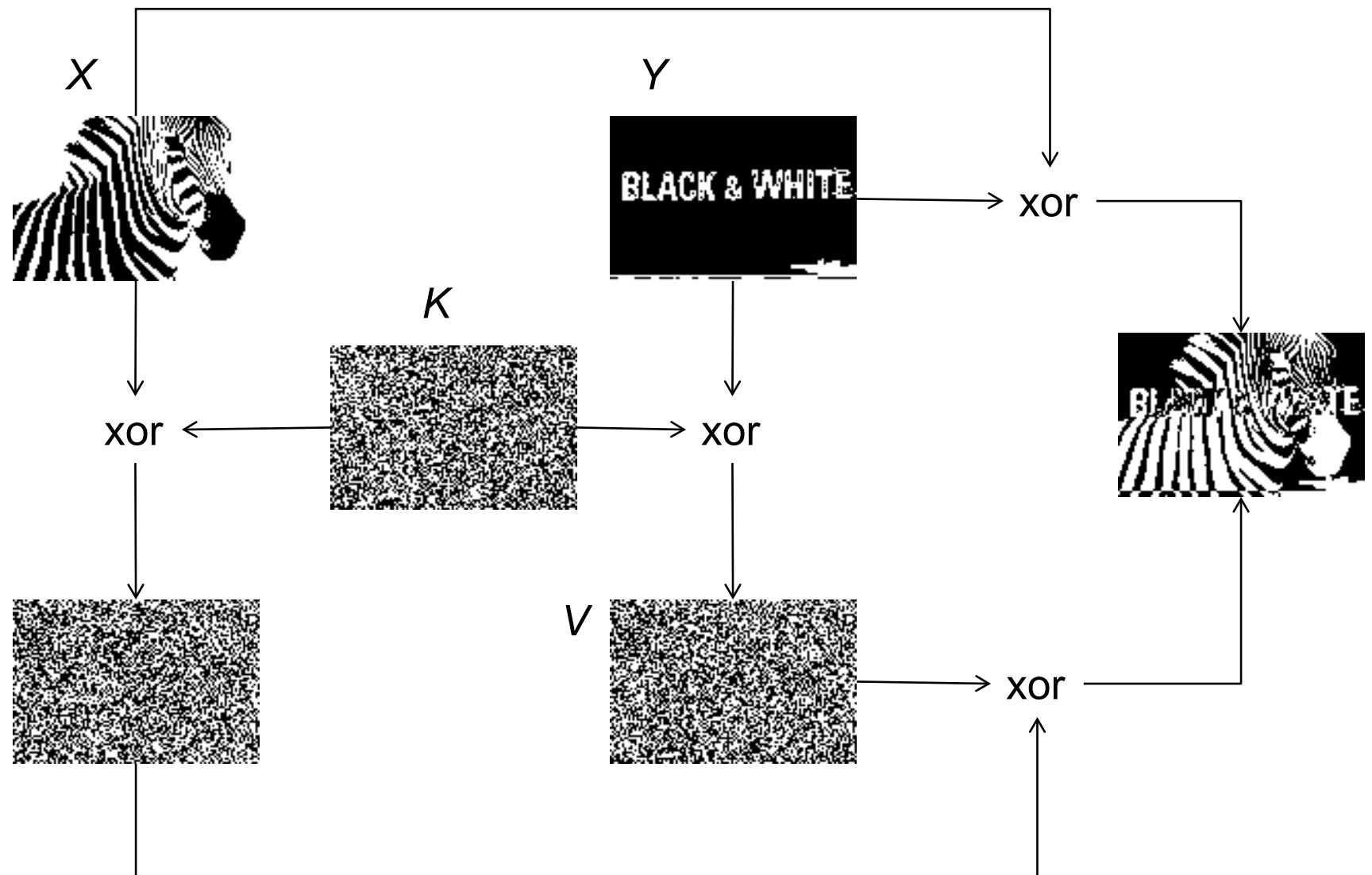
- Suppose X is a 80x120 image of an animal.
Each pixel is either black (0) or white (1).
The image can be represented as a (80x120)-bit sequence,
where each bit corresponds to a pixel.
- Y is another 80x120 pixels image rendering two words,
which is similarly represented as a sequence.

- Here is $X \oplus Y$:



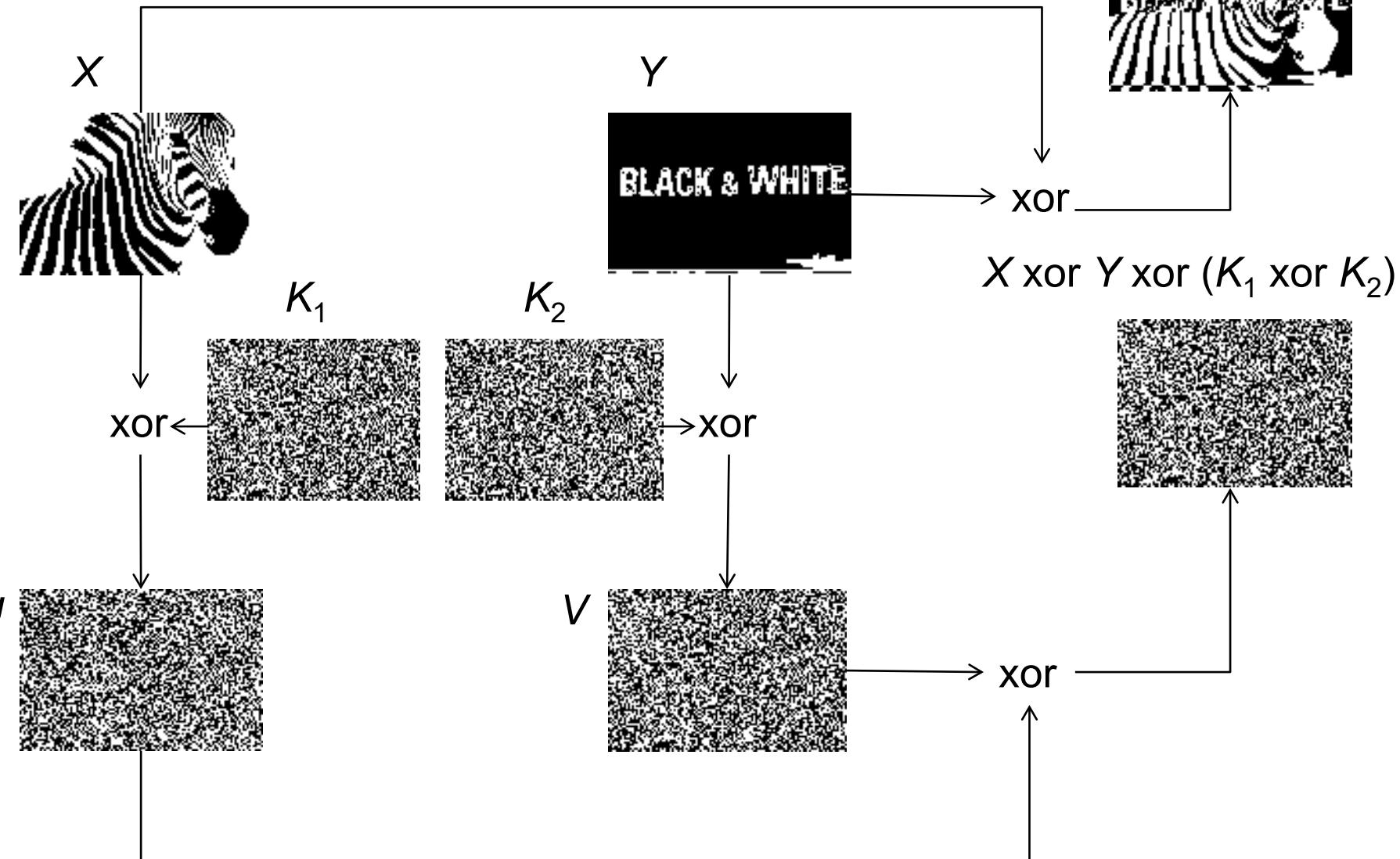
- So, what is X, Y ?

Stream Cipher Without IV



Stream Cipher with IV

$X \text{ xor } Y$



Role of IV

- During 2 different processes of encryption of the same plaintext:
 - The IVs are likely to be different
 - The 2 pseudorandom sequences are thus likely to be different
 - The ciphertexts are also likely to be different
- Furthermore, the xor-ing of two ciphertexts would **not** cancel the effect of the pseudorandom sequences, and reveal information of the plaintexts!
- Note that IV is not only used in stream cipher, but also adopted in other ciphers

(**Note:** In many documents, the description of ciphers (such as AES, RSA) does not include the IV, and thus doesn't specify how the IV is to be chosen. Typically, the IV is mentioned during discussion of implementation. E.g. for AES, the IV is only included in the “**mode of operation**” of AES.)

1.4 Attacks on Cryptosystem Implementations

A secure encryption scheme can be vulnerable if it is **not implemented properly**.

This section gives some examples:

- 1.4.1 – Reusing IV and one-time-pad key in encryption
- 1.4.2 – Predictable secret-key generation
- 1.4.3 – Designing your own cipher

(**To be studied later:** Using encryption for the wrong purpose, e.g. using encryption scheme to ensure *message integrity*)

1.4.1 Reusing IV, One-Time-Pad Key

Mishandling of IV

- Some applications overlooked IV generation.
As a result, under some situations, the same IV is **reused**.
- E.g. To encrypt a file F , the IV is derived from *the filename*.
It is quite common to have files with the same filename.

(Read “Schneier on Security, Microsoft RC4 Flaw”:

https://www.schneier.com/blog/archives/2005/01/microsoft_rc4_f.html
<http://eprint.iacr.org/2005/007.pdf>)

- E.g. When using AES under the “CBC mode”, the IV has to be **unpredictable** to prevent a certain type of attack.
(Hence, it is vulnerable to choose IV as 1, 2, 3,...).

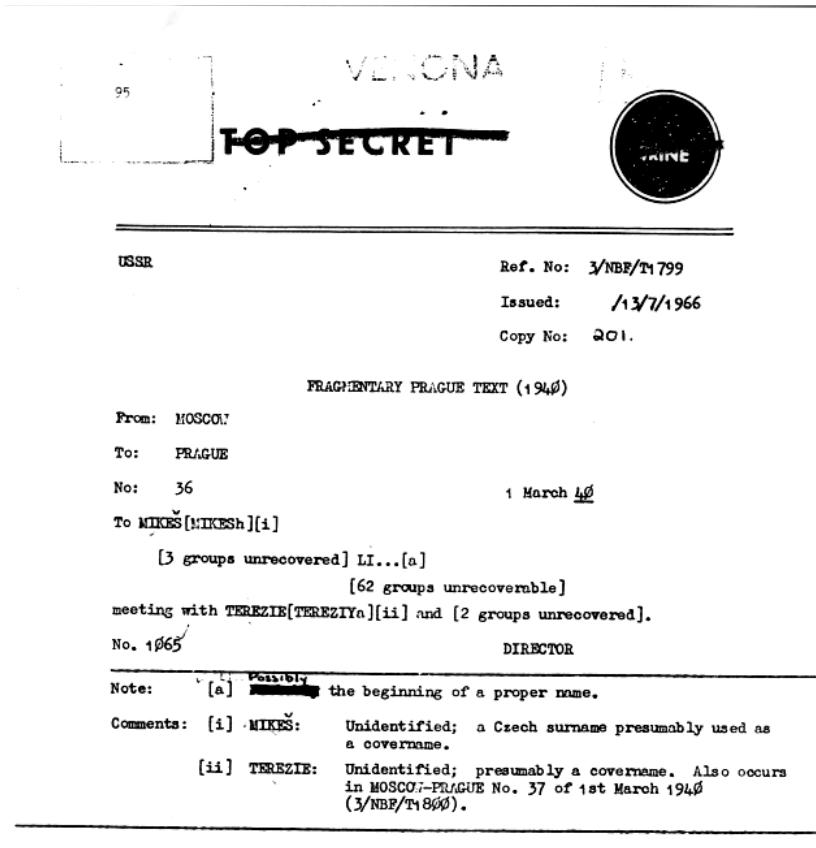
The well known BEAST attack exploits this:

(Optional: <http://resources.infosecinstitute.com/ssl-attacks/>)

Reusing One-Time-Pad Key

- The Venona project is a classic example on such failure

(Optional: https://www.nsa.gov/about/_files/cryptologic_heritage/publications/coldwar/venona_story.pdf)



1.4.2 Predictable Secret-Key Generation

Random Number Generation

- **Scenario 1:**
 - You are coding a program for a **simulation system**, for e.g. to simulate road traffic
 - In the program, you need a sequence of random numbers, for e.g. to decide the speed of the cars
 - *How to get these random numbers?*
- **Scenario 2:**
 - You are coding a program for a **security system**
 - In the program, you need a random number, for e.g. you need to generate a random number as a temporary **secret key**
 - *How to get these random numbers?*

To be Discussed in Tutorial

- In Java, what is the difference between the following?
 - `java.util.Random`
 - `java.security.SecureRandom`
- In C, what is the difference between using the following:

```
#include <time.h>
#include <stdlib.h>
    srand(time(NULL));
    int r = rand();
```

and a more complicated version below?

```
int byte_count = 64;
char data[64];
FILE *fp;
    fp = fopen("/dev/urandom", "r");
    fread(&data, 1, byte_count, fp);
    fclose(fp);
```

1.4.3 Designing Your Own Cipher

Caution!

- **Don't** design your own cryptosystem,
or even make a slight modification to existing scheme,
unless you has an in-depth knowledge of the topic!
- Read “Don’t roll your own crypto”:
<http://security.stackexchange.com/questions/2202/lessons-learned-and-misconceptions-regarding-encryption-and-cryptology/2210#2210>

1.5 Kerckhoffs' Principle vs Security through Obscurity

Kerckhoffs' Principle

- “A system should be secure even if everything about the system, *except the secret key*, is a public knowledge”

Security through Obscurity

- To hide the design of the system in order to achieve security
- *Is it good or bad??*

Examples (*Against Obscurity*)

- RC4:
 - Was introduced in 1987 and its algorithm was a **trade secret**
 - In 1994, a description of its algorithm was **anonymously posted** in a mailing group.
 - See <http://en.wikipedia.org/wiki/RC4>
- MIFARE Classic:
 - A contactless smartcard widely used in Europe employed a set of proprietary protocols/algorithms
 - However, they were **reverse-engineered** in 2007
 - It turned out that the encryption algorithms were already known to be weak (using only 48bits) and breakable
 - See <http://en.wikipedia.org/wiki/MIFARE>
 - Optional: Presentation video by the researcher who reverse-engineered it: <http://www.youtube.com/watch?gl=SG&hl=en-GB&v=QJyxUvMGLr0>.
The algorithm was revealed at 14:00.)

Examples (*Supporting Obscurity*)

- Usernames:
 - They are not secrets
 - However, it is *not* advisable to publish all the usernames
- Computer network structure & settings:
 - E.g. location of firewall and the firewall rules
 - These are not secrets, and many users within the organization may already know the settings
 - Still, it is *not* advisable to them
- The actual program used in a smart-card:
 - It is not advisable to publish it
 - If the program is published, an adversary may be able to identify vulnerability that was previously unknown, or carry out side-channel attacks
 - A sophisticated advisory may be able to reverse-engineer the code nevertheless

So, Should We Use Obscurity???

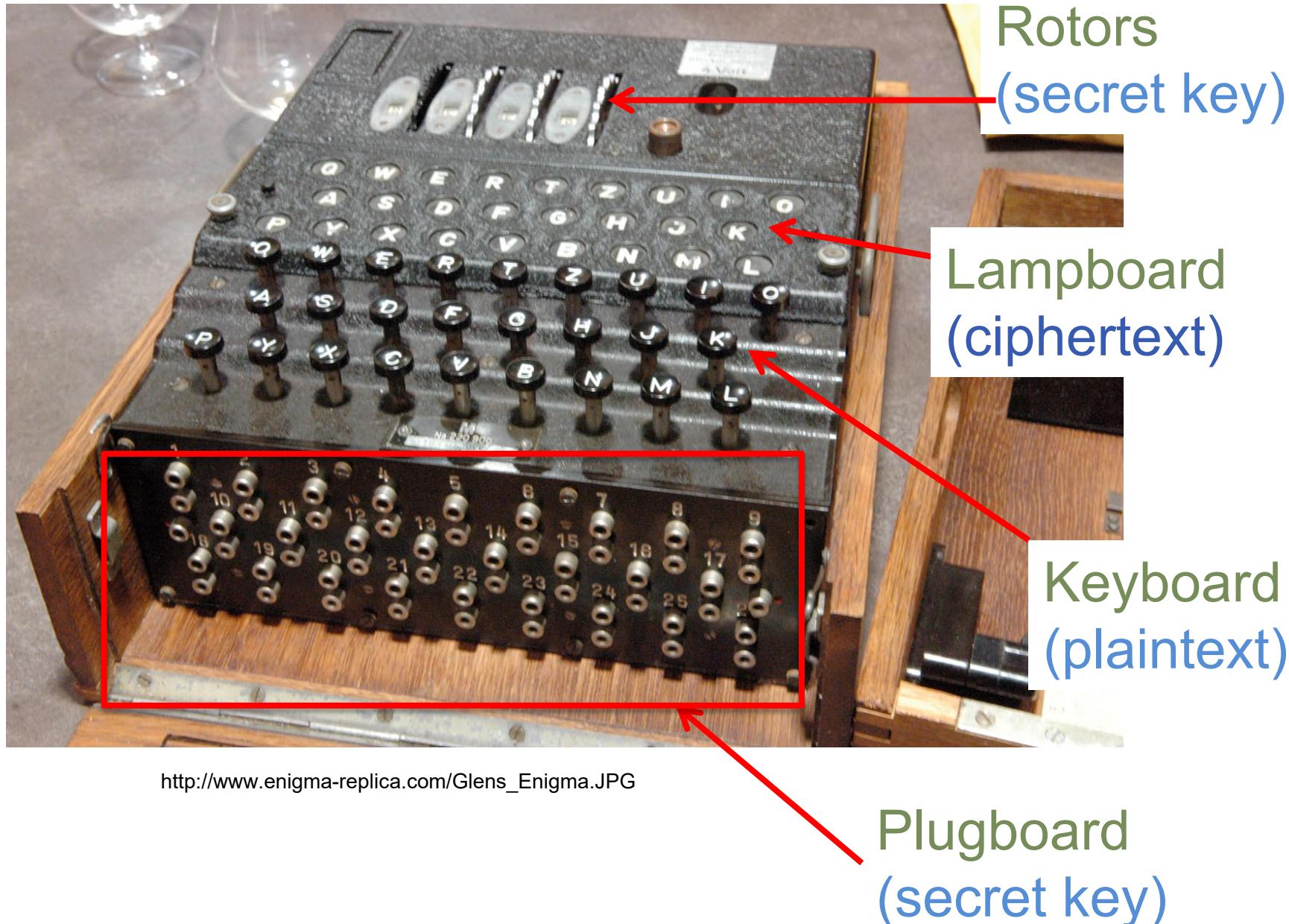
- In general, obscurity can be used as one layer in a ***defense in depth*** strategy
- It could deter or discourage novice attackers, but is ineffective against attackers with high skill and motivation
- The system **must remain secure** even if everything about it, *except its secret key*, becomes known
- In this module, we always assume that the attackers **know** the algorithms
- See:
 - <http://technet.microsoft.com/en-us/magazine/2008.06.obscurity.aspx>
 - http://en.wikipedia.org/wiki/Security_through_obscurity

1.6 Some Historical Facts

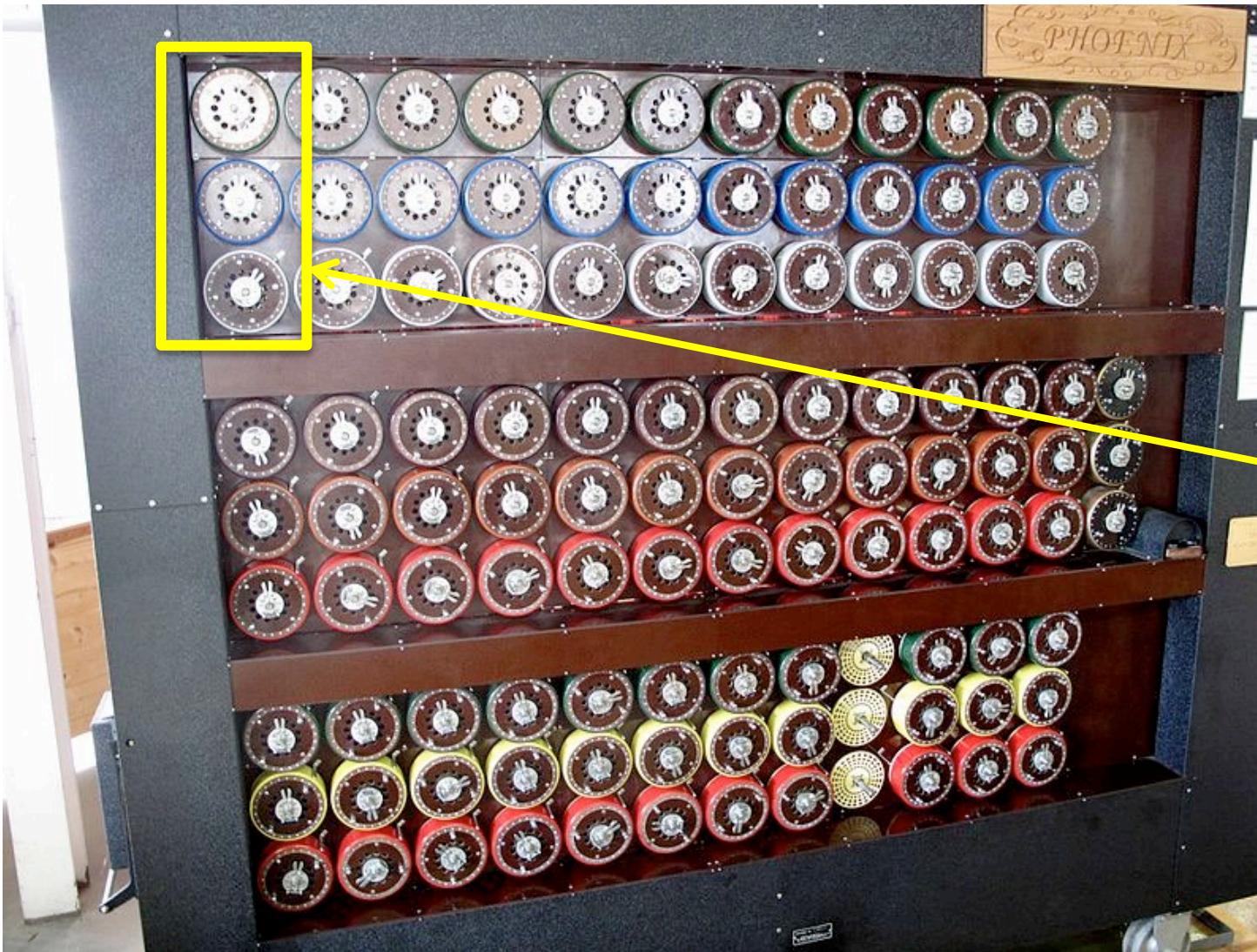
Cryptography: History

- Cryptography is closely related to warfare and can be traced back to ancient Greece
- Its role became significant when information is sent over the air
- WWII: Famous encryption machines include the Enigma, and the Bombe (that helped to break Engima)

Enigma (Replica)



Working Rebuilt Bombe at Bletchley Park Museum



Simulates
the 3
rotors
in one
Enigma
machine

http://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma#Crib-based_decryption

Modern Ciphers

DES (Data Encryption Standard):

- 1977: DES, 56 bits

(During cold war, cryptography, in particular DES was considered as “munition”, and subjected to export control. Currently, export of certain cryptography products is still controlled by US.

Read the crypto law survey’s section on Singapore at

<http://www.cryptolaw.org/cls2.htm>)

- 1998: A DES key broken in 56 hours
- Triple DES (112 bits) is still in used

AES (Advanced Encryption Standard):

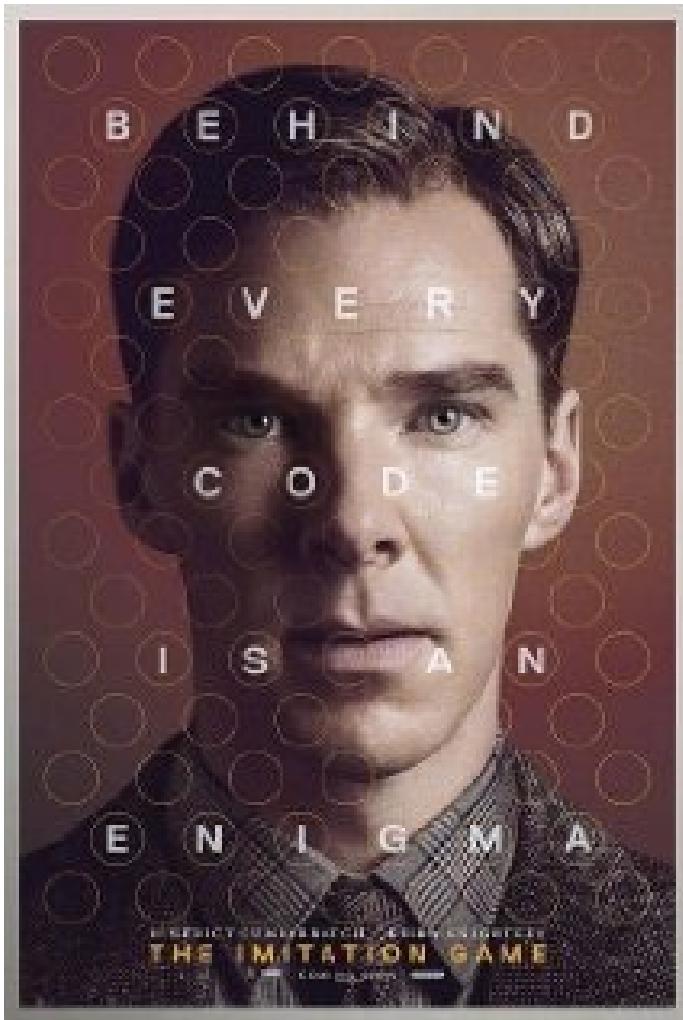
- 2001: NIST. 128, 192, 256 bits

Modern Ciphers

RC4:

- 1987: Designed by Ron Rivest (RSA Security), initially a trade secret
- 1994: Algorithm leaked in
- 1999: Used in widely popular **WEP** (for WiFi);
WEP implementation has 40 or 104-bit key
- 2001: A weakness in how WEP adopts RC4 is published by
Fluhrer, Mantin, Shamir
- 2005: A group from FBI demonstrated the attack
- Afterward: Industry switched to WPA2
(with WPA as an intermediate solution)

Movie About Encryption



“The Imitation Game”:

During World War II, mathematician **Alan Turing** tries to crack Enigma with help from fellow mathematicians
[\(http://www.imdb.com/title/tt2084970/\)](http://www.imdb.com/title/tt2084970/)

Sample Tutorial Questions

Question:

Bob encrypted a video file using Winzip, which employs the 256-bit key AES. He choose a 6-digit number as password.

Winzip generated the 256-bit key from the 6-digit password using a “hash” function, say SHA1.

Alice obtained the ciphertext.

Alice also knew that Bob used a 6-digit password.

Given a “guess” of the 256-bit key, Alice can determine whether the key can successfully decrypted the file.

How many guesses Alice really needed to make in order to get the video?

Lecture 2: Authentication (Password)

Topics:

2.1. Overview

2.2 Password (weak authentication)

 2.2.1 Intercepting password while bootstrapping

 2.2.2 Searching password (dictionary, guessing, exhaustive attacks)

 2.2.3 Stealing password

 2.2.4 Preventive measures

 2.2.5 ATM attacks

 2.2.6 Password reset: Security questions

2.3 Biometrics

2.4 Multi-factor authentication

 2.4.1: Case studies: SMS vs token (in tutorial)

2.1 Overview

Reading:

[PF2.1] excluding Federated Identity Management

[Gollman] also has good coverage on Password (Chapter 4.1 to 4.5)

Authentication

- **Authentication:** the process of assuring that the communicating entity, or the origin of a piece of information, is the one that it claims to be.
- Two types of authentication:
 - *Entity authentication:*
 - For connection-oriented communication
 - Communicating entity is *an entity involved in a connection*
 - Mechanisms: password, challenge and response
 - *Data-origin authentication:*
 - For connectionless communication
 - Communicating entity is *the origin of a piece of information*
 - Data-origin authenticity implies data integrity (see next slide)
 - Mechanisms: MAC or digital signature

Authenticity

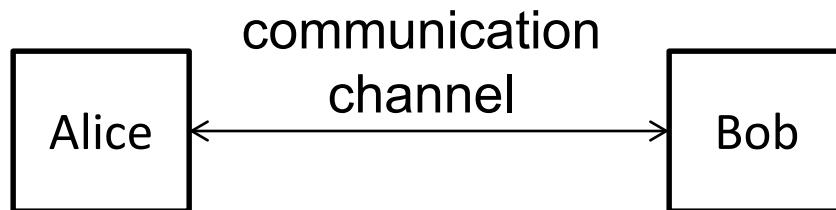
- ***Authentic*** (adjective): the claimed entity/origin is assured by supporting evidence
- ***Authenticity***: the condition of being authentic
- Authenticity and integrity:
 - Are they related? Yes.
 - Example: “In the context of an insecure channel, we can say that a message that has been modified in transit means that it no longer comes from its original source.”
 - In other words, a message whose integrity is compromised also means that its authenticity is compromised
 - Hence, *data-origin authenticity implies data integrity*
 - But data integrity does *not* imply data-origin authenticity
 - Authenticity is a stronger requirement than integrity

Authenticity

- Authenticity-preserving techniques also ensure integrity:
 - MAC & digital signature versus hash (*to be revisited later!*)
- Some notes:
 - Again, note that some documents use the term “integrity” to mean “authenticity”
 - Some even claim that authenticity does not necessarily give integrity
 - Hence, when reading a document, do pay attention to the context and the applications involved

Examples of Problem Ensuring Authenticity

Over different communication channels:



- Alice received a *phone call*, which claimed to be from the Police Department, and asked for information regarding her brother.
Authentic?
- Alice logged-in to *IVLE* and wondered whether the server that her laptop was interacting with is the *authentic* “IVLE”?
Conversely, why the IVLE’s server would be convinced that the entity logged in is the *authentic* “Alice”?
- Alice tried to connect to WiFi using her phone while at NUH’s bus-stop. Among the available WiFi network names (SSIDs), an item “NUS” is listed. Alice keyed in her user-id and password.
Was that WiFi access point *authentic*?

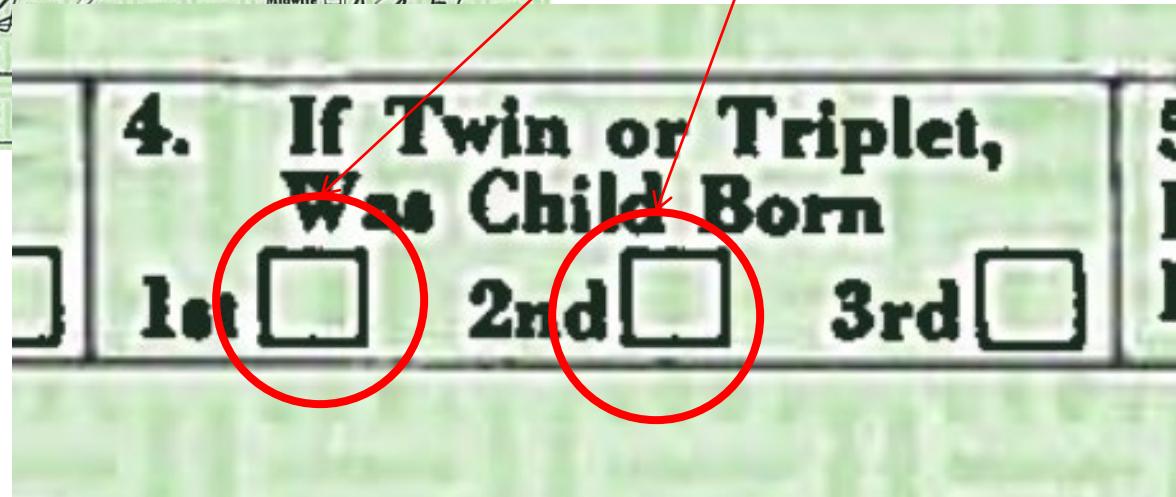
More Examples of Problem Ensuring Authenticity

Involving presented digital data or physical document:

- Bob submitted a *medical certificate (MC)* to the lecturer, indicating that he was unfit for exam.
Was the MC *authentic* (i.e. issued by the purported clinic)?
Or had Bob altered the date?
- Is the *birth certificate* (see next slide) released by the White House *authentic* (i.e. issued by the claimed Local Registrar)?
- Alice received an *email* from her lecturer notifying her that the quiz is cancelled.
Was the email *authentic* (i.e. sent by the lecturer)?

Is This Birth Certificate Authentic?

| STATE OF HAWAII | | CERTIFICATE OF LIVE BIRTH | | | DEPARTMENT OF HEALTH | | |
|--|--|--|---|----------------------------------|--------------------------|--------------|--|
| | | | | | FILE NUMBER 151 61 10641 | | |
| 1a. Child's First Name (Type or print) | | 1b. Middle Name | | 1c. Last Name | | | |
| BARACK | | HUSSEIN | | OBAMA, II | | | |
| 2. Sex Male | 3. This Birth Single <input checked="" type="checkbox"/> Twin <input type="checkbox"/> Triplet <input type="checkbox"/> | 4. If Twin or Triplet, Was Child Born 1st <input type="checkbox"/> 2nd <input type="checkbox"/> 3rd <input type="checkbox"/> | 5a. Birth Date | Month August | Day 4, | Year 1961 | |
| 6a. Place of Birth: City, Town or Rural Location Honolulu | | 6b. Island Oahu | | | | | |
| 6c. Name of Hospital or Institution (If not in hospital or institution, give street address) Kapiolani Maternity & Gynecological Hospital | | | 6d. Is Place of Birth Inside City or Town Limits? If no, give judicial district Yes <input checked="" type="checkbox"/> No <input type="checkbox"/> | | | | |
| 7a. Usual Residence of Mother: City, Town or Rural Location Honolulu | | 7b. Island Oahu | 7c. County and State or Foreign Country Honolulu, Hawaii | | | | |
| 7d. Street Address 6085 Kalanianaole Highway | | | 7e. Is Residence Inside City or Town Limits? If no, give judicial district Yes <input checked="" type="checkbox"/> No <input type="checkbox"/> | | | | |
| 7f. Mother's Mailing Address | | | 7g. Is Residence on a Farm or Plantation? Yes <input type="checkbox"/> No <input checked="" type="checkbox"/> | | | | |
| 8. Full Name of Father BARACK HUSSEIN OBAMA | | | 9. Race of Father African | | | | |
| 10. Age of Father 25 | 11. Birthplace (Island, State or Foreign Country) Kenya, East Africa | 12a. Usual Occupation Student | 12b. Kind of Business or Industry University | | | | |
| 13. Full Maiden Name of Mother STANLEY ANN DUNHAM | | | 14. Race of Mother Caucasian | | | | |
| 15. Age of Mother 18 | 16. Birthplace (Island, State or Foreign Country) Wichita, Kansas | 17a. Type of Occupation Outside Home During Pregnancy None | 17b. Date Last Worked | | | | |
| I certify that the above stated information is true and correct to the best of my knowledge. | | 18a. Signature of Parent or Other Informant Dawn Dunham Obama | | 18b. Date of Signature 8-7-61 | | | |
| I hereby certify that this child was born alive on the date and hour stated above. | | 19a. Signature of Attendant David A. [Signature] | | 19b. Date of Signature 8-8-61 | | | |
| 20. Date Accepted by Local Reg. AUG - 8 1961 | 21. Signature of Local Registrar V. Lee | | | | | | |
| 23. Evidence for Delayed Filing or Alteration | | | | | | | |

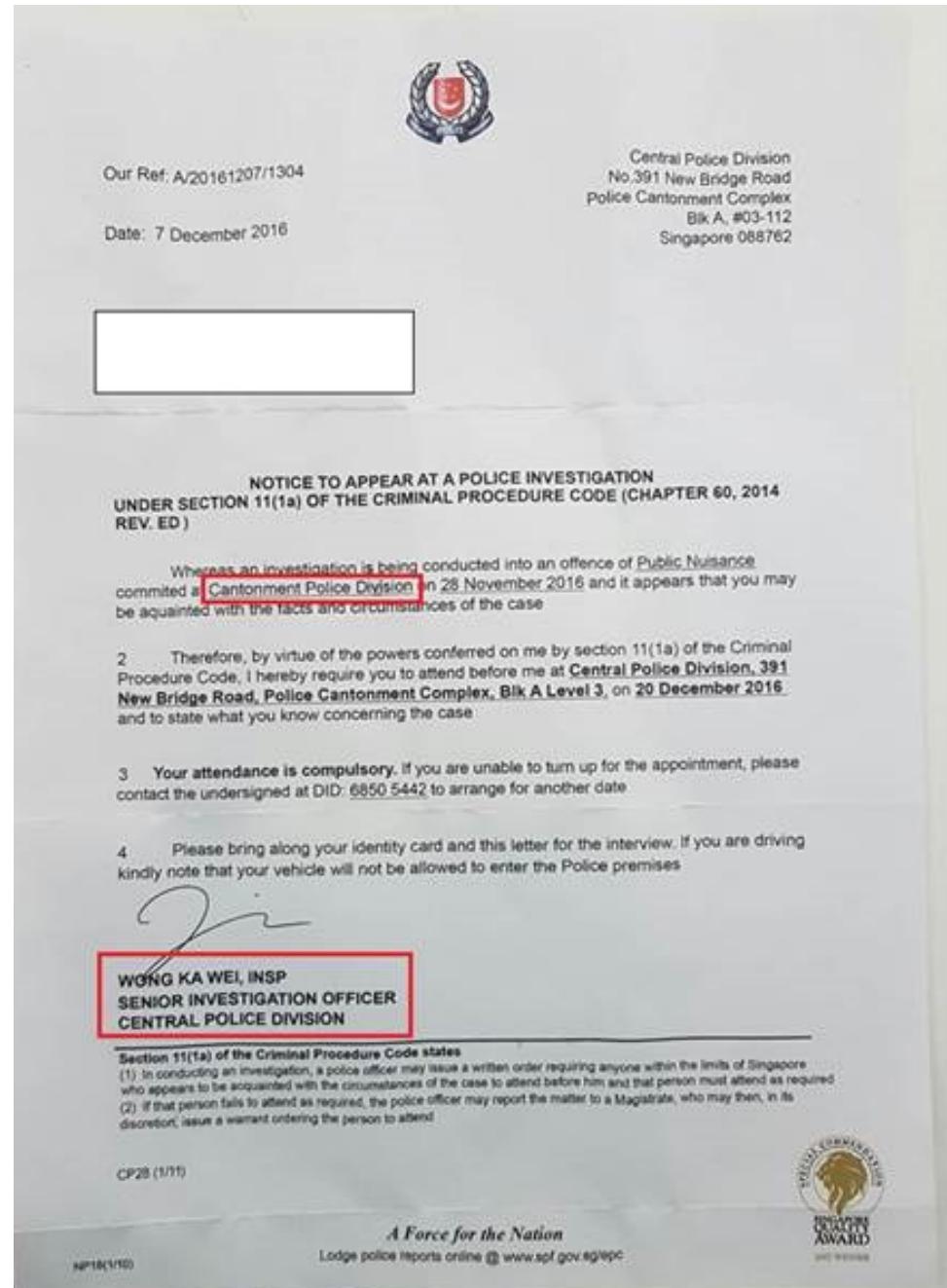


Is This Letter Authentic? (Actual Case in Singapore)

From:

https://www.police.gov.sg/news-and-publications/media-releases/20161217_others_advisory_spf_letters,

December 16, 2016



2.2 Password (Weak authentication)

Password: An Authentication System

Stage 1: Bootstrapping

- Server and a user establish a common password
- The server keeps track of a file recording the *identity* (i.e. *userid*, *username*) and the corresponding *password*

Stage 2: Authentication

- The server authenticates an entity
- If the entity gives the correct password corresponding to the claimed identity, the entity is deemed authentic

Password: An Authentication System

- The **identity** does not need to be kept secret:
 - It could be: username in a computer system, bank account no, customer id, etc.
- The **password** is a secret:
 - Only the authentic user and the server know it
 - The fact that an entity knows the password implies that it is either the server or the authentic user

Question: Analyze a password system where no identity is involved, i.e., just password.

You can read:

<https://technet.microsoft.com/en-us/library/cc512578.aspx>

Identification, Authentication, Authorization

The differences?

| Process | Provided By | To Answer | Attributes | Uniqueness Requirement |
|----------------|-------------|-------------------------|------------------------------|------------------------|
| Identification | Principal | “Who are you?” | Public assertion | Yes (locally) |
| Authentication | Principal | “How can you prove it?” | Secret response | No |
| Authorization | System | “What can I do?” | Token/ticket, access control | - |

From: <https://technet.microsoft.com/en-us/library/cc512578.aspx>

Stage 1: Bootstrapping

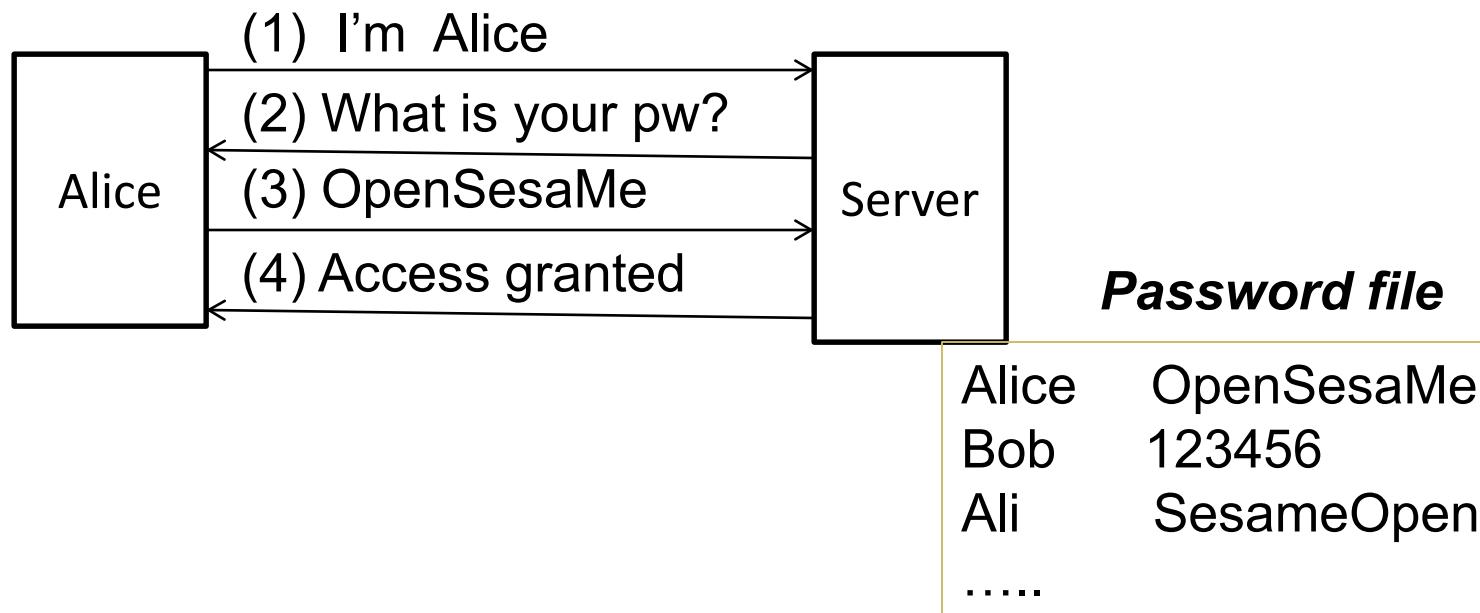
- The password is to be established during bootstrapping
- This can be done by either:
 1. The server (user) chooses a password, and sends it to the user (server) through another communication channel
 2. Default password

Question: Describe some bootstrapping mechanisms that you have encountered (e.g. NUSNET, Singpass, WiFi router)

Stage 2: Password-based Authentication

- Typical interaction:
 - User → Server : My name is *Alice*
 - Server → User : OK. *Alice*, what is your password?
 - User → Server : *OpenSesame*
 - Server : OK. You are indeed *Alice*.
- Alternatively, authentication can be carried out without interactions:
 - User just sends the following SMS to a server:
Userid: *Alice@nus.edu.sg*. Password: *OpenSesame*.
Instruction: Unsubscribe (from your mailing list.
No more junk mail please)

System Diagram



Weak Authentication System and Replay Attack

- Password system is classified as a “***weak authentication***” system
- A weak authentication is one that subjected to this simple “***replay attack***”: information sniffed from the communication channel can be used to impersonate the user at a later time
- In contrast, under “***strong authentication***”:
 - Information sniffed during the process can't be used to impersonate the user
 - We will briefly look into this in PKI later

Question (Terminologies): What are “*Sniff*” and “*Spoof*”?

Attacks on Password System

Different possible attacks:

- Attack the bootstrapping
- Searching for the password:
 - Guessing
 - Dictionary attacks
 - Exhaustive attacks
- Stealing the password:
 - Eavesdropping: sniff the network, use key logger
 - Phishing
 - Spoofing login screen
 - Password caching
 - Insider attacks

2.2.1 Attack the Bootstrapping

Possible Attacks on Bootstrapping

- Attacker may intercept the password during bootstrapping:
 - Example: if the password is sent through postal mail, an attacker could steal the mail to get the password
- An attacker uses the “default” passwords:
 - There are many reported incidents on this attack (e.g. IP camera, WiFi router)
 - See <http://www.pcworld.com/article/2033821/widely-used-wireless-ip-cameras-open-to-hijacking-over-the-internet-researchers-say.html>

Read (Mirai botnet attack):

- <http://www.computerworld.com/article/3134097/security/chinese-firm-admits-its-hacked-products-were-behind-fridays-ddos-attack.html>

Default Password on IP Camera: Real Example



Question

Question: ([Gollmann] Pg 64)

You are shipping WLAN access points.

Access to these devices is protected by password.

- What are the implications of shipping all access points with the same *default password*?
- What are the implications of shipping each access point with its *individual password*?

(Hint: Argue from the viewpoint of usability vs security)

2.2.2 Searching for the Password

[PF2.1] Guessing the Password from Social Information

- The attacker gathers some social information about the user, and infer the password
 - E.g. mobile phone number, spouse's name
- Password guessing types:
 - Online guessing: an attacker directly interacts with the authentication system
 - Offline guessing: an attacker can obtain the password file from the authentication system

Dictionary Attacks

- The attacker tries different passwords during login sessions
- The attacker can employ **exhaustive search**: tries all combinations
Is it feasible? See the table on possible *key space sizes* of different character sets and password lengths
- Alternatively, the attacker can restrict the search space to a large collection of *probable passwords*:
 - Words from English dictionary, known compromised passwords, other language dictionaries, etc.
 - This is known as ***dictionary attack***

Table 3-1. Possible Keystreams by Password Length and Character Set Size

| Char. Set Size | Character Types | | | | Password Length | | | | |
|----------------------|-----------------|------------------|----------------------------------|----------------------------|-----------------|-------------|-------------|-------------|-------------|
| | Digits | Letters | Symbols | Other | 4 | 8 | 12 | 16 | 20 |
| 10 | Decimal | | | | $1*10^4$ | $1*10^8$ | $1*10^{12}$ | $1*10^{16}$ | $1*10^{20}$ |
| 16 | Hexa-decimal | | | | $7*10^4$ | $4*10^9$ | $3*10^{14}$ | $2*10^{19}$ | $1*10^{24}$ |
| 26 | | Case-insensitive | | | $5*10^5$ | $2*10^{11}$ | $1*10^{17}$ | $4*10^{22}$ | $2*10^{28}$ |
| 36 | Decimal | Case-insensitive | | | $2*10^6$ | $3*10^{12}$ | $5*10^{18}$ | $8*10^{24}$ | $1*10^{31}$ |
| 46 | Decimal | Case-insensitive | 10 common ⁷ | | $4*10^6$ | $2*10^{13}$ | $9*10^{19}$ | $4*10^{26}$ | $2*10^{33}$ |
| 52 | | Upper and lower | | | $7*10^6$ | $5*10^{13}$ | $4*10^{20}$ | $3*10^{27}$ | $2*10^{34}$ |
| 62 | Decimal | Upper and lower | | | $1*10^7$ | $2*10^{14}$ | $3*10^{21}$ | $5*10^{28}$ | $7*10^{35}$ |
| 72 | Decimal | Upper and lower | 10 common | | $3*10^7$ | $7*10^{14}$ | $2*10^{22}$ | $5*10^{29}$ | $1*10^{37}$ |
| 95 | Decimal | Upper and lower | All symbols on standard keyboard | | $8*10^7$ | $7*10^{15}$ | $5*10^{23}$ | $4*10^{31}$ | $4*10^{39}$ |
| 222 | Decimal | Upper and lower | All symbols on standard keyboard | All other ASCII characters | $2*10^9$ | $6*10^{18}$ | $1*10^{28}$ | $3*10^{37}$ | $8*10^{46}$ |

Table from: Guide to Enterprise Password Management (Draft), NIST, 2009 <http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf>

Dictionary Attacks

- **Hybrid attack:** it is possible to carry out exhaustive search together with dictionary attack
- Example: try all combinations of 2 words from the dictionary, and exhaustively try all possible capitalizations of each word, substituting “a” by “@”, etc.
- See list of “2014 worst password” reported by SplashData:
<http://www.prweb.com/releases/2015/01/prweb12456779.htm>

Question: Download a password dictionary.
Is your password listed in the dictionary?

Presenting SplashData's "Worst Passwords of 2014":

- 1 123456 (Unchanged from 2013)
- 2 password (Unchanged)
- 3 12345 (Up 17)
- 4 12345678 (Down 1)
- 5 qwerty (Down 1)
- 6 1234567890 (Unchanged)
- 7 1234 (Up 9)
- 8 baseball (New)
- 9 dragon (New)
- 10 football (New)
- 11 1234567 (Down 4)
- 12 monkey (Up 5)
- 13 letmein (Up 1)
- 14 abc123 (Down 9)
- 15 111111 (Down 8)
- 16 mustang (New)
- 17 access (New)
- 18 shadow (Unchanged)
- 19 master (New)
- 20 michael (New)
- 21 superman (New)
- 22 696969 (New)
- 23 123123 (Down 12)
- 24 batman (New)
- 25 trustno1 (Down 1)

Famous Case



From: Wikipedia

Ben Hall
@Ben_Hall

Follow ▾

Ouch. Mark Zuckerberg's social media accounts have been hacked

↑ 7 Replies

Mark Zuckerberg @finkd Hey, @finkd You were in Linkedin Database with the password "dadada" ! DM for proof..

12 Faves 9 Retweets

5 Jun 2016 at 20:01 via Twitter Web Client

Continue

Pinterest works best if you switch to our iPhone-friendly app

Pinterest for iPhone ★★★★★ 700,000+ 5-star reviews

Log in with browser

Hacked By OurMine Team - Read @zuck

575 Retweets 368 Likes



His hacked password was: *****

2.2.3 Stealing the Password

Shoulder Surfing, Sniffing

- ***Shoulder surfing***: look-over-the-shoulder attack
- ***Sniffing***: listening/intercepting the communication channel:
 - Some systems and protocols simply send the password over a public network in clear (i.e. unencrypted)
 - Examples: FTP, Telnet, HTTP
- ***Sniffing*** a wireless keyboard:
See <http://arstechnica.com/security/2015/01/meet-keysweeper-the-10-usb-charger-that-steals-ms-keyboard-strokes/>
- Other method: using sound made by a keyboard:
([L. Zhuang, F. Zhou, J.D. Tygar, “Keyboard Acoustic Emanations Revisited”, 2005](#))

Question (Terminology): What is a “*side channel attack*” ?

Some Fun Videos to Watch: Live Password Leakage



<http://securityaffairs.co/wordpress/35856/cyber-crime/tv5monde-investigation-details.html>

<http://www.bbc.com/news/world-europe-32248779>

In a live interview, a TV5Monde staffer accidentally revealed a password used to access the broadcaster's social media account!

Key-Logger

- A ***key-logger*** captures/records the keystrokes, and sends the information back to the attacker, via a “covert channel”
- *By software:*
Some computer viruses are designed as a ***key-logger***
- *By hardware:*
Hardware key-logger: see the next slide for an example
- See “Hardware-based keyloggers” in
http://en.wikipedia.org/wiki/Keystroke_logging

Hardware Key-Logger



From http://en.wikipedia.org/wiki/Keystroke_logging

Question (Terminology): What is a “*covert channel*”?

Login Spoofing

- Attacker displays a “spoofed” (fake) login screen



- Prevention:
 - Some systems have a ***secure attention key*** or ***secure attention sequence*** (e.g. Ctrl+Alt+Del for Window NT)
 - When they are pressed, the system starts the trusted login processing

Phishing

- Similar to login spoofing
- The user is tricked to voluntarily sends the password to the attacker over the network
- Phishing attacks ask for password under some false pretense. For example:

★ Lynn Luckett

IT Care

21 January 2015 2:31 pm

LL



Attn NUS Staff:

An attempt was made to connect your account from a new computer. For your account security, click the link below and fill accurate details to protect your account.

Copy or Click here: <http://www.pjserver.com/form/forms/form1.html>

IT Care.

© Copyright 2001-2015 National University of Singapore. All Rights Reserved.

This email is confidential and intended solely for the use of the individual to whom it is addressed. If you are not the intended recipient, be advised that you have received this email in error, and that any use, dissemination, forwarding, printing, or copying of this email is prohibited. If you have received this email in error, please contact the sender.

A Real Recent Phishing Attempt (in NUS)

From: ITCARE
To: [REDACTED]
Subject: [Ticket #645159] Someone has accessed your account
Date: Monday, March 27, 2017 9:35:44 AM
Importance: High

Dear [REDACTED]

Someone just tried to sign in to your account. We have stopped this sign-in attempt.

Details:

IP Address: 95.108.142.138

Location: Russia

You are advised to change your password immediately.

[Change NUSNET Password](#)

Please [Sign In](#) to NUSNET password page.

Note:

- Your password must be at least 8 characters in length.
- Your password cannot contain your userID or any part of your name.
- You cannot re-use any of your 6 old passwords.
- You cannot change your password more than once in a day.

Another Example: POSB Phishing

Secure | https://www.channelnewsasia.com/news/singapore/phishing-scheme-targeting-posb-customers... :

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Singapore Edition | Watch TV | Sign in | All Sections

Singapore

Phishing scheme targeting POSB customers: DBS

Sample of the malicious email:

-----Original Message-----

From: customerservice@posbbank.com.sg This is not a DBS/POSB email address

Date : 05/05/2018 - 11:43 AM Given To : This is not a DBS/POSB email address

To : This is not a DBS/POSB email address

Subject : Security Update



Dear Customer,

Kindly be informed that Singapore Banks has been under attack by hackers, and this has cost some customers to lose their money to this <https://virutallin.gq/secure/update>. The Monetary Authority of Singapore to enact a new law mandating all customer This is not a DBS/POSB email address to keep their money safe with banks. This is not a DBS/POSB email address

Kindly click on www.posb.com.sg/secure/update to update your account with us and to keep your money safe. Failure to follow this instructions might result in the deactivation of your account. Please note that we will not be responsible for any Link to phishing website account if you fail to update.

Sorry For any inconvenience this may have caused you.

Thank you for using DBS Bank

Screenshot from the DBS phishing alert, showing a sample of the malicious email.

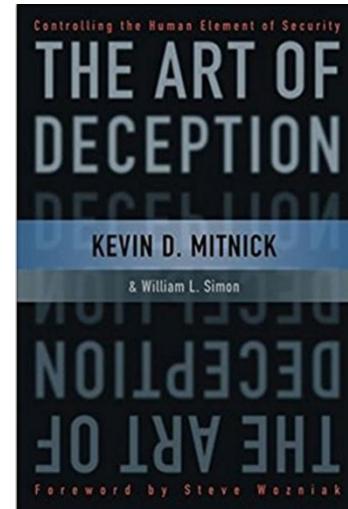
From: Channel News Asia, 4 May, 2018

Spear Phishing

- Phishing can be targeted to a particular small group of users (e.g. NUS staff in the above example)
- Such attack is generally known as ***spear phishing***, which is an example of ***targeted attacks***
- Phishing attack is a type of ***social engineering*** attack
- Wiki definition of social engineering:
“Social engineering, in the context of information security, refers to psychological manipulation of people into performing actions or divulging confidential information.”
See http://en.wikipedia.org/wiki/Social_engineering_%28security%29

More on Phishing

- Phishing of passwords is typically done through emails, but can also be carried out over phone calls
- Spear-phishing can be very effective
See [PFpage275], Sidebar 4-11
- More on social engineering techniques:
Kevin D. **Mitnick** and William L. Simon,
“The Art of Deception: Controlling the Human Element of Security”, 2003



Question (Terminology):

What are **Phishing**, **Pharming**, **Vishing** and **Smishing**?

(You can read for e.g.: <http://csbweb.com/phishing.htm>)

Preventing Phishing

- User education: phishing drill

BS Batam Sea View Resort via OneBooking.com <Club55@travelrefundshq.com>
Your Booking is Confirmed for Batam Sea View Resort
This message was sent with High importance.

OneBooking.com Booking No.:12782601890

Thanks Jitendera Sarda!

Your Booking for **Batam Sea View Resort** in **Indonesia** Is Confirmed

You'll pay the property directly. Batam Sea View Resort in Indonesia handles all payments, so please check below for more information.

[Change your booking](#)

Your reservation : 2 nights, 1 Superior room
Check in : Wednesday, 9 Aug 2017 (from 15:00)
Check out : Thursday, 10 Aug 2017 (until 12:00)
Price : S \$149

*You will pay direct to Batam Sea View Resort. 7% Tax is excluded. 10% service charge is excluded.
This reservation cannot be cancelled free of charge.*

[Cancel your booking](#)

This Superior Room has a satellite TV and air conditioning

Guest Name Jitendera Sarda
Number of Guests max 2 people
Meal Plan Breakfast is included

Are you expecting this email? Does the email address ends with the domain, Onebooking.com?

Mouse over the hyperlink without clicking it. Does it shows the domain, OneBooking.com?

Flustering recipient by creating a sense of urgency and scaring are common tactics used in Phishing email.

Preventing Phishing

- Phishing repository site:
 - Example: phishtank.com (submit suspected phishes, track the status of your submissions, verify other users' submissions)
- However, it can be tricky to accurately determine if an unsolicited email is a phishing
 - Example: SonicWall Phishing IQ Test
<https://www.sonicwall.com/phishing/>
 - *You can test your own phishing-spotting skill!*
- Any good/secure way of verifying a suspected phishing email?
 - When in doubt, call for help/clarification!?

Password Caching

- When using a shared workstation (for e.g. a browser in airport), information keyed-in could be cached
- The next user may able to see the cache
- Prevention: Clear the browser's cache and close the browser when using a shared workstation

Insider Attack

Some examples:

- A malicious system admin who steals the password file
- The system admin's account is compromised (e.g. password stolen via phishing), leading to a lost of password file

2.2.4 Preventive Measures

Use Strong Password

- Randomly chosen:
 - A password is chosen randomly among all possible keys using an *automated password generator*
 - High “entropy” but difficult to remember:
e.g. 3n5dcvUD9cfm (10 characters)
- User selection:
 - Mnemonic method: Pbmbval!
 - Altered passphrases: Dressed*2*tge*9z
 - Combining and altering word: B@nkC@mera

Remark: Pbmbval! is no longer a good choice since it had appeared as examples in many document on password selection
- **Read** page 3-10 of “Guide to Enterprise Password Management (Draft)”, NIST, 2009
<http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf>

Password Protection

- **Limited login attempts:**
 - Add delay into login session
 - Add security questions
 - Lock the account after a few failed attempts
- **Password checker:**
 - Check for weak password when user registers/ changes password (for e.g. using password dictionary)
- **Password metering:**
 - Indicate weak, average, strong passwords

Password Protection

- **Password ageing:**
 - Users must regularly change passwords
 - Nevertheless, many believe that frequent changes of passwords actually lower security
 - See https://www.schneier.com/blog/archives/2016/08/frequent_passwo.html
- **Password usage policy:**
 - Rule set by an organization to ensure that users use strong passwords, and minimize password loss
 - Example: the policy may state that a password has to be at least 10 characters

Question:

What is NUSNET password policy?
Does the password expire?

Protecting Password File

- Recap: the *password file* stores userid+password
- It could be leaked, due to:
insider attack, accidental leakage, hacked system, etc.
- There are many well-known incidents where unprotected or weakly protected password files are leaked, leading to a large number of passwords being compromised
- See “2012 LinkedIn Hack”:
https://en.wikipedia.org/wiki/2012_LinkedIn_hack
- Hence, it is desired to add an *additional layer of* protection to the password file

Hashed Password (Revisit This Slide after Hash is Covered)

- Passwords should be “hashed” and stored in the password files.
(Textbook ([PF]pg 46) uses the term “encrypted”. Note that this is inaccurate. For encryption, there is a way to recover the password from the ciphertext. For cryptographically secure hash, it is infeasible to recover the password from its hashed value.)
- During authentication, the password entered by the entity is hashed, and compared with the the value stored in the password file

Password in clear

| | |
|---------|------------|
| Alice | OpenSesaMe |
| Bob | 123456 |
| Ali | SesameOpen |
| Charles | SesameOpen |

Hashed password

| | |
|---------|-------------|
| Alice | X3lad=3adfV |
| Bob | 3Dv6usgawer |
| Ali | da5DGDSDFd3 |
| Charles | da5DGDSDFd3 |

$$\text{“da5DGDSDFd3”} = \text{Hash(“SesameOpen”)}$$

Hashed Password (Revisit This Slide after Hash is Covered)

- It is desired that the same password will be hashed into *two different values* for two different userid. Why? (See tutorial)
- This can be achieved by using *salt*

Password in clear

| | |
|---------|------------|
| Alice | OpenSesaMe |
| Bob | 123456 |
| Ali | SesameOpen |
| Charles | SesameOpen |

Salted-hashed password

| | | |
|----------|-------|-------------|
| Alice, | Adf3, | 39Gkaj10Dmf |
| Bob, | a3gh, | d978bjklDFD |
| Ali, | f8ad, | DJk34hoaev7 |
| Charles, | 10vd, | K108ELvio2B |

“DJk34hoaev7”= Hash(“f8adSesameOpen”)
“K108ELvio2B”= Hash(“10vdSesameOpen”)

2.2.5 Security Questions

Read

https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet

Optional:

Ariel Rabkin, “*Personal Knowledge Questions for Fallback Authentication: Security Questions in The Era of Facebook*”, Usable Privacy and Security, 2008

Usage and Attacks

- Security questions can be viewed as a mechanism for ***fallback authentication*** or a ***self-services password reset***:
 - + *Enhancing usability*: a user can still login even if password is lost
 - + *Reducing cost*: it reduces operating cost of helpdesk
 - *Weakening security*: attackers have another mean to obtain access
- Common “secret” questions?
 - Name your pet, aunt’s middle name, movie...
 - Problem: not really secret!
- See [PF2.1] SideBar 2-1 & 2-2 on known past incidents:
 - US vice presidential candidate Sarah Palin’s Yahoo! email hack
 - Attack by George Bronk by scanning Facebook pages

Choices of Security Questions

(From:https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet)

- ***Memorable***: If users can't remember their answers to their security questions, you have achieved nothing
- ***Consistent***: The user's answers should not change over time. For instance, asking "What is the name of your significant other?" may have a different answer 5 years from now
- ***Nearly universal***: The security questions should apply to a wide audience of possible
- ***Safe***: The answers to security questions should not be something that is easily guessed, or research (e.g., something that is matter of public record)

Question: Give example of “bad” security questions

2.2.6 ATM Attacks

ATM Card

- To get authenticated, the user has to present a *card* and the *PIN*
- ***The card*** contains a magnetic stripe, which stores the *user account id*
- Essentially, the magnetic stripe *simplifies the input of account id* into the ATM system: instead of keying it in, just insert the card
- ***The PIN*** plays the role of password
- Data are encoded into the magnetic stripe using ***well-known standards***. Given a valid card, an attacker can “copy” the card by reading the info from the card, and write it to a spoofed card.



This card can be purchased from ebay ☺

ATM Skimmer

- An ATM skimmer steals the victim's account id (username) and PIN (password)
- The skimmer consists of:
 1. A card-reader attached on top of existing ATM reader
 2. A camera overlooking the keypad, or a spoofed key-pad on top of existing keypad
 3. Some means to record and transmit the information back to the attacker
- With the information obtained from:
 - (1): the attacker can spoof the victim's ATM card
 - (2): the attacker obtain the PIN
- Well-known incidents in Singapore: DBS in 2012

“\$1 million stolen from the bank accounts of 700 DBS and POSB customers.”

See <http://news.asiaone.com/News/Latest+News/Singapore/Story/A1Story20120223-329820.html>

Self-Explanatory Images of ATM SKIMMING



Synopsis:

Fictitious card reader and cellular telephone with a video camera attached to ATM machine. The fictitious card reader is flush to compromised ATM whereas the others are recessed. A façade of ATM colored molding is attached to upper part of ATM. The façade conceals a cellular phone camera which records the PIN number.

Fun Video to Watch: Very Big ATM Skimmer



https://www.liveleak.com/view?i=bea_1457038390

Another (really big) ATM skimmer!

Preventive Measures

- Install anti-skimmer device: a device that prevents external card reader to be attached onto the ATM



- Shield the keypad



- User awareness
 - Use newer chip-based (EMV) cards, which use encryption
- See: <https://en.wikipedia.org/wiki/EMV>,
https://www.youtube.com/watch?v=B2iABG53h_0

Some Fun Videos to Watch: POS Skimmer Installation



https://www.youtube.com/watch?v=_BFRD8_LrcM

CCTV caught someone deploying a Point-Of-Sale skimmer (similar to ATM skimmer)

More video:

“Why Chip Credit Cards Are Still Not Safe From Fraud”

<https://www.youtube.com/watch?v=gJo9PfsplsY>

2.3 Biometrics

Reading:

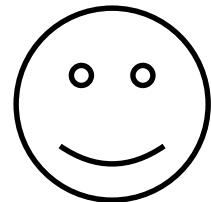
[PF] page 53-64

Biometrics

- Biometrics use *unique physical characteristics* of a person for authentication
- During ***enrollment***, a ***reference template*** of an user's biometric data is constructed and stored (similar to bootstrapping in password system)
- During ***verification***, biometric sample data of the person-in-question is captured and compared with the template using a *matching algorithm*
- The algorithm decides whether to accept or reject
- Biometrics can be used for:
 - *Verification* (our focus in this lecture): 1:1 verification whether the person is the claimed person
 - *Identification*: 1: n comparison to identify the person from a database of many persons

Process Diagram

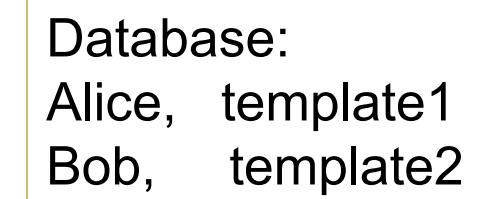
Enrollment



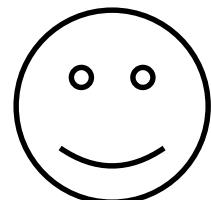
Bob



biometric data



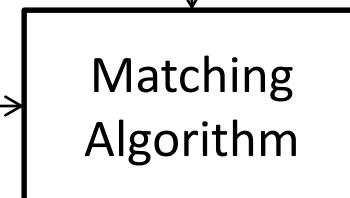
Verification (Authentication)



Bob



biometric data,
identity



accept

reject

Differences between Biometric and Password

| Password | Biometric |
|---|-----------------------------|
| Can be changed (revoked) | Can't |
| Need to remember | Don't have to |
| <i>Zero non-matched rate</i> | <i>Probability of error</i> |
| Users can pass the password to another person | Not possible |

Matching Algorithm: Similarity/Inexact Matching

- Unlike password, there are inevitable *noises* in capturing the biometric data, leading to *error* in making the matching decision: FMR (False match rate) and FNMR (false non-match rate) .

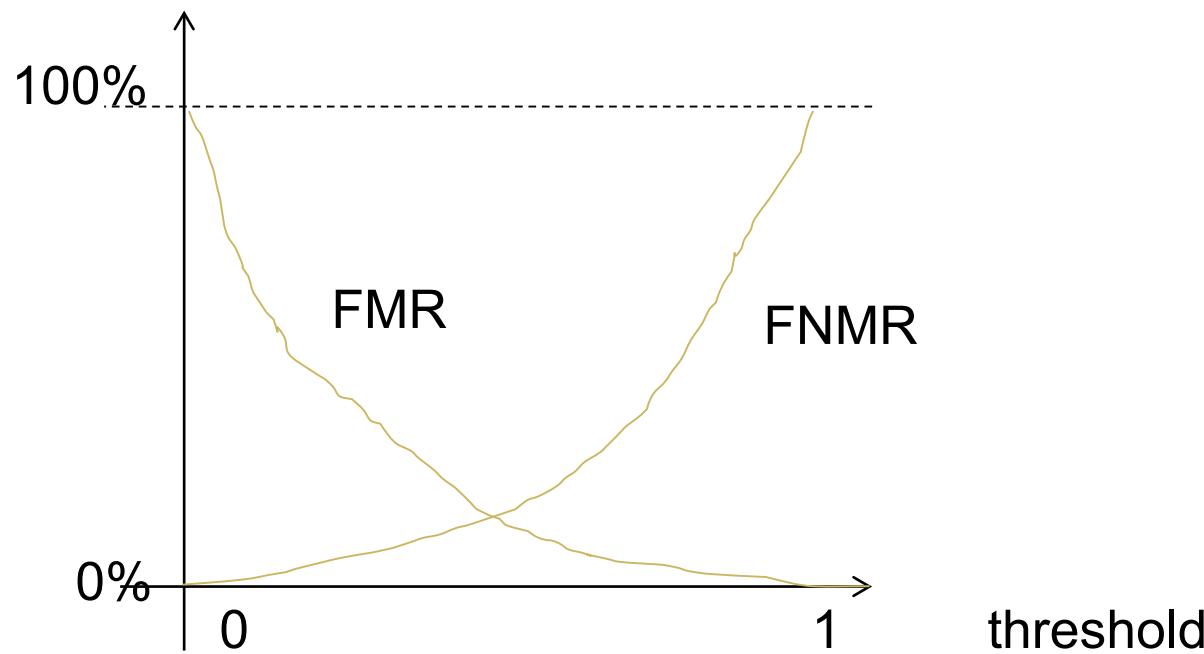
$$\text{FMR} = \frac{\text{number of successful false matches (B)}}{\text{number of attempted false matches (B+D)}}$$

$$\text{FNMR} = \frac{\text{number of rejected genuine matches (C)}}{\text{number of attempted genuine matches (A+C)}}$$

| | | accepted/ | rejected/ |
|-----------------|---|-----------|-----------|
| | | match | non-match |
| genuine attempt | A | C | |
| | B | D | |

Threshold Value Selection

- The matching algorithm typically makes decision based on some adjustable *threshold*
- By adjusting the threshold, the FMR and FNMR can be adjusted:
 - Lower threshold → more relax in accepting
 - Higher threshold → more stringent in accepting



How to set the threshold? It depends on applications

Other Types of Errors

Equal error rate (EER):

- The rate when FNMR = FMR

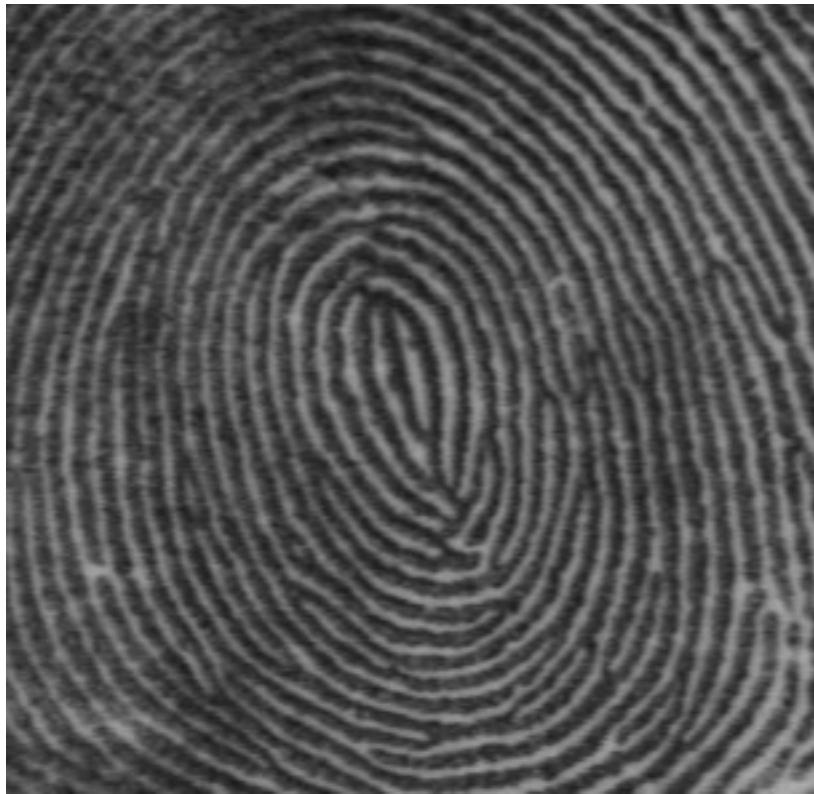
False-to-enroll rate (FER):

- Some users' biometric data can't be captured for enrollment
- For example: due to injury

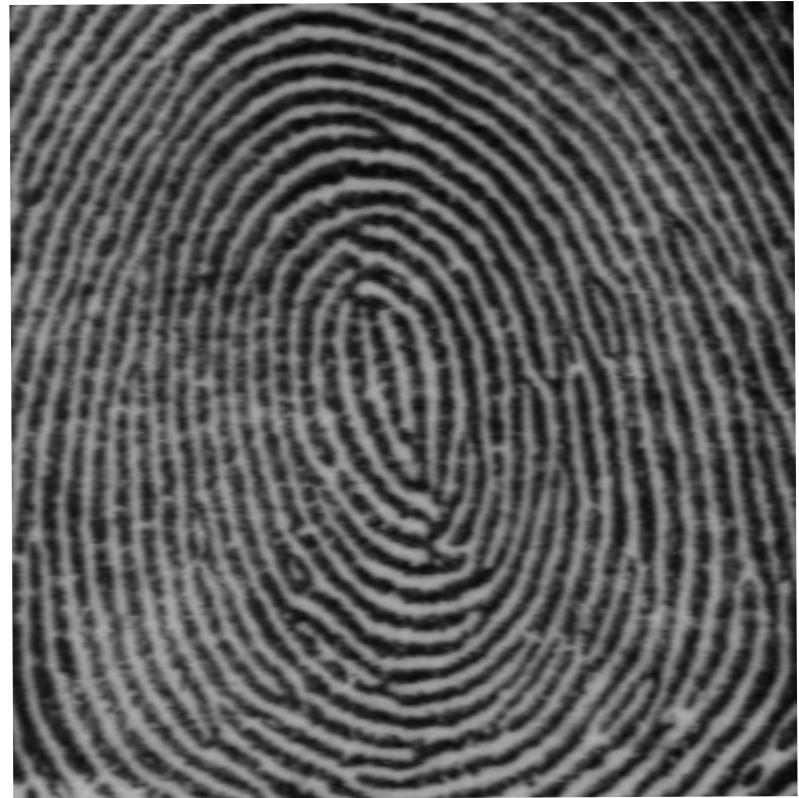
Failure-to-capture rate (FTC):

- A user's biometric data may fail to be captured during authentication
- For examples: fingers are too dry, dirty, etc

Examples on Fingerprint

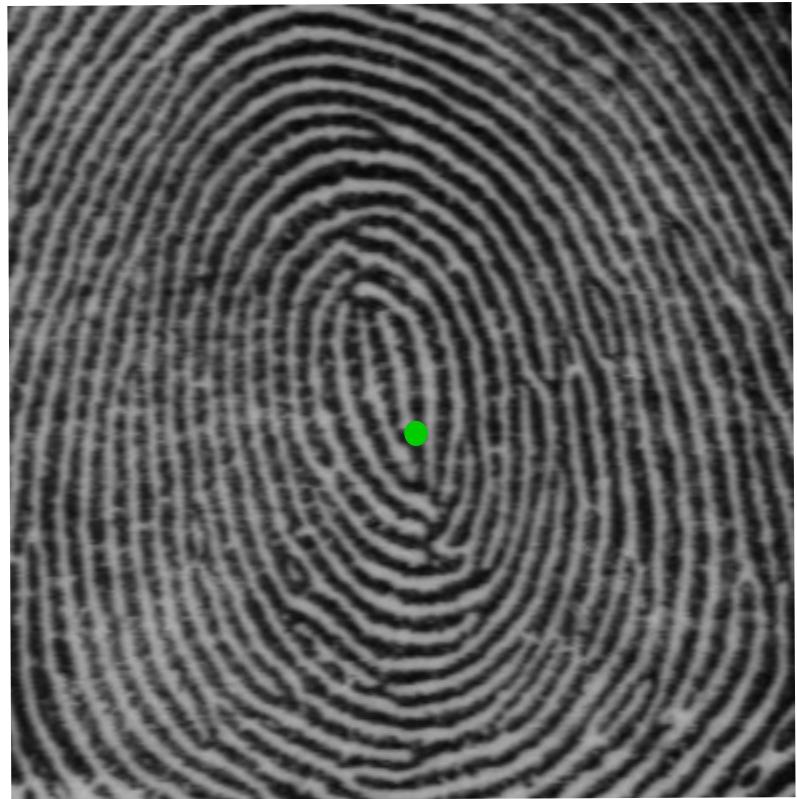
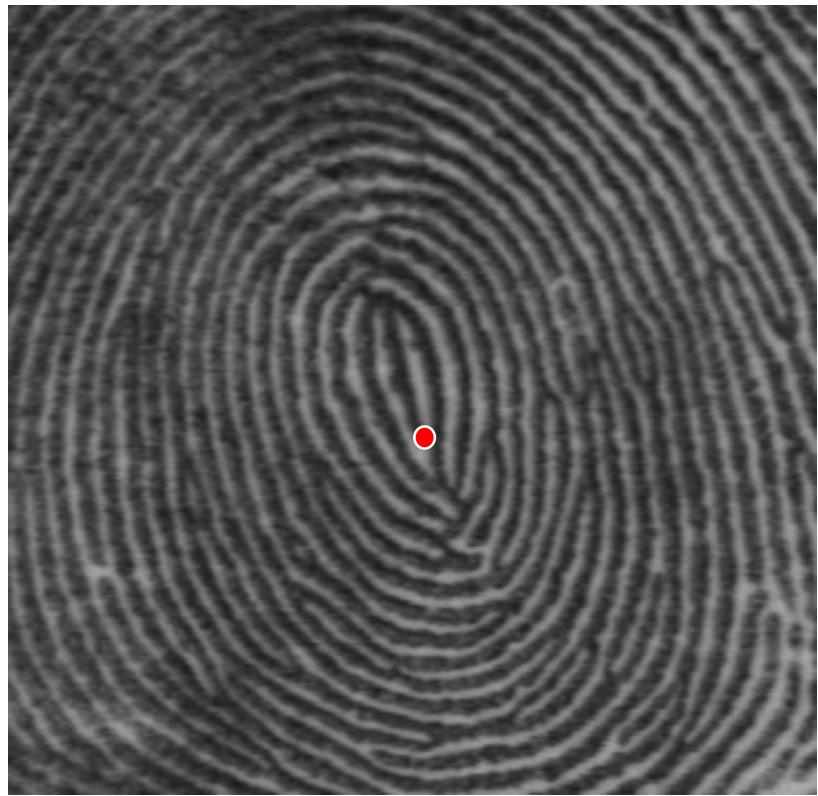


First scan of a finger



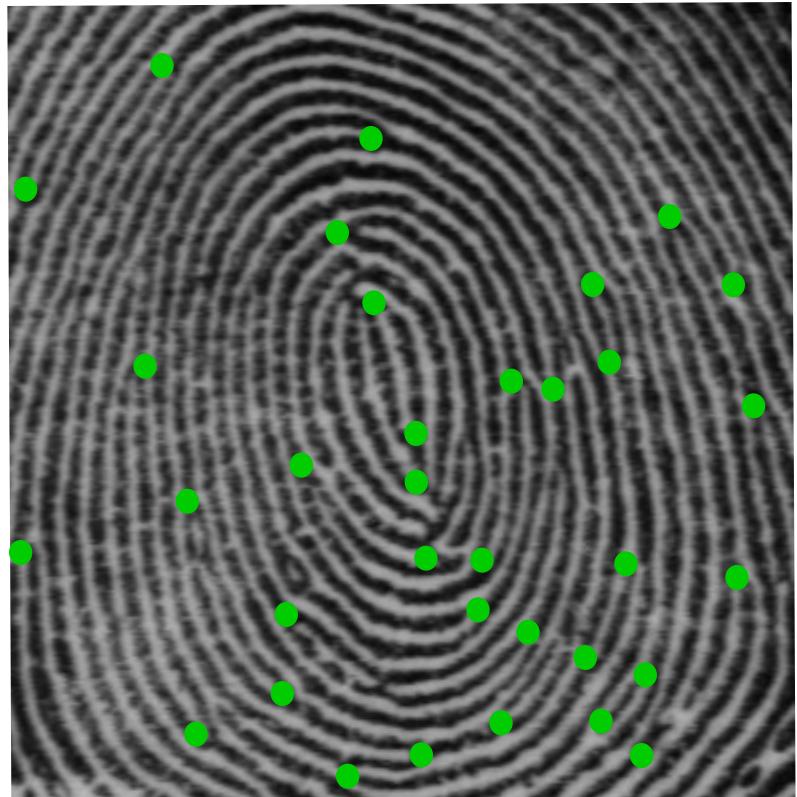
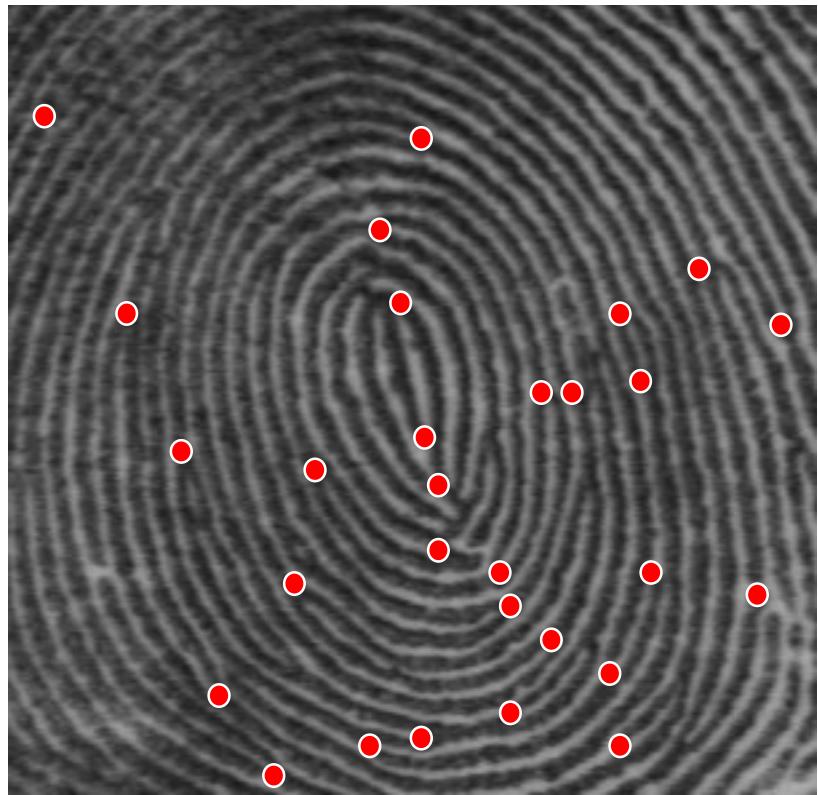
Another scan
of the same finger

Fingerprint: Background



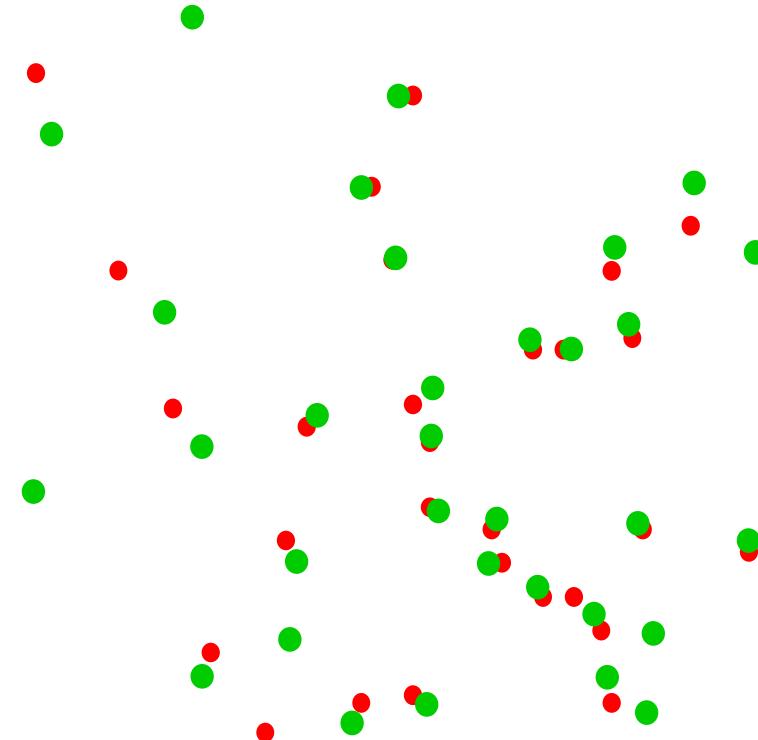
A feature point

Fingerprint: Background



The set of feature points
(known as *minutiae* for fingerprint)

The features points extracted from the two scans are similar,
but *not exactly the same*!



How Good is Fingerprint as a Biometric?

- Performance depends on the quality of the scanner
- EER can range from 0.5 to 5% depending on quality of scanners
- See result of Fingerprint Verification Competition FVC2006
<http://bias.csr.unibo.it/fvc2006/default.asp>

Other Forms of Biometrics

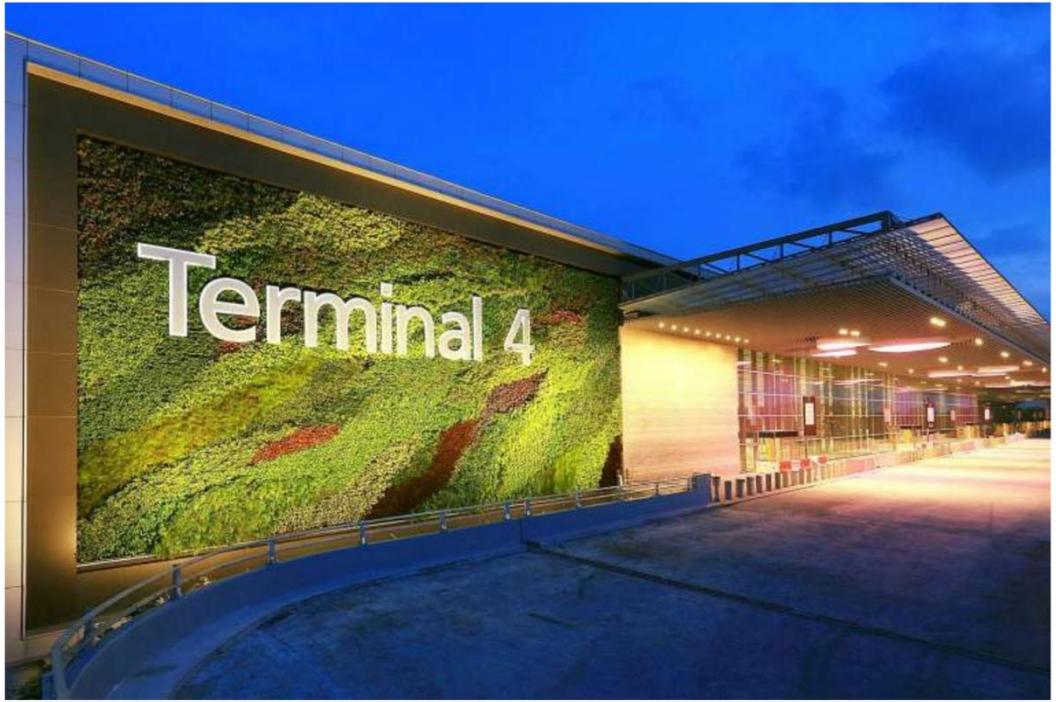
- Palm print, palm veins, hand geometry, face, iris, retina, DNA
- Others?
Tounge, odour/scent

The Straits Times,
Mar 12, 2017

ST Multiple biometric check x

www.straitstimes.com/singapore/multiple-biome... 🔍 Q ☆ ⌂

Multiple biometric checks at Changi Airport Terminal 4

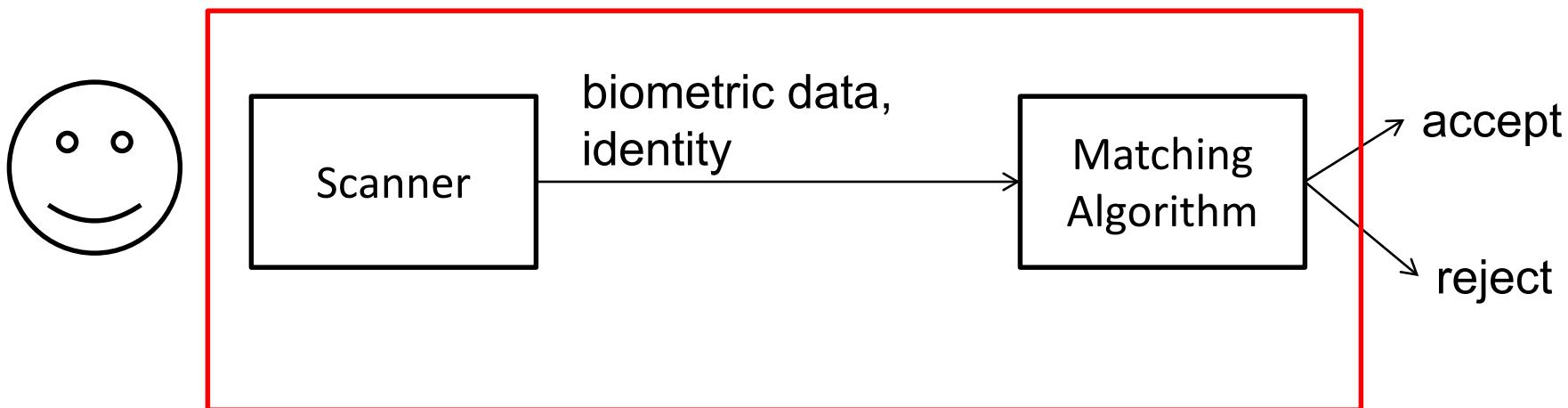


Changi Airport Terminal 4 will be Singapore's first checkpoint with multiple biometric capabilities. PHOTO: CHANGI AIRPORT GROUP

PUBLISHED MAR 12, 2017, 5:00 AM SGT

Security of a Biometric System

- The scanner is assumed to be secured: no tampering is possible



- Yes, some biometric data could be spoofed as seen in movies
- See <http://www.wikihow.com/Fake-Fingerprints> on how to make a fake fingerprint
- Some biometric systems include ***liveness detection*** to verify that the entity scanned by the scanner is indeed “live” instead of spoofed materials, say a photograph
(For example: temperature detection in fingerprint scanner)

2.4 *n*-Factor Authentication (2FA)

Reading:

[PF2.1] pg 65-70 (excluding Federated Identity Management)

n-factor Authentication

- Require at least two different authentication “factors”
- Commonly-used factors:
 1. What you know: password, PIN
 2. What you have: smart card, ATM card, mobile phone, security/OTP token
 3. Who you are: biometrics
- Other possible factors [Gollmann]:
 1. where you are
 2. what you do
- It is called an ***2-factor authentication*** if 2 factors are employed
- MAS (Monetary Authority of Singapore) expects all banks in Singapore to provide 2-factor authentication for e-banking

MAS Compliance Checklist

- MAS compliance checklist for Internet Banking and technology risk management guidelines, item 26:

<http://www.mas.gov.sg/~media/MAS/Regulations%20and%20Financial%20Stability/Regulatory%20and%20Supervisory%20Framework/Risk%20Management/IBTRM%20Checklist.pdf>

| | | Supplementary, for fast access | | | | |
|-----|-------|--|--------------------------|--------------------------|--------------------------|--------------------------|
| 25. | 4.3.5 | Procedures and monitoring tools to track system performance, server processes, traffic volumes, transaction duration and capacity utilisation on a continual basis are put in place to ensure a high level of availability of internet banking services. | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| 26. | 4.4.2 | Two-factor authentication at login for all types of internet banking systems and for authorising transactions is implemented. | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| 27. | 4.4.3 | For high value transactions or for changes to sensitive customer data (e.g., customer office | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

What You Have: OTP Token

One-Time Password (OTP) token:

- A hardware that generates one time password (i.e. password that can be used only once)
- Each token and the server share *some secret*
- There are two types:
 1. **Time-based:** Based on the shared secret *and current time interval*, a password K is generated.
Now, both server and the user has a common password K .
(See “TOTP: Time-Based One-Time Password Algorithm”, RFC 6238)
 2. **Sequence-based:** An event (for e.g. user pressing the button) triggers the change of the password

*: Not to be confused with “One-Time Pad”

Example of 2FA (1): Password + OTP Token

Registration:

- The server issues a OTP token to the user, which contains a “secret key” that the server knows
- User sets a password

Authentication:

- (1) User “presses” the token, which then computes and displays a one-time-password (OTP)
- (2) User sends username, password and OTP to server
- (3) Since the server has the “secret key”, the server can also compute the OTP
- (4) Server verifies that both OTP and password are correct

New Trend of OTP Token: Mobile App as a Soft Token!

4/11/2017

DBS rolls out 'soft' tokens to replace all hardware tokens by June 2018, Singapore News & Top Stories - The Straits Times

THE STRAITS TIMES

DBS rolls out 'soft' tokens to replace all hardware tokens by June 2018



DBS digibank

Coming soon: The new digital token



An e-mail sent by DBS to customers this week informing them of the new digital token. PHOTO: DBS

Another New Trend of OTP Token: Authenticator App

The image shows a split-screen view. On the left, a web browser window displays the Google 2-Step Verification page. The URL in the address bar is <https://accounts.google.com/signin/v2/challenge/totp?service=mail&passive=true&rm=false&continue=https%3A%2F%2Fmail.google.com%2Fmail%2Fm%2F%3Fhl%3Dus%26passive%3D1>. The page itself has a "Welcome" message, the user's email address (redacted), and a "2-Step Verification" section asking for a code from the "Google Authenticator app". A blue input field is provided for entering the code, and a checkbox for "Don't ask again on this computer" is checked. Below these are "More options" and a "NEXT" button. On the right, a smartphone screen shows the Google Authenticator app interface, displaying the verification code "183 331" and the account information "Google [redacted]@gmail.com". The phone's status bar at the top shows signal strength, battery level (46%), and the time (00:17).

Another New Trend of OTP Token: Authenticator App Setting

The screenshot shows a web browser window with the URL <https://myaccount.google.com/signinoptions/two-step-verification>. The page title is "2-Step Verification".

The main content area is titled "Set up alternative second step" and includes the instruction: "Set up at least one backup option so that you can sign in even if your other second steps aren't available."

The page lists several backup options:

- Backup codes**: These printable one-time passcodes allow you to sign in when away from your phone, like when you're traveling. Includes a "SET UP" button.
- Google prompt**: Get a Google prompt on your phone and just tap **Yes** to sign in. Includes an "ADD PHONE" button.
- Authenticator app**: Use the Authenticator app to get free verification codes, even when your phone is offline. Available for Android and iPhone. Includes a "SET UP" button.
- Backup phone**: Add a backup phone so you can still sign in if you lose your phone. Includes an "ADD PHONE" button.
- Security Key**: A Security Key is a small physical device used for signing in. It plugs into your computer's USB port. Includes a "Learn more" link and an "ADD SECURITY KEY" button.

Example of 2FA (2): Password + Mobile Phone (SMS)

Registration:

- User gives the server his mobile phone number and password

Authentication:

- (1) User sends password and username to server
- (2) Server verifies that the password is correct
Server sends a one-time-password (OTP) to the user *via SMS*
- (3) User receives the SMS and enters the OTP
- (4) Server verifies that the OTP is correct

Examples:

Singpass, Internet banking,

SMS OTP Security

- Is SMS OTP secure?
- No!
Read: https://www.schneier.com/blog/archives/2016/08/nist_is_no_long.html
- From NIST's "Digital Authentication Guideline":
"[Out of band verification] using SMS is deprecated, and will no longer be allowed in future releases of this guidance."
- Possible security threats:
 - Interception of cellular networks' channel
 - SMS messages are stored as plaintext by the Short Message Service Center (SMSC)
 - Malware/trojan on smartphones:
"Swearing Trojan" fakes base station in China attacking 2FA online banking: <https://blog.checkpoint.com/2017/03/21/swearing-trojan-continues-rage-even-authors-arrest/>
- Expert opinion: still better than just userid+password

Example of 2FA (3): Smartcard + Fingerprint (Door Access System)

Registration:

- The server issues a smartcard to the user (which contains a secret key K)
- The user enrolls his/her fingerprint

Authentication:

(1) User insert smartcard to the reader.

The reader obtains the user identity, and verifies whether the smartcard is authentic.

If so, continue.

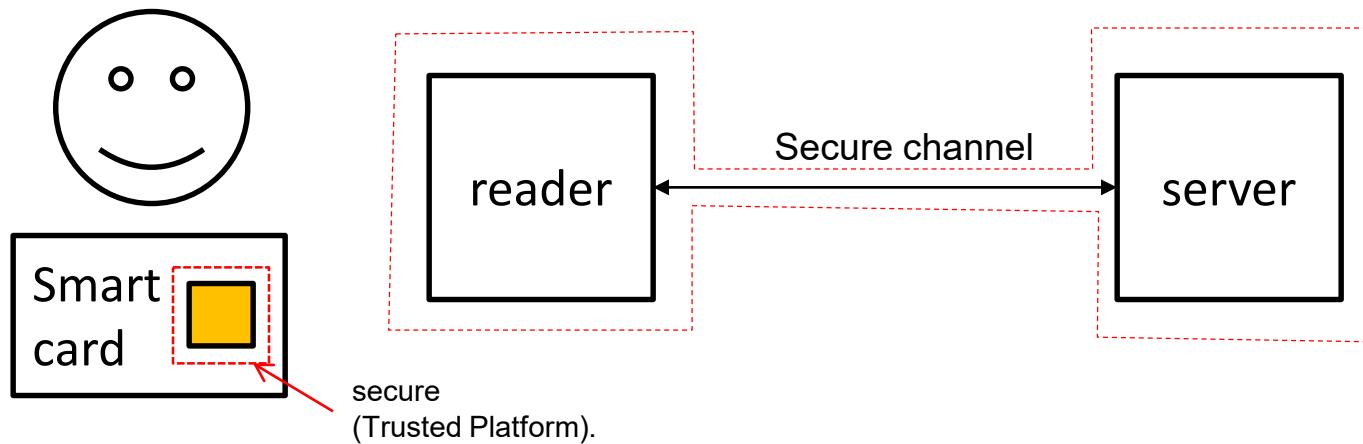
(2) User presents fingerprint to the reader.

The reader performs matching to verify that it is authentic.

If so, open the door.

Security Requirements

- Very often, information on the user identity, the secret K , and the fingerprint template are not stored in the reader
- The reader has a secure communication channel to a *server* that stores these info



- In this case, we also assume that *reader and server are secure*, i.e. attackers are unable to access them

Security Requirements

Some notes:

1. A smart card has this security feature:
Even if an attacker has a physical access to the card,
it is extremely difficult, if not impossible, to extract a secret
stored in the card
2. What are the actual two factors?
3. What is the role of the secret?
4. It is possible to eliminate the need of the server,
e.g. by storing the fingerprint in the card,
and storing a small secret key in the reader.
Question: how to achieve this?

Sample Question (To-be-Discussed in Tutorial)

Tutorial Question:

Comparing the three 2-FA systems, which one is more “secure”?

Hypothetically, we also adopt the first two for door access system.

Are there attacks that one can prevent, but not the another?

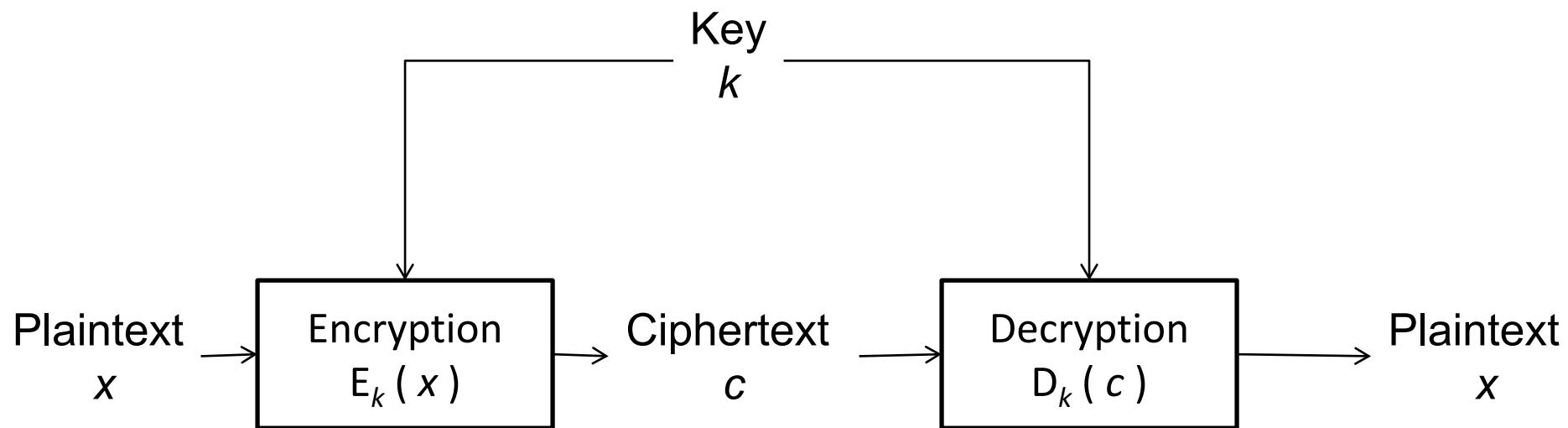
Lecture 3: Authenticity (Data Origin: MAC & Signature)

- 3.1. Crypto background 1: Public Key Cryptography (PKC)
- 3.2 Crypto background 2: Cryptographic hash
- 3.3 Data integrity (Hash)
- 3.4 Data authenticity (MAC, Signature)
- 3.5 Some attacks & pitfalls:
 - 3.5.1 Birthday Attack on hash
 - 3.5.2 Design flaw: Using encryption for authenticity
- 3.6 Application of hash: Password file protection (revisited)

3.1. Crypto Background: Public Key Cryptography (PKC)

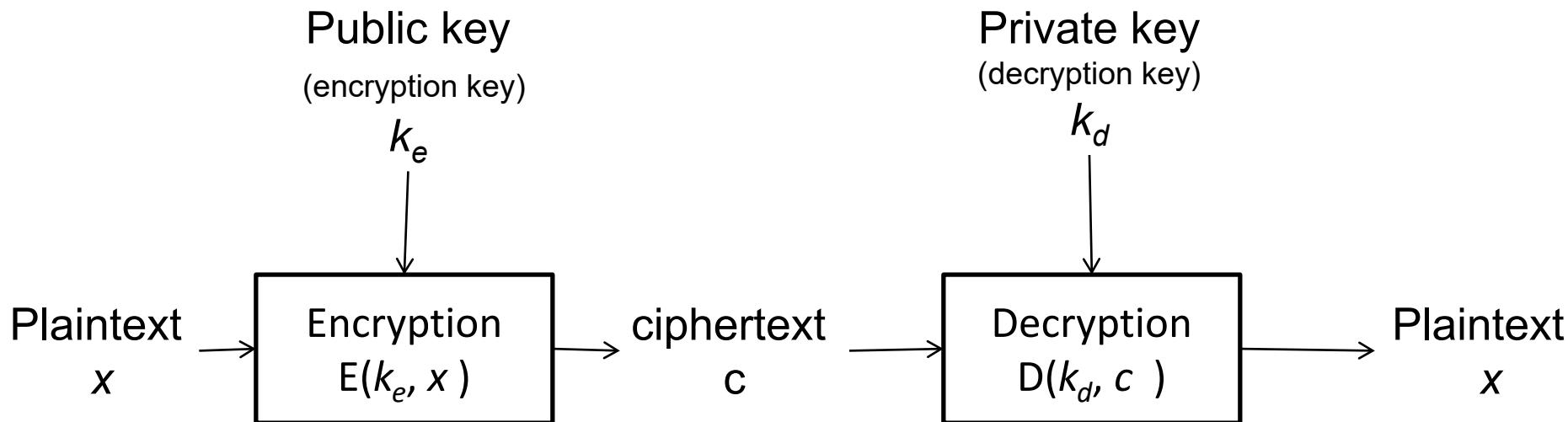
Overview: Symmetric-Key Scheme

A symmetric-key encryption scheme uses the same key for both encryption and decryption



Overview: Public-Key Scheme

A public-key (asymmetric-key) scheme uses *two different keys* for encryption and decryption



Note:

Very often, the *decryption key* consists of two parts $\langle k_e, k_d \rangle$, where k_e is the encryption (public) key.

When it is clear in the context, we also call k_d the private key, although both k_e and k_d are required during decryption.

Why is It Called “Public Key”?

- In a typical usage, the owner Alice *keeps her private key secret*; but tells everyone of her public key (e.g. by posting it in her Facebook)
- A following usage scenario is possible
- Suppose Bob has a **plaintext x** for Alice. Bob can encrypts it (using **Alice's public key**), and posts the **ciphertext c** on Alice's Facebook.
- Another entity, Eve, also obtains Alice's public key and the posted ciphertext c from Alice's Facebook page. Yet, without Alice's private key, Eve is unable to derive x .
- Alice, with her secret **private key**, can decrypt and obtain the plaintext x

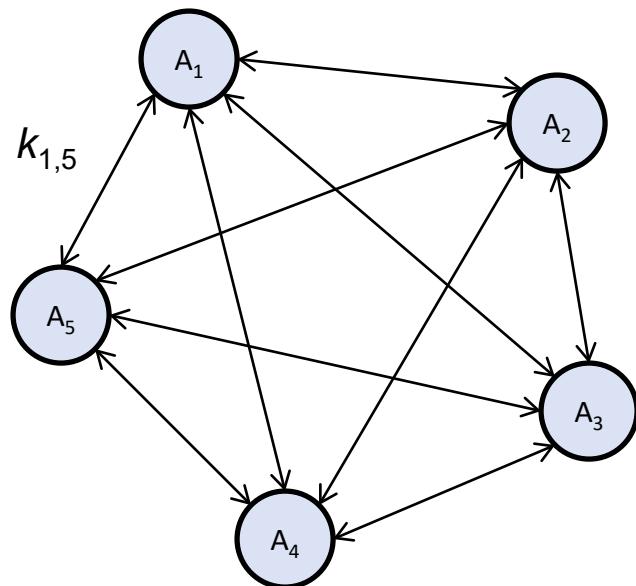
Security Requirements of a Public Key Scheme

- Two *security requirements*:
 1. Given the public key and the ciphertext, but not the private key, it is difficult to determine the plaintext
 2. Without a knowledge of the private key, the ciphertext resembles a sequence of random values
- The requirements imply that it must be difficult to get the private key from the public key

Public-Key Scheme and Key Management Issue

- Suppose we have multiple entities, A_1, A_2, \dots, A_n :
 - Each of them can compute a pair of <private key, public key>
 - Each announces his/her public key, but keeps the private key secret
- Now, suppose A_i wants to encrypt a message m to be read only by A_j :
 - A_i can use A_j 's public key to encrypt m
 - By the property of PKC, only A_j can decrypt it
- If we don't use the PKC, then, any two entities must share a symmetric key:
 - Many keys are required especially if the no of entities is large
 - Establishing all the secret keys is also difficult

Number of Keys Required (with n Entities)

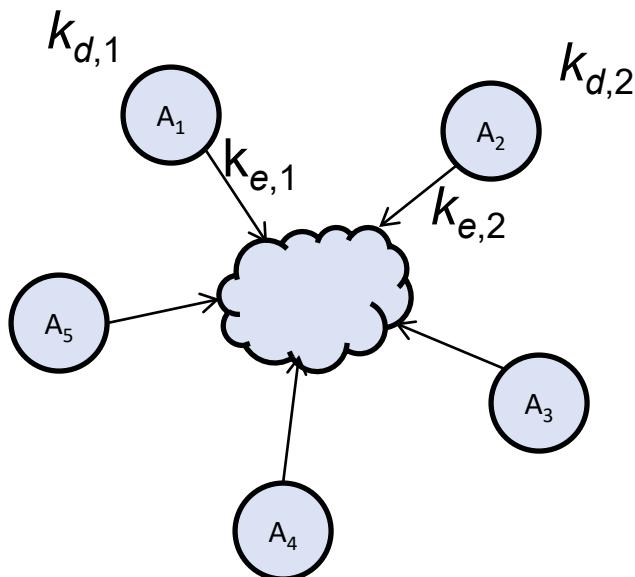


Symmetric-key setting:

Every pair of entities requires one key

Let $k_{i,j}$ be the key shared by A_i and A_j

Total number of keys needed: $n(n-1)/2$



Public-key setting:

Each entity A_i publish its public key $k_{e,i}$ and keep its private key $k_{d,i}$

Total number of public keys: n

Total number of private keys: n

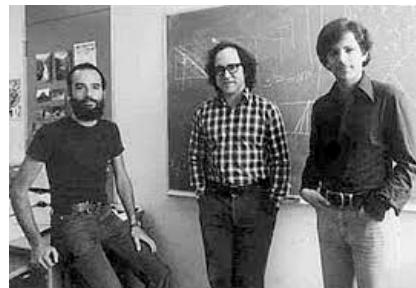
Popular PKC Schemes

- RSA: Key size ~2,048 bits (256 bytes)
- ElGamal: ElGamal can exploit techniques in Elliptic Curve Cryptography (ECC), thus reducing the key size to ~ 300 bits
- Historical notes:



The idea of an asymmetric public-private key cryptosystem is attributed to Whitfield Diffie and Martin Hellman, who published the concept in 1976

Diffie and Hellman, 2015 Turing Award winners



Ron Rivest, Adi Shamir, and Leonard Adleman proposed RSA algorithm in 1977

Rivest, Shamir and Adleman, 2002 Turing Award winners

“Classroom” RSA

Optional

Public key: a composite modulus n , the encryption exponent e :

- The modulus n is the product of two large primes p and q , i.e. $n=pq$
- The values of p and q are not revealed to the public

Private key: d

The relationship among the 3 numbers $n=pq$, e , d :

$$de \bmod (p-1)(q-1) = 1$$

Encryption: Given message m , compute the ciphertext c :

$$c = m^e \bmod n$$

Decryption: Given ciphertext c , compute the plaintext m :

$$m = c^d \bmod n$$

(Note: For RSA mathematical background, see: [PF12.3] or Susanna Epp, “Discrete Mathematics with Applications”, 4th ed, Chapter 8, Section 4)

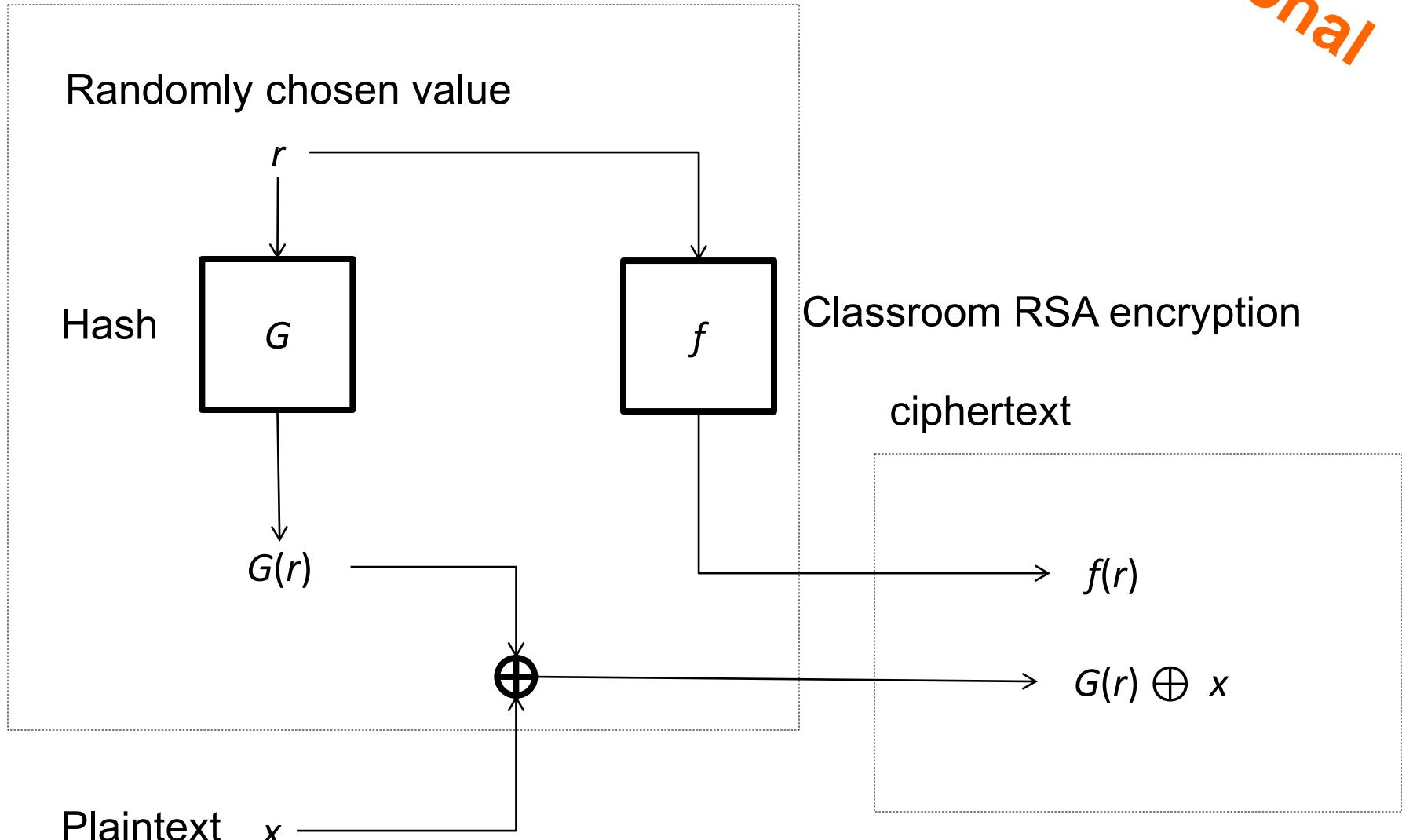
RSA Standards

- Similar to symmetric-key encryption, some form of **IV** is required so that encryption of the *same* plaintext at different times would give different ciphertexts
- Hence additional mechanisms need to be added to the “classroom” RSA
- The standard Public-Key Cryptography Standards (PKCS)#1 (http://en.wikipedia.org/wiki/PKCS_1) adds “optimal padding” to achieve the above

(**Note:** the “classroom RSA” is not “semantic secure”, and leaks some information about the plaintext. PKCS#1 attempts to fix such a problem.)

RSA with Padding (Just to illustrate how complicated it could be)

Optional



Pitfall: Using RSA in the Symmetric-Key Setting

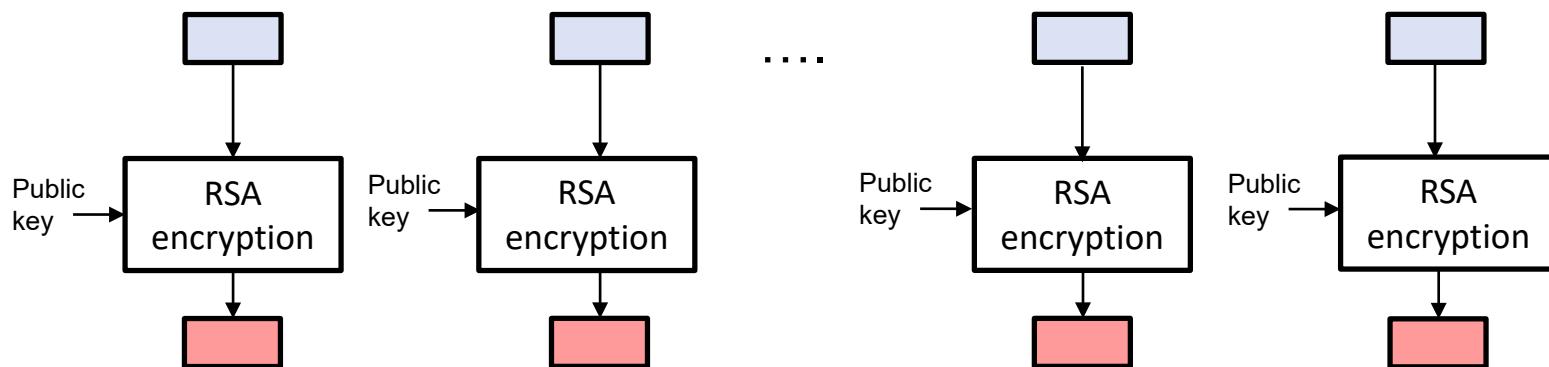
Consider a sample scenario:

“For his Final Year Project, Bob is tasked to write an app that employs an end-to-end encryption to send images from a mobile phone to another mobile phone over WiFi. The sender and recipient use SMS to establish the required symmetric key.”

- Bob feels that RSA is cool 😊
- Instead of employing AES, Bob employs RSA as the encryption scheme: the public/secret key pair is treated as the symmetric key
- Bob claims that RSA is “more secure” than AES: it can be proved to be as difficult as factorization, which is believed to be hard

Pitfall: Using RSA in the Symmetric-Key Setting

- Bob's RSA implementation:
To encrypt a large image (say 1MB),
Bob divides the file into chunks of 128 bytes,
and then apply RSA to each chunk
- Implementation diagram:



Issue 1: Efficiency/Performance

RSA is significantly slower than AES:

- Comparing with the same key strength (e.g. by following NIST recommendation): 128-bit AES \approx 3072-bit RSA
- Crypto++ Benchmarks
(<https://www.cryptopp.com/benchmarks.html>): a C++ library

| Algorithm | MiB/Second | Cycles Per Byte | Microseconds to Setup Key and IV | Cycles to Setup Key and IV |
|-----------------------|------------|-----------------|----------------------------------|----------------------------|
| AES/GCM (2K tables) | 102 | 17.2 | 2.946 | 5391 |
| AES/GCM (64K tables) | 108 | 16.1 | 11.546 | 21130 |
| AES/CCM | 61 | 28.6 | 0.888 | 1625 |
| AES/EAX | 61 | 28.8 | 1.757 | 3216 |
| AES/CTR (128-bit key) | 139 | 12.6 | 0.698 | 1277 |
| AES/CTR (192-bit key) | 113 | 15.4 | 0.707 | 1293 |
| AES/CTR (256-bit key) | 96 | 18.2 | 0.756 | 1383 |

In different order of magnitude!

| Operation | Milliseconds/Operation | Megacycles/Operation |
|---------------------|------------------------|----------------------|
| RSA 1024 Encryption | 0.08 | 0.14 |
| RSA 1024 Decryption | 1.46 | 2.68 |
| RSA 2048 Encryption | 0.16 | 0.29 |
| RSA 2048 Decryption | 6.08 | 11.12 |

Solution

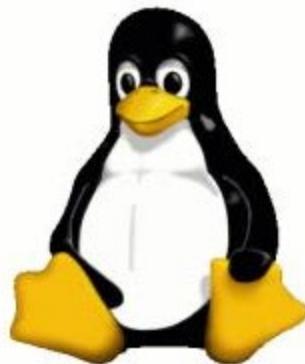
When a large file F is to be encrypted under the public-key setting, for efficiency, the following steps are carried out:

1. Randomly choose an AES key k
2. Encrypt F using AES with k as the key, to produce the ciphertext C
3. Encrypt k using RSA to produce the ciphertext q
4. The final ciphertext consists of two components: (q, C)

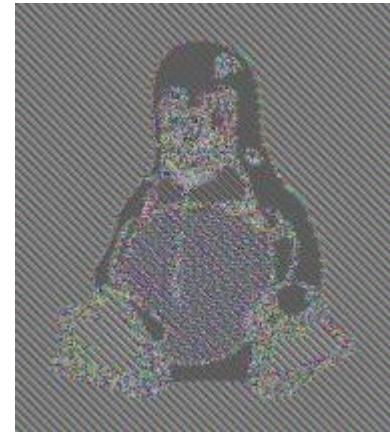
Question: What about decryption steps?

Issue 2: Extension to Multiple Blocks

- Dividing the plaintext into pieces, and independently apply encryption to each of them *could leak* important information!
- Note that this issue applies to symmetric-key encryption as well
- Suppose the image below is divided into blocks, and encrypted with some *deterministic encryption scheme** using the same key
- Since it is deterministic, any two plaintext blocks that are exactly the same (e.g. from the white background) will be encrypted into the same ciphertext



Plaintext



Ciphertext

Issue 2: Extension to Multiple Blocks

Some additional notes: *

- An encryption scheme is “**deterministic**” in a sense that the encryption algorithm always produces **the same output** (i.e. ciphertext) when given the same input (i.e. the key and plaintext)
- An example: AES without the IV
- In contrast, a “**probabilistic**” encryption scheme produces **different ciphertexts** even with the same input is given
- AES is deterministic.
However, if we employ **AES with a randomly-chosen IV**, then it becomes probabilistic (since the IV is different)

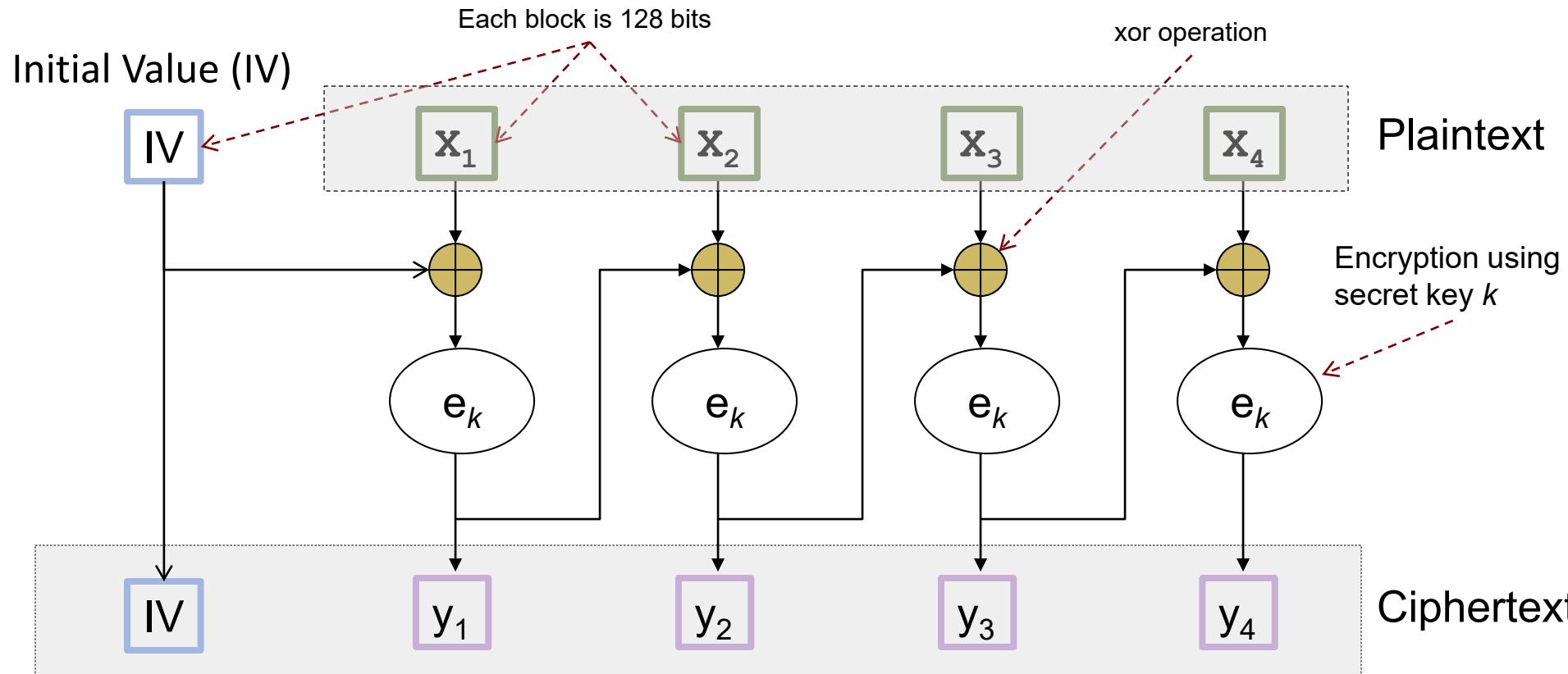
Solution

- To prevent the leakage of information shown in the previous slide, **additional mechanisms** are required
- A ***mode-of-operation*** describes how the blocks are to be “**linked**” so that different blocks at different locations would give different ciphertext, even if all the blocks have the same content
- Popular mode-of-operation is *Cipher Block Chaining* (CBC)
- Avoid the *Electronic Codebook (ECB)* mode, where “we can see the penguin”!

Question: Why not just randomly choose an IV **for each block**, and hence achieve a probabilistic encryption so as to prevent the leakage?

(Answer: It will significantly increase the size of the final ciphertext.
See also Slide 64.)

Mode-of-Operation: Cipher Block Chaining (CBC) on AES



Note: In the above figure, we treat **IV as part of the final ciphertext**. The terminology is inconsistent in the literature. Some documents may state that “the final message to be sent are the IV and the ciphertext” (i.e. IV is not called the “ciphertext”). In this module, when it is crucial, we will explicitly state whether IV is excluded (e.g. AES without IV).

Issue 3: Security of RSA

- RSA is not necessarily “more secure” than AES
- It can be shown that, getting ***the private key from the public key*** is as difficult as factorization
- But, it is not known whether the problem of getting ***the plaintext from the ciphertext and public key*** is as difficult as factorization
- As mentioned before, the “classroom” RSA has to be modified so that different encryptions of the same plaintext lead to different ciphertexts (i.e. probabilistic)
- Such modification is not straightforward (e.g. PKCS#1)

Remarks on PKC Strengths

- The main strength of RSA and PKC is the “public-key” setting, which allows an entity in the public to perform an encryption **without** a (pre-established) pair-wise *secret key*
- This “secret-key-less” feature is also very useful for providing **authentication**: more in this lecture later
- In practice, PKC is rarely used to encrypt a large data file (as already discussed earlier)

Question:

Assuming that a PKC encryption scheme on average takes 1M cycles to decrypt 16 bytes.

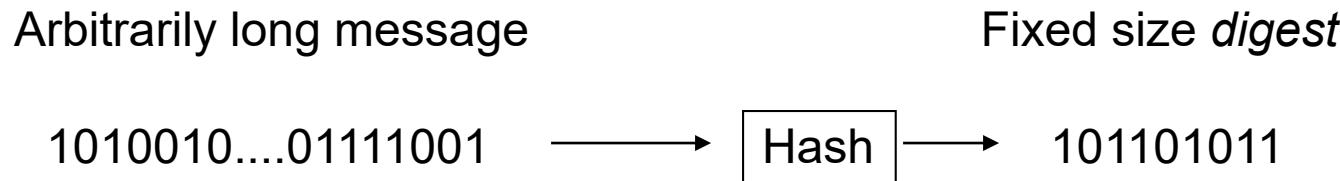
How long does it take to decrypt a 4 GBytes movie on a single-core 4GHz chip?

(For comparison, note that RSA-OAEP take 42M cycles per invocation [Gollmann] pg272.)

3.2. Crypto Background: Cryptographic Hash

Hash (With No Secret Key Involved)

A (*cryptographic*) **hash** is a function that takes an arbitrarily long message as input, and outputs a *fixed-size* (say 160 bits) **digest**

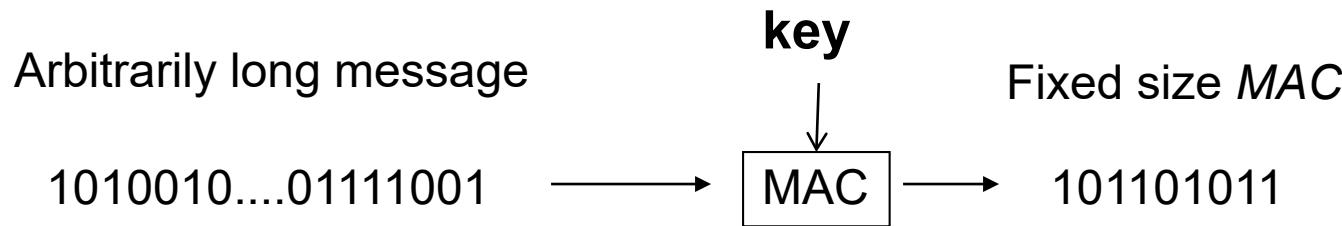


Security requirements:

- “Preimage resistant”: given a digest d , it is difficult to find a m such that $h(m) = d$
- “Second-preimage resistant”: given m_1 , it is difficult to find a second preimage $m_2 \neq m_1$ such that $h(m_1) = h(m_2)$
- “Collision-resistant”: it is difficult to find **two different messages** $m_1 \neq m_2$ that “hash” into the same digest, i.e. $h(m_1) = h(m_2)$

MAC (aka Keyed-Hash, with a Secret Key *Involved*)

A **keyed-hash** is a function that takes an arbitrarily long message and a **secret key** as input, and outputs a fixed-size (say 160 bits) **MAC (Message Authentication Code)**



Security requirement:

- Without knowing the key, it is difficult to forge the MAC

Popular Hash

- SHA-0, SHA-1, SHA-2, SHA-3

Some historical notes:

- SHA-0 was published by NIST in **1993**.
It produces a 160-bits digest.
It was withdrawn shortly after publication and superseded by the revised version SHA-1 in **1995**.
- In **1998**, an attack that finds collision of SHA-0 in 2^{61} operations was discovered.
(Simply using the straight forward birthday attack, a collision can be found in $2^{160/2} = 2^{80}$ operations).
In **2004**, a collision was found, using 80,000 CPU hours.
In **2005**, Wang Xiaoyun et al. (Shandong University) gave an attack that can finds collision in 2^{39} operations.

Popular Hash (Historical Notes)

- **SHA-1** is a popular standard. It produces 160-bits message digest. It is employed in SSL, SSH, etc.
- In **2005**, Xiaoyun Wang et al. gave a method of finding collision in SHA-1 using 2^{69} operations, which was later improved to 2^{63} .
In Feb **2017**, researchers from CWI and Google announced the first successful SHA1 collision (<https://shattered.io>) – see the next 2 slides
- In **2001**, NIST published SHA-224, SHA-256, SHA-384, SHA-512, collectively known as **SHA-2**.
The number in the name indicates the digest length.
- No known attack on full SHA-2 but there are known attacks on “partial” SHA-2, for e.g. attack on a 41-round SHA-256 (whereas the full SHA-256 takes 64 rounds).
- In Nov **2007**, NIST called for proposal of **SHA-3**.
In Oct **2012**, NIST announced the winner, Keccak (pronounced “catch-ack”).
(See: NIST announcement <http://www.nist.gov/itl/csd/sha-100212.cfm>)

Recent Successful Collision Attacks on SHA-1 (Feb 2017)



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

SHAttered
(shattered.io)
Feb 23, 2017

Announcing the first SHA1 collision

February 23, 2017

Posted by Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), Pierre Karpman (CWI Amsterdam),
Ange Albertini (Google), Yarik Markov (Google), Alex Petit Bianco (Google), Clement Baisse (Google)

Cryptographic hash functions like SHA-1 are a cryptographer's swiss army knife. You'll find that hashes play a role in browser security, managing code repositories, or even just detecting duplicate files in storage. Hash functions compress large amounts of data into a small message digest. As a cryptographic requirement for wide-spread use, finding two messages that lead to the same digest should be computationally infeasible. Over time however, this requirement can fail due to [attacks on the mathematical underpinnings](#) of hash functions or to increases in computational power.

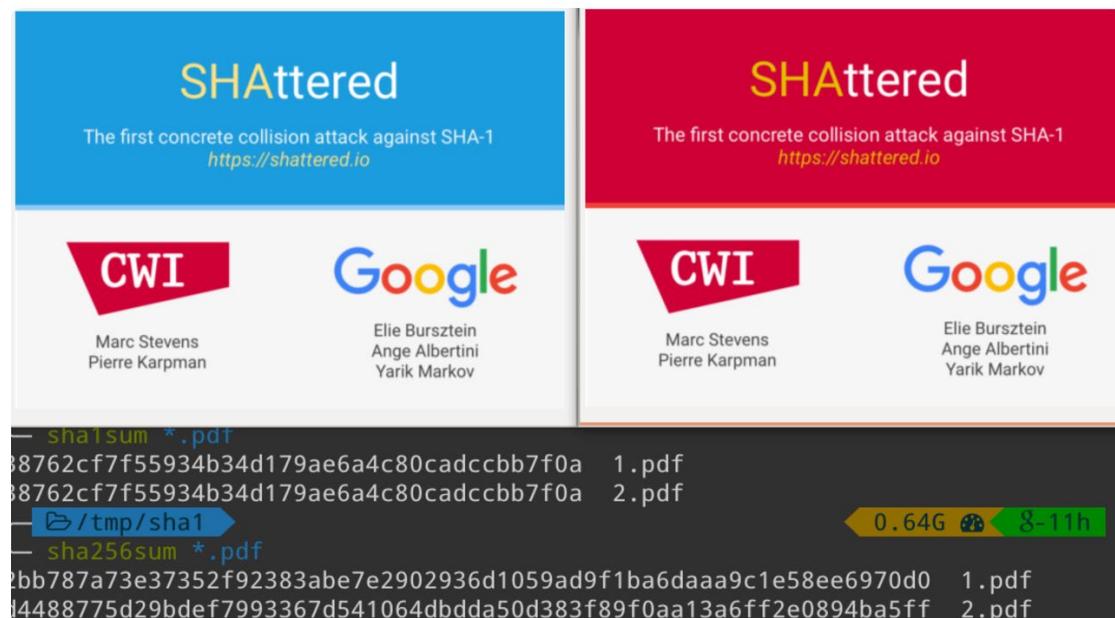
Today, more than 20 years after of SHA-1 was first introduced, we are announcing the first practical technique for generating a collision. This represents the culmination of two years of research that sprung from a collaboration between the [CWI Institute in Amsterdam](#) and Google. We've summarized how we went about generating a collision below. As a proof of the attack, we are [releasing two PDFs](#) that have identical SHA-1

Recent Successful Collision Attacks on SHA-1 (Feb 2017)

- Done by a team from CWI and Google
- Two PDF files with the same hash values as attack proof:
 - <https://shattered.io/static/shattered-1.pdf>
 - <https://shattered.io/static/shattered-2.pdf>
- Defense mechanisms:
 - Use SHA-256 or SHA-3 as replacement
 - Visit shattered.io to test your PDF

Read

<https://shattered.io/>



Other Popular (but Obsolete) Hash

- MD5

Some historical notes:

- Designed by Rivest, who also invented MD, MD2, MD3, MD4, MD6.
- MD6 was submitted to NIST SHA-3 competition, but did not advance to the second round of the competition.
- MD5 was widely used. It produces 128-bit digest.
- In **1996**, Dobbertin announced a collision of the compress function of MD5.
- In **2004**, collision was announced by Xiaoyu Wang et al.
The attack was reported to take one hour.
- In **2006**, Klima give an algorithm that can find collision within one minute on a single notebook!

Security implication: ***Do not*** use MD5!

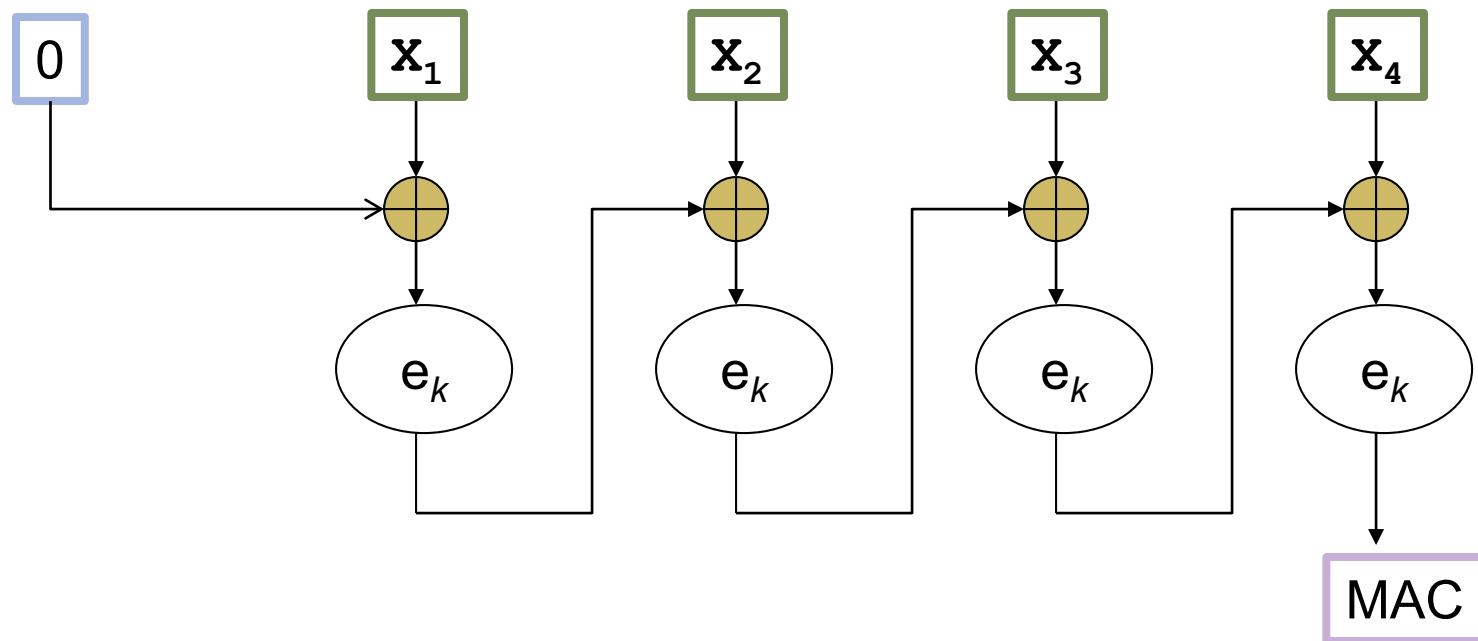
Popular Keyed-Hash (MAC)

- **CBC-MAC:**
 - Based on AES operated under CBC mode
- **HMAC:**
 - Based on any iterative cryptographic hash function (e.g. SHA)
 - Hashed-based MAC
 - Standardized under RFC 2104
(<http://tools.ietf.org/html/rfc2104>)

CBC-MAC

Initial Value (IV)

Optional



HMAC

Optional

$$\text{HMAC}_k(x) = \text{SHA-1}((K \oplus opad) \parallel \text{SHA-1}((K \oplus ipad) \parallel x))$$

where:

$opad = 3636\dots36$ (outer pad)

$ipad = 5c5c\dots5c$ (inner pad)

(Note: the values above are in hexadecimal)

3.3. Data Integrity: Hash (Without Secret Keys)

Authenticity and Integrity Problems

Recall the examples given in last lecture:

- Bob submitted a medical certificate to the lecturer, indicating that he was unfit for exam.
Was the certificate **authentic**: i.e. was it issued by the clinic?
Or had Bob altered the date?
- Alice received an email from the lecturer.
Is the email **authentic**: i.e. sent by the lecturer?

Another **important example**:

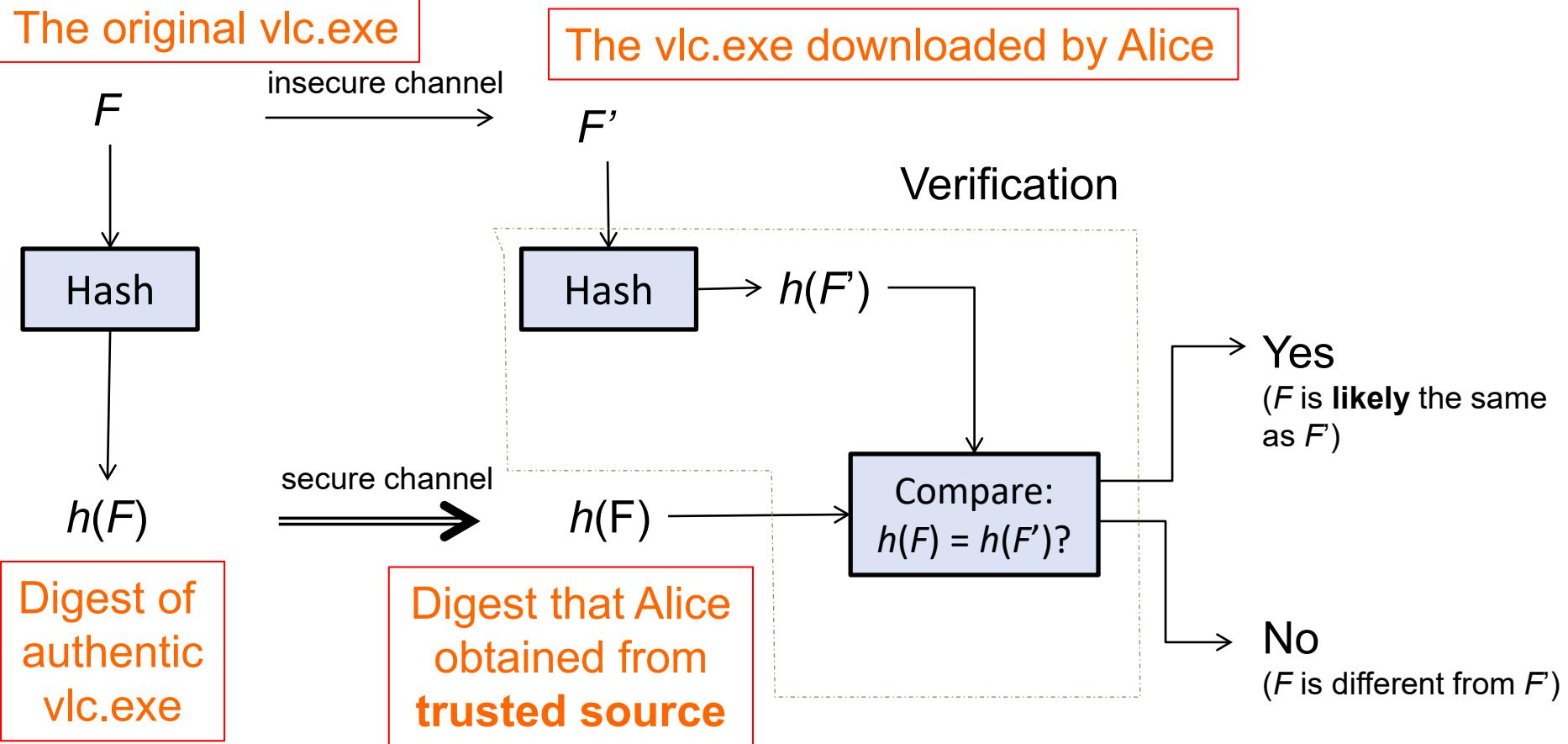
- Alice downloaded a **software** vlc-2.15-win32.exe from the Web.
Is the downloaded file **authentic**?
(See the “checksum” in <http://get.videolan.org/vlc/2.1.5/win32/vlc-2.1.5-win32.exe>)

Using (Unkeyed) Hash for *Integrity* Protection

Assuming that **there is a secure channel** to send a short piece of information, the steps are as follows:

- Let F be the *original* data file
- Alice obtains the digest $h(F)$ from the secure channel
- Alice then obtains a file, say F' , whose origin claims that the file is F
- Alice computes and compares the two digests $h(F)$ and $h(F')$:
 - If $h(F) = h(F')$, then $F = F'$ (with a very high confidence)
 - If $h(F) \neq h(F')$, then $F \neq F'$, i.e. the file integrity is compromised

Using (Unkeyed) Hash for *Integrity* Protection



vlc-2.1.5-win32.exe

Thanks! Your download will start in **0** seconds...

If you have a problem, [click here](#). SHA-1 checksum: bc5e2b879c110c7702973fa3c380550ea2856689



The digest
(checksum)

Sponsored link

DECATHLON



Quechua 2 Seconds Easy 2 Tent

Sponsored link



Gmail for Work

Look more professional with
custom email from Google Apps.

[Start free trial](#)

VLC media player

[VLC](#) a free and open source cross-platform multimedia player and framework that plays most multimedia files a:

Some Remarks

- We may argue that with the digest, the verifier can be assured that the data is “**authentic**”, and thus the “authenticity” of the data origin is achieved
- Nevertheless, in many literatures/documents, when there is no secret key involved, we refer to the problem as an “**integrity**” issue
- The requirement of *a secure channel*, e.g. digest posted on a webpage which is sent over HTTPS
- How can provide (data-origin) authenticity?
Use **MAC** or **digital signature**

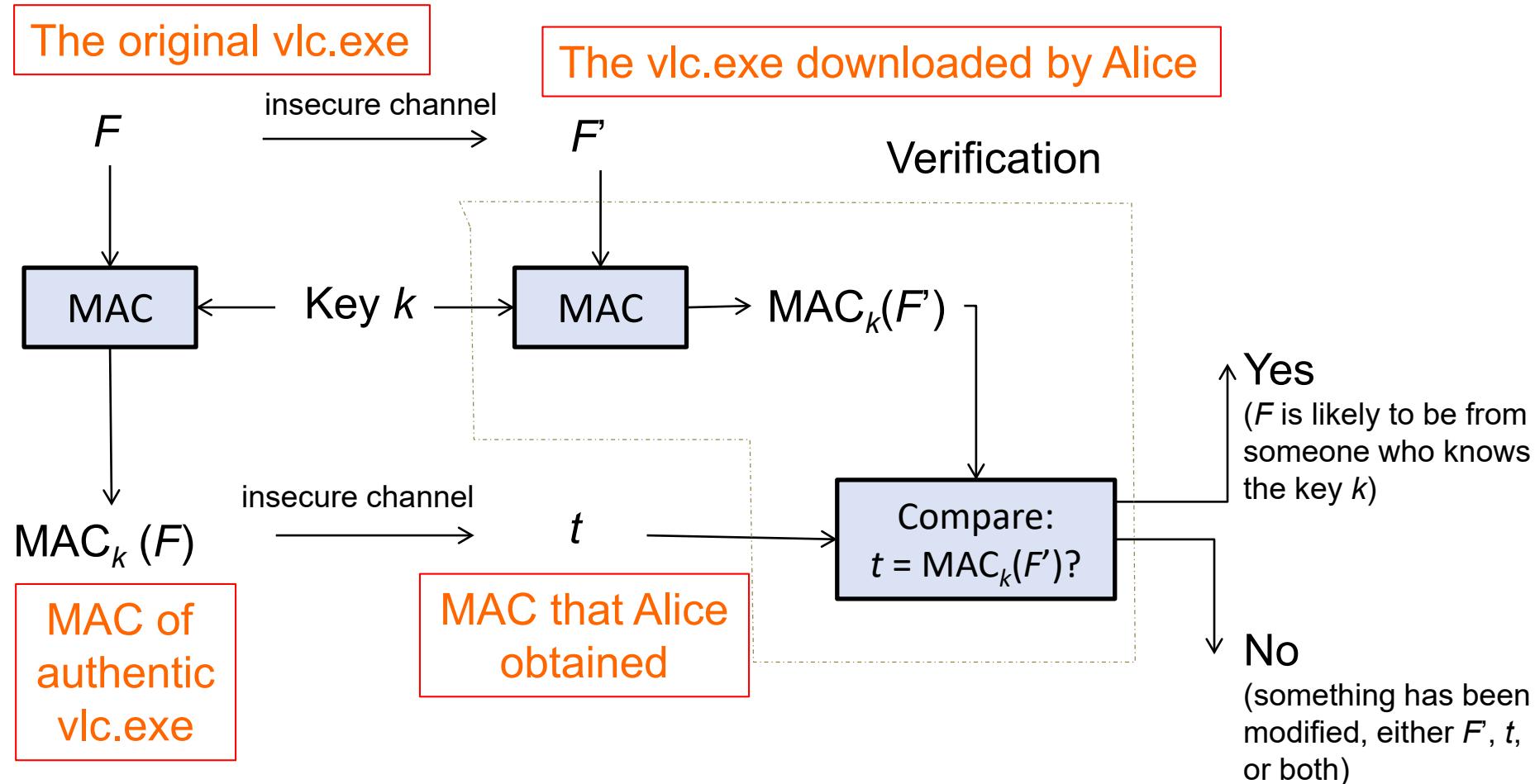
3.4. Data-Origin Authenticity: MAC and Signature

MAC and Signature

- In the previous vlc.exe example, the digest **could be faked**:
 - E.g. the server hosting the web page could be compromised
 - Or in the first place, it was obtained from a “spoofed” webpage
- In other words, we *don't have* a secure channel to deliver the digest
- In such scenarios, we can protect the digest with the help of some secrets:
 - In the symmetric-key setting: **MAC**
 - In the public-key setting: **digital signature**

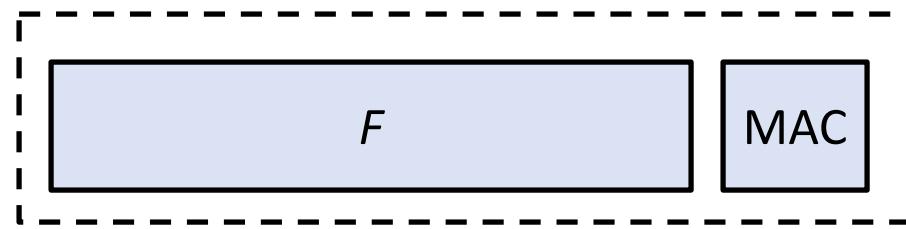
MAC (Message Authentication Code)

- In this setting, the MAC might also be modified by attacker
- If such a case happened, it can be detected with a high probability



Some Remarks

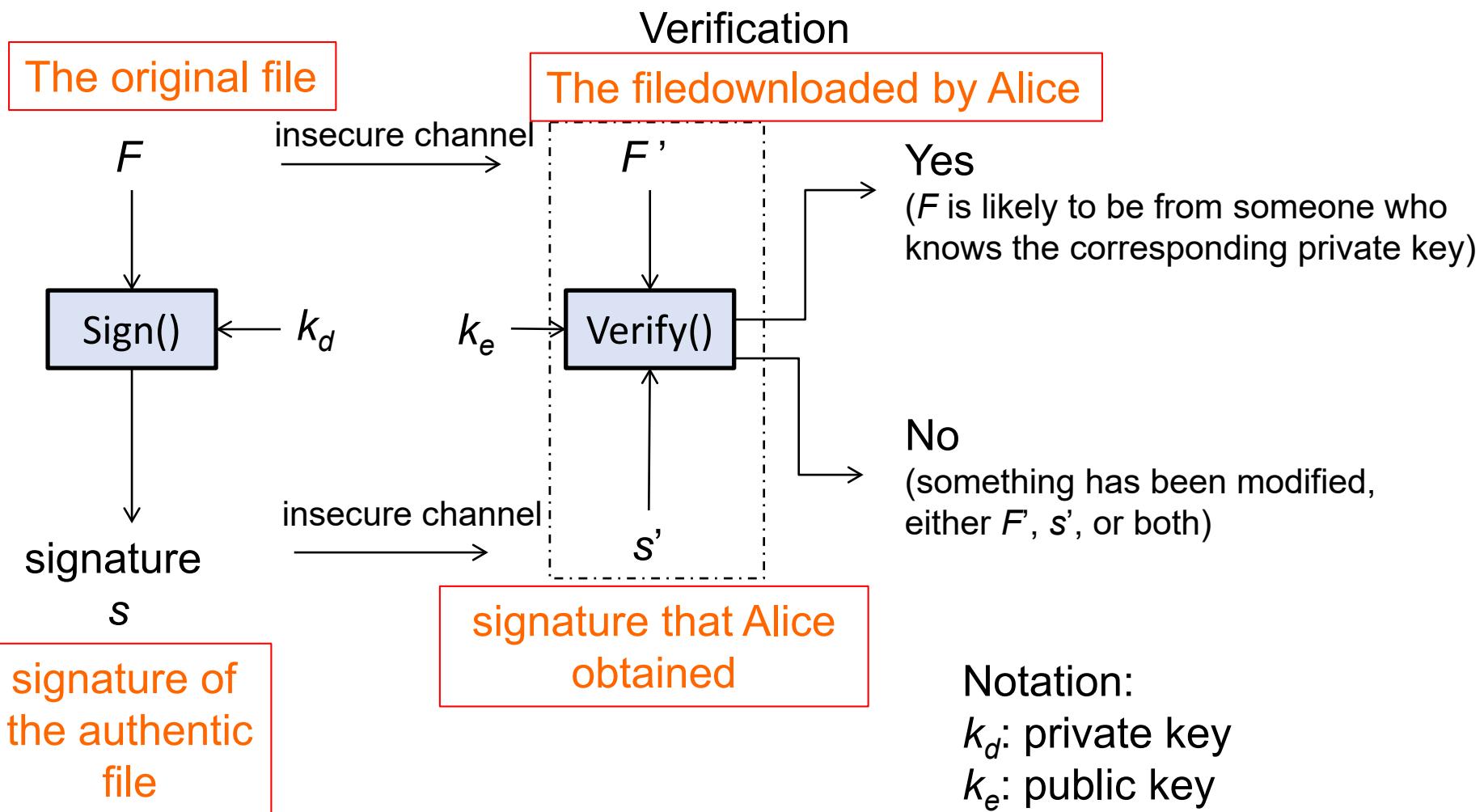
- Note that there is no issue with confidentiality: the **data F** can be sent **in clear**
- Typically, the MAC is **appended** to F , then they are stored as a single file, or transmitted together through a communication channel (Hence, MAC is also called the *authentication tag*)



- Later, an entity who wants to verify the *authenticity* of F , can carry out the verification process using the secret key

Digital Signature

This is essentially the asymmetric-key version of MAC



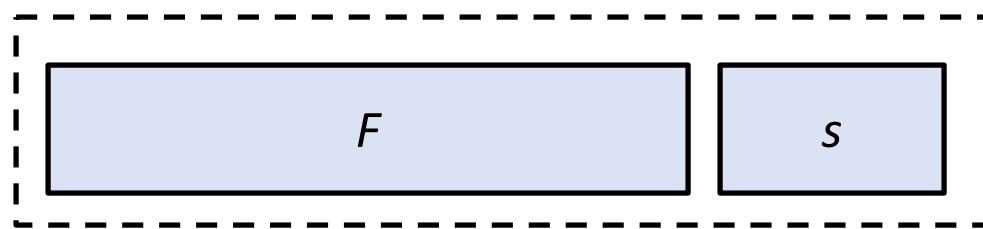
Notation:

k_d : private key

k_e : public key

Some Remarks

- Note that the signature is computed using the private key and F
- Likewise, the computed signature is typically **appended** to F and stored as a single file



- When we say that “**Alice signs the file F** ”, we mean that Alice computes the signature s , and then appends it to F
- Later, the authenticity of F can be verified by **anyone** who knows the signer’s public key
- Why? The valid signature can be computed only by someone knowing the private key:
if the signature is valid, then F must be authentic

What's so Special about Signature (Compared to MAC)?

- We can view the *digital signature* as the counterpart of the *handwritten signature* in a legal document:
 - A legal document is authentic or certified if it has the **correct** handwritten signature
 - **No one**, except the authentic signer, can forge the signature
- In addition to authenticity, signature scheme also achieves ***non-repudiation***:
 - *Assurance that someone cannot deny his/her previous commitments or actions*

Two Scenarios of Non-Repudiation Provisioning

Using only **MAC** (non-repudiation is *not* achieved):

- Suppose Alice sent Bob a message appended with a MAC, which is computed with a secret key **shared by** Alice and Bob
- Later Alice denied that she had sent the message
- When confronted by Bob, Alice claimed that Bob generated the MAC

Using **digital signature** (non-repudiation *is* achieved):

- Suppose Alice sent Bob a message signed using her private key
- Later, she wanted to deny that she had sent the message
- However, she was unable to do so: only the person who knows the **private key** (i.e. Alice) can sign the message
- Hence, the signature is a **proof** that Alice is aware of the sent message

Popular Signature Schemes

- Many signature schemes incorporate both PKC and cryptographic hash in their designs
- The algorithm ***sign()*** usually corresponds to decryption using private key, and the algorithm ***verify()*** corresponds to encryption using public key: the details are omitted here
- However, do refer to the operations as ***sign()*** and ***verify()***
- For examples:
 - RSASSA-PSS, RSASSA-PKCS1:
a signature scheme based on RSA
 - DSA (Digital Signature Algorithm):
a standard by FIPS that is based on ElGamal

3.5. Some Attacks and Pitfalls

3.5.1 Birthday Attack on Hash

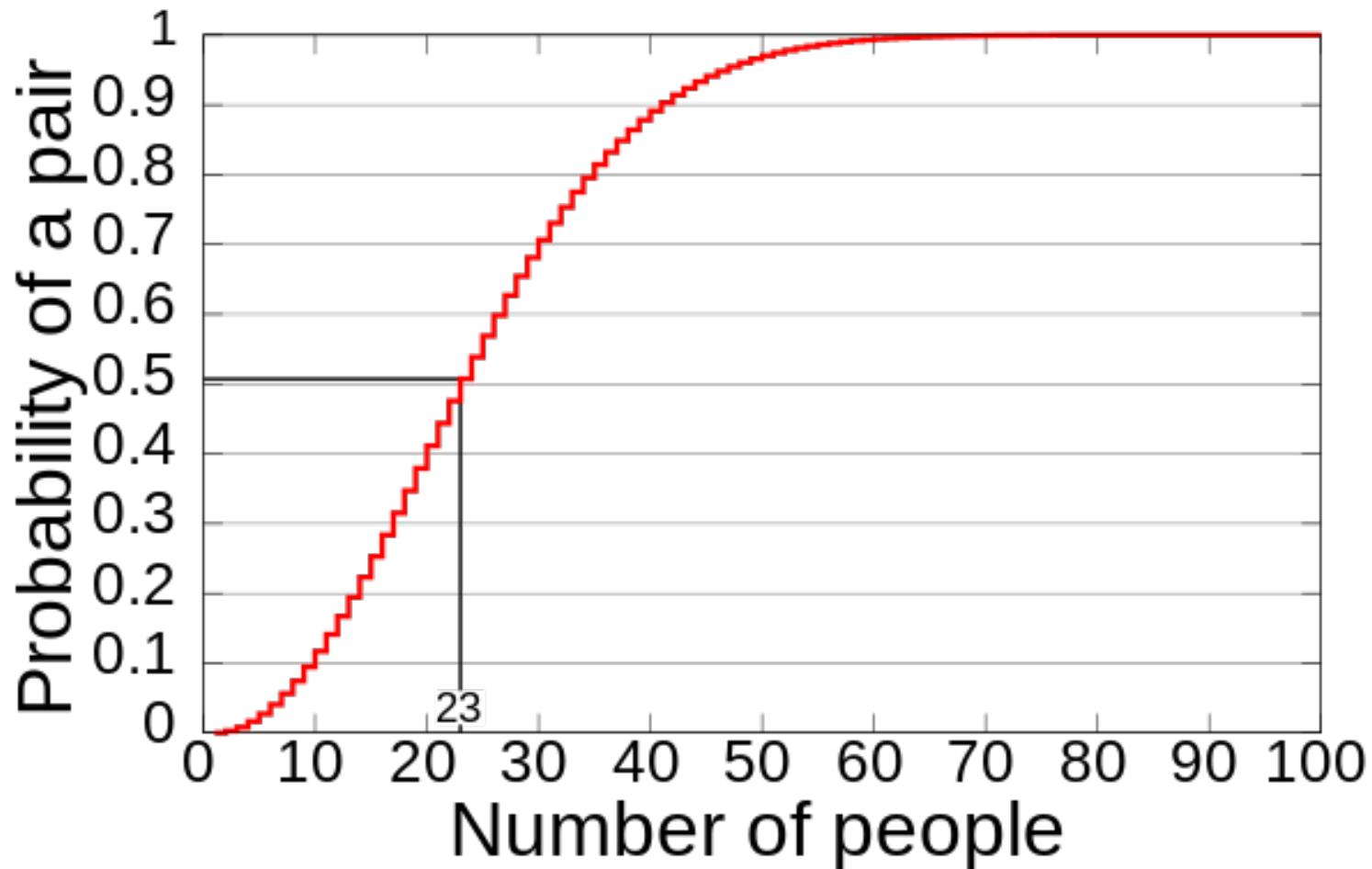
Hash and Birthday Attack

- Hash functions are designed to make a **collision** difficult to find (i.e. to be **collision resistant**)
- Recall that a *collision* involves two different messages m_1 , m_2 that give the same digest, i.e.:
$$h(m_1) = h(m_2) \text{ and } m_1 \neq m_2$$
- Nevertheless, all hash functions are subjected to **birthday attack** (similar to exhaustive search on encryption schemes)

Birthday Paradox/Problem

- “How many students need be randomly selected so that, with probability more than 0.5, there is *a pair of students* having the same birthday”
- Answer: **23**
- The (low) number needed may surprise many people!
- See: https://en.wikipedia.org/wiki/Birthday_problem
- *Why is that possible?*
- There are **366** possible birthdays (including 29 February)
- By **the pigeonhole principle**, the probability reaches 100% when the number of students reaches $366+1=367$
- This is *not* about fixing on one individual and comparing his/her birthday to everyone else's birthday
- But about comparing between *every possible pair* of students = $23 \times 22 / 2 = 253$ comparisons

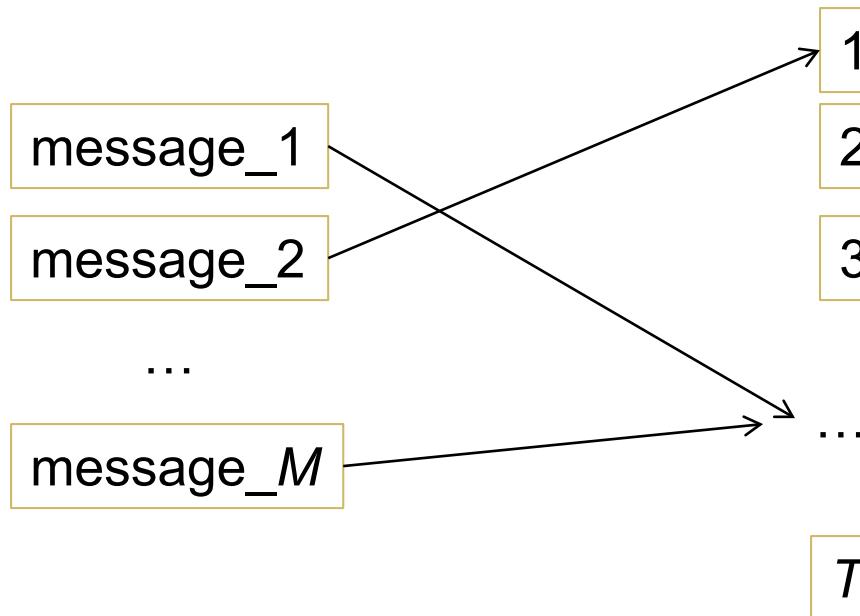
Birthday Paradox/Problem



Ref: https://en.wikipedia.org/wiki/Birthday_problem

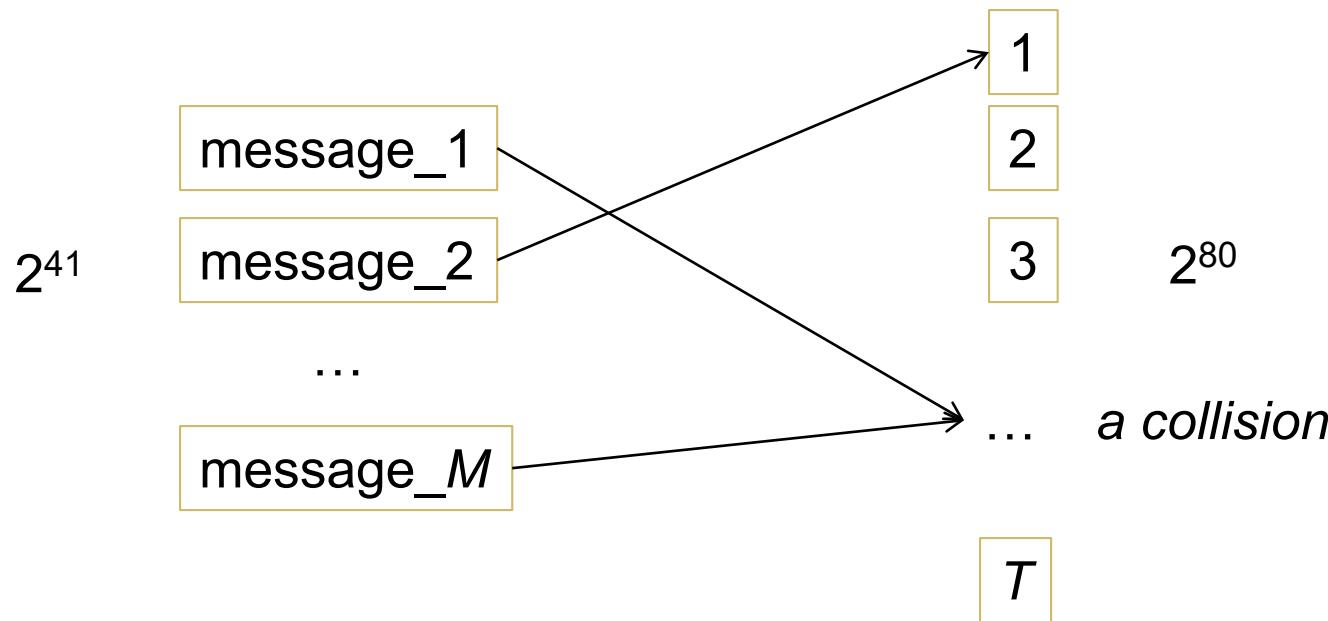
Birthday Paradox/Problem

- Birthday paradox can be applied to hash function
- Suppose we have M messages, and each message is tagged with a value randomly chosen from $\{1, 2, 3, \dots, T\}$
- If $M > 1.17 T^{0.5}$, then with probability more than 0.5, there is *a pair of messages* tagged with the same value
- This result leads to **birthday attack** on a hash function: to reduce the complexity/work-factor of finding a collision



Birthday Attack on Hash

- Suppose the digest of a hash is 80 bits: $T = 2^{80}$
- Now, an attacker wants to find a collision
- If the attacker randomly generates 2^{41} messages ($M = 2^{41} = 2^{1 + 80 \cdot 0.5}$), then $M > 1.17 T^{0.5}$
- Hence, with probability more than 0.5, among the 2^{41} messages, ***two of them give the same digest!***



In general, the **probability that a collision occurs** $\approx 1 - \exp(-M^2/(2T))$

Implication of Birthday Attack for Digest Length

- Birthday attack has a serious consequence on the **digest length** required by a hash function to be collision resistant
- Read the NIST cryptographic key-length recommendation, including on several popular hash functions:
<http://www.keylength.com/en/4/>
- When the key length for a symmetric-key is **112**, the corresponding recommended length for digest is at least **224**
- Why must the digest length be significantly larger?
(Answer: birthday attack, of course)

3.5.2 Design Flaw: Using Encryption for Authenticity

Misuse of Encryption: A Case Study

- Encryption schemes may provide *false sense* of security
- Consider the following case involving **the design of a mobile app** from company XYZ:
- The mobile phone and a server share a secret 256-bit key k .
The server can send **instructions** to the mobile phone via SMS.
*(Note that an SMS consists of only readable ASCII characters. We assume, however, that there is a way to **encode** a binary string using the readable characters).*
- Suppose **the format** of the instruction is:

$X \ P$

where: X is a 8-bits string specifying the type of operation, and P is a 248-bits string specifying the parameter.

- If an operation doesn't take in parameter, P will be ignored

Misuse of Encryption: A Case Study

- There is a total of **16 valid instructions**, such as:
 - 00000000 P : adds P into the contact list
 - 11110000 P : rings for P seconds
 - 10101010 : self-destruct (brick) the phone!
- An instruction is to be **encrypted** using as 256-bit AES, encoded to readable characters, and then sent as an SMS
- After a mobile phone received the SMS, it **decrypts** it. If the instruction is invalid, it ignores the instruction. Otherwise, it executes the instruction.
- The company XYZ claims that: “*256-bits AES provides high level of security, and in fact is classified as Type 1 by NSA. Hence the communication is secure. Even if the attackers have compromised the base station, they are still unable to break the security*”.
- **What's wrong with this?**
And what cryptographic technique should be used instead?

Misuse of Encryption: A Case Study (Concluding Remarks)

- Encryption is designed to provide **confidentiality**
- It does **not**, however, guarantee integrity and authenticity
- In the case above, the XYZ company actually wants to achieve “authenticity” (or “integrity”??)
- But it wrongly employs encryption to achieve that
- A secure design could use **MAC** instead of encryption

Notes:

- There are still some details being omitted in this case
- Simply adding MAC to the instructions is not secure, for example against “replay attacks”
- To prevent replay attacks, a “nonce” is required

3.6 Password File Protection (Revisited)

Hashed Password (From Lecture 2)

- Passwords should be “hashed” and stored in the password files.
(Textbook ([PF]pg 46) uses the term “encrypted”. Note that this is inaccurate. For encryption, there is a way to recover the password from the ciphertext. For cryptographically secure hash, it is infeasible to recover the password from its hashed value.)
- During authentication, the password entered by the entity is hashed, and compared with the the value stored in the password file

Password in clear

| | |
|---------|------------|
| Alice | OpenSesaMe |
| Bob | 123456 |
| Ali | SesameOpen |
| Charles | SesameOpen |

Hashed password

| | |
|---------|-------------|
| Alice | X3lad=3adfV |
| Bob | 3Dv6usgawer |
| Ali | da5DGDSDFd3 |
| Charles | da5DGDSDFd3 |

$$\text{“da5DGDSDFd3”} = \text{Hash(“SesameOpen”)}$$

Hashed Password (From Lecture 2)

- It is desired that the same password will be hashed into *two different values* for two different userid.
Why? (see tutorial)
- This can be achieved by using *salt*

Password in clear

| | |
|---------|------------|
| Alice | OpenSesaMe |
| Bob | 123456 |
| Ali | SesameOpen |
| Charles | SesameOpen |

Salted-hashed password

| | | |
|----------|-------|-------------|
| Alice, | Adf3, | 39Gkaj10Dmf |
| Bob, | a3gh, | d978bjklDFD |
| Ali, | f8ad, | DJk34hoaev7 |
| Charles, | 10vd, | K108ELvio2B |

“DJk34hoaev7”= Hash(“f8adSesameOpen”)
“K108ELvio2B”= Hash(“10vdSesameOpen”)

Interesting Sample Question:

- Back to the example given on Slide 17
(the attack on a penguin-image encryption)
- One remedy is as follow:
“each block is to be encrypted with an IV”
- Although this is secure, it **increases the ciphertext size**,
since the IV for each block has to be included in the ciphertext
- Suppose each IV is 16 bits, there will be an overhead of 16
bits per block
- Hence, with the remedy, it is desirable to keep the size of IV
small

Interesting Sample Question:

- Now, suppose it has been still decided to use 16 bits for the IV, and each image has around $2^{10} = 1,024$ blocks
- Below are two ways of choosing the IVs:
 1. Start from a randomly-chosen number for the first block, and increment it by one for subsequent blocks.
Hence, the IVs for the 1st, 2nd, 3rd, .. block are:
 $r, (r+1) \text{ mod } 2^{10}, (r+2) \text{ mod } 2^{10}, \dots$
where r is a randomly chosen value.
 2. Independently and randomly choose an IV for each block
- Which one would you prefer?
- Would you still prefer AES working in Cipher Block Chaining (CBC) mode?

Lecture 4: PKI + Channel Security

4.1 Public key distribution

4.2 Public Key Infrastructure (PKI)

 4.2.1 Certificate

 4.2.2 CA & Trust Relationship

4.3 Limitations/attacks on PKI

4.4 Strong authentication

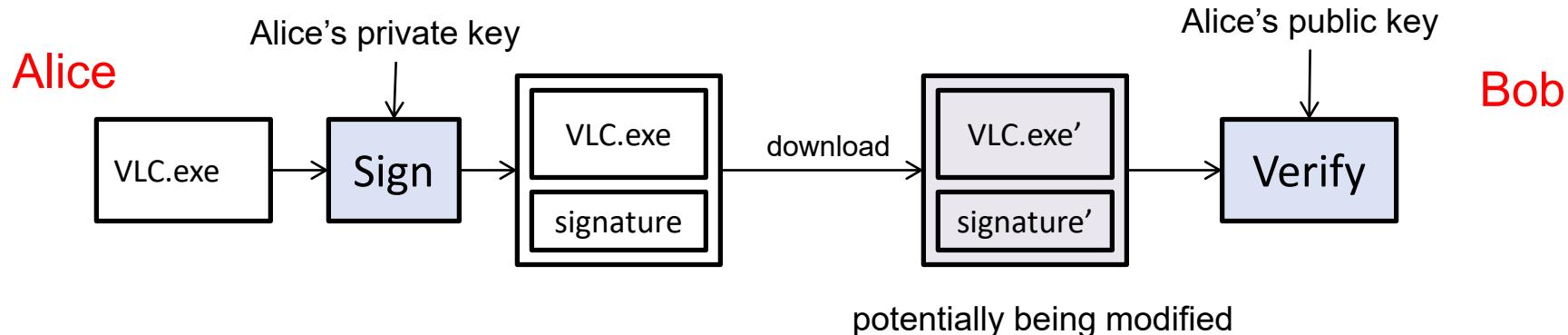
4.5 Putting all together: Securing a communication channel

4.6 Putting all Together: HTTPS

4.1 Public Key Distribution

Motivation 1: Signed Application Example

Recall the previous example of **downloading VLC.exe from a website**



As mentioned before, a method to assure the authenticity of the downloaded program VLC.exe is as follows:

1. The developer, say Alice, signed the VLC.exe file using her *private key*
2. A user, say Bob, who has downloaded the signed file VLC.exe from an unverified source (e.g. CNET download site), can verify the authenticity of the file using *Alice's public key*

Motivation 2: “Signed” Email (using PGP Public Key) Example

- Alice, with email account `alice@comp.nus.edu.sg`, sent an email to Bob.
Alice has a pair of **“PGP” public-private key**.
Alice’s email is signed using her PGP **private key** (see the next slide for the actual email sent).
- After Bob has received the email, with **Alice’s public key**, he can check its authenticity by verifying the signature
- *Any possible issues for PKC to be used/deployed securely?*
 - To carry out the authenticity, Bob **needs to know Alice’s public key** 😞
 - Now, we are now back to square one:
a secure channel is needed for Alice to send her public key to Bob

Example of “Signed” Email using PGP Public Key

Date: Wed, 07 Mar 2007 03:22:08 +0800
From: Alice Ho <alice@comp.nus.edu.sg>
User-Agent: Thunderbird 1.5.0.10 (Windows/20070221)
MIME-Version: 1.0
To: bob@comp.nus.edu.sg
Subject: My first signed email
X-Enigmail-Version: 0.94.2.0
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit

Header (unsigned,
i.e. not included in
computing the signature)

-----BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

Dear Bob,

This is my very first signed email and I want you to keep it =)

Regards,

Alice Ho

-----BEGIN PGP SIGNATURE-----

Version: GnuPG v1.4.3 (MingW32)

Comment: Using GnuPG with Mozilla - <http://enigmail.mozdev.org>

iD8DBQFF7b9XMJcr5kFKO4IRAk+yAKC7JVI1eY+aHEAqqCeVdYGOE10PmwCg9DrE

ArgWymKbDn17m9W1leVeQqM=

=EkxE

-----END PGP SIGNATURE-----

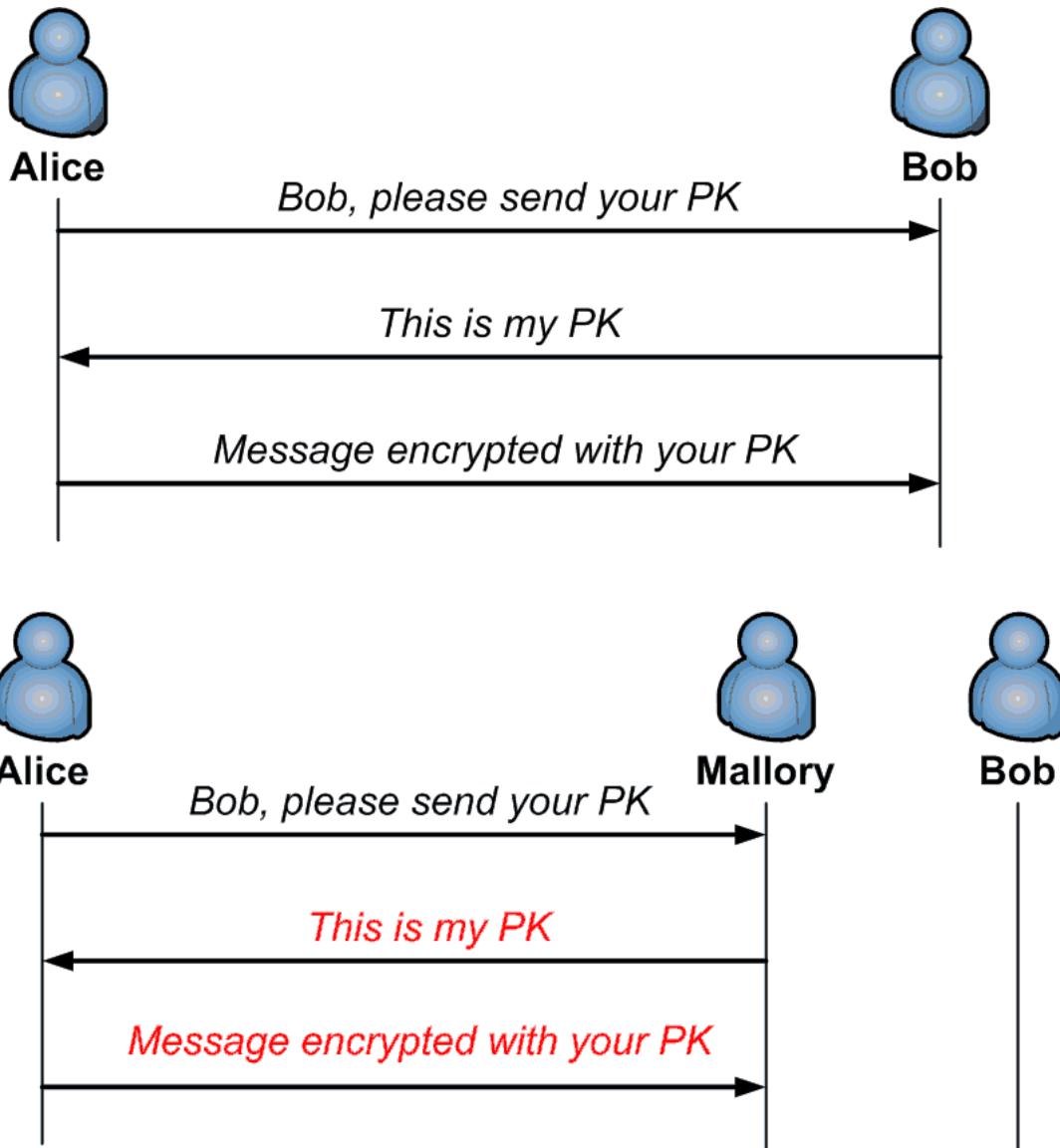
(signed)
Message

Signature

Motivation: If a Public-Key is Distributed *Insecurely*

Intended & assumed communication

Actual (intercepted) communication



Key Distribution Problem and Possible Methods

- The previous two examples illustrate the need for a mechanism to securely distribute/broadcast public keys
- With a public key “securely” distributed, we can use it for encryption (*confidentiality*) and signature verification (*authenticity*)
- **Important questions:**
 - How can we use a secure channel to address this public-key distribution problem?
 - How different is the secure channel requirement between the public-key and secret-key settings?

Secure Channel Requirement in Public-Key Setting

- Compared to the symmetric-key setting (SK encryption, MAC), public-key setting **doesn't** require a secure channel to send **the secret key from Alice to Bob**
- Yet, we still *need* a secure channel for Alice to send her **public key** to Bob!
- Nevertheless, the public-key setting is arguably **easier to handle**:

| Requirement Aspect | Symmetric-Key Setting | Public-Key Setting |
|--|---|---|
| No. of times a secure channel is required | For every pair of entities : $n.(n-1)/2$ | Each entity just needs to securely broadcast its public key : n |
| Item to be transmitted | Shared secret key | (Publicly-published) public key |
| Secure channel timing requirement (e.g. when a new entity needs to securely talk to another entity) | A secure channel is needed to deliver both parties' newly-set secret key | Previously-announced public key(s) just need to be made accessible to a party requiring the key(s) |

Possible Public-Key Distribution Methods

- We would look into 3 different possible methods:
 1. Public announcement
 2. Publicly available directory
 3. Public Key Infrastructure (PKI)

Public-Key Distribution Method: (1) Public Announcement

- The owner **broadcasts** his/her public key
- For example: by sending it to friends via email, or publishing it on a **website**
- Many owners list their “PGP public key” in a blog, personal webpage, etc.

For example: Bruce Schneier’s

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

Contact Bruce Schneier

For feedback on content, please e-mail Bruce Schneier: schneier@schneier.com

For other website issues (browser compatibility problems, etc.), please e-mail: webmaster@schneier.com

Password Safe Support

Password Safe is now an open source project -- please see its [Sourceforge page](#) for feature req and bug reports.

Keys

OTR (IM) Fingerprint

8FB810D4 A2B73FAE 935FF3AE BA5EFFE2 9A98966F

PGP Key

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v2.0.21 (MingW32)

```
mQINBFIpG2IBEACuiDv9Lo8UW0eUh9sUvB1ltnCGMIgJczcdSlHXNoApf0uEmTPwngIpmkeOdXniLeEHv2ea09813IjtIfvo2YfnqFQ2lSn+UUfnCf+nh6jYAnyEOClidr8oXN5Lx91XfRCdU17oGYW6azTIKZqxLQticf0GvCaXYHdBaAqU5E1C20sC6CnVIlqIxrx/kjzvQdhZ1ig8LPu90l7ltsf6BevEI0wSLJFRZXF3mHb9iYNhJnz+gWj/SXBWcgJpPblH0dOo8gyF/R58HMh8NPo9nQg09bWmo/TMPzdX5DERGMaZ92tg34i6bFjGj2oflu22o8Wl0Zn07iXAKJKG6BLcnOT4tpqCWkrM2YBr+eD7BR92qRaJQ3T8fm2ohYHiLjqkvH7/LjpGTilcdwkHmuJr9pD/MJQZR5BsyyWg0a6A35jvViAvAozkz+wF86TCIdPGBj9q+vB++F3MZDL/qREIWeUn1cu01JobJlrb648eyLkxhbeu3z1G1IuzNfc8a1/Wz9rPJZpOehf/woddiDkxnYvgyyyXo/t7/7kaMJg1W6VVVKVgGmWEFHoL93pcKXZdqImCsUTk362v8qrb3RlhG/zgFHBR1jcvAVbeP+Y7HayeO756iWewGiy/9Z5d1s1MV594fhXM9BzwMNfbosZBiviljvOEyTSpm3q0fHx/tQARAQABtCBzY2huZWllciA8c2NobmVpZXJAc2NobmVpZXIUy29tPokCOQQTAQIAIwUCUiikbYg1bAwc1CQgHawIBbhUIAgkKCwQWAgMBAb4BAheAAoJELSOKiztrOpnODKp/3PAsx0r2/6D48GLqTmUBwJiK6z4EmNaMmwElvgzeadc7DknzSgHKWDcDCZPx1lDRvkdaX7kKq+zuSAfzEtK+KZ4jm0ahn5bpdzp+j8YHvym+jXcmv+jSiGdtQmCybT0B1xPvrVnxK7uRr6M+KRxtZ8Qfnkf1nObR01lw47e1cyGdhp5kX0dMb2hr40cfnxC
```

Public-Key Distribution Method: (1) Public Announcement

Limitations:

- Not standardized, and thus there is no systematic way to find/verify the public key when needed
- Eventually, we still need to trust the “entity” distributing the public key:
 - In the previous example on Schneier’s PGP key, the *website* needs to be trusted

Exercise:

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

Public-Key Distribution Method: (2) Publicly-Available Directory

If Bob wants to find the public key associated with the name
alice@yahoo.com.sg

he can search the public directory by querying a server

Public-Key Directory:

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

Directory Server



Public-Key Distribution Method: (2) Publicly-Available Directory

Potential issues:

- Anyone can post his/her public key into the server,
e.g. <https://pgp.mit.edu/>
- It is not easy to have a “secure” public directory:
Suppose the server receives a request to post a public key,
how does the server verify that the information is **correct**?
- Eventually, some entity need to be **trusted**:
in this case, the website <https://pgp.mit.edu/>
- Furthermore, even if a user trusts the website operator,
how does the user know that the “website” **visited**
is indeed <https://pgp.mit.edu/> as claimed?

Public-Key Distribution Method: (3) PKI + Certificate

- PKI is a **standardized** system that distributes public keys
- (Again, when reading a document, note that there are different definitions of “Public Key Infrastructure”)
- PKI’s objectives:
 - To make public-key cryptography **deployable** on a large scale
 - To make public keys verifiable without requiring any two communicating parties to **directly trust** each other
 - To manage public & private key pairs throughout their entire key lifecycle

Public-Key Distribution Method: (3) PKI + Certificate

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

4.2 Public Key Infrastructure

4.2.1 Certificate

Certificate Authority (CA)

- A CA:
 - Issues and signs digital certificates
 - Keeps a directory of public keys (more on this later)
 - It also has its own public-private key pair:
we assume that the CA's public key **has been securely distributed** to all entities involved
- Most OSes and browsers have a few **pre-loaded CAs' public keys**: they are known as the “root” CAs
- Stringent operational requirements for a CA:
 - For example, it must pass **WebTrust audit** (<http://www.webtrust.org/homepage-documents/item76002.pdf>)

Certificate Authority (CA)

Example: Root CAs in Chrome browser:

The screenshot shows the Google Chrome settings page with a red circle highlighting the 'Manage certificates' and 'Manage HTTPS/SSL certificates and sites' options under the 'Content settings' section. A red box highlights the 'Trusted Root Certification Authorities' tab in the 'Certificates' dialog window.

Certificates

Intended purpose: <All>

Trusted Root Certification Authorities Trusted Publishers Untrusted Publishers

| Issued To | Issued By | Expirati... | Friendly ... |
|-------------------------------------|---------------------------------|-------------|--------------|
| AddTrust External CA Root | AddTrust External CA Root | 30-May-20 | The USE... |
| AffirmTrust Commercial | AffirmTrust Commercial | 31-Dec-30 | AffirmTr... |
| Baltimore CyberTrust Root | Baltimore CyberTrust Root | 13-May-25 | DigiCert... |
| Certum CA | Certum CA | 11-Jun-27 | Certum... |
| Certum Trusted Network CA | Certum Trusted Network CA | 31-Dec-29 | Certum... |
| Class 3 Public Primary Certifica... | Class 3 Public Primary Certi... | 02-Aug-28 | VeriSign... |
| COMODO RSA Certification Au... | COMODO RSA Certification ... | 19-Jan-38 | COMOD... |
| Copyright (c) 1997 Microsoft C... | Copyright (c) 1997 Microso... | 31-Dec-99 | Microsof... |
| Deutsche Telekom Root CA 2 | Deutsche Telekom Root CA 2 | 10-Jul-19 | Deutsch... |
| DIGI TA... | DIGI TA... | 10-Nov-21 | DIGI C... |

Import... Export... Remove Advanced

Certificate intended purposes

Server Authentication, Client Authentication, Secure Email, Code Signing, Time Stamping, Encrypting File System, IP security tunnel termination, IP security user

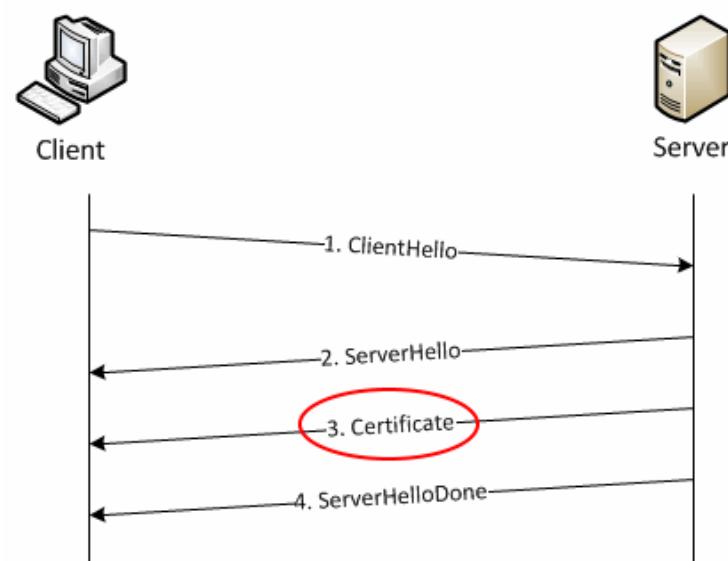
View Close

Certificate: Content and Usage

- A ***certificate*** is a digital document that contains at least the following main items:
 - The identity of an owner, for e.g. alice@yahoo.com
 - The public key of the owner
 - The time window that this certificate is valid
 - The signature of the CA

(It also has additional information like the intended purpose of the certificate: e.g. client authentication, secure email, ...)

- A certificate is widely used by Internet applications: SSL/TLS, S/MIME, SSH, ...
- Sample usage in the **SSL/TLS handshake protocol**:



Role of a Certificate

- **Important question:** *Can a certificate-based PKI work without a publicly-available directory server?*
- Recall the method of using publicly-available directory
- Now, treat CA as the **directory server**
- Note that there are **two issues** of retrieving the public key from the directory server whenever needed:
 - The CA becomes a bottleneck
 - Bob, the verifier, needs to have online access to the CA at the verification point
- Using certificates is a “smart” way of avoiding the above limitations!

Without Certificate

We assume that Bob has the public key of CA. Hence the authenticity of the messages exchanged between them (i.e. Steps 2,3) can be verified.

alice@yahoo.com.sg

alice@yahoo.com

alice@cs.nyu.edu

apple@google.com

.....

x1s34adf39

asd3123411

2s3dasdf233

a323fasdfas

Directory Server (CA)



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

signature: xsdewsdesd



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

CA’s public key

(Step 2) Bob: What is the public key of alice@yahoo.com.sg?

(Step 3) CA: The public key of alice@yahoo.com.sg is x1s34adf39 and it is valid until 1 Sep 2020

With a Certificate

An “offline” CA would sign the message **beforehand** and pass it to **Alice**. Such a signed message is the certificate.

alice@yahoo.com.sg

alice@yahoo.com

alice@cs.nyu.edu

apple@google.com

.....

x1s34adf39

asd3123411

2s3dasdf233

a323fasdfas

Directory Server (CA)

(Step 2) Bob **verifies** that the signature in the **certificate** is indeed signed by the CA.

Since no one except the CA can produce the valid signature, the **authenticity of the information in the certificate** is as good as **coming directly from the CA**.

Alice

Bob

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

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

signature: xsdewsdesd

Name : alice@yahoo.com.sg
Public key: x1s34adf39
Valid until: 1 Sep 2020
Signature of the CA

CA’s public key

Role of a Certificate: No Required Directory Server

- A CA (as certificate issuer) basically binds an entity with his/her public key
- Now, with the certificate, Bob can obtain Alice's public key, and verifies its authenticity, ***even without a connection*** to the CA
- Notice, however, there is still a need to check *that the certificate hasn't been revoked*:
 - Online CRL Distribution Point or OCSP Responder (to be discussed later)

X.509 Digital Certificate Standard

- Standardization bodies:
 - ITU-T X.509:

Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
 - The Public-Key Infrastructure (X.509) Working Group (PKIX):

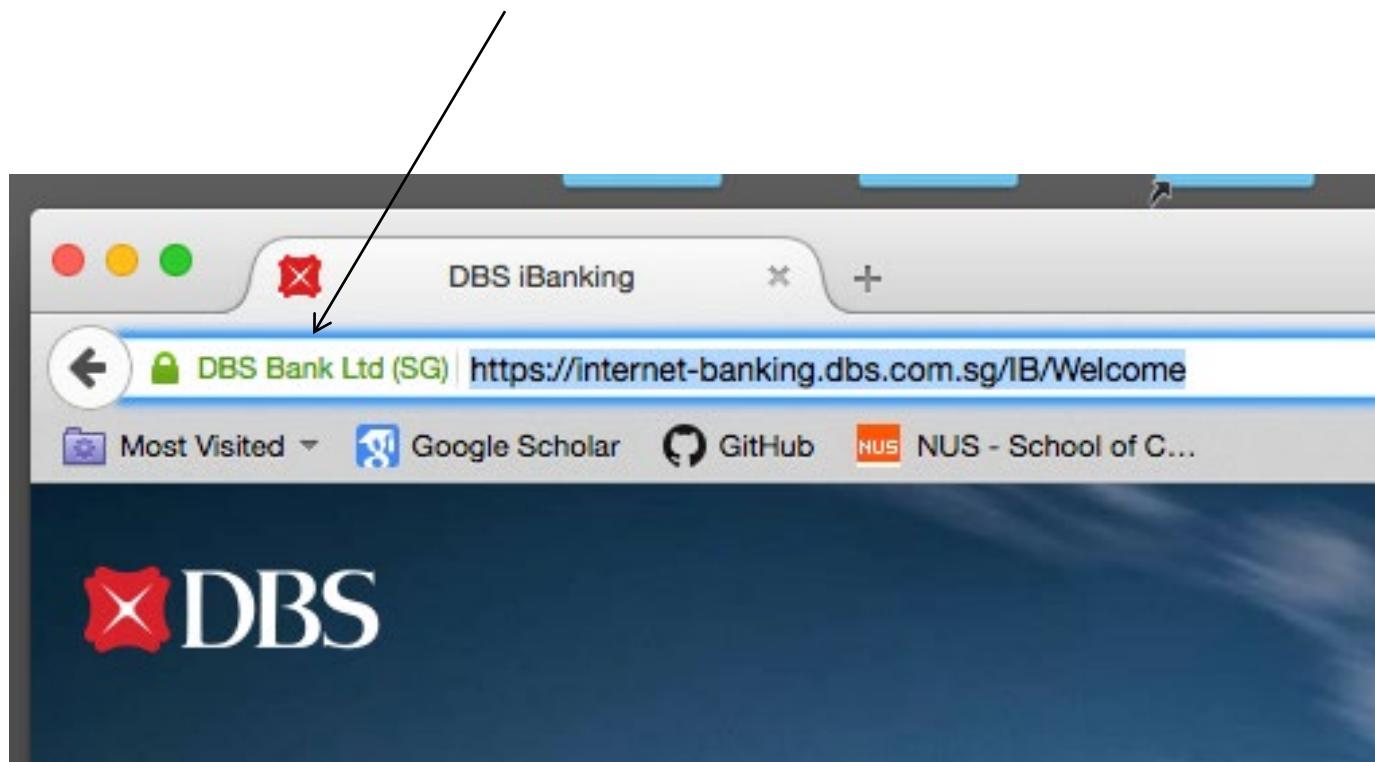
IETF working group that creates Internet standards on issues related PKI based on X.509 certificates
- Structure of an X.509 v3 digital certificate:
 - Certificate:
 - Version Number
 - Serial Number
 - Signature Algorithm ID (*Note Signature Algorithm below too*)
 - **Issuer Name**
 - Validity period: Not Before, Not After

X.509 Digital Certificate Standard

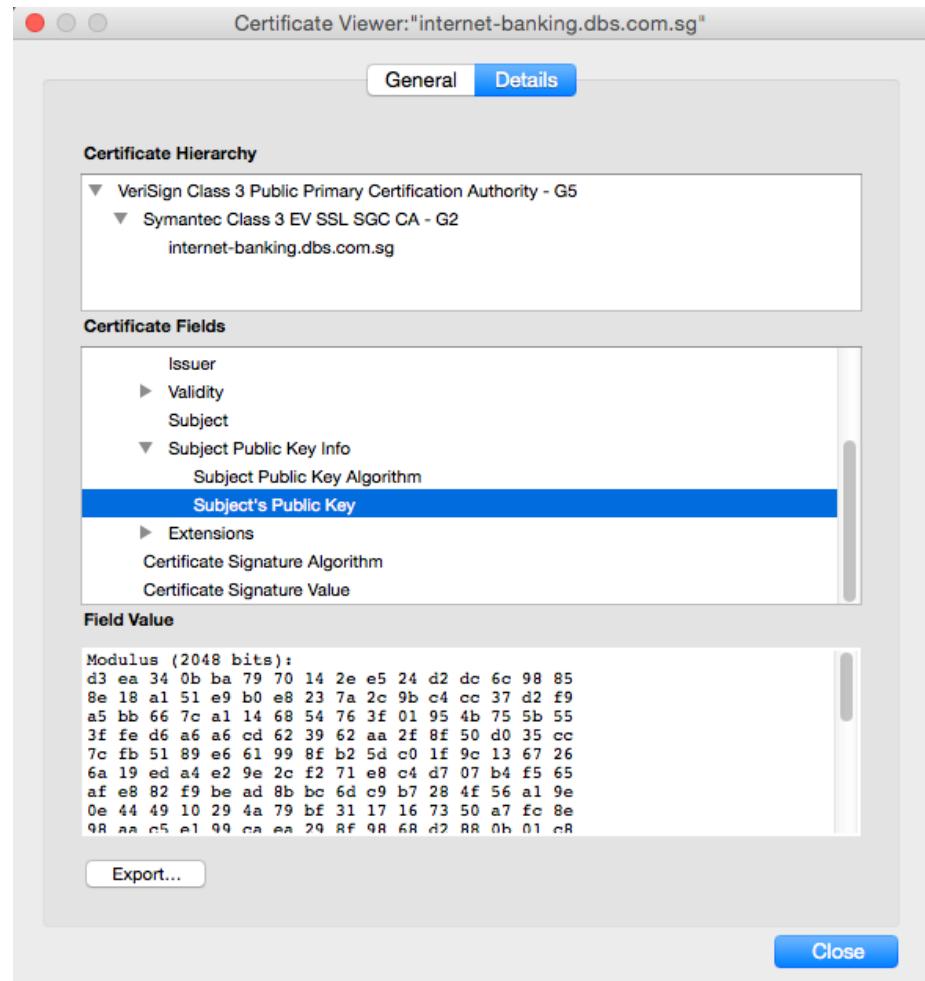
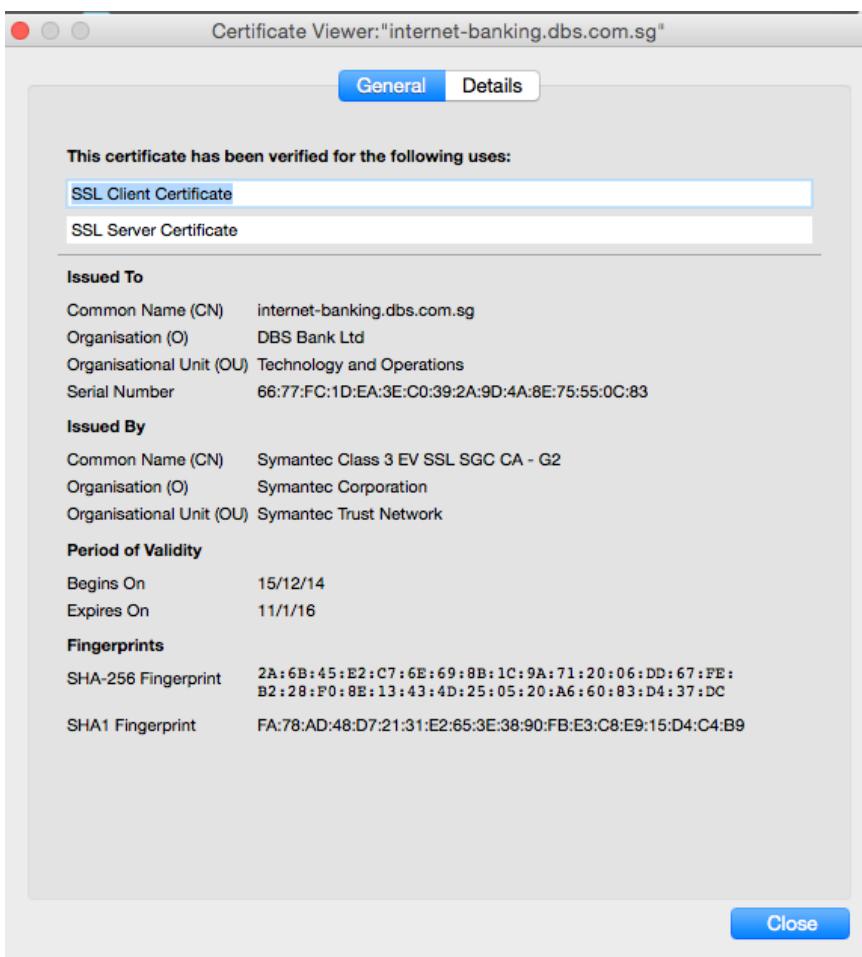
- Subject Name
- Subject Public Key Info: Public Key Algorithm, Subject Public Key
- Issuer Unique Identifier (optional)
- Subject Unique Identifier (optional)
- Extensions (optional)
- Certificate Signature Algorithm
- Certificate Signature
- **Distinguished Name (DN)** to identify an entity
(e.g. issuer and subject names):
 - Common attribute types:
Country (C), State (S), Locale (L), Organization name (O),
Organizational unit name (OU), **Common name (CN)**
 - **Common name** can be an individual user or any other entity,
e.g. a web server

Example of Certificate

- Visit <https://internet-banking.dbs.com.sg/IB/Welcome>
- Check its certificate's detail
(For Firefox, click the address bar)



Example of Certificate



How Do I Get a Certificate?

- Get a Root CA to issue you one:
 - Paid ones: \$10 - \$50 / year (not costly)
- “*Let's Encrypt*” provides (basic) TLS certs at **no charge**:
 - Launched in April 2016
 - A certificate is valid for 90 days
 - Its renewal can take place at anytime
 - Automated process of cert creation, validation, signing, installation, and renewal
 - No of issued certs: **1M** (March 2016) to **380M** (Sept 2018)
- Firefox Telemetry:
 - **77%** of all page loads via Firefox are now encrypted
 - It is predicted that it will reach **90%** by the end of 2019

Certificate: Summary

- A certificate is simply a document signed by a CA that specifies:
 1. An identity
 2. The associated public key
 3. The time window that this certificate is valid
 4. The signature of the CA
- The certificate “certifies” that the public key indeed belongs to the stated identity
- We assume that Bob already has the CA’s public key installed in his machine

4.2.2 Certificate Authority & Trust Relationship

Responsibility of a CA

- The CA, besides issuing certificate, is also responsible to **verify** that the information is correct
- For instance, if someone wants request for a certificate for www.nus.edu.sg, the CA should check that the applicant indeed **owns** the above **domain name**
- This may involve some manual checking and thus it could be costly, especially for the **Extended Validation SSL** (EV SSL) certificates:
 - Initiative by CA/Browser Forum
 - The highest “class” of SSL certificates with more stringent checks done
 - Activate both the padlock and the green address bar in major browsers!

What are Checked by a CA before a Certificate Issuance?

- **Domain Validation (DV)** SSL certificate:
 - Issued if the purchaser can demonstrate the right to administratively **manage a domain name**,
(e.g. response to email sent to the email contact in whois details, publishing a DNS TXT record)
- **Organization Validation (OV)** SSL certificate:
 - Issued if the purchaser additionally has an organization's actual **existence as a legal entity**
- **Extended Validation (EV)** SSL certificate:
 - Issued if the purchaser can persuade the cert provider of its legal identity, including **manual verification checks** by a human

Types of SSL Certificates

- Read: <https://www.ssl.com/article/dv-ov-and-ev-certificates/>
- Summarized below:

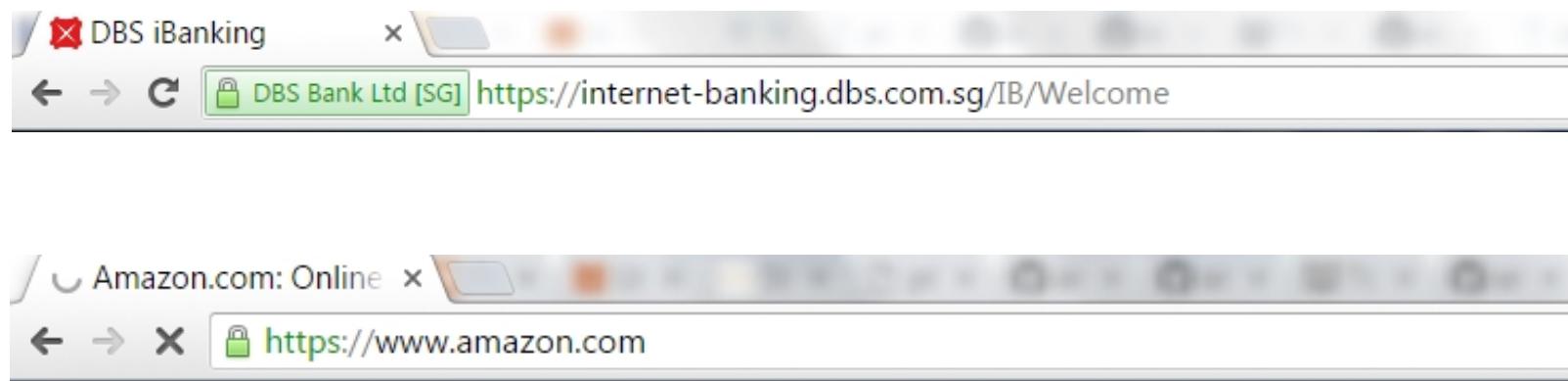
TLS Certificate Level Summaries

| Certificate type | HTTPS encrypted? | Padlock displayed? | Domain validated? | Address validated? | Identity validation | Green address bar? |
|------------------|------------------|--------------------|-------------------|--------------------|---------------------|--------------------|
| DV | Yes | Yes | Yes | No | None | No |
| OV | Yes | Yes | Yes | Yes | Good | No |
| EV | Yes | Yes | Yes | Yes | Strong | Yes |

Source: PCI Security Standards Council, "Best Practices for Securing E-commerce",
https://www.pcisecuritystandards.org/pdfs/best_practices_securing_ecommerce.pdf

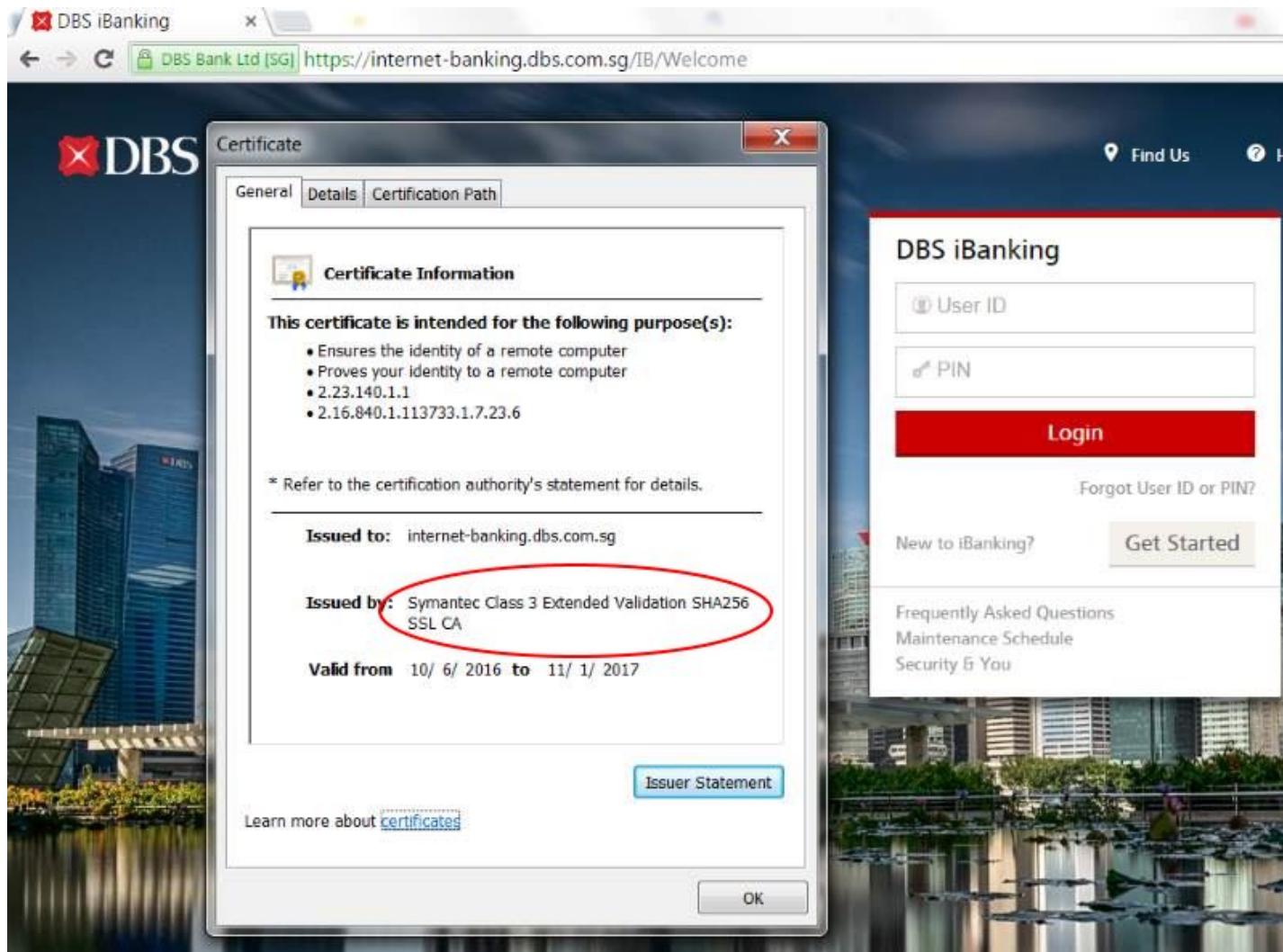
Browser UI Security Indicators

- Browsers offer users different **visual-based security indicators** on different types of certificates
- Examples: Two different domains as shown by Chrome browser below
- Can you guess which types of certificates used?



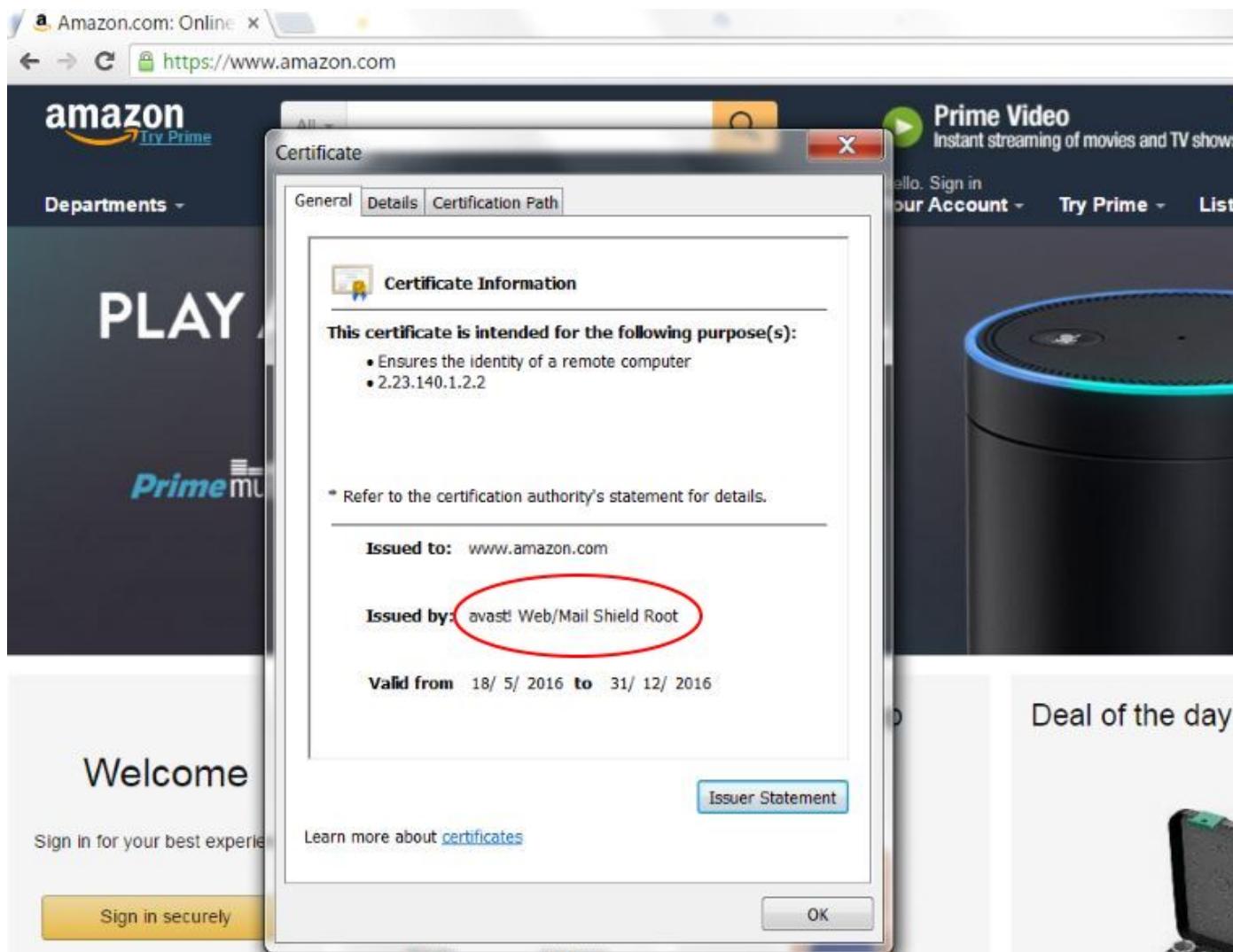
EV SSL Certificate

- DBS Internet banking:



Standard SSL Certificate

- www.amazon.com:



Browser UI Security Indicators

Browser UI Security INDICATORS as of March 2017:

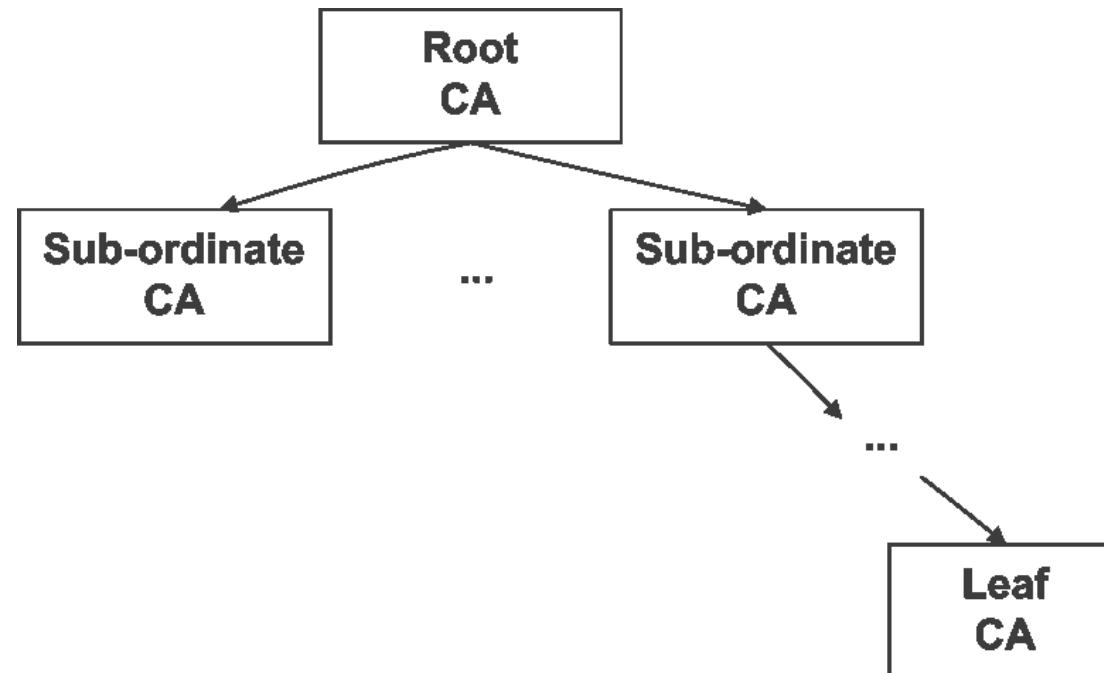
Updated URL UIs indicated by orange outlined box

| Browser UI Security Indicator: | HTTP only (no certificate) | DV certificate | OV certificate | EV certificate |
|--------------------------------|-------------------------------|------------------------------|--------------------------|---|
| Chrome 56 (Windows) | (i) www.example.com | Secure https://case | Secure https://www | Trustwave Holdings, Inc. [US] https://www.trust |
| Chrome 56 (Android) | www.example.com | https://casetrusted.com | https://www.example | https://www.trust.com |
| Edge 20 (Windows) | example.com | casecurity.org | example.com | Symantec Corporation [US] symantec.com |
| Firefox 51 (Windows) | (i) www.example.com | (i) https://casetrusted.com | (i) https://www.example | (i) COMODO CA Limited [GB] https://crt.sh |
| Safari 10 (Mac) | (i) www.example.com | (i) https://casetrusted.com | (i) https://www.example | GMO GlobalSign Inc |
| Safari 10 (iOS) | example.com | casecurity.org | example.com | DigiCert, Inc. |
| OperaMini 23 (Android) | www.example.com | casecurity.org | www.example.com | www.globalsign.com/en/ |
| UC Mini 10 (Android) | Example Domain | CA Security Council | Example Domain | https://www.digicert.com |
| UC Browser 10.8.6.889 (iOS) | example.com | CA Security Council | example.com | SSL & Digital Certificates by GlobalSign |

Source [https://casetrusted.com/wp-content/uploads/2017/03/CASC-
Browser-UI-Security-Indicators.pdf](https://casetrusted.com/wp-content/uploads/2017/03/CASC-Browser-UI-Security-Indicators.pdf)

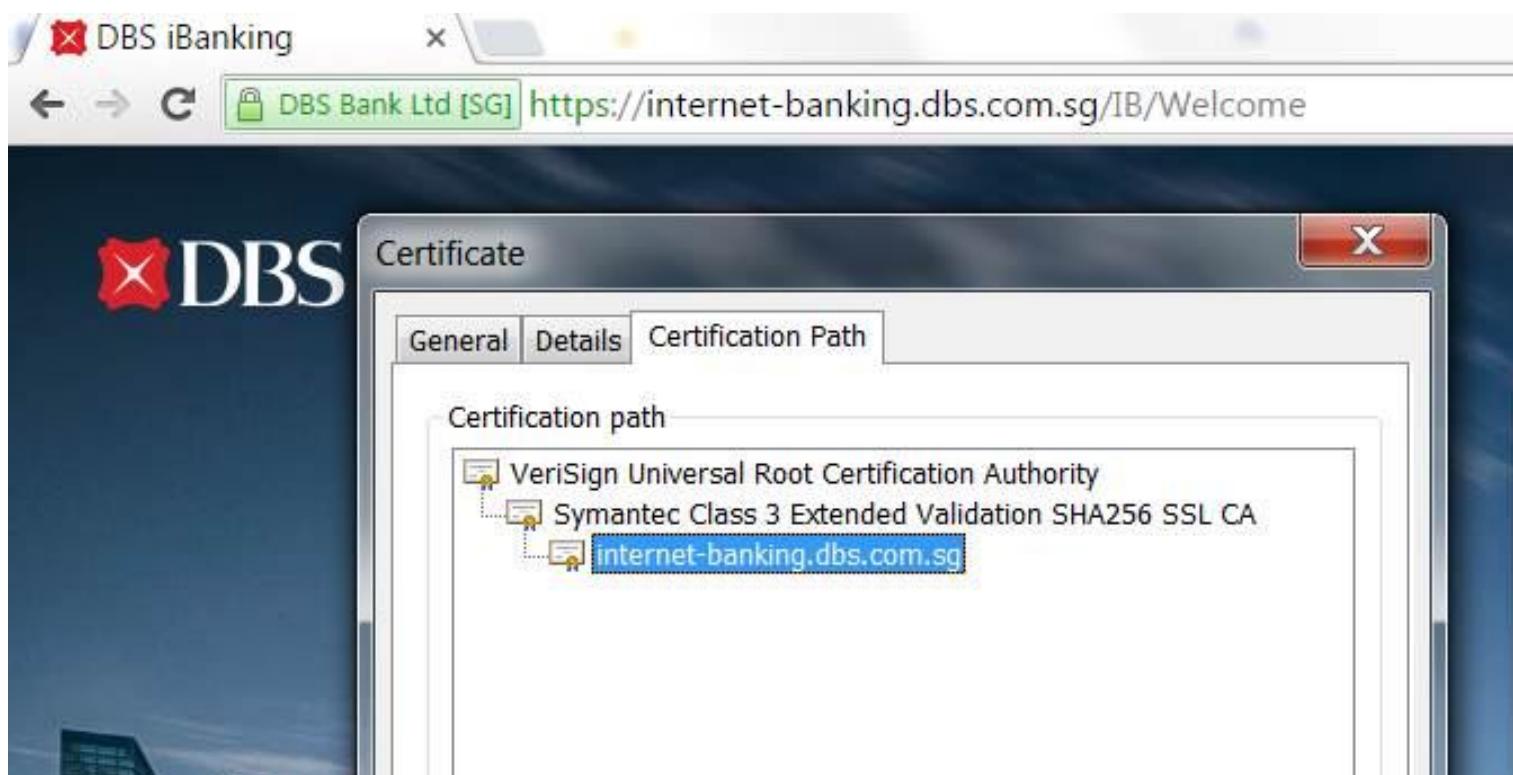
Types of CA

- There are **3 different types** of CA:
 - Root CA: whose certificate is self-signed
 - Sub-ordinate/intermediate CA: Tier 1, 2, ...
 - Leaf CA



Types of CA: A Real Example

- DBS Internet banking website:



Hierarchy of Trust

Trust inference:

Bob trusts CA #1: because Bob trusts the *Root CA*, and the *Root CA* certifies CA #1

Bob also trusts CA #3 and all certificates signed by CA #3: since CA #1 certifies CA #3.

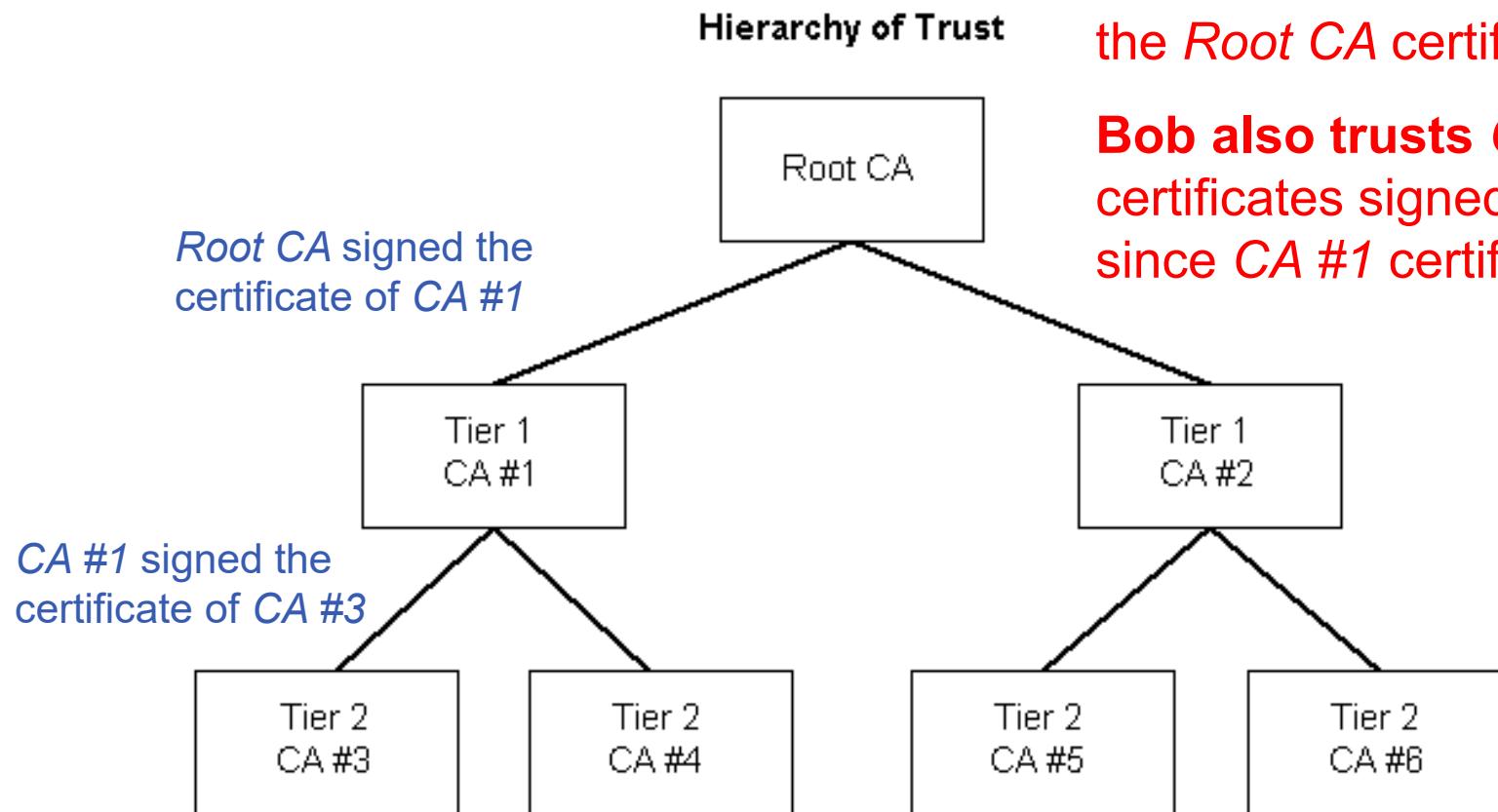


image from
<https://msdn.microsoft.com/en-us/library/windows/desktop/aa382479%28v=vs.85%29.aspx>

See [PF] pages 117-121 for a detailed explanation of **Trust**

Certification Chain/Path Verification: An Example

- Suppose Alice's certificate is issued & signed by CA_1 , which is a tier-1 intermediate CA
- Bob doesn't have the public key of CA_1
- *Why should Bob do?*
- In the first place, Alice, anticipating the Bob might not have CA_1 's public key, can send Bob her email, her certificate, and CA_1 certificate (*see the next slide*)
- Now, Bob can:
 - Verify CA_1 's certificate: using root CA's public key
 - Verify Alice's certificate: using the verified CA_1 's public key
 - Verify Alice's email: using Alice's verified public key
- If Alice doesn't attach CA_1 's certificate, then Bob has to obtain it from other sources

Certification Chain/Path Verification: An Example

- Illustration:

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

Signature: xsdewsdesd

Name : alice@yaho.com.sg
Public key: x1s34adf39
Valid until: 1 Sep 2019
Signature of the CA₁

Name : CA₁
Public key: x3141342
Valid until: 1 Sep 2020

Note: CA₁ can issue certificates

Signature of the Root CA

- In our example, CA₁'s certificate clearly indicates that CA₁ is a CA that can issue certificate
- Without that “Note” portion, the certificate owner can't issue other certificates

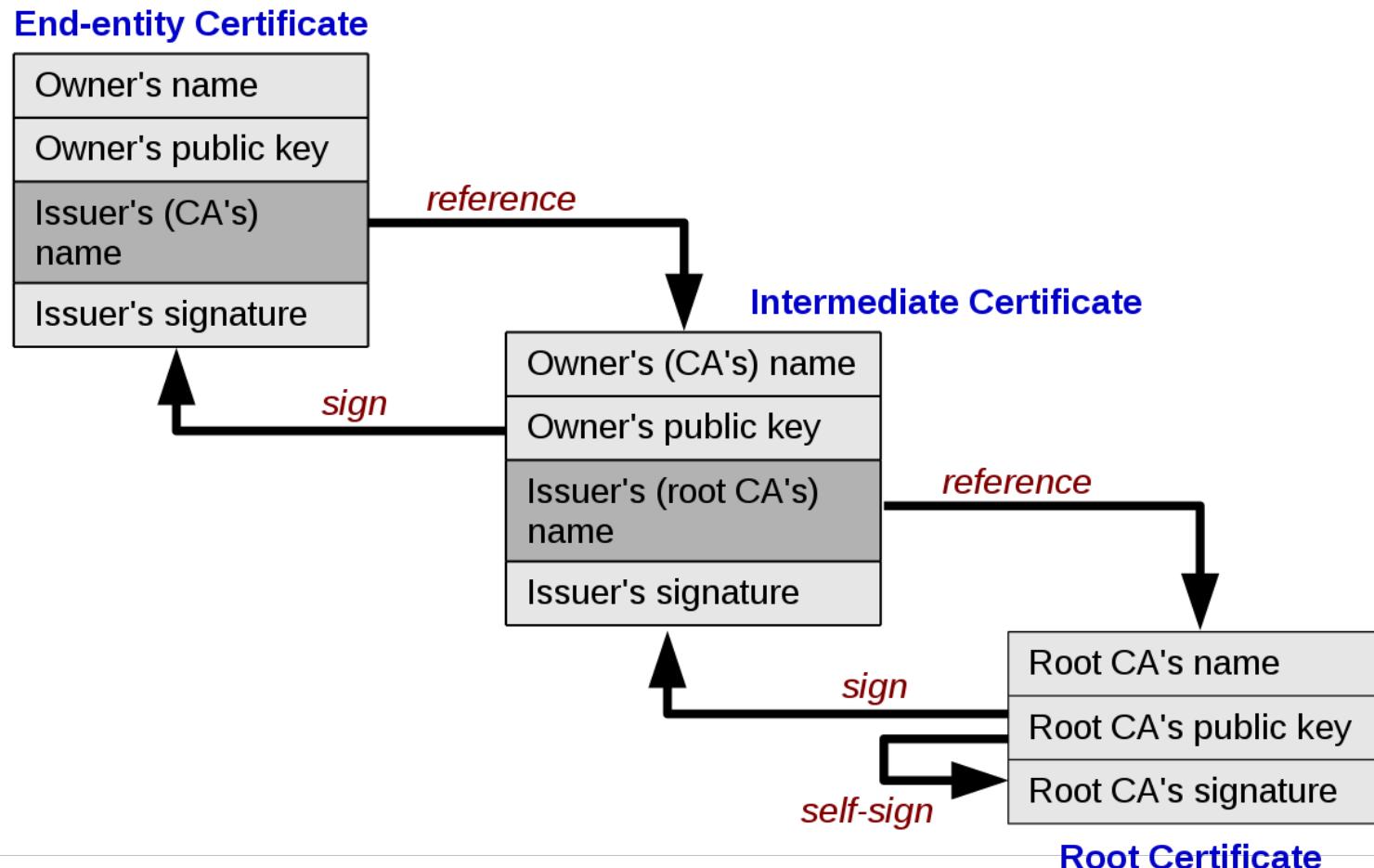
Certification Chain: Definition

Certification chain/path:

- A **list of certificates** starting with an **end-entity certificate** followed by one or more CA certificates (with the last one being a self-signed **root certificate**)
- For each certificate (except the last one):
 - The issuer matches the subject of the next certificate in the list
 - It is signed by the secret key of the next certificate in the list
- The last certificate in the list, i.e. the root CA's, is the **trust anchor**

Certification Chain: Diagram and Verification

- How does a certificate chain get verified?



Souce: Wikipedia

Some Questions:

- Occasionally, while surfing the web, you may encounter this warning message:

www.example.com uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.

Option 1: Get me out of here.

Option 2: I know the risk. Accept the certificate.

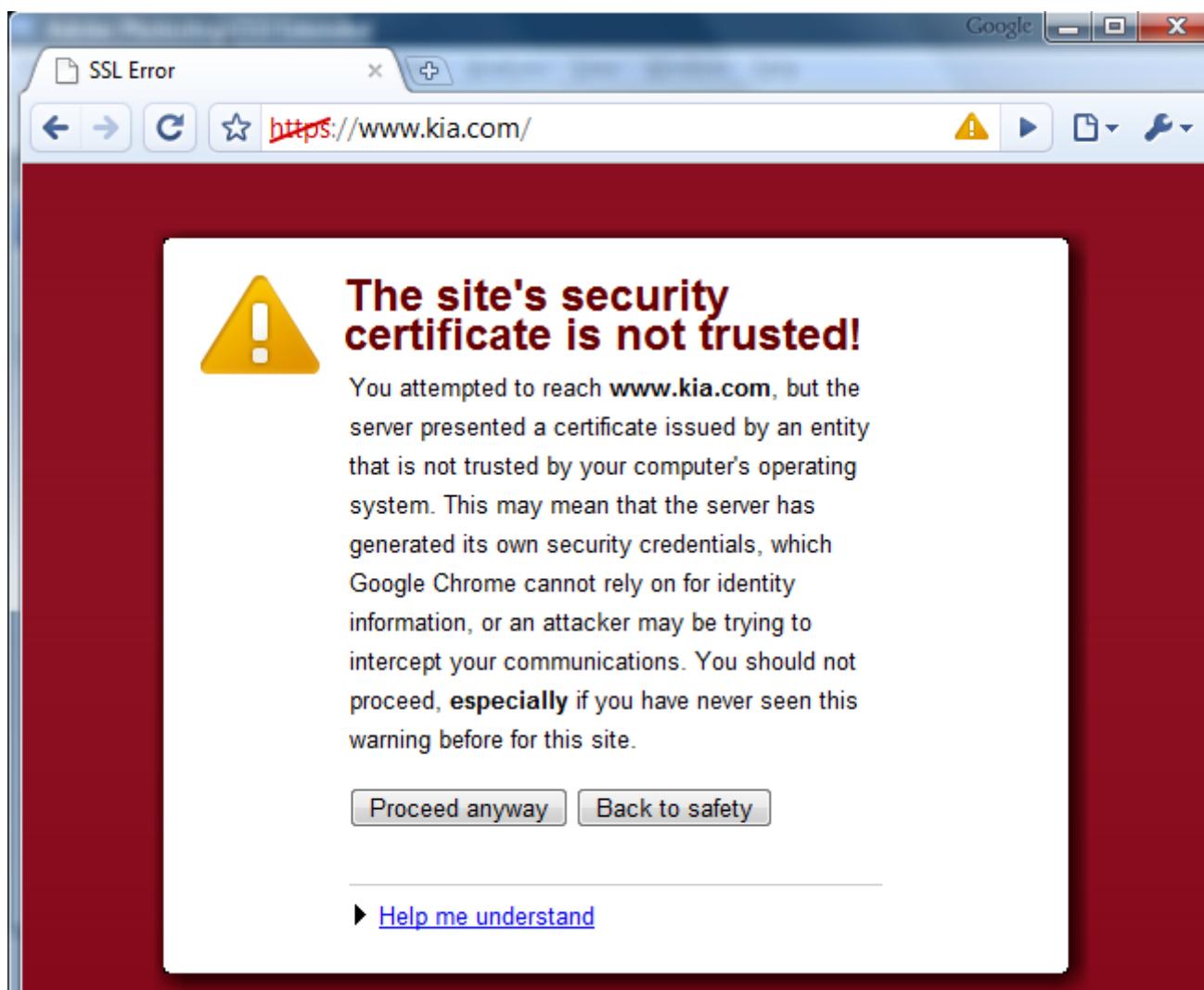
What is going on here? What are the security implications?

- While installing a new package using a package manager (this applied to MAC OS, Linux, cgywin, etc.), say apt-get, you may also encounter similar message:

Packages server certificate verification failed.

What is going on here? Should you continue the installation?

Some Questions: Sample Alert Box



Another Issue: Certificate Revocation

- **Non-expired certificates** can be **revoked** for different reasons:
 - Private key was compromise
 - Issuing CA was compromise
 - Entity left an organization
 - Business entity closed
- A verifier needs to check if a certificate in question is **still valid**, although the certificate is not expired yet
- Different approaches to certificate revocation:
 - **Certificate Revocation List (CRL)**:
CA periodically signs and publishes a revocation list
 - **Online Certificate Status Protocol (OCSP)**:
OCSP Responder validates a cert in question
- An ***online*** CRL Distribution Point or OCSP Responder is needed

Another Issue: Certificate Revocation

- As of Firefox 28, Mozilla have announced they are **deprecating** CRL in favor of OCSP
- Some OCSP problems:
 - **Privacy:** OCSP Responder knows certificates that you are validating
 - **Soft-fail validation:** Some browsers proceed in the event of no reply to an OCSP request (no reply *is* a “good” reply)
- Solution/improvement?
 - ***OCSP stapling:*** allows a certificate to be accompanied or “stapled” by a (time-stamped) **OCSP response** signed by CA
 - **Part of TLS handshake:** clients **do not** need to contact CA or OCSP Responder
 - **Drawback:** increased network cost

4.3 Limitations/Attacks on PKI

Compromised CAs

CA breach incidents:

Four CAs Have Been Compromised Since June

Posted by **SoulSkill** on Friday October 28 2011, @04:08PM
from the four-whole-californias-wow dept.

- DigiNotar (Netherlands):
 - Resulted in 500+ fraudulent certificates, including for *.google.com, *.mozilla.com, *.windowsupdate.com, *.torproject.org, in Sept 2011
 - Immediately removed by major browsers
 - Declared bankrupt within the same month
- Turktrust (Turkey):
 - Its sub-ordinate CA, *.EGO.GOV.TR, issued *.gmail.com certificates
 - Fraudulent certificates were used against Google Web properties

Abuse by CA

- There are so many CAs: Some of them could be malicious
- A **rogue CA** can practically forge any certificate.
Here is a well-known incident.
- **Trustwave** issued a “sub-ordinate root certificate”, which can then issue other certificates, to an organization for monitoring the network. With this certificate, the organization can “**spoof**” X.509 certificates, and hence is able to act as the **man-in-the-middle** of any SSL/TLS connection.
- See:
ComputerWorld, *Trustwave admits issuing man-in-the-middle digital certificate; Mozilla debates punishment*, Feb 8 2012.
<http://www.computerworld.com/article/2501291/internet/trustwave-admits-issuing-man-in-the-middle-digital-certificate--mozilla-debates-punishment.html>

Another Famous Case of CA's Abuse (or Ignorance?)

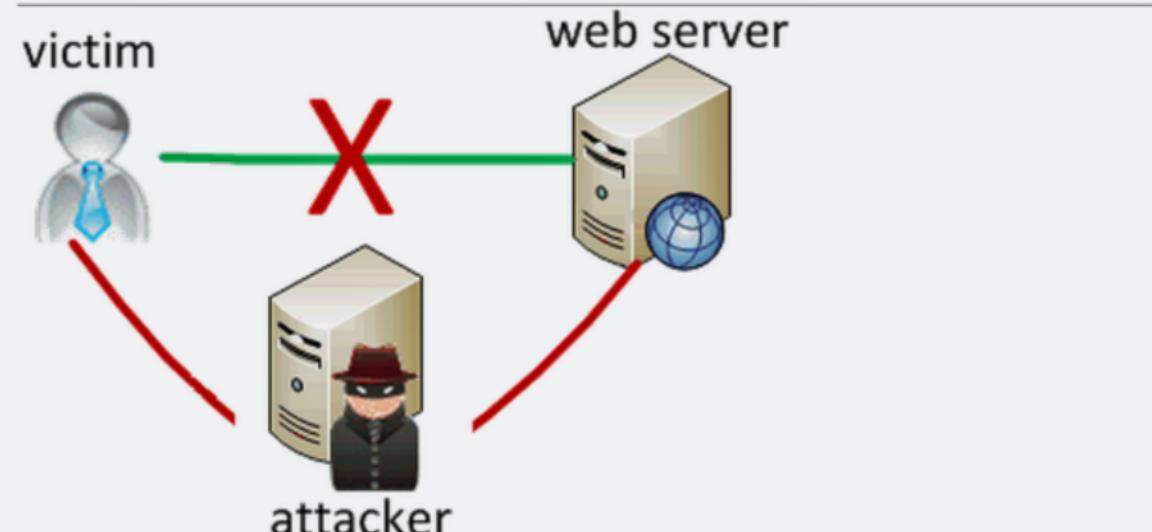
- Lenovo's **SuperFish scandal** (*reserved for class presentation*)

BIZ & IT —

Lenovo PCs ship with man-in-the-middle adware that breaks HTTPS connections [Updated]

Superfish may make it trivial for attackers to spoof any HTTPS website.

DAN GOODIN - 2/20/2015, 12:36 AM



victim

web server

attacker

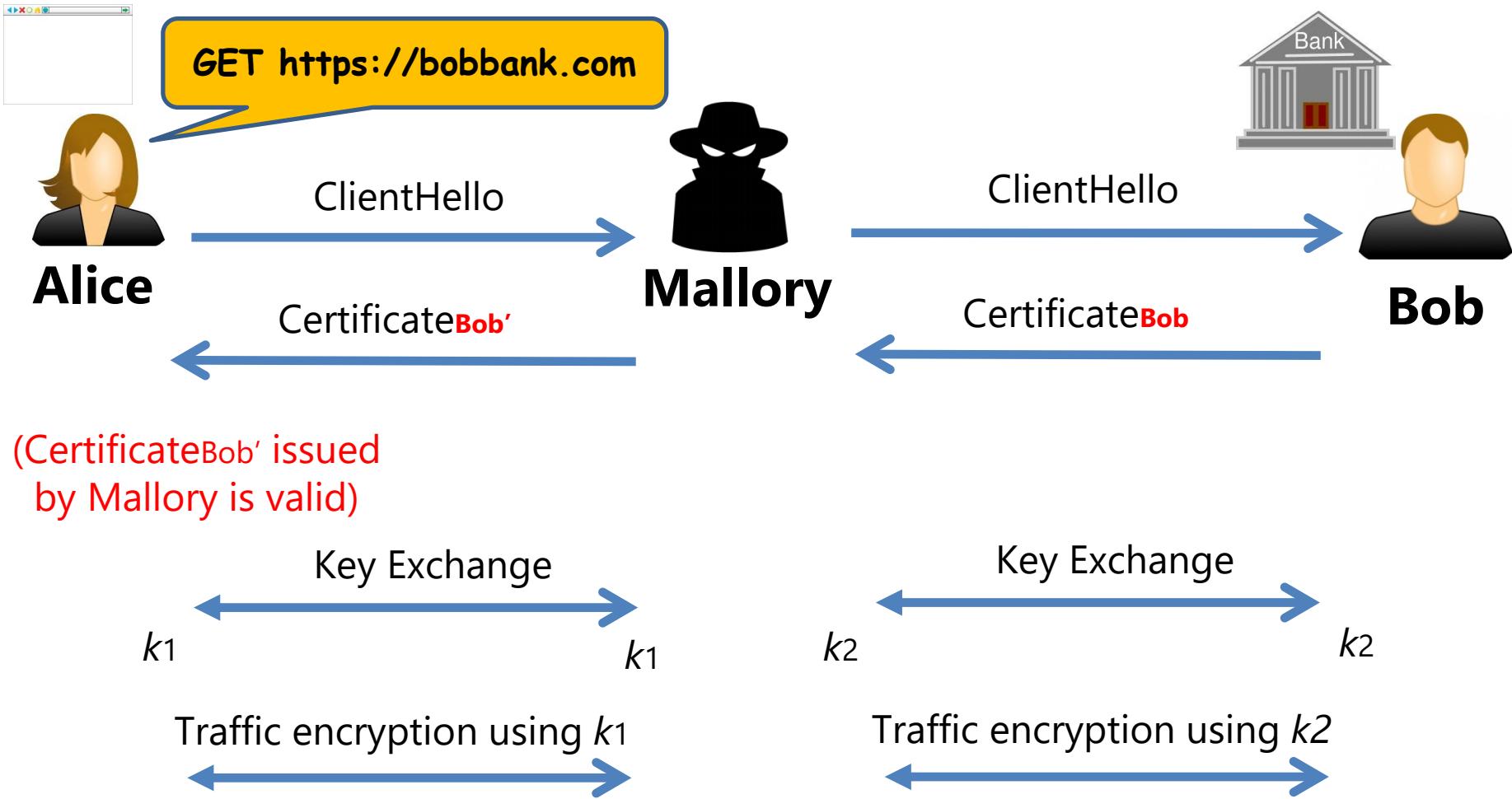
Web Programming Step by Step, 2nd Edition

Source: <https://arstechnica.com>

Weak Browser Trust Model

- **Browser trust model:**
 - A **pre-loaded list** of widely-used root CAs compiled by web browser developers
 - An form of *Certificate Trust List (CTL) approach*, where a list of CAs' certificates are compiled by a “trusted” authority
- **Security issue:**
 - Trust anchor: the **union** of all root CAs
 - Question: **which root CA** is the one used from the root-CA list?
 - Certification is only as *strong as the weakest root CA!*
- *See real-world analyses in:*
 - Peter Eckersley, Jesse Burns, “An observatory for the SSLiverse”, Defcon 18, 2010.
 - Peter Eckersley Jesse Burns, “Is the SSLiverse a Safe Place?”, 27th Chaos Communication Congress (CCC), 2010.

MITM Attack by a Rouge CA



Mallory performs a *proxy re-encryption*.
He can see all traffic, and also modify data!
(To be discussed in Tutorial)

Implementation Bugs: E.g. Null-byte Injection Attack

- There are quite a number of well-known implementation bugs leading to severe vulnerability. Here is one example.
- Some browsers ignore the substrings in the entity's identity/name field after the null characters **when displaying it in the address bar, but include them when verifying the certificate.**

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

(a) The common name in the cert when it is **being verified:**

“www.comp.nus.edu.sg\0.hacker.com”

(b) The browser displays it as:

“www.comp.nus.edu.sg”

- As a result, the user **thought** he/she is connecting via https to www.comp.nus.edu.sg, **but in fact** to www.comp.nus.edu.sg\0.hacker.com.
- See also:

www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/

Question (Terminologies): What is CVE?

Social Engineering

Malicious hackers may carry out ***typosquatting***.

For example:

1. A hacker **registered** for a domain name `luminus.nvs.edu.sg`, and **obtained a valid certificate** of the name
2. The hacker employs a **phishing attack**, tricking a victim to click on the above link, which is a spoofed site of `luminus.nus.edu.sg`
3. The address bar of the victim's browser **correctly displays** `https://luminus.nvs.edu.sg`, but the victim doesn't notice that, and log in using the victim's credential

It is also possible that the hacker doesn't carry out step 2. He just waits and hopes that some students accidentally type the wrong address `luminus.nvs.edu.sg`.

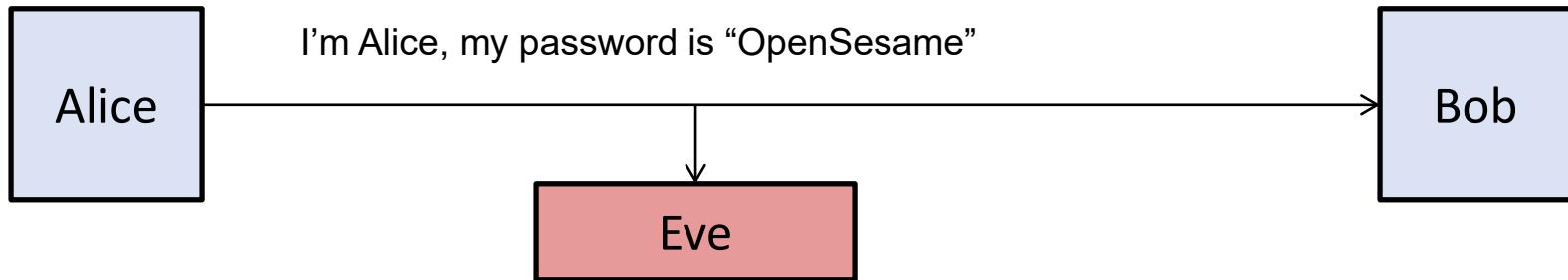
Read <http://en.wikipedia.org/wiki/Typosquatting>

4.4 Strong Authentication

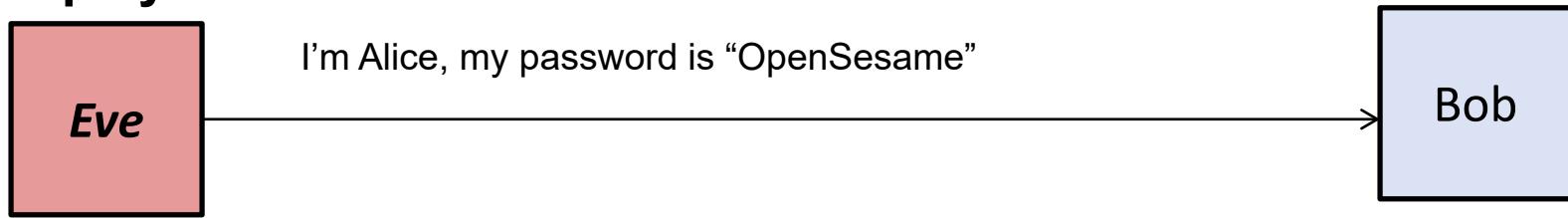
Password and Weak Authentication

We've mentioned that password is a weak authentication system: an eavesdropper can get the password, and **replay** it

Eve eavesdrops:



Eve replays:



The **secret** that is shared between Alice and Bob is the password. 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 k .

Both of them agreed on an encryption scheme, say AES.

(1) Alice sends to Bob a hello message:

“Hi, I’m Alice”

(2) (Challenge) Bob randomly picks a plaintext m .

Bob computes:

$$y = E_k(m)$$

and sends y to Alice.

(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: 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?

Unilateral Strong Authentication using PKC

Suppose Alice wants to authenticate Bob.

- (1) (Challenge) Alice chooses a **random number r** and sends to Bob:
“Bob, here is your challenge”, r
- (2) (Response) Bob uses his private key to sign r .
Bob also attaches his certificate:
 $\text{sign}(r)$, Bob’s certificate
- (3) Alice verifies Bob’s certificate, extracts Bob’s public key from the certificate, and then verifies that the signature is correct.

An eavesdropper can’t derive Bob’s private key and replay the response because the challenge is likely to be different.

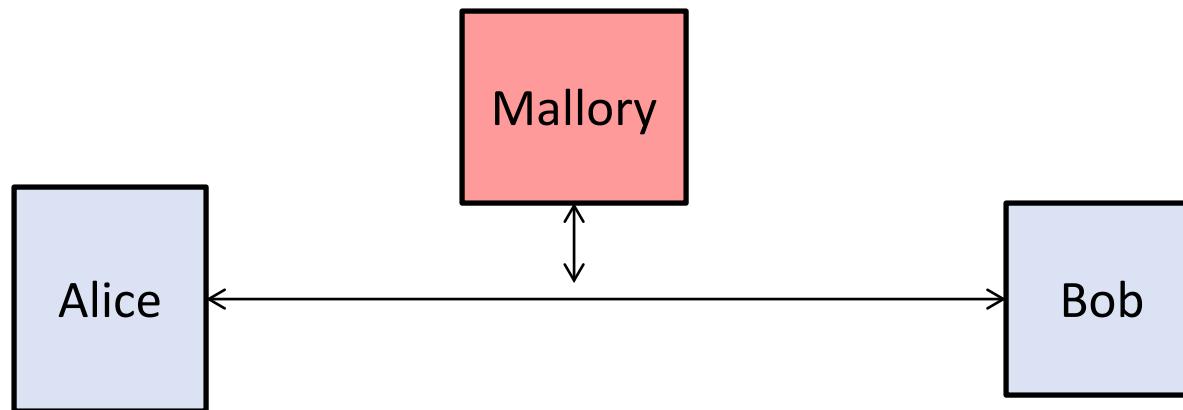
The value r is also known as the ***cryptographic nonce*** (or simply ***nonce***).

Question: Which component in the above ensure freshness?

Remark: the shown protocols have omitted many details.
Designing a secure authentication protocol is **not easy**.

Is Authentication Alone Sufficient?

- You may wonder what comes 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.
- Later Mallory pretends to be Alice!

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 is a new secret k (a.k.a **session key**) established by Alice and Bob
- The process of establishing a secret between Alice and Bob is called **key exchange** or **key agreement**

Authenticated Key Exchange

- If the process is incorporated with authentication, then it is called ***authenticated key exchange*** or ***station-to-station protocol*** (if the key-agreement used is Diffie-Hellman):
 - To carry out the protocol, Alice and Bob must have a way to know **each other's public key** (e.g. using PKI)
 - For unilateral authentication, only **one party** needs to have public key
 - After the protocol has completed, **a set of session keys** is thus established: e.g. 1 key for encryption & 1 key for MAC; 1 key for each direction of encryption, etc.

4.5 Putting All Together: Securing a Communication Channel

Secure Channel/Communication 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 tools?
- “**A Secure channel**” establishes, between 2 programs, a data channel that has *confidentiality*, *integrity*, *authenticity* against a computationally-bounded “network attacker” (i.e. Mallory)
- Common example:
Imagine that Alice wants to visit a website Bob.com.
- How to protect the **authenticity** (that Bob.com is authentic), and **confidentiality & integrity** of the communication?
- We have discussed some important necessary mechanism:
authenticated key exchange, PKI, encryption, ...
(How about message-ordering protection?)

Securing Alice's Communication with Bob.com

(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 t, k**

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

*(Details of how t and k can be established are omitted.
See station-to-station protocol for details.)*

Securing Alice's Communication with Bob.com

(Step 2)

Subsequent communication between Alice and Bob.com will be protected by t , k 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:

$$E_k(i \parallel m_i) \parallel MAC_t(E_k(i \parallel m_i))$$

where: i is the sequence number,

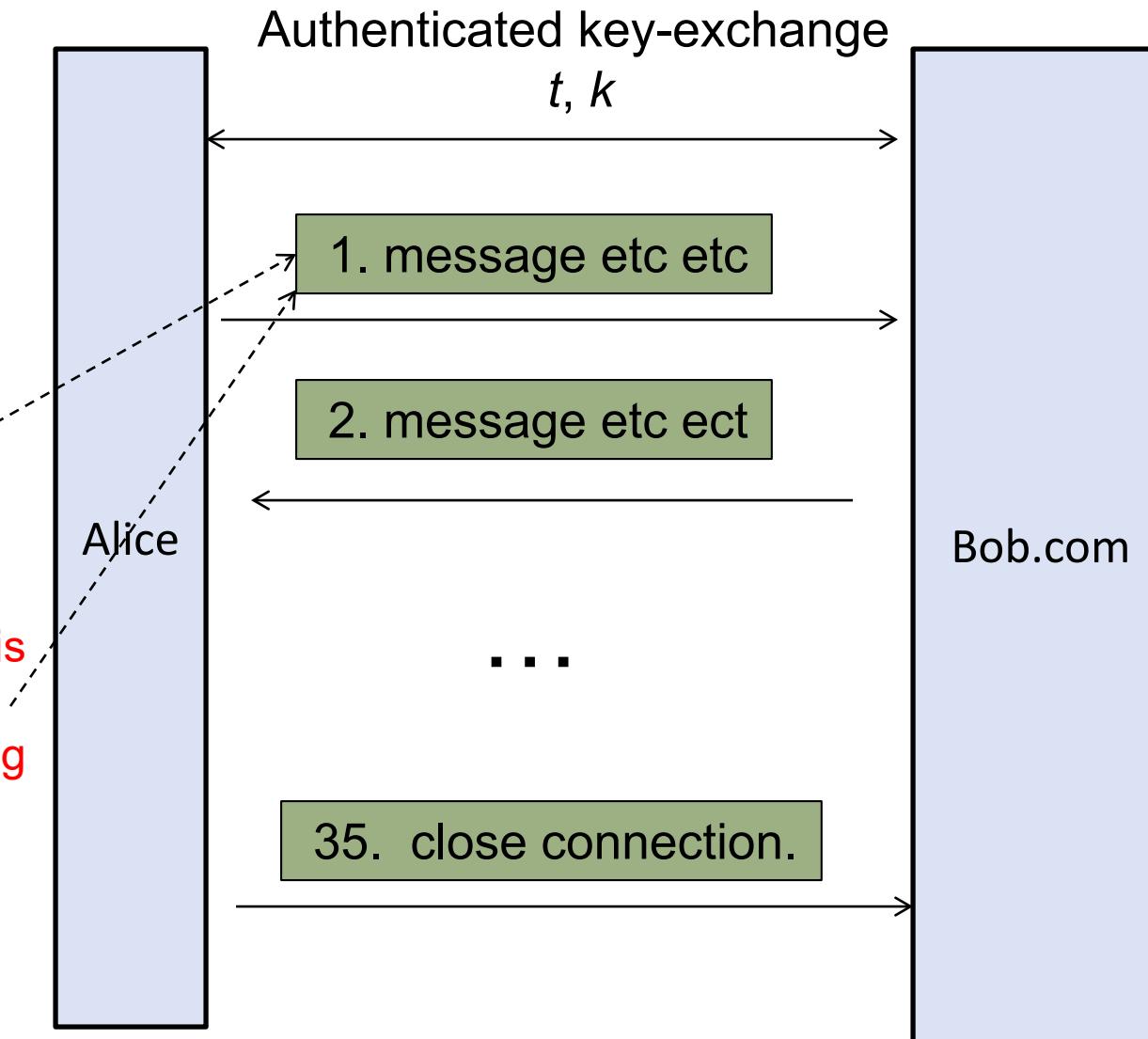
\parallel refers to concatenation

(Optional) The technique above is known as “encrypt-then-MAC”. There are other variants of **authenticated encryption** called “MAC-then-encrypt” and “MAC-and-encrypt”.

Secure Communication between Alice and Bob.com

Authenticity is protected by MAC using t as key

Confidentiality is protected by encryption using k as key.



Question: Why do we need the sequence number?

Secure Communication and PKI

- 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: suppose **Alice** visits **Bob.com**, and wants to verify that the entity she's communicating 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
- (Note: this is a case of unilateral authentication)

4.6 Putting All Together: HTTPS

HTTPS Protocol

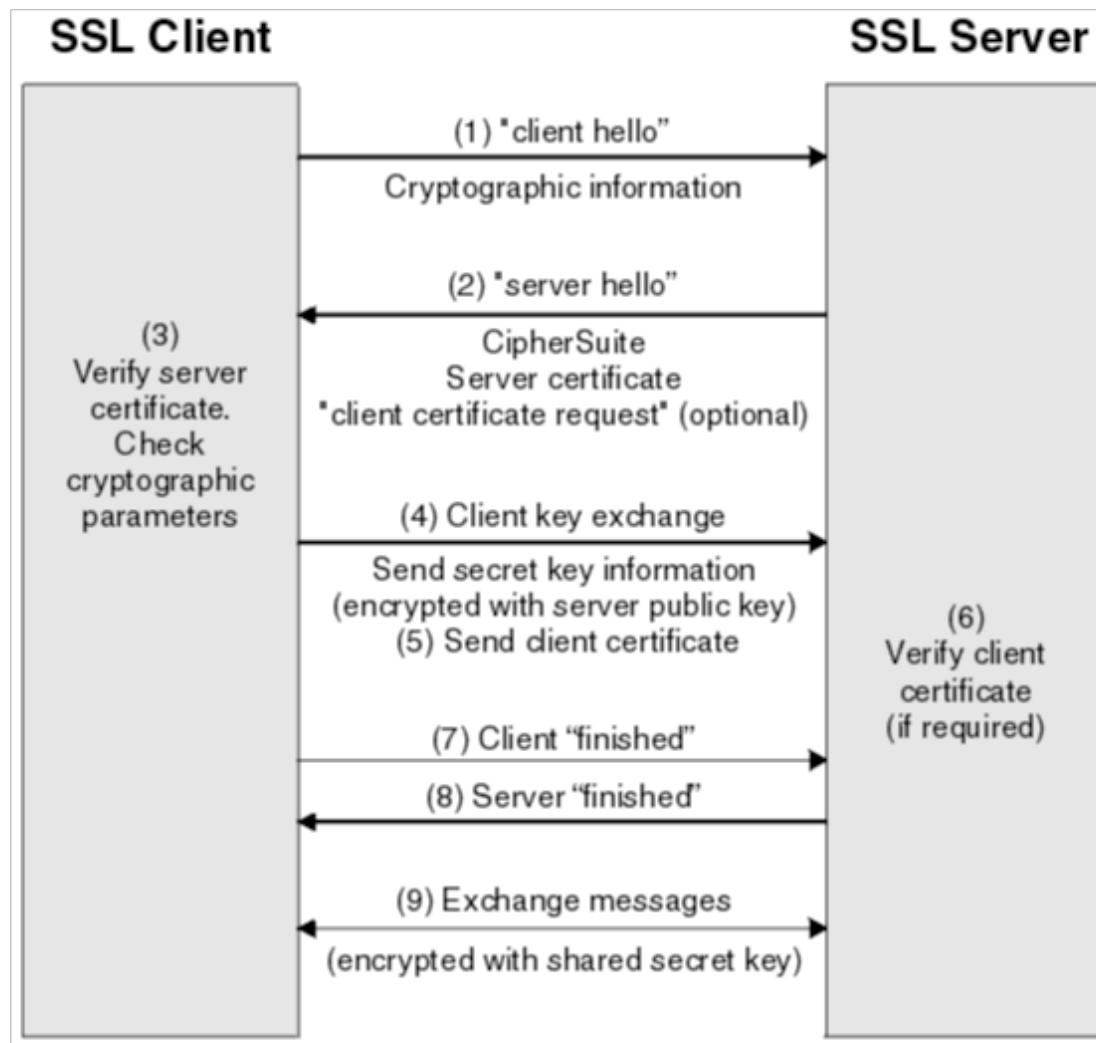
- **HTTPS (HTTP Secure)** is widely used to secure Web traffic
- HTTPS is built on top of SSL/TLS: **HTTPS = HTTP + SSL**
Hence, HTTPS is also called: HTTP over SSL,
or HTTP over TLS
- Transport Layer Security (**TLS**) is a protocol to secure
communication using cryptographic means:
TLS 1.2 [2008],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

HTTPS Protocol

- How does HTTPS work **at the high level?**
 - Ciphers negotiation
 - Authenticated Key Exchange (AKE):
the exchange of session key, which also authenticates
the identities of parties involved
 - Symmetric-key based secure communication
 - Re-negotiation (if needed)

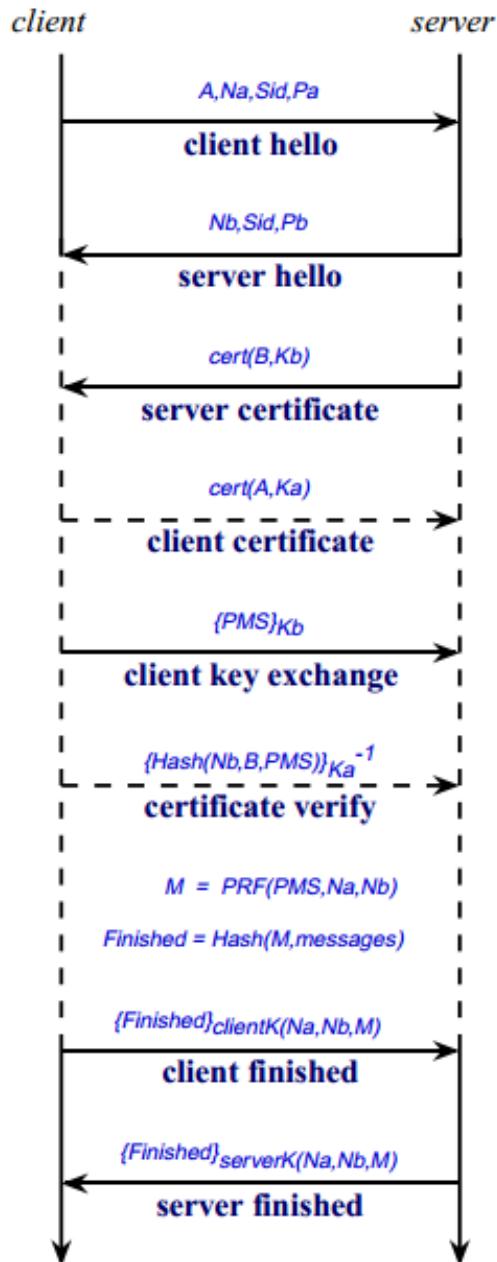
Question: Alice is in a café. She uses the free WiFi to upload her assignment to IVLE (which uses HTTPS). The café owner controls the WiFi router, and thus can inspect every packet going through the network. Can the café owner get Alice's report?

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)



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

HTTPS Protocol in Action: An Example

The screenshot shows a web browser window for Amazon.com. At the top, the address bar indicates a secure connection (https://www.amazon.com). The main content area features several promotional banners: "SEE SOMETHING NEW, EVERY DAY.", "TAKE A LOOK", "Shop school supplies", and decorative items like sunglasses and a rainbow. The navigation bar includes links for "Deliver to Singapore", "Your Amazon.com", "Today's Deals", language selection ("EN"), and user account options ("Hello, Sign in", "Account & Lists", "Orders", "Try Prime"). A shopping cart icon shows 0 items.

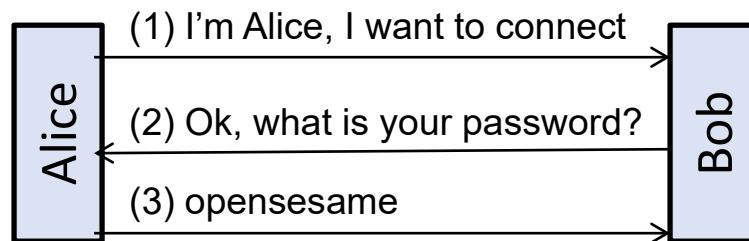
Below the main content, the browser's developer tools are visible, specifically the Network tab. The "Security" sub-tab is selected, displaying a green message: "This page is secure (valid HTTPS)". It lists two main points: "Certificate - valid and trusted" and "Connection - secure (strong TLS 1.2)". Under the connection point, it details the encryption method: "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." The text "[TLS 1.2]" and "[AES_128_GCM]" is highlighted with red boxes.

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

Side Remark: What is a Protocol?

Protocol Definition and Example

- Slides 60, 62 and 68-69 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 want to connect”
 - Alice ← Bob: “Ok, what is your password?”
 - Alice → Bob: “opensesame”
- It can also be visually shown as below:



Assignment 1

- Any questions so far?
- Please ask your TAs, discuss in Luminus Forum
- Available ***consultation sessions on A1***
this week and next week
- Please submit the answers before A1 due date

Mid-Term Quiz

- Please refer to the announcement released in LumiNUS!
- Time: **Friday, 4 October 2018, 10:20-11:30am**
(please do come by 10:05am as usual if you can)
- Venue: **LT19** (the usual lecture venue)
- Weightage: **15%** of your final mark
- **Format:**
 - Part A: Multiple choice questions
 - Part B: Security terminology questions
 - Part C: Short-answer and scenario-based questions
(similar to tutorial questions)

Mid-Term Quiz

- **Scope:**
 - Lectures 0-5, Tutorials 1-4
 - The details of techniques in your assignments
are not examinable
- **Open book:**
 - Please be self-sufficient
 - **No** communication devices
 - NUS approved calculators are fine: but you *should be able* to answer the solutions even without a calculator
- Past-year mid-term quiz will be uploaded soon
- Consultation hours during the recess week will be open

Lecture 5: Network Security

- 5.1 Background: Network layers
- 5.2 Name resolution and attacks
- 5.3 Denial of Service (Dos) attacks
- 5.4 Useful network security tools
- 5.5 Protection: Securing the communication channel using cryptography
- 5.6 Protection: Firewall
- 5.7 Protection: Network security management

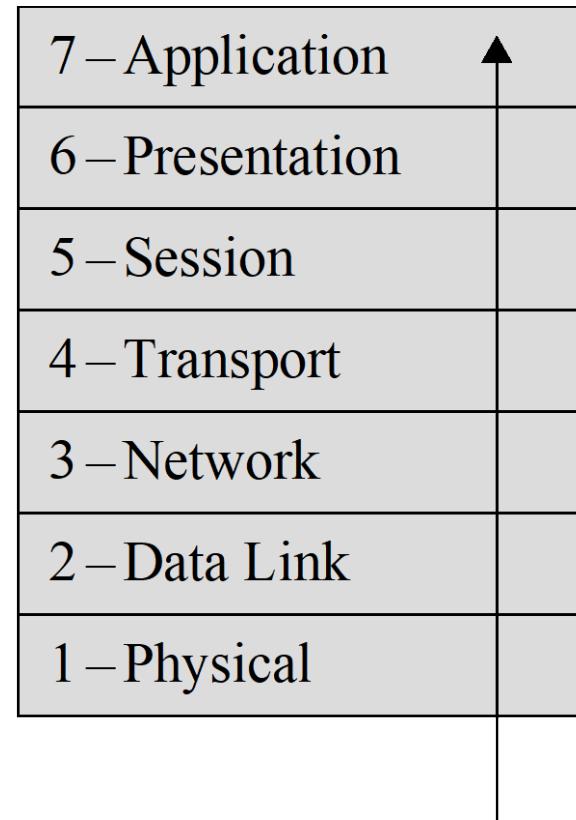
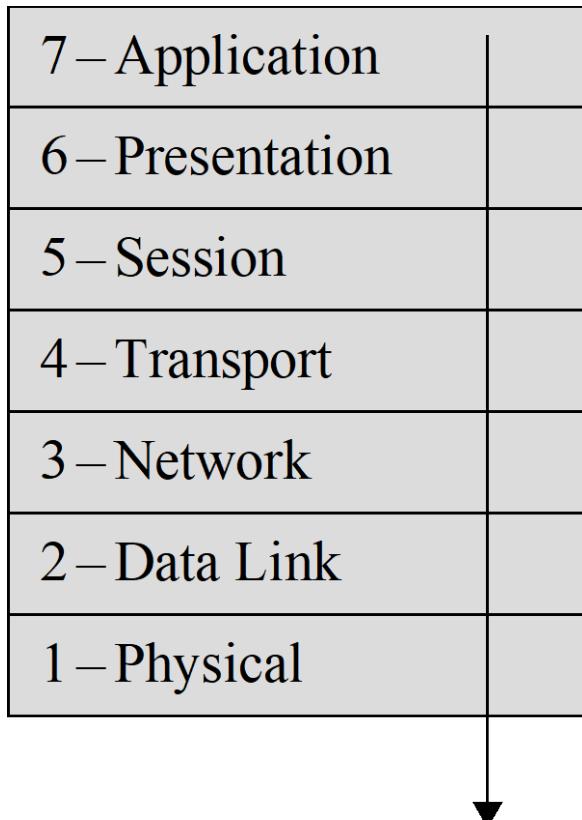
See: [PF6.1], [PF6.2],[PF6.4],[PF6.6], [PF6.9]

5.1 Background: Network Layers

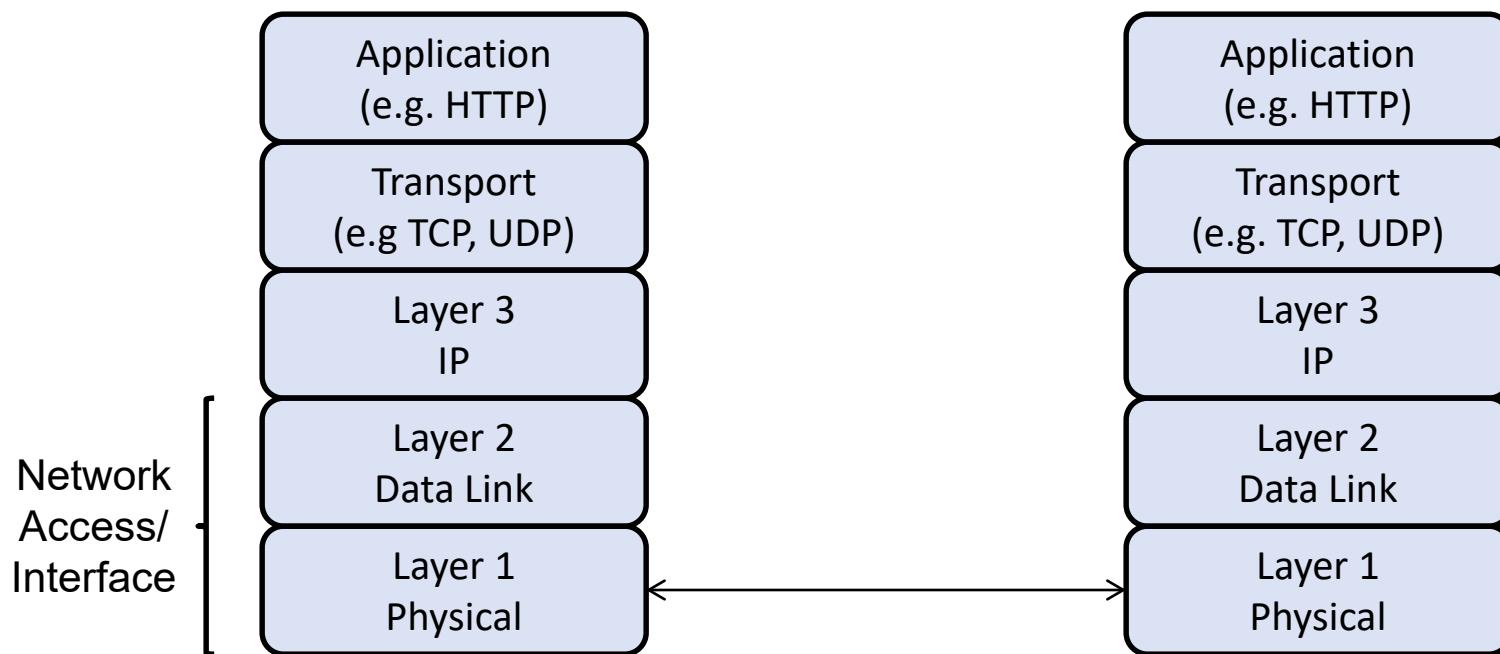
The OSI Seven-Layer Network Model

The ***Open Systems Interconnection (OSI) model:***

a conceptual/reference model that standardizes the
communication functions of a telecommunication/computing system

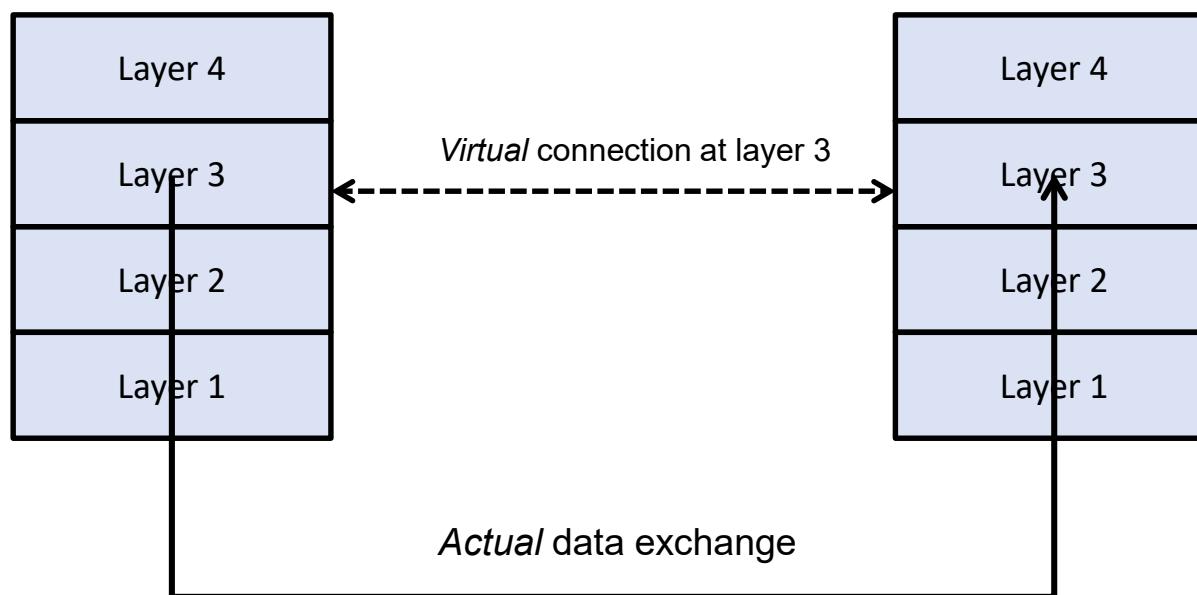


The Internet (TCP/IP) Reference Model



Why Network Layering?

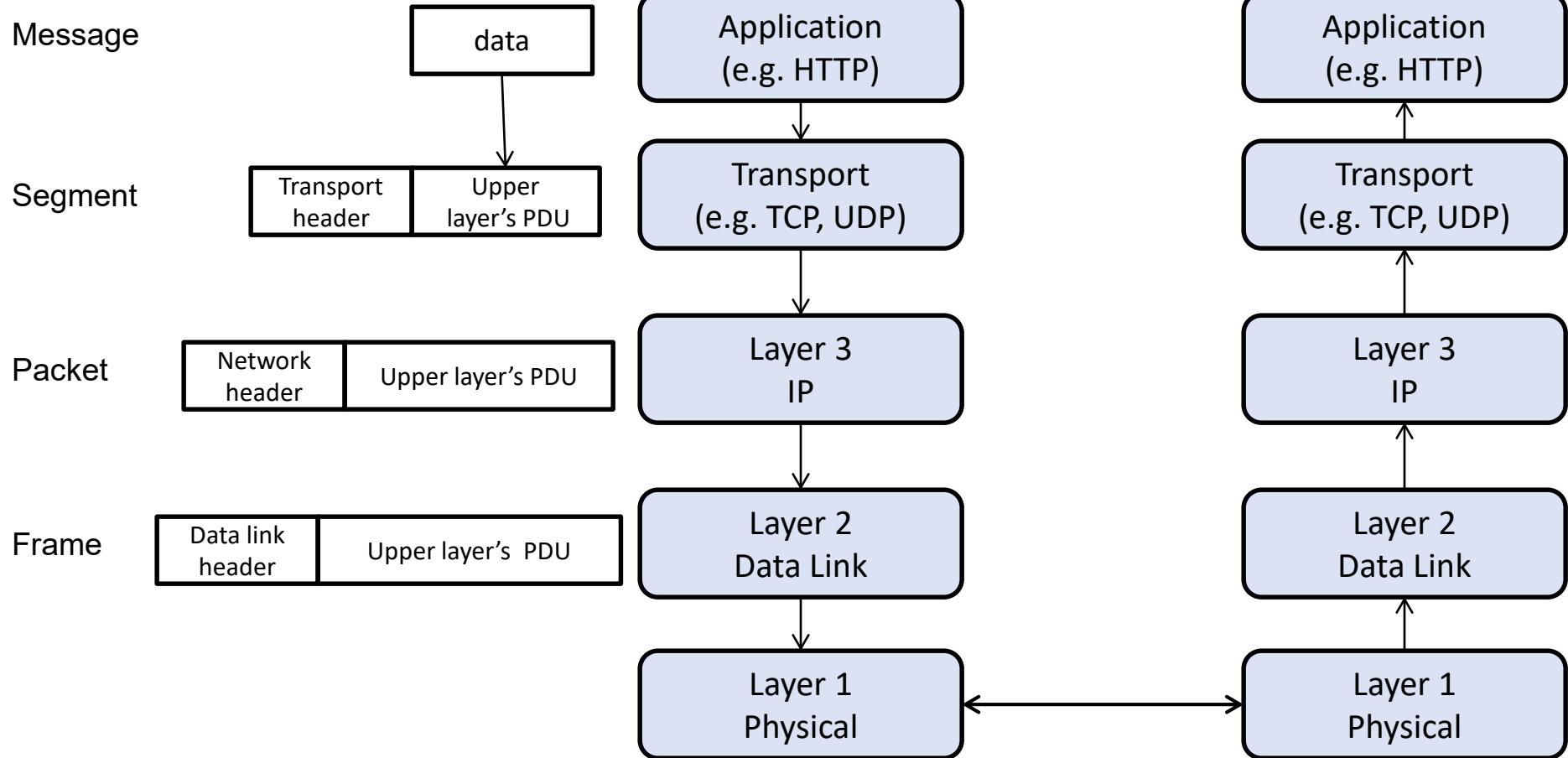
- It partitions a complex communication system into several **abstraction layers**
- The **peer entities** at the same layer N “conceptually” communicate with each other by executing a protocol *at that layer*



Why Network Layering?

- The layer- N protocol is built **on top of** a virtual connection at layer $N-1$ below
- Example of a protocol at **layer 4** (end-to-end protocol):
 - (1) $A \rightarrow B$: “hello”
 - (2) $A \leftarrow B$: certificate of B
- The “virtual connection” at layer 3 sends the message “hello” from A to B in Step (1), and sends B ’s certificate in Step (2)
- At layer N :
 - A message to be sent is called:
layer- N protocol data unit (PDU)
 - **Encapsulation** of upper (i.e. $N+1$) layer’s PDU

Network Layers and Message Encapsulation



General PDU format:



Header: meta data

Payload: the message/information intended to be sent

Internet Layers: Different Addressing Schemes

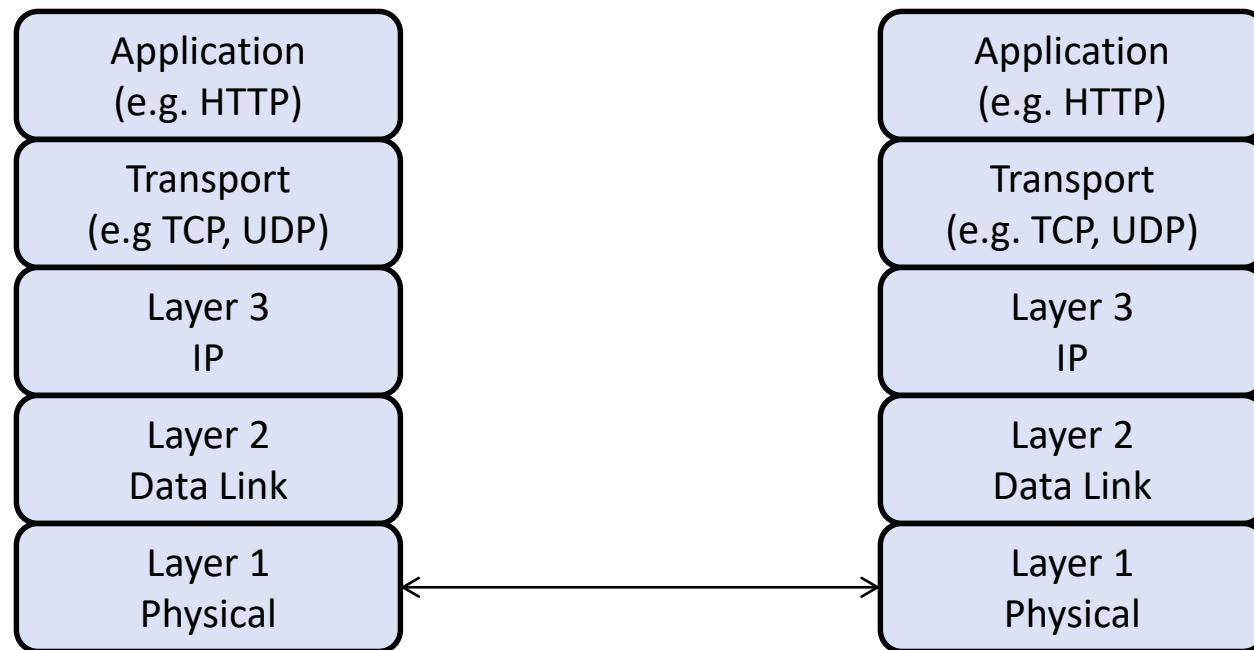
- Different addressing schemes at different layers

Domain name
www.comp.nus.edu.sg

Port number
80

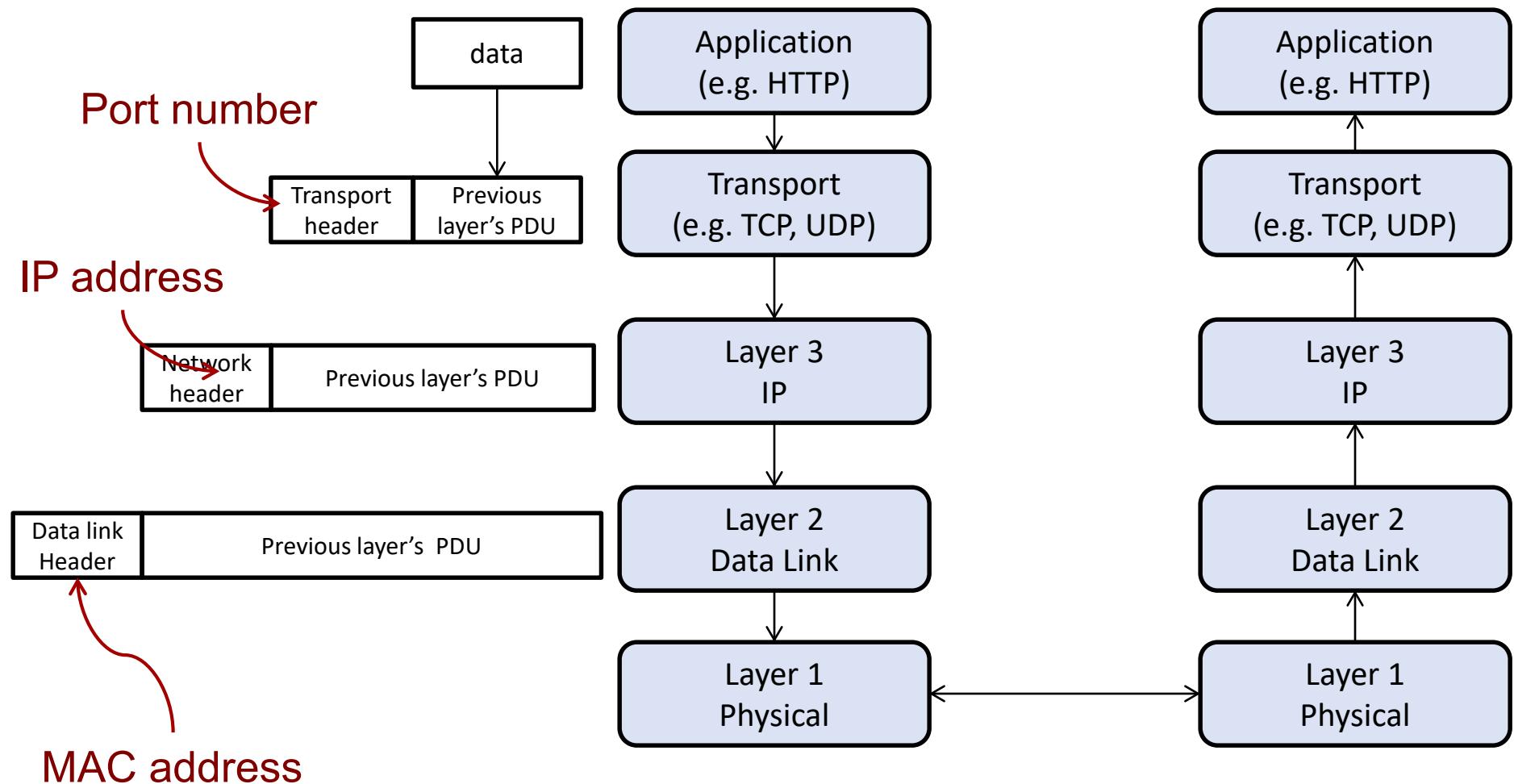
IP address
132.127.12.3

MAC address
01-02-03-04-05-06



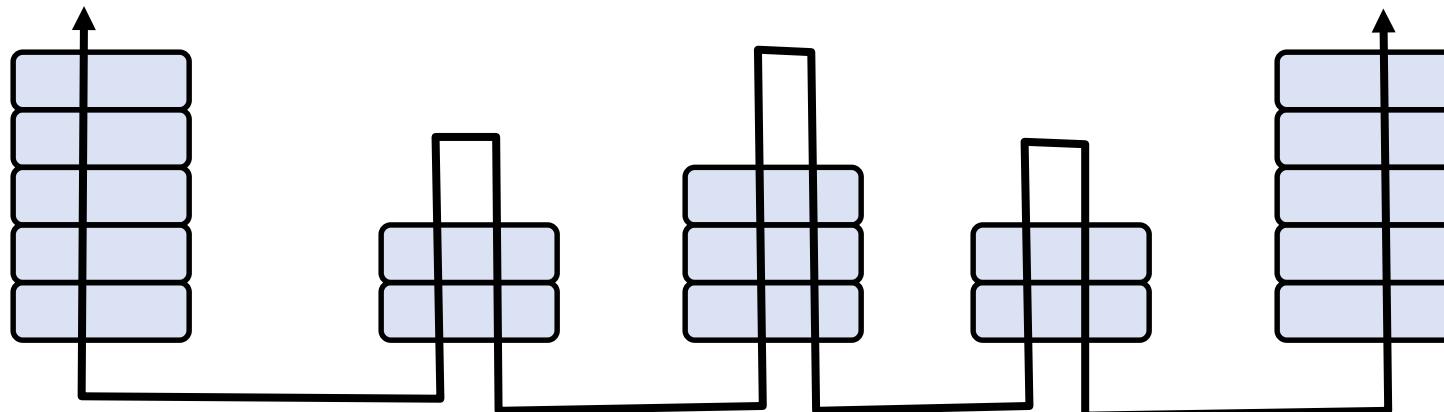
Note: MAC (medium access control) is *not* to be confused with crypto's MAC

Addressing at Various Layers



Multiple Hops from Sender to Destination

- Note that data may go through **multiple hops**
- Can you guess each device type in the diagram below?
- Some networking devices: *router, switch, hub, repeater*



Security Weakness

- The original Internet design **does not** take into account of *intentional* attacks
- Possible threats (from Lecture 1):
 - **Interception**: unauthorized viewing
 - **Modification**: unauthorized change
 - **Fabrication**: unauthorized creation
 - **Interruption**: preventing authorized access
- Attacker at any layer can modify the **data and the header**:
 - Consider the **source IP address** in the IP header, which indicates the sender address
 - Without any protection mechanisms, an attacker can easily send a packet with **spoofed** “source” IP address

5.2 Name Resolution and Attacks

Naming Schemes and Resolution

- Each peer entity has a *name*
- On a single node, at different layer, the name can be **different**

www.comp.nus.edu.sg

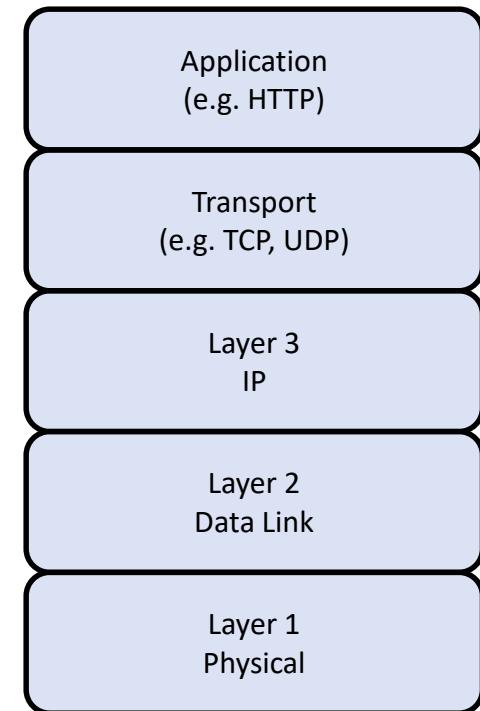
Domain name

132.127.12.3

IP address

01-02-03-04-05-06

MAC address



Naming Schemes and Resolution

- When a peer entity uses the virtual connection in the layer below, it needs to find out the corresponding ***name mapping***
- Example: finding the IP address of a domain name
- Protocols that perform name mappings are known as “***resolution***” protocols
- Many initial design of resolution protocols **didn't** take security into account, and thus easy for attackers to **manipulate the outcome**

Resolution Protocols

- Domain Name System (DNS):
 - Maps **domain name** to **IP address**
 - A hierarchical decentralized naming system
 - An attacker can target the association of domain name with IP address

In this module, we only consider a *basic* type of DNS attack

- Address Resolution Protocol (ARP):
 - Associate/map **IP address** (logical address) with/to **MAC address** (physical address)
 - Use a *broadcast mechanism* on a local network
 - An attacker **on the local network** can target the association

DNS (Domain Name System)

- Given a domain name (e.g. www.comp.nus.edu.sg), its IP address can be found by either looking up a locally stored **host table**, or by **querying a DNS server**. The process is known as ***name resolution***.
- The entity (a.k.a client) that initiates the query is called the ***resolver***
- If the address is found, we say that the domain name is ***resolved***

```
$ nslookup www.comp.nus.edu.sg
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
www.comp.nus.edu.sg canonical name =
www0.comp.nus.edu.sg.
Name:        www0.comp.nus.edu.sg
Address:    137.132.80.57

$
```

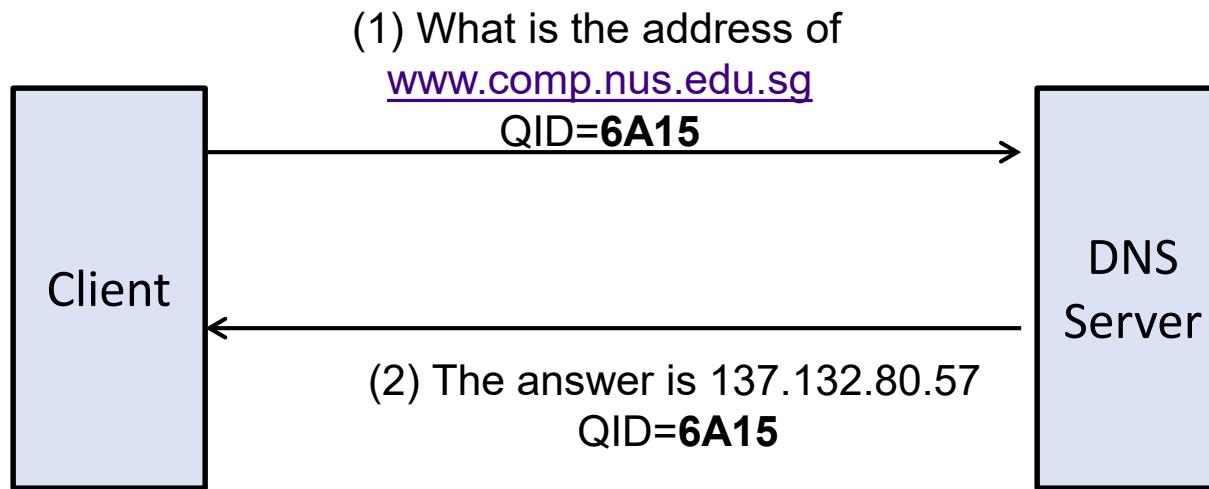
The domain name to look up

Address of the DNS server

Result of the query

(Lightweight) Authentication of DNS Query

- Each query contains a 16-bit number, known as **Query ID (QID)**
- The response from the name server must also contain a QID
- If the QID in the response doesn't match the QID in the query, the client rejects the answer
- Note that **no** encryption/MAC is involved



Remark: In the original design consideration, the QID is probably *not* meant for authentication, but as an efficient way to match multiple queries

Local DNS Attack Scenario

Alice:

- is using a café's free/open (without protection) WiFi to surf the web
- wants to visit the webpage www.comp.nus.edu.sg
- types the domain name into the browser's address bar
- **Alice's browser:**
 - makes a query to a DNS server to determine the IP address
 - then connects to the IP address (after the browser obtains the IP address)

Local DNS Attack Scenario

Active Attacker (Mallory):

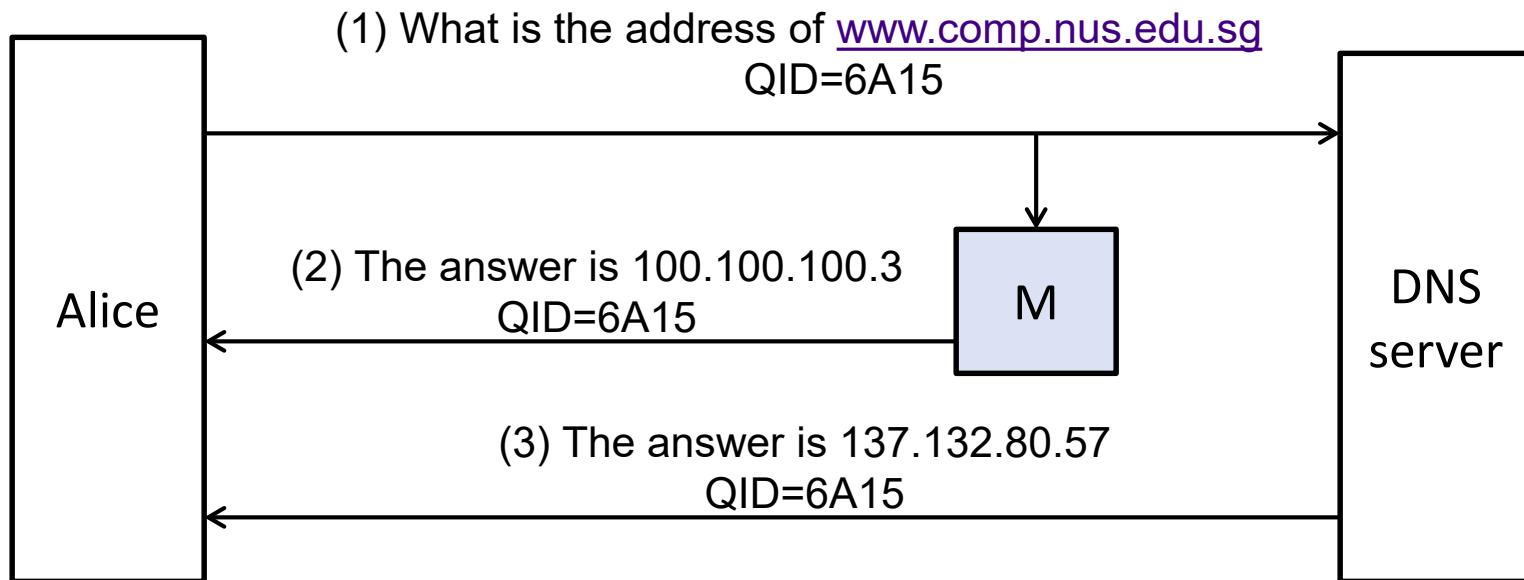
- We consider an attacker at the *physical layer*:
for example, he/she can be another person in the café
- Since the WiFi is not protected, the attacker can:
 - Sniff data from the communication channel
 - Spoof data into the communication channel
- Attacker, however, can't remove/modify data already sent by Alice
- Attacker also owns a Web server (e.g. with IP address 100.100.100.3), which is a spoofed SoC website

The Attack

(See [PF] page 409)

- (1) Alice asks for the address
- (2) Mallory sniffs and knows about it.
She quickly spoofs a reply with the same QID.
- (3) DNS server also sends a reply.
Since Mallory is closer to Alice,
Mallory's reply is likely to reach Alice first.

Alice takes the *first* reply as answer, and connects to 100.100.100.3.



Some Remarks

- DNS operates at the **application layer**.
- Although the attacker is at the physical layer, for ease of analysis, we can assume that the attacker is just below the application layer. That is, there exists some virtual connection that can send the message across.
Hence, the previous slide doesn't mention about the MAC and IP addresses, etc., of the DNS server.
- The DNS is an important component as it resolves the domain name. Hence, an DNS server can be the "**single-point-of-failure**" for the network.
- A DoS attacks, instead of attacking a Web server, could attack the **DNS server** instead.
E.g. see attack on WikiLeaks (See [PF6.5] pg. 414, [PF] pg. 485),
StarHub attack 2016 (next slide).

Recent DNS Attacks

The screenshot shows a news article from www.channelnewsasia.com/news/singapore/broadband-service-outages-due-to-ddos-attacks-starhub. The article title is "Broadband service outages due to DDoS attacks: StarHub". It was posted on 25 Oct 2016 at 15:36 and updated on 25 Oct 2016 at 23:07. The article discusses two recent broadband service outages that hit StarHub, which were confirmed to be intentional and likely malicious attacks on its servers. The text also quotes StarHub's media statement regarding the inspection of network logs and the confirmation of DDoS attacks on their domain name servers (DNS). The screenshot includes social sharing icons (Facebook, Twitter, LinkedIn, Email, More), a print icon, and font size/contrast controls.

Broadband service outages due to DDoS attacks: StarHub

Posted 25 Oct 2016 15:36 Updated 25 Oct 2016 23:07

719 More

SINGAPORE: The two recent broadband service outages that hit StarHub were the result of "intentional and likely malicious attacks" on its servers, the telco confirmed on Tuesday (Oct 25), adding that the attacks were "unprecedented in scale, nature and complexity".

In a media statement, StarHub said: "We have completed inspecting and analysing network logs from the home broadband incidents on Oct 22 and Oct 24 and we are now able to confirm that we had experienced intentional and likely malicious distributed denial-of-service (DDoS) attacks on our domain name servers (DNS).

"These two recent attacks that we experienced were unprecedented in scale, nature and complexity," it said.

Starhub said that the DDoS attacks caused temporary web connection issue for some of its home broadband customers. "On both occasions, we mitigated the attacks by filtering unwanted traffic and increasing our DNS capacity, and restored service within two hours. No impact was observed

Channel News Asia,
25 Oct 2016

5.3 Denial of Service Attacks

DOS Attacks

- **Availability:** the property of being accessible and usable upon demand by an authorized entity
- **Denial of service (DoS):**
 - The prevention of authorized access to resources or the delaying of time-critical operations
 - An attack on availability
- **Types of DoS attacks:**

| | Stopping Service | Exhausting Resources |
|----------------------|---|---|
| Local Attack | <ul style="list-style-type: none">• Process killing• Process crashing• System reconfiguring | <ul style="list-style-type: none">• Spawning processes• Filling up file system |
| Remote Attack | Sending malformed packet attacks | <i>Packet flooding</i> |

DoS Attack Types

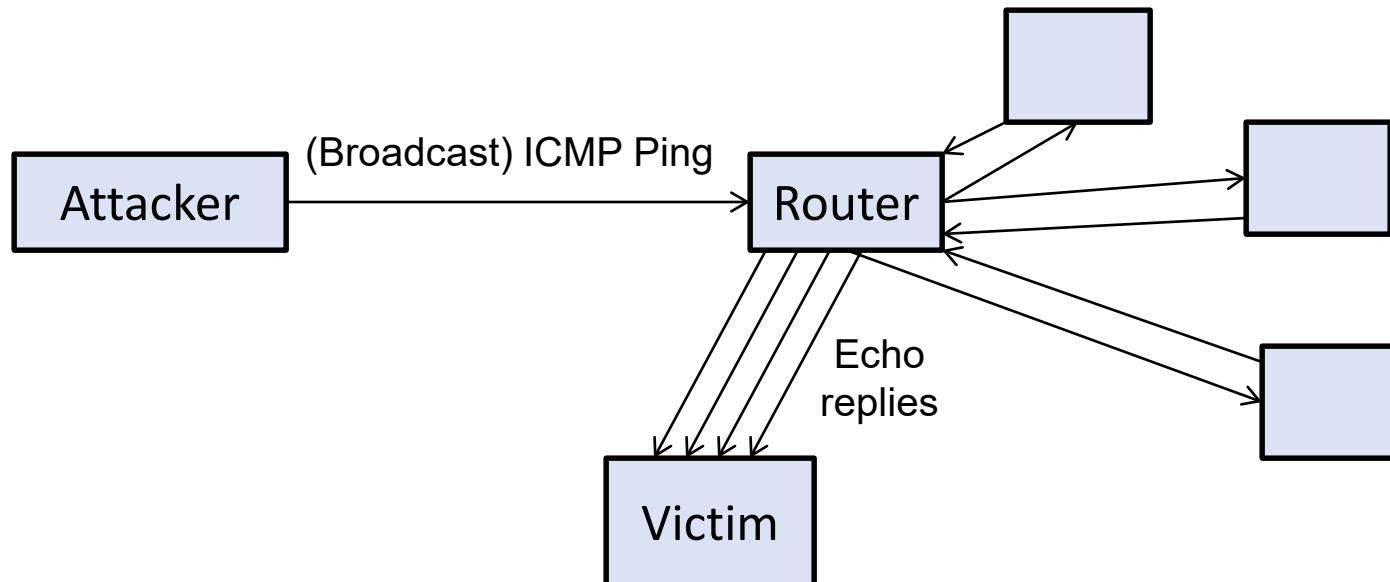
- Local attacks can be more easily tracked
- Sending **malformed attack** remotely does not usually work on updated OSes
- **Packet flooding** attacks:
 - Many effective DOS attacks simply remotely **flood** the victims with overwhelming requests/data
 - The attacker can *amplify* small traffic to obtain large traffic, typically by using available **public servers** (Internet infrastructure), such as DNS, NTP, CharGen

ICMP/Smurf Flood Attack

[PF] page 404

- (1) An attacker sends the “**ICMP PING**” request to a router, instructing the router to **broadcast** this request. The request’ source IP address is spoofed with the victim IP address.
- (2) The router broadcasts this request.
- (3) Each entity who has received this request, replies to it by sending an “**Echo reply**” to the source, which is the victim

The victim is thus overwhelmed with “**Echo reply**” from the entire network. The attacker takes advantage of the ***amplification*** effect.



ICMP/Smurf Flood Attack: Preventive Measures

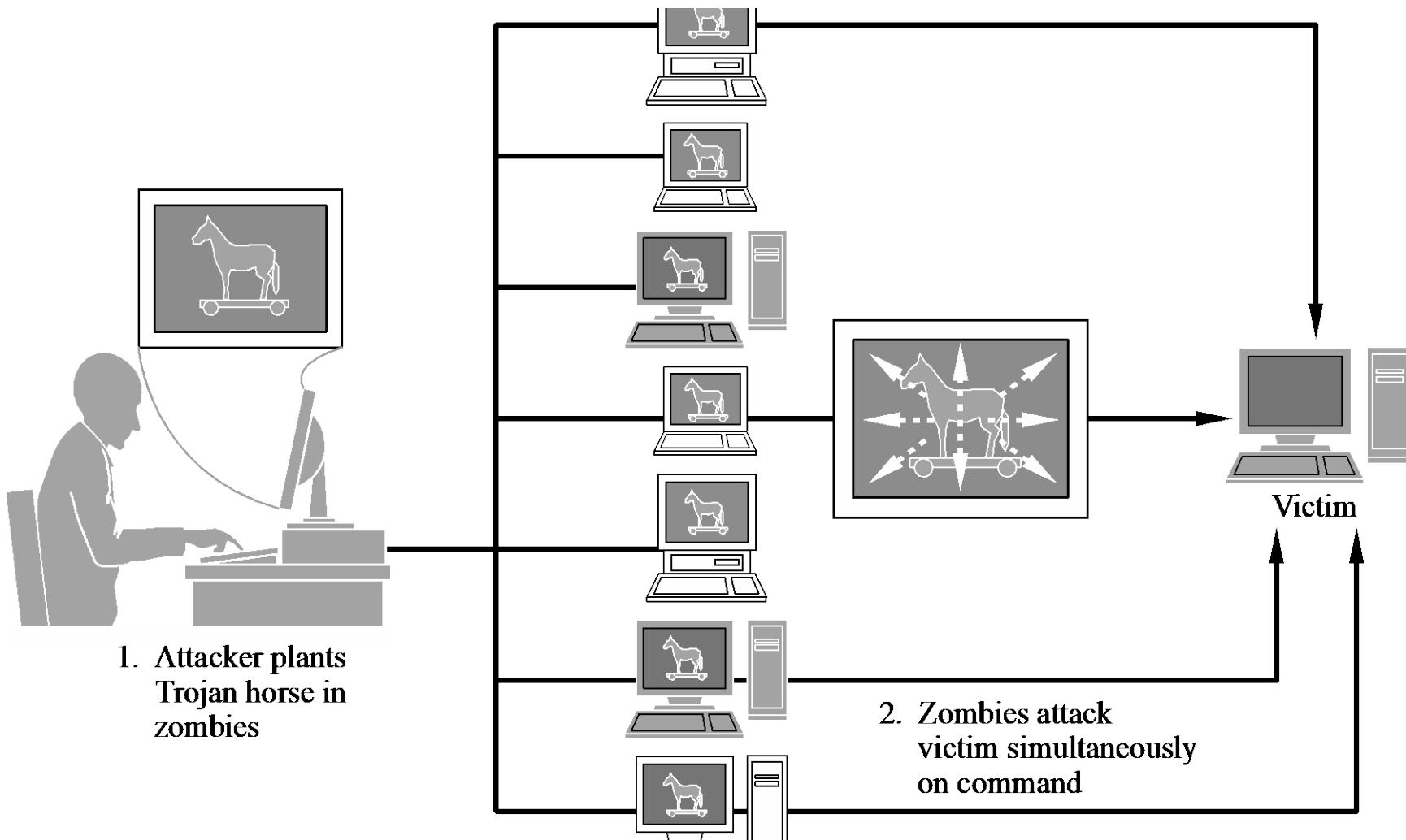
- Is this attack technique still effective?
- *No!*
- Why not?
- Most routers are now configured *not* to broadcast the requests
- To prevent the attack, this measure simply *disables* a feature that was previously thought to be useful

See: IP broadcasting, http://en.wikipedia.org/wiki/Broadcast_address

Example of Application-Layer DoS Attack (HTTP Get)

- Simply flood a Web server with **HTTP requests**
- Example: MyDoom worm, which targeted SCO's website.
Attacks started on Feb 12, 2004.
This is after the SCO's legal actions and public statements
against Linux.
- For this attack to be effective, **a large number** of attackers
are required.
(Since each attacker can send requests at a low rate only).
- When DoS is carried out by large number of attackers,
this is called **Distributed Denial of Service (DDoS)**.

Distributed Denial of Service (DDoS)



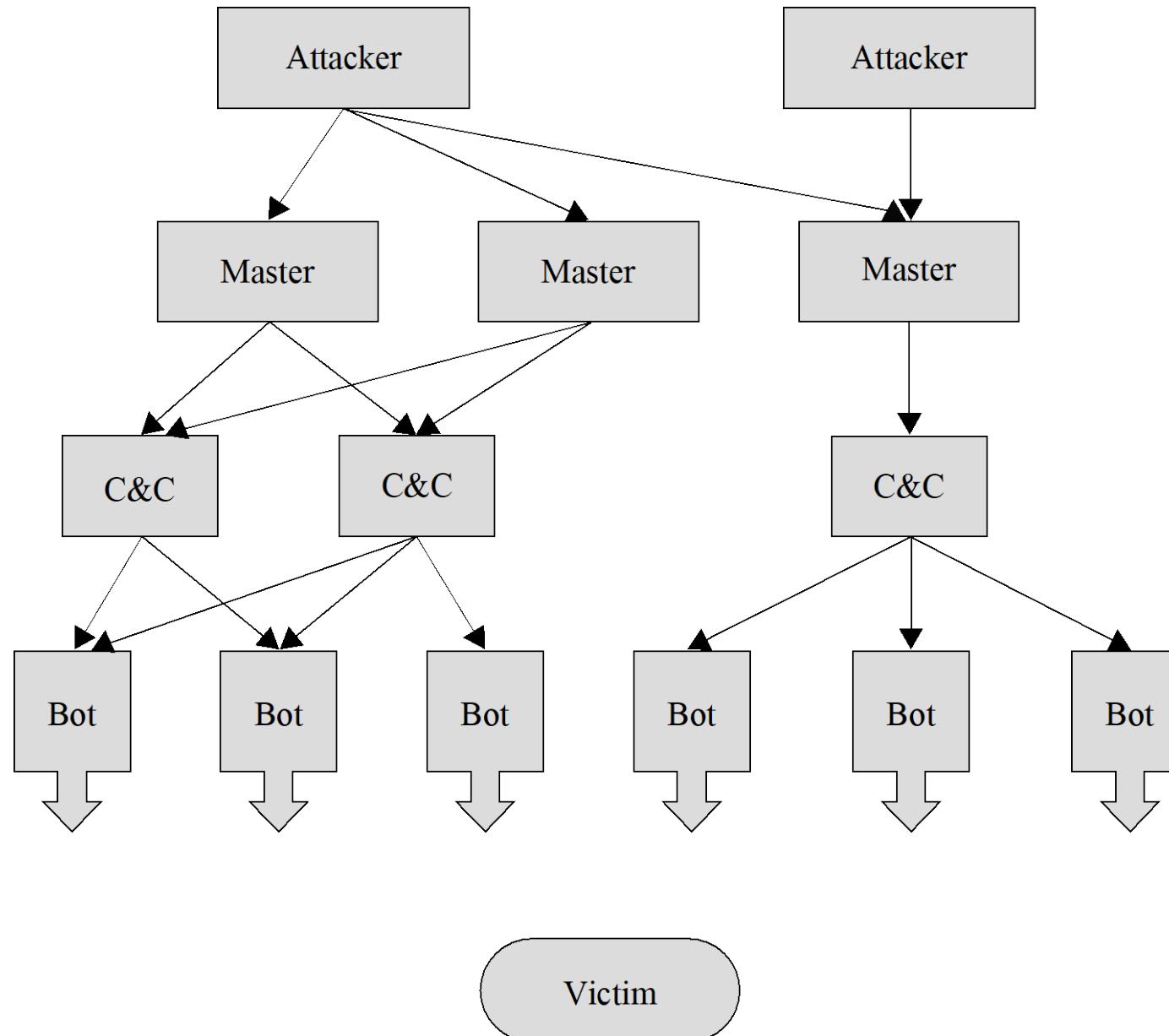
From *Security in Computing, Fifth Edition*, by Charles P. Pfleeger, et al. (ISBN: 9780134085043). Copyright 2015 by Pearson Education, Inc. All rights reserved.

Botnet

- A **bot** (aka *zombie*) is a compromised machine
- A **botnet** (aka *zombie army*) is a large collection of connected bots, communicating via covert channels
- A botnet has a **command-and-control** mechanism, and thus can be controlled by an individual to carry out DDOS
- Possible usages of a botnet:
 - DDoS flooding, vulnerability scanning, anonymizing HTTP proxy, email address harvesting, cipher breaking!
- See Wiki for the size of known botnets:
<http://en.wikipedia.org/wiki/Botnet>

Question: Why **covert** channels are used by a botnet?

Botnet



IoT Devices: a New Trend



TECHNOLOGY

M1 launches commercial IoT network in boost to Singapore's Smart Nation drive



All Sections



By Kevin Kwang
@KevinKwangCNA

07 Aug 2017 06:20PM
(Updated: 07 Aug 2017 09:28PM)



Smart rubbish bins enabled with sensors will trigger an alert to cleaners to clear them after a certain level is met. (Photo: M1)

Channel News Asia,
7 Aug 2017

5.4 Useful Tools

Wireshark (a Packets Analyzer)

- **Wireshark:** a popular free open-source network packet analyzer, <https://www.wireshark.org/>.
- What does Wireshark exactly capture?

Generally performs capturing at the link layer.

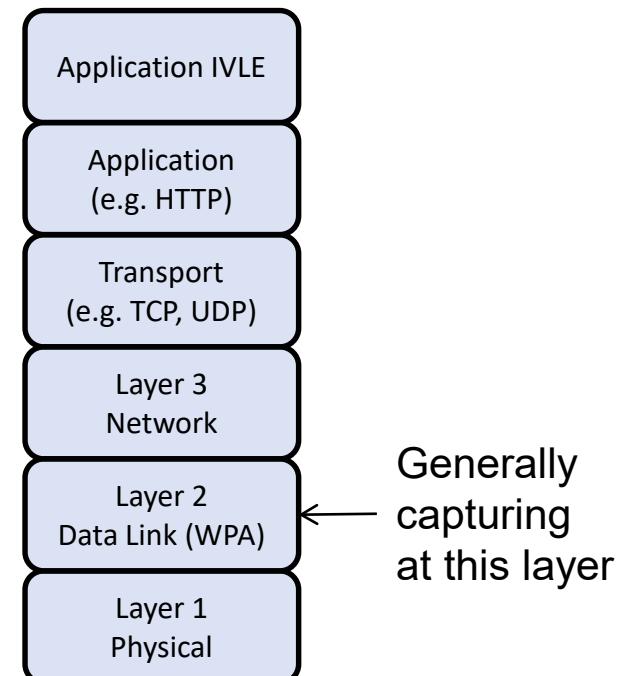
This depends on OS and hardware.

Essentially captures “interactions” between the OS and the network card driver.

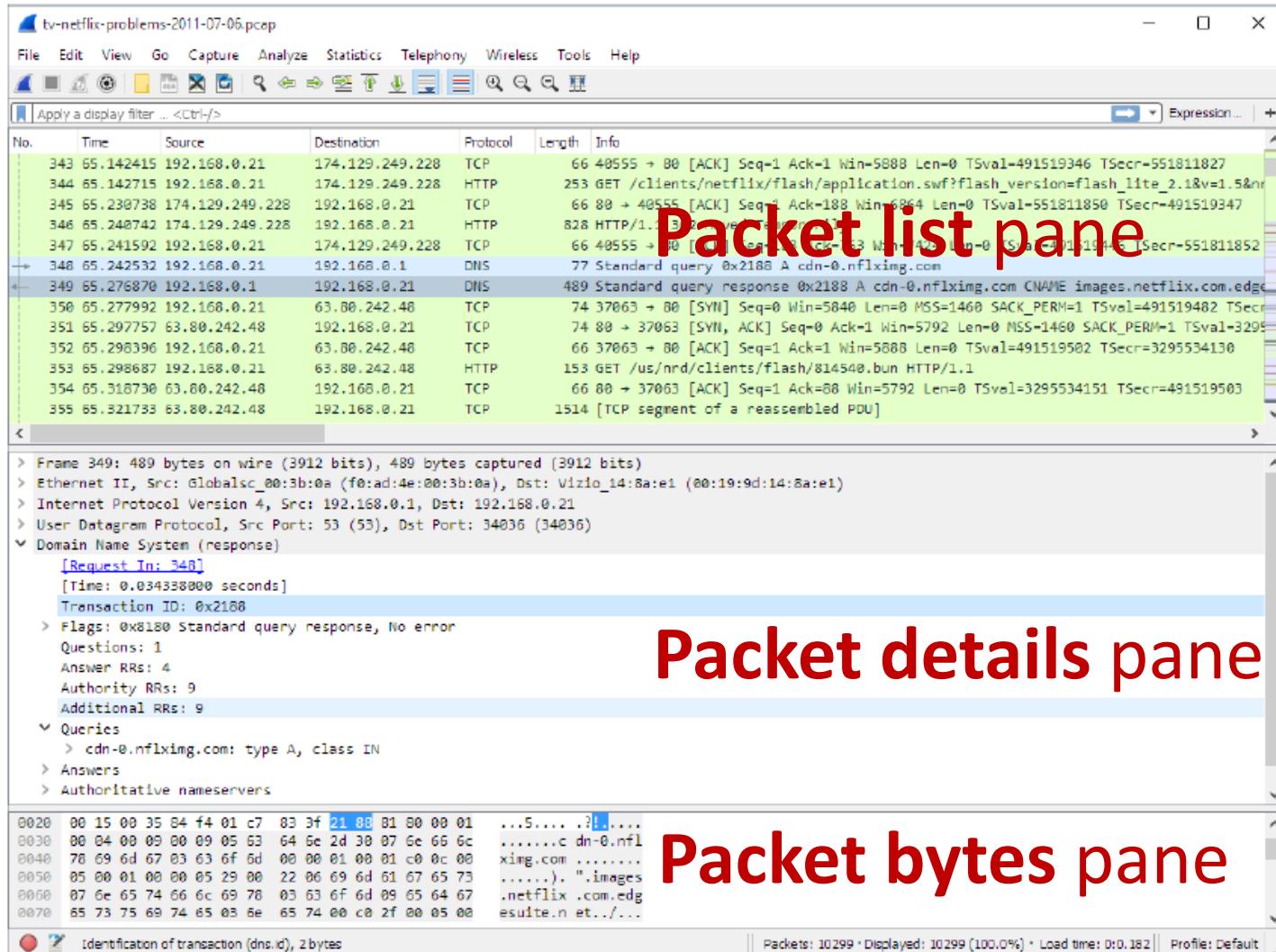
See the following FAQ for more info:

<https://ask.wireshark.org/questions/22956/where-exactly-wireshark-does-captures-packets>

(**Demo:** Wireshark)

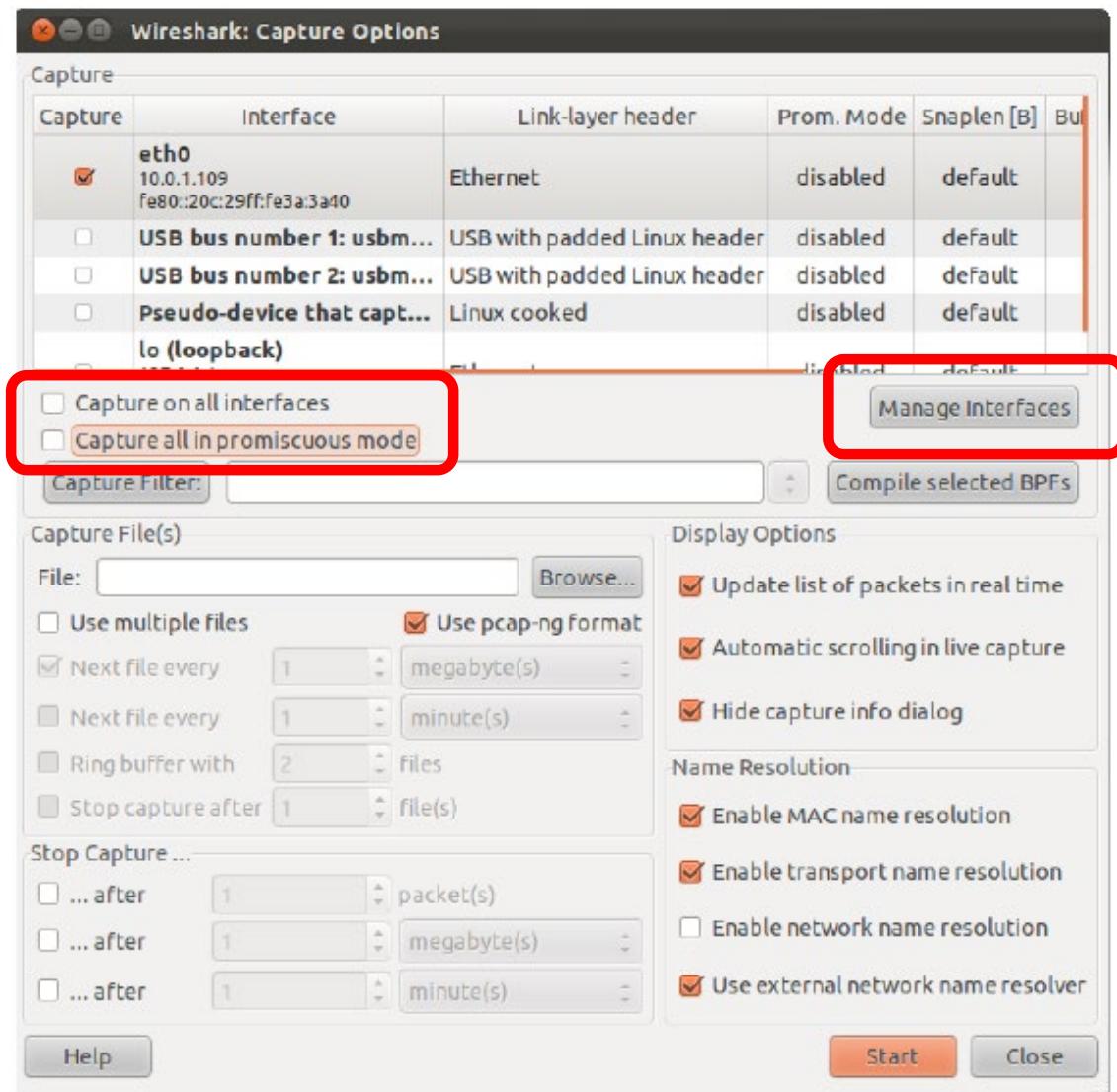


Wireshark (a Packets Analyzer)



For Wireshark usage, see: Wireshark User's Guide,
https://www.wireshark.org/docs/wsug_html/

Sniffing using Wireshark: Capture Options



Sniffing using Wireshark: Popup Menu

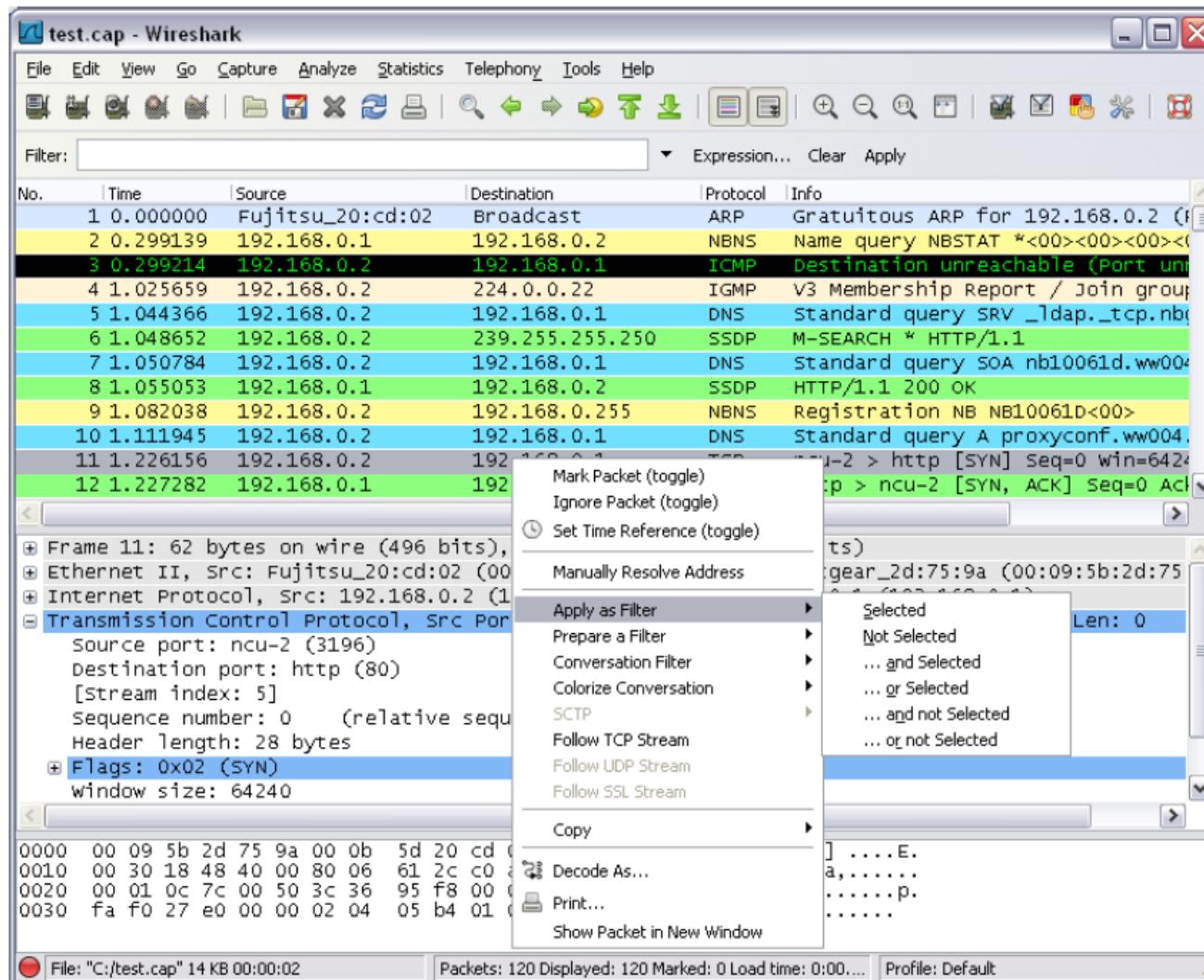
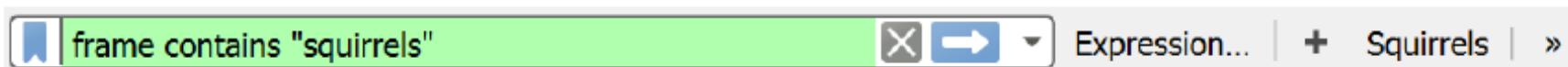


Figure 59. Pop-up menu of the “Packet List” pane

Sniffing using Wireshark: Filter

You need to specify a good (while-viewing) filter:



Filter comparison operators

Table 20. Display Filter comparison operators

| English | C-like | Description and example |
|-------------|--------|--|
| eq | == | Equal. <code>ip.src==10.0.0.5</code> |
| ne | != | Not equal. <code>ip.src!=10.0.0.5</code> |
| gt | > | Greater than. <code>frame.len > 10</code> |
| lt | < | Less than. <code>frame.len < 128</code> |
| ge | >= | Greater than or equal to. <code>frame.len ge 0x100</code> |
| le | <= | Less than or equal to. <code>frame.len <= 0x20</code> |
| contains | | Protocol, field or slice contains a value. <code>sip.To contains "a1762"</code> |
| matches | ~ | Protocol or text field match Perl regular expression. <code>http.host matches "acme\.(org com net)"</code> |
| bitwise_and | & | Compare bit field value. <code>tcp.flags & 0x02</code> |

Sniffing using Wireshark: Follow TCP Stream

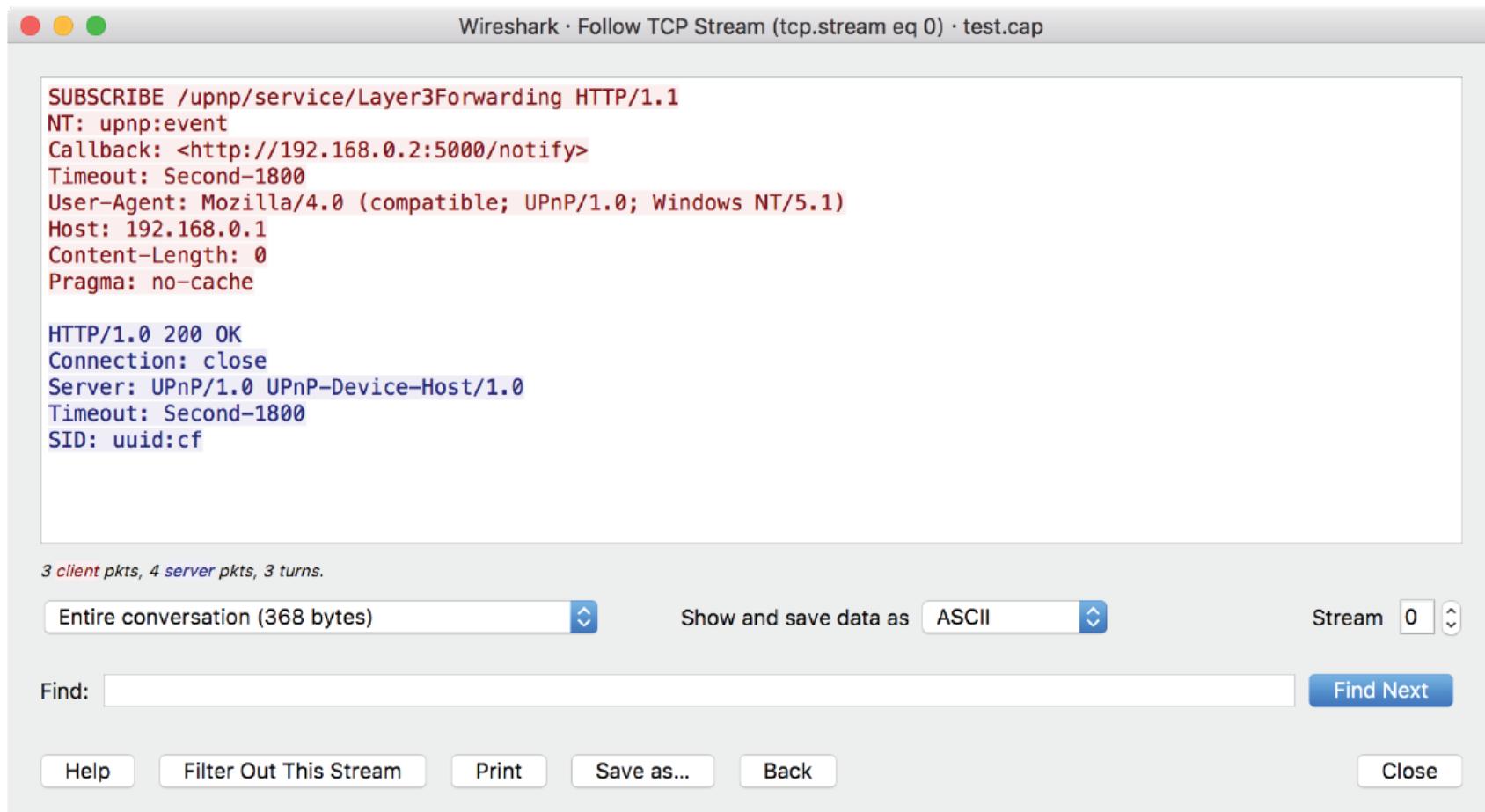
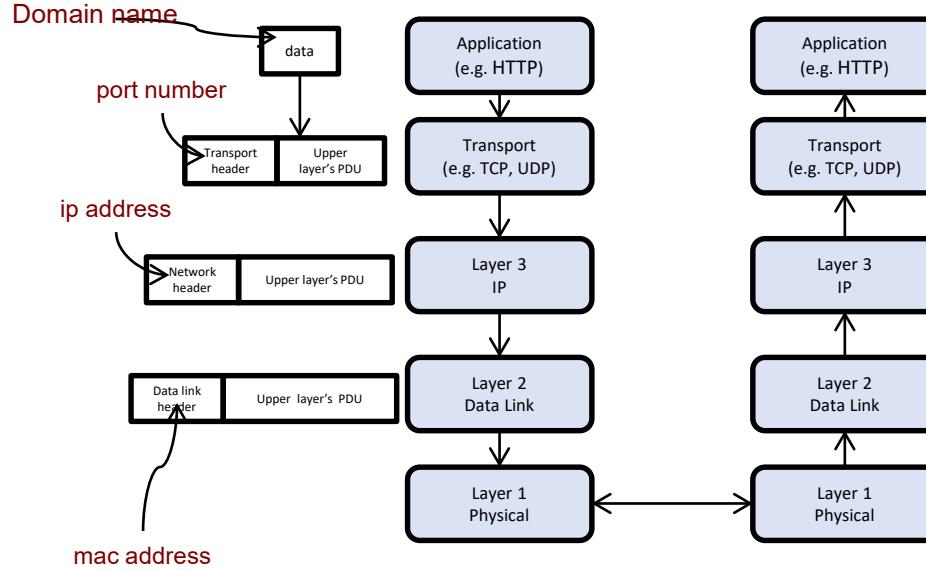


Figure 67. The “Follow TCP Stream” dialog box

Source: Wireshark User's Guide

Nmap (Port Scanner)

- What is a port?



- When a server receives an incoming packet, it will decide **which application process** to handle that packet based on the port no
- By saying that a process/service is “*listening*” to a particular port, we mean that the process **is running and ready to process** packets with that particular port no

Nmap (Port Scanner)

- When a port is “*open*”, there exist such a process running in the server
- See well-known port numbers:
https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers#Well-known_ports.
- ***Port scanning***: the process of determining which ports are open on hosts in a network
- Ports are “doors” into each machine, hence port scanning is like **knocking at the doors**
- ***Port scanner***:
 - A tool for performing port scanning
 - Is useful for both attacker and network administrator to scan for vulnerabilities
 - E.g. **Nmap** (very popular!)

Nmap (Port Scanner)

- Nmap is a full featured port-scanning tool:
 - Command-line tool, with GUI frontend
 - Installation:
`sudo apt-get install nmap, zenmap`
 - Usage: `nmap [Scan Type(s)] [Options] {target specification}`
 - Examples:
 - TCP ACK scan (a stealthier scan): `nmap -sA`
 - OS fingerprinting: `nmap -O`
 - Service/version detection: `nmap -sV`

(Demo: Nmap)

Nmap: Sample Output

```
Nmap scan report
192.168.1.1 / somehost.com (online) ping results
address: 192.168.1.1 (ipv4)
hostnames: somehost.com (user)
The 83 ports scanned but not shown below are in state: closed
Port      State       Service Reason      Product Version Extra info
21        tcp        open      ftp      syn-ack    ProFTPD   1.3.1
22        tcp        filtered ssh      no-response
25        tcp        filtered smtp     no-response
80        tcp        open      http     syn-ack    Apache    2.2.3    (CentOS)
106       tcp        open      pop3pw   syn-ack    poppassd
110       tcp        open      pop3     syn-ack    Courier   pop3d
111       tcp        filtered rpcbind  no-response
113       tcp        filtered auth     no-response
143       tcp        open      imap     syn-ack    Courier   Imapd    released
2004      tcp      open      http     syn-ack    Apache    2.2.3    (CentOS)
443       tcp        open      http     syn-ack    Apache    2.2.3    (CentOS)
465       tcp        open      unknown  syn-ack
646       tcp        filtered ldap     no-response
993       tcp        open      imap     syn-ack    Courier   Imapd    released
2004      tcp      open      http     syn-ack
995       tcp        open      nfs      no-response
2049      tcp        filtered nfs      no-response
3306      tcp        open      mysql    syn-ack    MySQL    5.0.45
8443      tcp        open      unknown  syn-ack
34 sec. scanned
1 host(s) scanned
1 host(s) online
0 host(s) offline
```

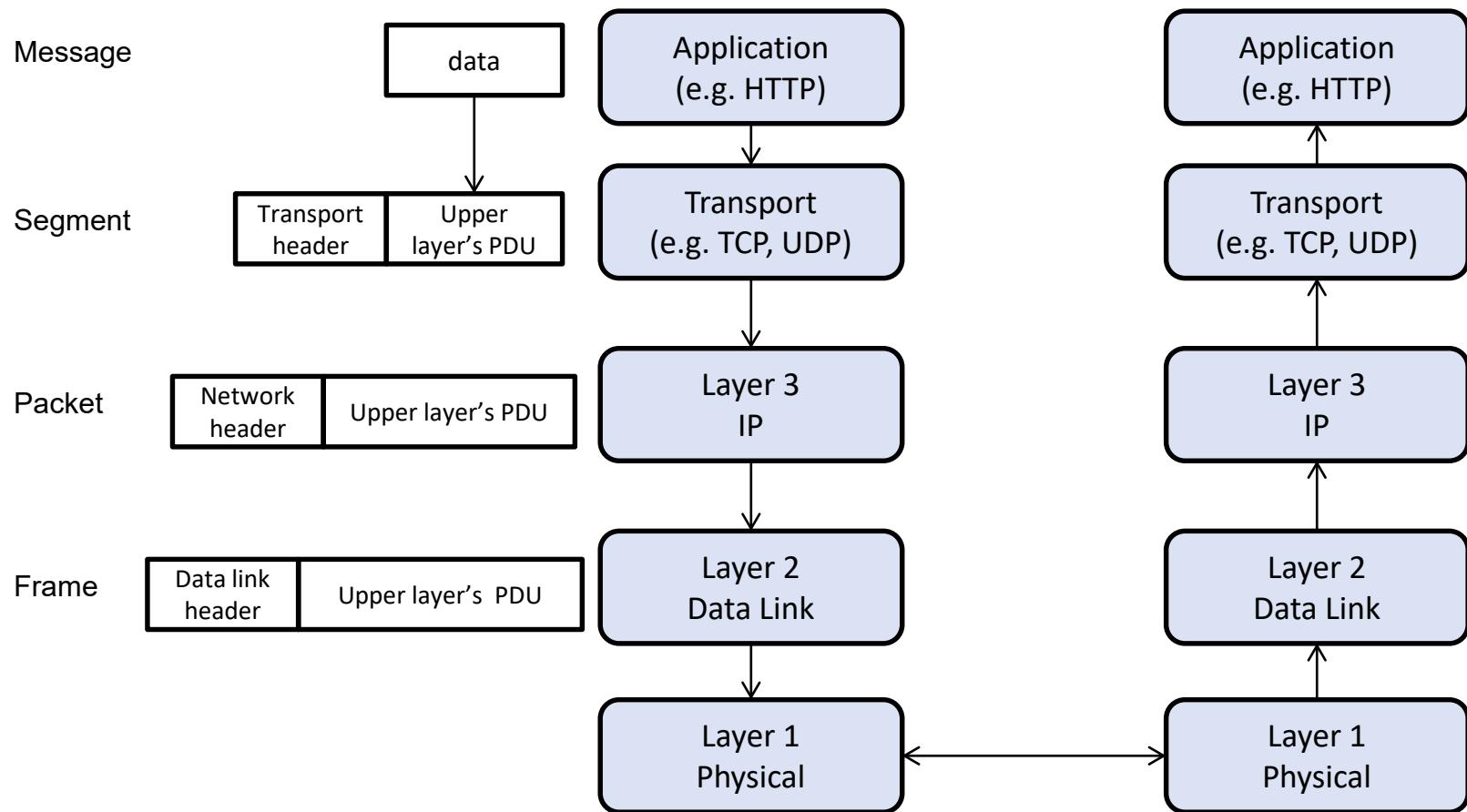
5.5 Protection: Securing the Communication Channel using Cryptography

Cryptography for Securing Network Communication

- Several cryptographic techniques to achieve **confidentiality** (encryption) and **authenticity** (MAC, PKI, strong authentication) over a public communication channel, even if the adversary can sniff & spoof data
- There are **many** security protocols that essentially achieve that, but operates at **different “layers”**
- Prominent **protocols**:
 - TLS/SSL
 - Wi-Fi Protected Access II (WPA2)
 - Internet Protocol Security (IPsec)

Security Protocols at Different Layers

- Recall the network layering shown previously
- TLS/SSL, WPA, IPSEC protect **different** layers



Remarks on Security Protocols and Network Layering

- Very often, when referring to a **security protocol**, we indicate the “**layer**” that the protocol **targets to protect**
- **Complication:** some protections *span across* multiple layers, or do *not* provide full protection of the targeted layer
- When analyzing an **attack**, it is also insightful to figure out at **what layer the attacker resides**
- **Complication:** likewise, some attacks *span across* multiple layers. In such situations, trying hard to pinpoint the layer could sometimes be very confusing

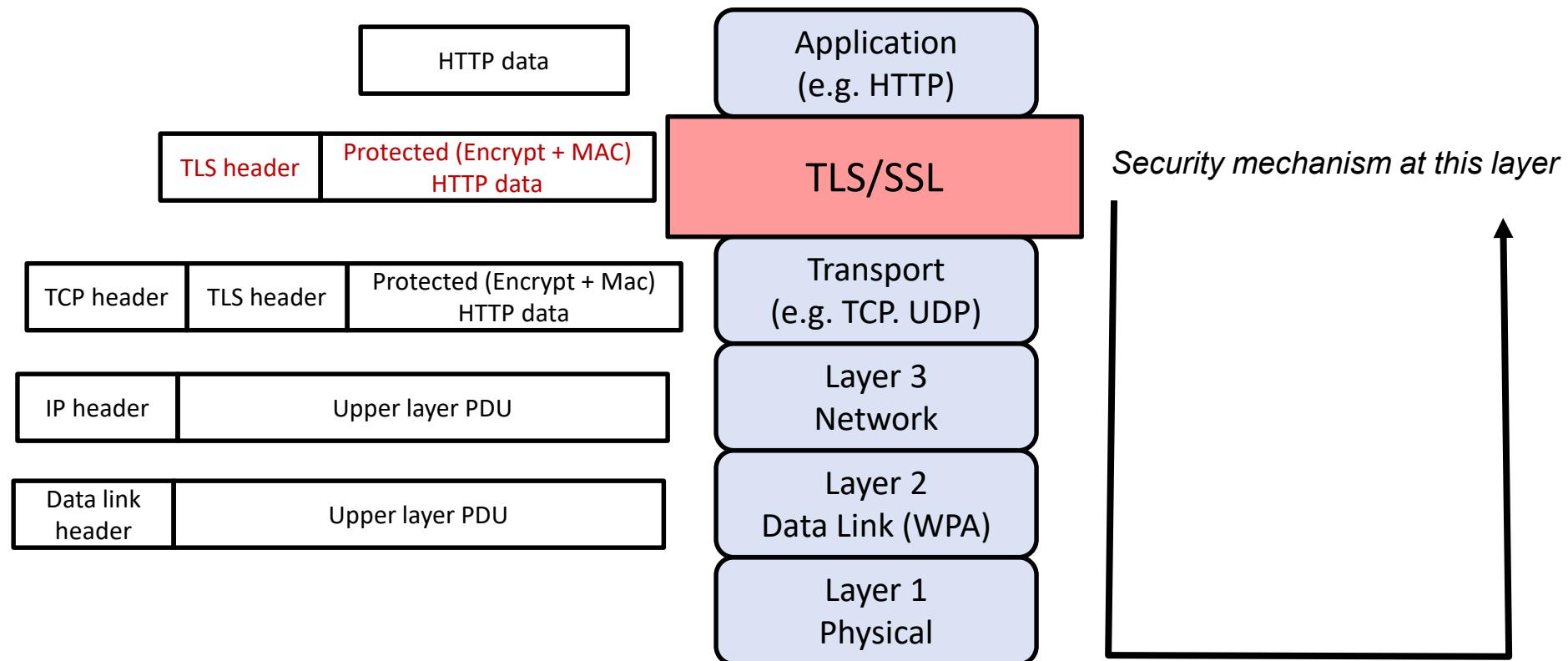
Remark on Security Protocols and Network Layering

- Below are the **general guideline** and example
- A security protocol that **protects layer k** would protect information from **that layer and above against** an attacker sitting at **layer $k-1$ and below**
- **Example:** what happens if an attacker resides at layer 1, and there is a security protocol that protects layer 3?
- What is protected by the security protocol:
the information generated in layer 3 and above
- What is *not* protected:
the information generated in layer 2

1. SSL/TLS

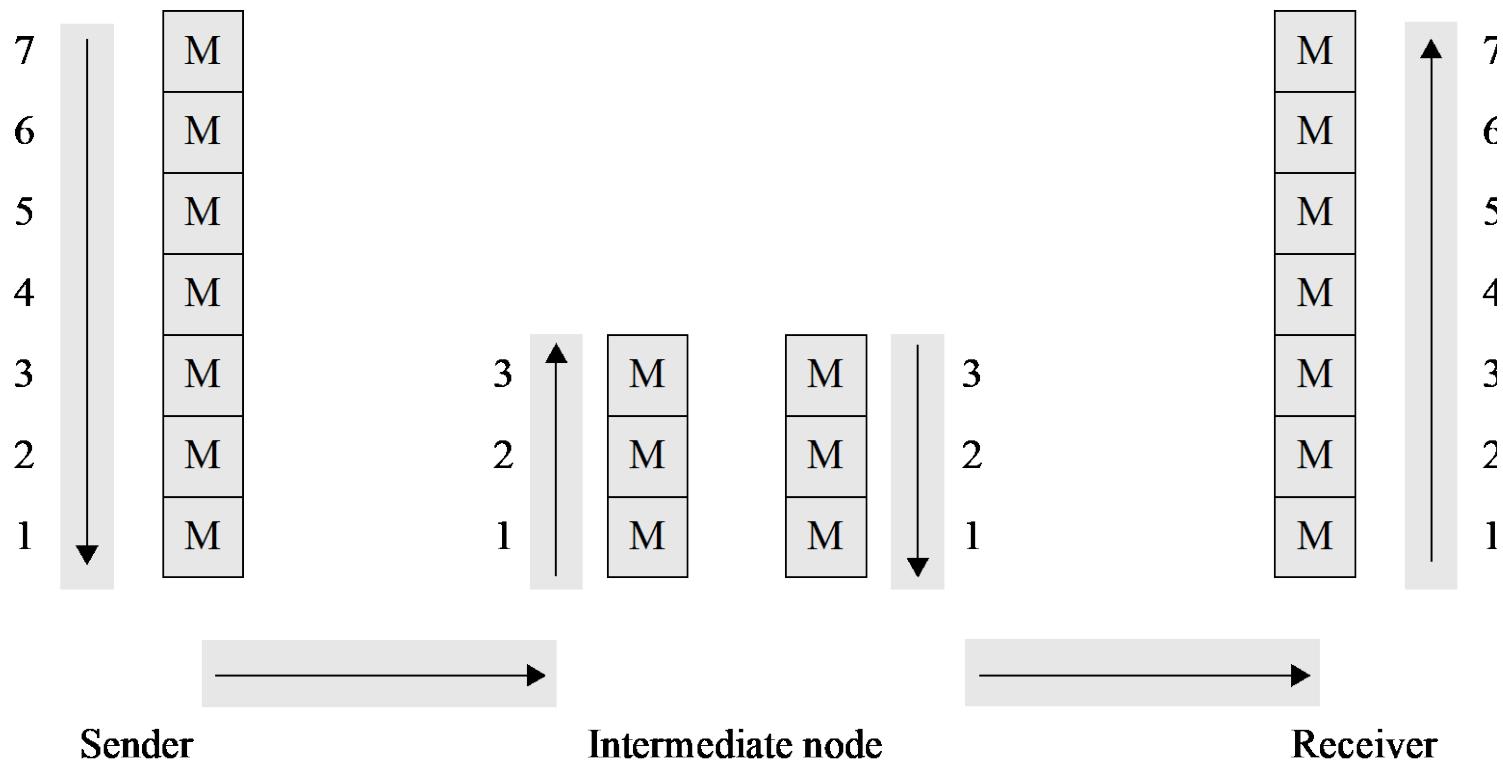
- The SSL/TLS sit on top of **transport layer**
- We can imagine that, when an application (e.g. browser or email agent) wants to send data to the other end point, it first pass the data and the destination IP address to SSL/TLS
- Next, SSL/TLS first “protects” the data using encryption (for confidentiality) and MAC (for authenticity), and then instructs the transport layer to send the protected data
- An **end-to-end encryption** is performed

SSL/TLS Location



The receiver end-point **decrypts** the received data at the corresponding layer

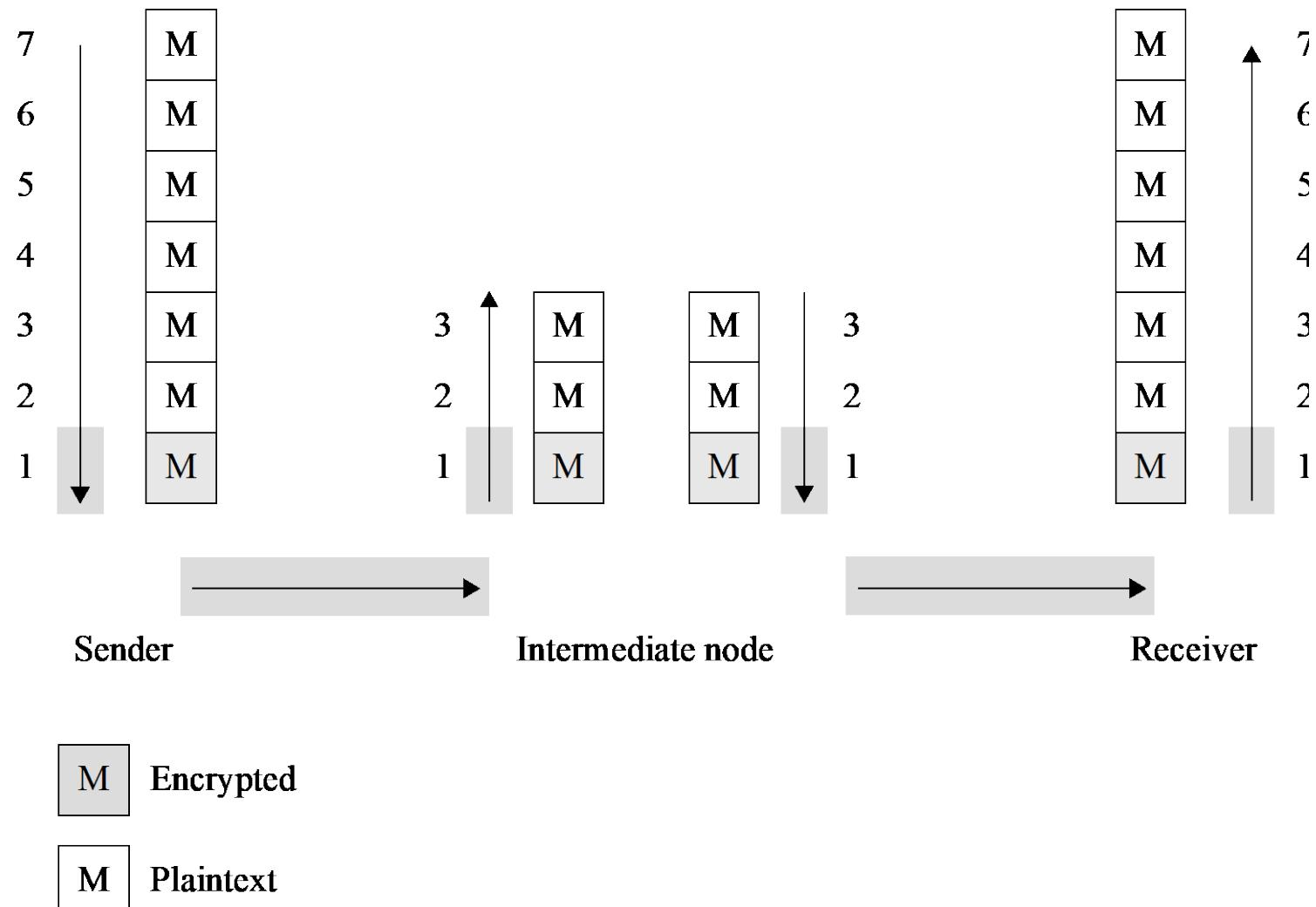
Terminology: End-to-End Encryption



M Encrypted

M Plaintext

Terminology: Link (Hop-by-Hop) Encryption



Sample Usage Scenario

- Alice accesses LumiNUS web application to upload her report `a.pdf` to the LumiNUS server
- Note that LumiNUS uses HTTPS, which in turn employs SSL/TLS

Alice's machine carries the following:

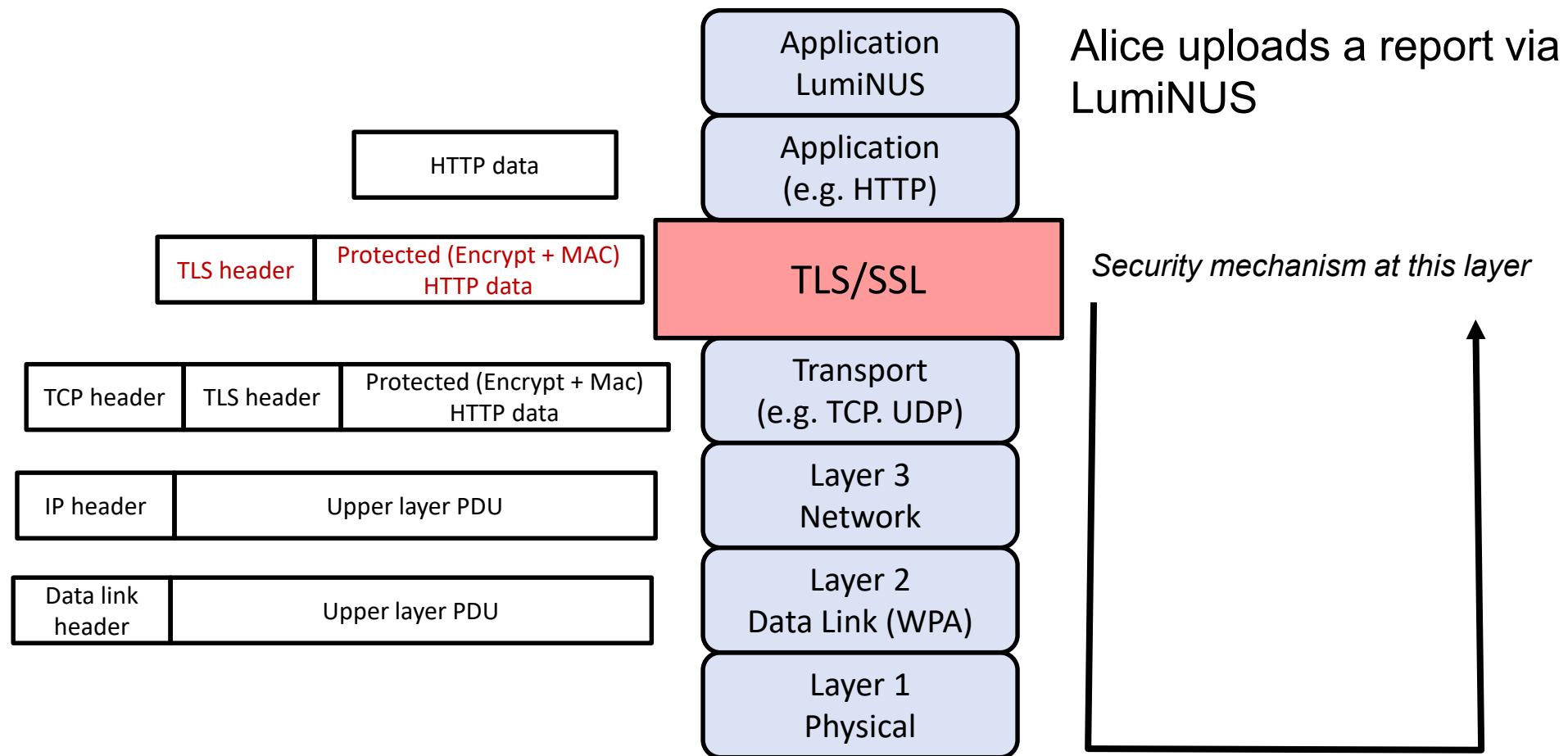
1. The “LumiNUS client” passes the file `a.pdf` to HTTPS, and then to TLS
2. TLS protects the data by encryption and MAC
3. TLS passes the protected data to the transport layer

Sample Usage Scenario

The LumiNUS server carries out the following:

1. The transport layer passes the protected data to TLS
 2. TLS decrypt the data and verify the MAC for integrity
 3. TLS passes the decrypted data to LumiNUS application
-
- ***Remark:*** Many details are omitted in the description.
For instance, the “handshaking”, whereby the two parties establishing the session keys.

Sample Usage Scenario



The receiver end-point **decrypts** the received data at the corresponding layer

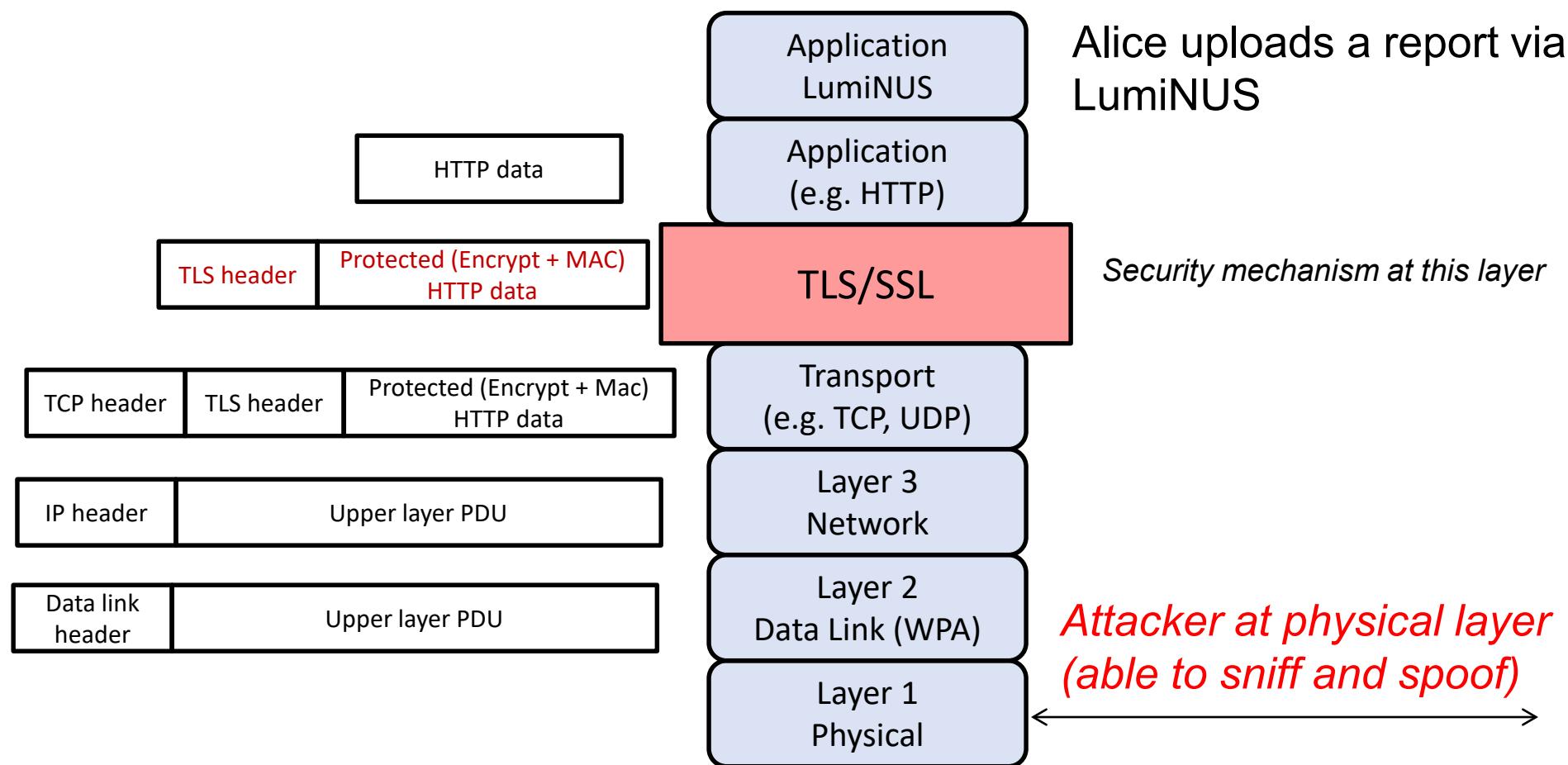
Attack Scenario 1: Attacker at Physical Layer

- Suppose that there is an attacker **at the physical layer**, who can sniff and spoof message at that layer
- **For example**, Alice uploads her report in a cafe using a free/open WiFi (without WPA protection). Hence, anyone in the café has access to the physical layer, and thus can sniff and spoof messages in that layer.

Question: Can the attacker learn:

1. Alice's uploaded report?
2. The fact that Alice is visiting LumiNUS website
(i.e. can the attacker learn the website's IP address)?

Attack Scenario 1: Attacker at Physical Layer



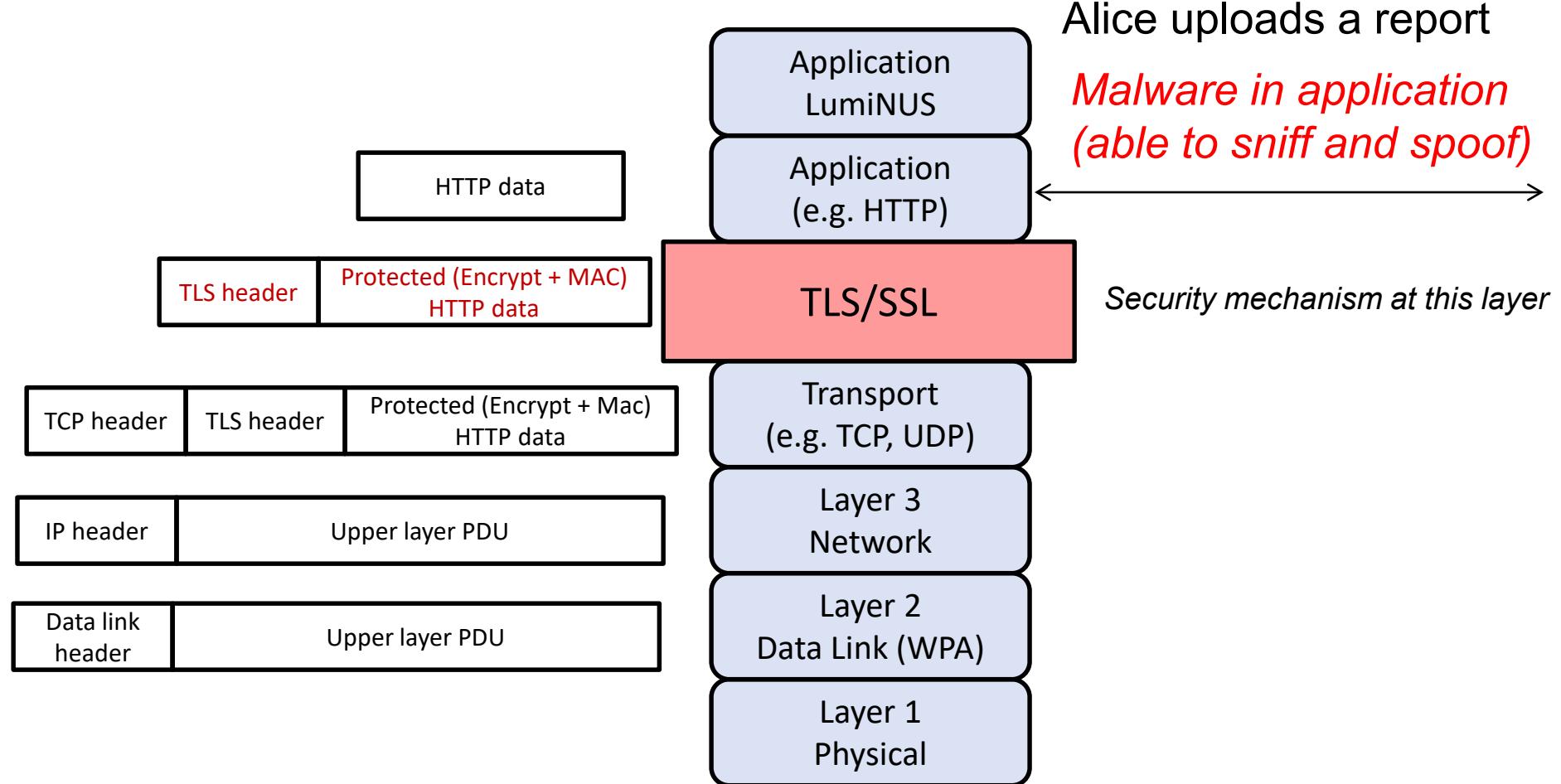
Attack Scenario 2: Attacker at Application Layer

- Suppose that there is an adversary at the **application layer**
- For example, a malicious JavaScript is injected into LumiNUS and being executed by Alice's browser

Question: Can the malicious script learn:

1. Alice's report?
2. *Alice's MAC address?*

Attack Scenario 2: Attacker at Application Layer



2. WPA2

WiFi Protected Access II (WPA2):

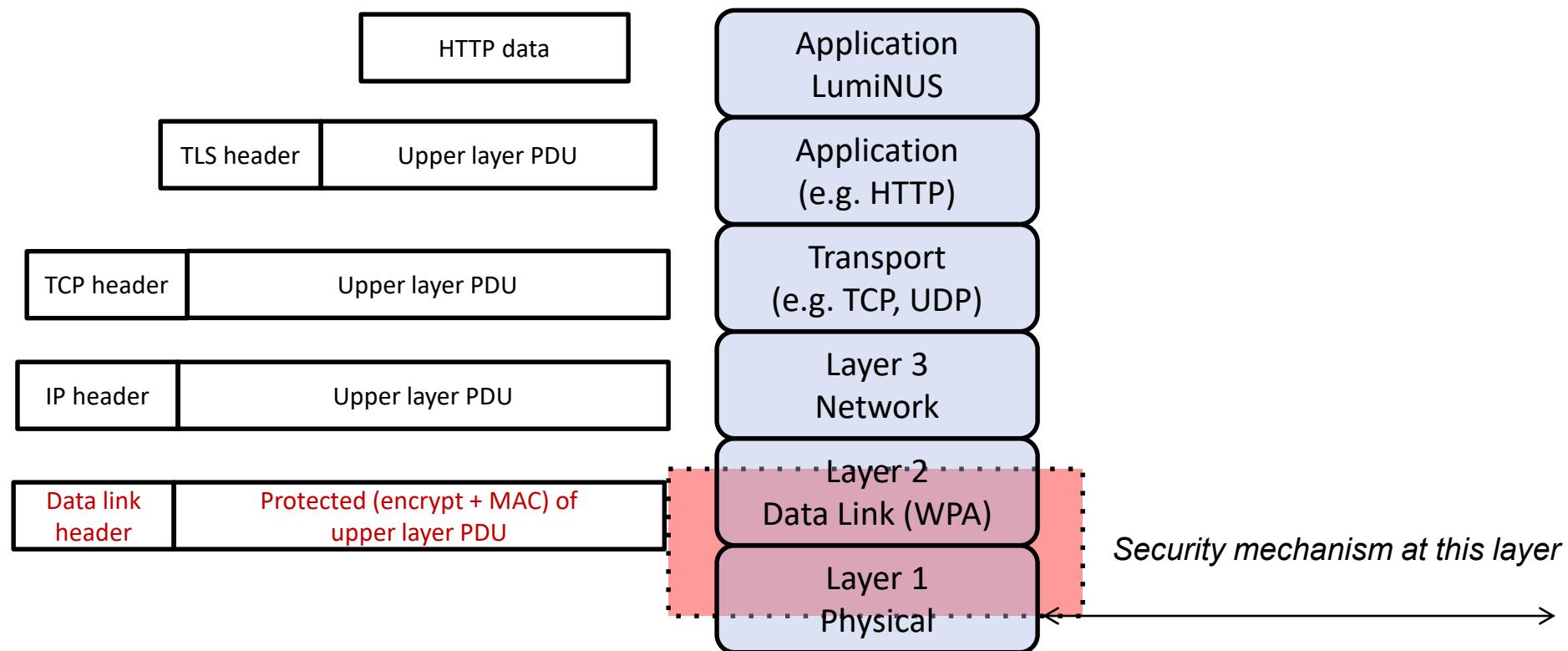
- A popular protocol employed in home WiFi access point
- More secure than WEP (broken), WPA

The protections provided:

- WPA2 provides protection at layer 2 (link) and layer 1 (physical)
- Note: *not* all information in layer 2 are protected

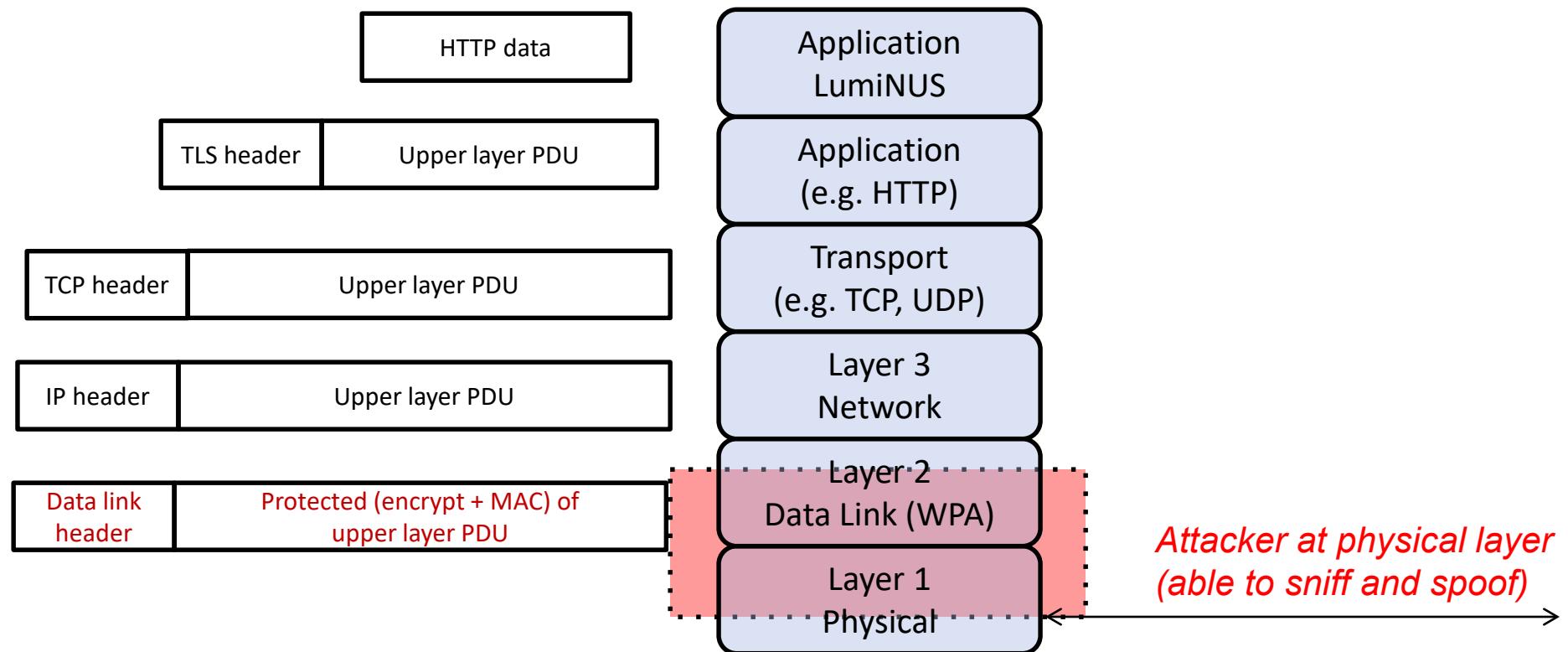
WPA2 and Network Layers

Alice uploads a report



Attacker at Physical Layer

Alice uploads a report

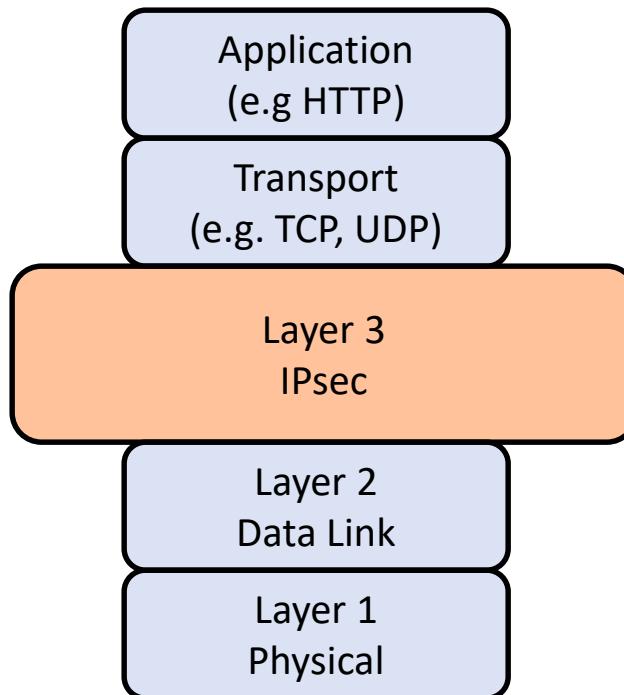


Question: Can the attacker learn:

1. *Alice's report?*
2. *The fact that Alice is visiting LumiNUS website?*
3. *The MAC address (link layer)?* not clear

3. IPsec

- IPsec provides **integrity/authenticity** protection of IP address, but *not* confidentiality
- Hence, attackers are unable to “spoof” the source IP address
- But they can still learn the source and destination IP addresses of the sniffed packets



Remarks on IPsec

IPsec is a mechanism whose goal is *to protect the IP layer*

Its description:

- “Internet Protocol Security (IPsec) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used in protecting data flows between a pair of hosts (*host-to-host*), between a pair of security gateways (*network-to-network*), or between a security gateway and a host (*network-to-host*).^[1]
- Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPsec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.
- IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at Application layer. Hence, only IPsec protects any application traffic over an IP network. Applications can be automatically secured by IPsec at the IP layer.”

-Wiki

Question

Question: Explain the underlined sentences

- “**Internet Protocol Security (IPsec)** is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used in protecting data flows between a pair of hosts (*host-to-host*), between a pair of security gateways (*network-to-network*), or between a security gateway and a host (*network-to-host*).^[1]
- Internet Protocol security (IPsec) uses cryptographic security services to protect communications over Internet Protocol (IP) networks. IPsec supports network-level peer authentication, data origin authentication, data integrity, data confidentiality (encryption), and replay protection.
- IPsec is an end-to-end security scheme operating in the Internet Layer of the Internet Protocol Suite, while some other Internet security systems in widespread use, such as Transport Layer Security (TLS) and Secure Shell (SSH), operate in the upper layers at Application layer. Hence, only IPsec protects any application traffic over an IP network. Applications can be automatically secured by IPsec at the IP layer.”

-Wiki

5.6 Protection: Firewall

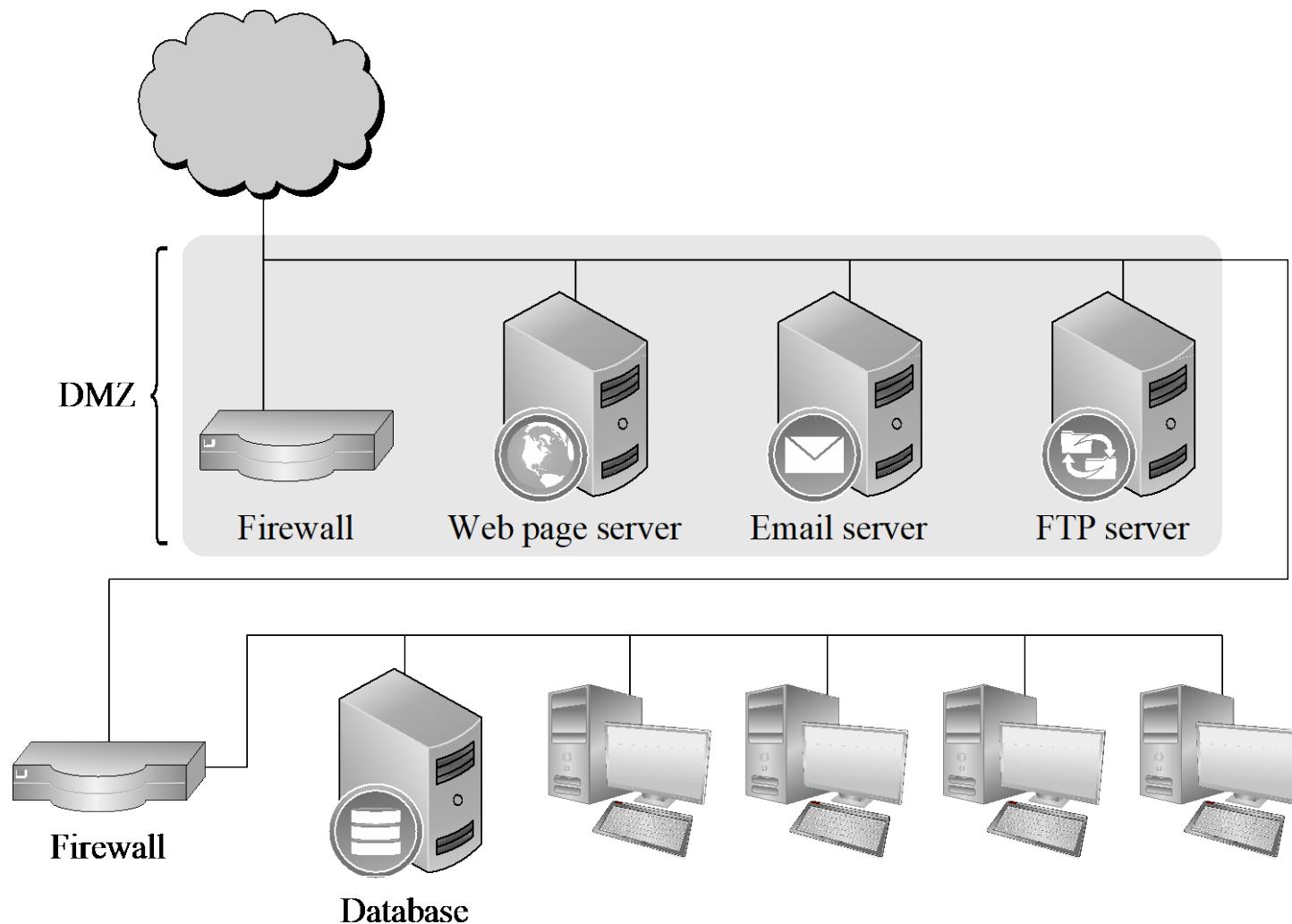
Motivation

- Having SSL/TLS and WPA2 is still *not* sufficient to protect the network:
 - There are concerns of **DOS** that they can't prevent
 - Many services and applications can't be protected by SSL/TLS & WPA2: e.g. **DNS spoofing**
(It is not practical, due to efficiency, to establish SSL/TLS to the DNS server for DNS query)
- There is a need to control **the flow of traffic** between networks, especially between the untrusted public network (Internet) and the trusted internal network

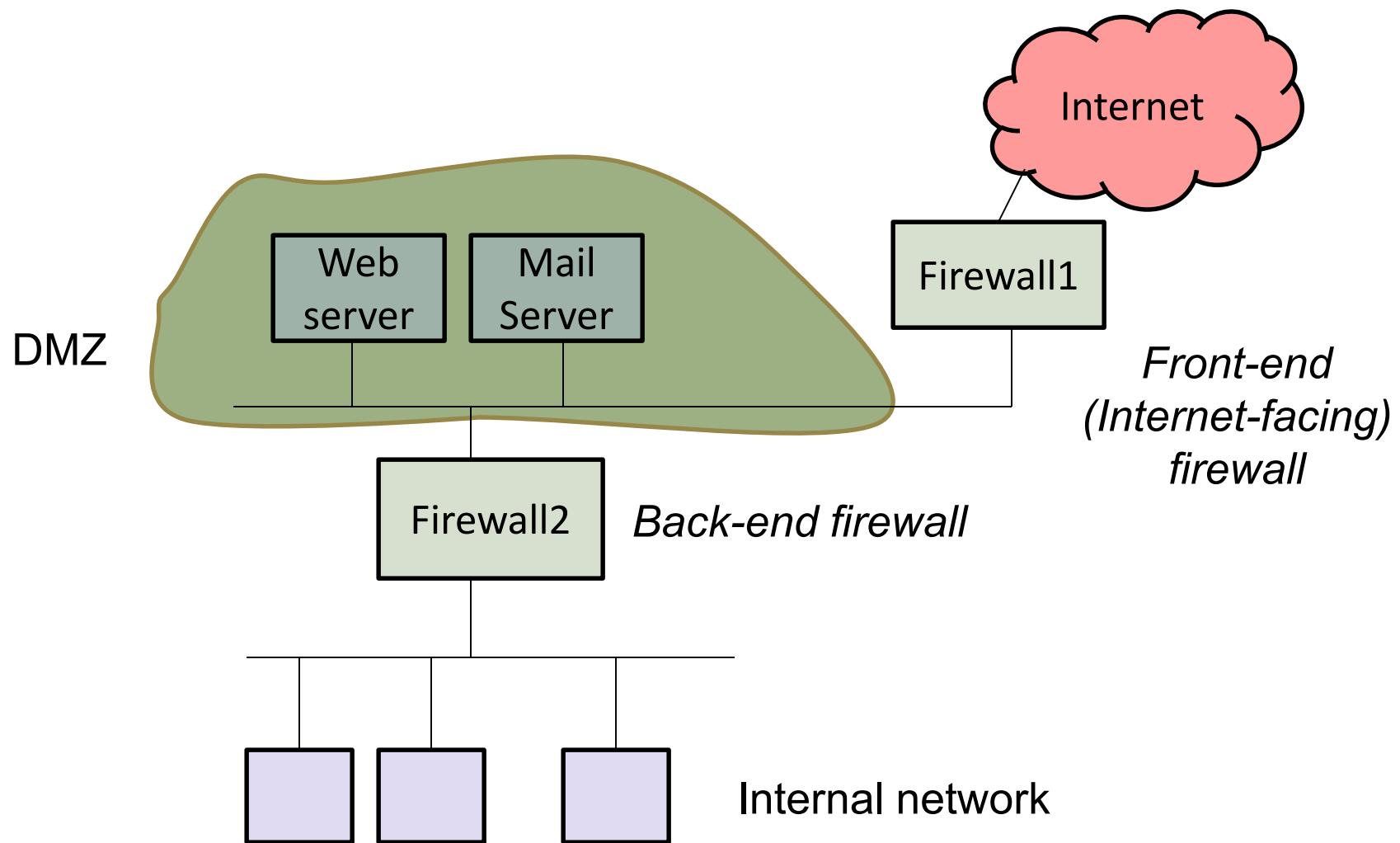
Firewall

- Firewall:
 - Sits at border between networks
 - Looks at addresses, services, other characteristics of traffic
 - Controls what traffic is allowed to enter the network (ingress filtering), or leave the network (egress filtering)
- **Definition:** “*Firewall are devices or programs that control the flow of network traffic between networks or hosts that employ differing security postures.*”
(From “Guidelines on Firewalls and Firewall Policy”, NIST, special publication 800-41
<http://csrc.nist.gov/publications/nistpubs/800-41-Rev1/sp800-41-rev1.pdf>.)
- **DMZ (Demilitarized Zone):**
 - A small sub-network that exposes the organization’s external service to the (untrusted) Internet
 - The original military term: an area between states in which military operations are not permitted

Demilitarized Zone (DMZ)



Typical 2-Firewall Setting



Firewall Design (Read [PF] pg 453)

- A firewall enforces a **set of rules** provided by the network administrator
- Examples of rules for Firewall2 (back-end firewall):
 - Block HTTP
 - Allow from Internal network to Mail Server: SMTP, POP3
- Examples of rules for Firewall1 (front-end firewall):
 - Allow from anywhere to Mail Server: SMTP only
- How the rules are to be specified differs on different devices and software
- The next slide gives a typical design/configuration (from [PF] pg. 453)

Sample Firewall Configuration

| Rule No | Protocol Type | Source Address | Destination Address | Designation Port | Action |
|---------|---------------|----------------|---------------------|------------------|--------|
| 1 | TCP | * | 192.168.1.* | 25 | Permit |
| 2 | TCP | * | 192.168.1.* | 69 | Permit |
| 3 | TCP | 192.168.1.* | * | 80 | Permit |
| 4 | TCP | * | 192.168.1.18 | 80 | Permit |
| 5 | TCP | * | 192.168.1.* | * | Deny |
| 6 | UDP | * | 192.168.1.* | * | Deny |

* (any) matches **any** value

The table is processed in a **top-down manner**

The **first matching rule** determines the action taken

Hence: put your most specific rule *first*, and put your most general rule *last*

Types of Firewall

The textbook [PF] lists **6** types of firewalls

The literature, including NIST's document (NIST 800-41), usually groups firewalls into **3 types**:

1. (Traditional) packet filters:

- Filters packets based on information in *packet headers*

2. Stateful-inspection (packet filters):

- Maintains a *state table* of all active connections
- Filters packets based on active connection states

3. Application proxy:

- Understands application logic
- Acts as a relay of application-level traffic

(Details are not required for this module)

Comparison of Firewall Types [PF]

Optional

| Packet Filter | Stateful Inspection | Application Proxy | Circuit Gateway | Guard | Personal Firewall |
|--|--|---|--|--|---|
| Simplest decision-making rules, packet by packet | Correlates data across packets | Simulates effect of an application program | Joins two subnetworks | Implements any conditions that can be programmed | Similar to packet filter, but getting more complex |
| Sees only addresses and service protocol type | Can see addresses and data | Sees and analyzes full data portion of pack | Sees addresses and data | Sees and analyzes full content of data | Can see full data portion |
| Auditing limited because of speed limitations | Auditing possible | Auditing likely | Auditing likely | Auditing likely | Auditing likely |
| Screens based on connection rules | Screens based on information across multiple packets—in either headers or data | Screens based on behavior of application | Screens based on address | Screens based on interpretation of content | Typically, screens based on content of each packet individually, based on address or content |
| Complex addressing rules can make configuration tricky | Usually preconfigured to detect certain attack signatures | Simple proxies can substitute for complex decision rules, but proxies must be aware of application's behavior | Relatively simple addressing rules; make configuration straightforward | Complex guard functionality; can be difficult to define and program accurately | Usually starts in mode to deny all inbound traffic; adds addresses and functions to trust as they arise |

5.7 Protection: Network Security Management

Network Security Management

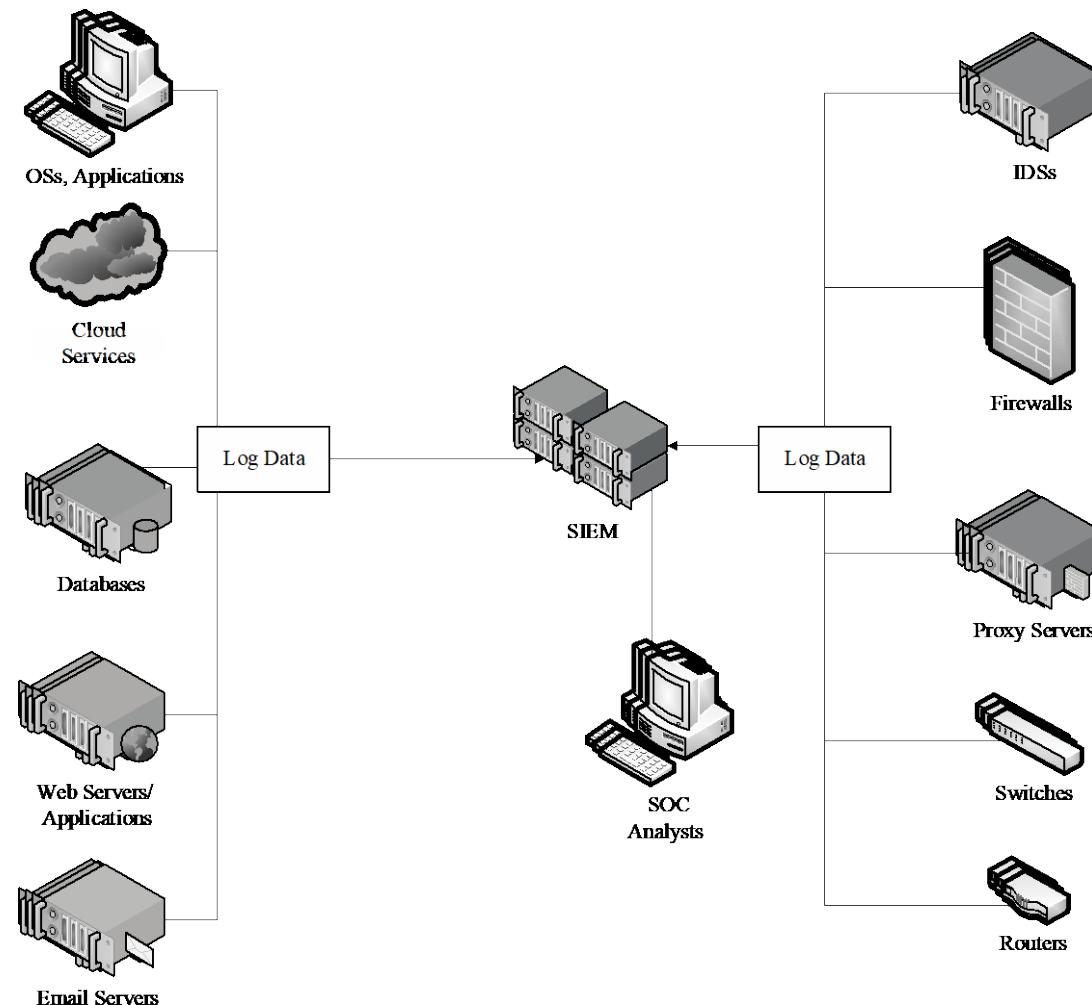
There is a need to *continuously* monitor and adjust network characteristics

(Details on this are omitted. See [PF6.9])

Some terms:

- **Security Operations Center (SOC):**
 - A centralized unit in an organization that monitors the IT systems and deals with security issues
- **Security Information and Event Management (SIEM):**
 - Pronounced as “SIM”
 - Provides **real-time analysis** of security alerts generated by network hardware and applications
 - May include the following **capabilities**: data aggregation & correlation, event alerting, compliance report generation, forensic analysis

Security Information and Event Management (SIEM)



End of Part 1 of CS2107

- We're are halfway in the module:
Great and thanks!
- Do enjoy your recess week
- Good luck for your Assignment 1!
- All the best for your mid-term quiz too!
- See you again in Week 7

Reading:

See [Gollmann] Chapter 5 & 7

See [PF] Chapter 2.2 (pages 72 – 85)

See [Andersen] Chapter 4 up to 4.2.4

Read Wiki http://en.wikipedia.org/wiki/File_system_permissions

Lecture 6: Access Control

6.1 Overview of layering model in computer system design

6.2 Access control model

6.3 Access control matrix

6.4 Intermediate control

6.5 Access control in UNIX/Linux

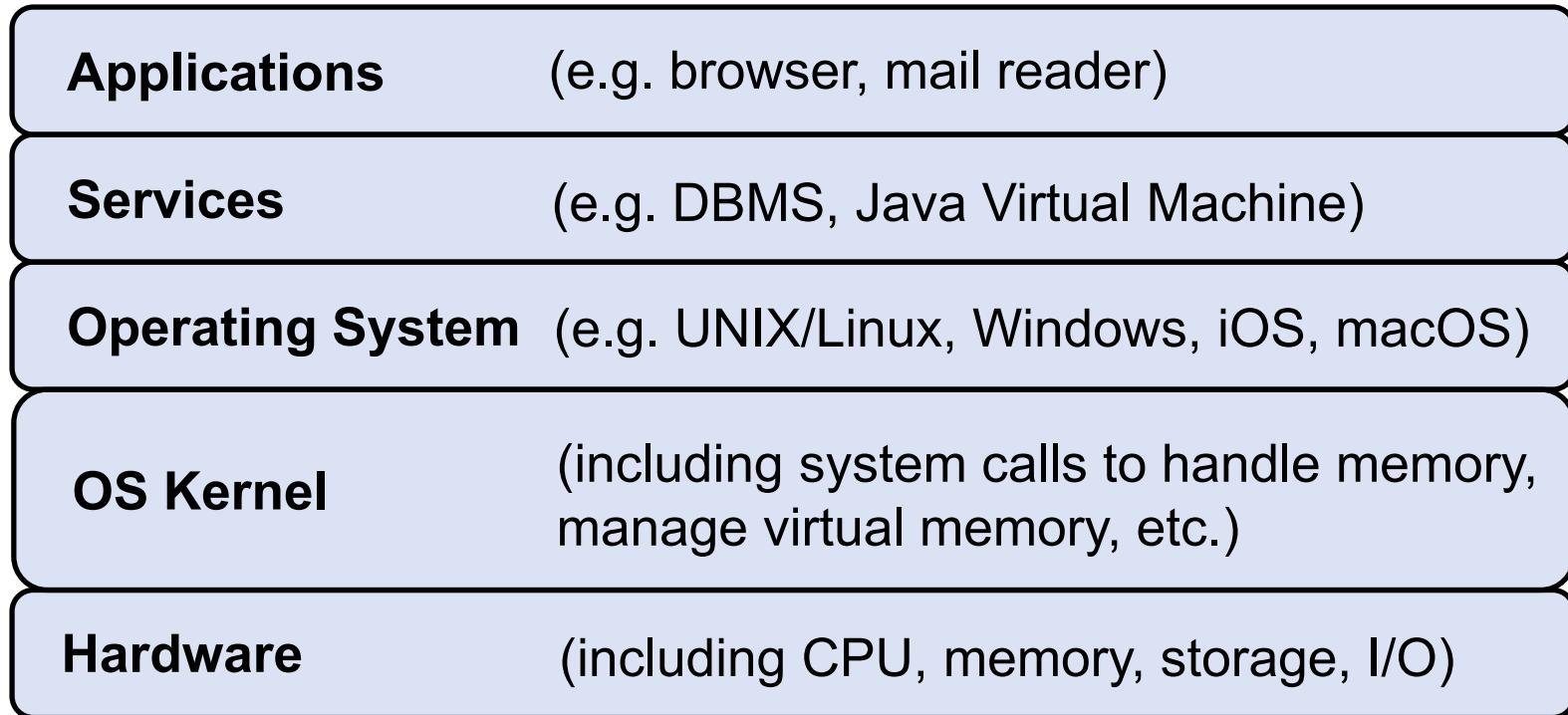
6.6* UNIX/Linux: Elevated privilege (controlled invocation)

* **Warning:** This part could be the most abstract and *complicated* notion in the module.

Complexity is bad for security. See https://www.schneier.com/news/archives/2012/12/complexity_the_worst.html.

6.1 Overview of Layering Model in Computer System Design

Layers in Computer System



Remarks:

1. We can view OS kernel as part of the OS.
2. These layers are used as a *guideline*.

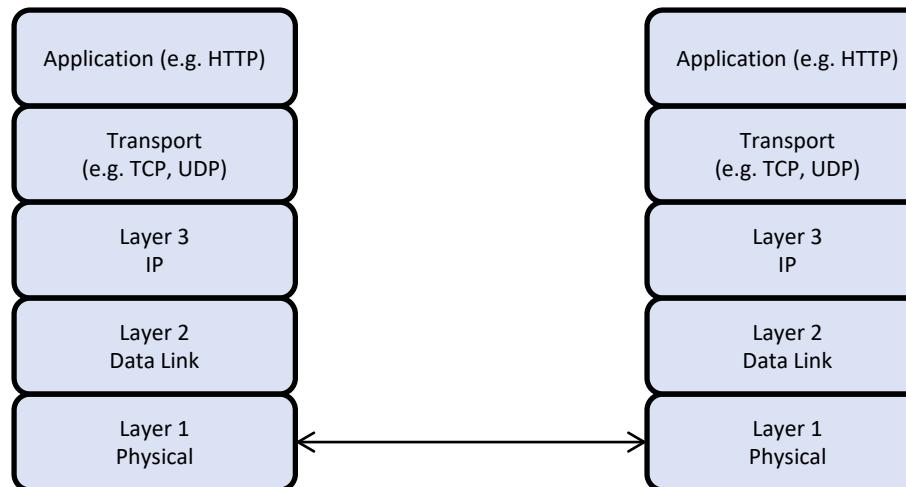
Actual systems typically don't have distinct layers

(for example, a windowing system may span across multiple “layers”).

Compared to Layers in Communication Network

Network:

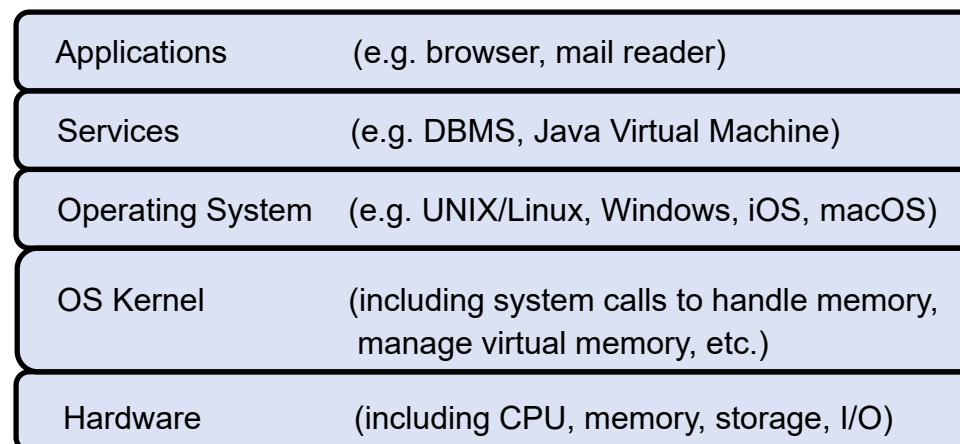
- The **boundary** is more well defined
- Information/data **flows** from the topmost layer down to the lowest layer, and is transmitted from the lowest layer to the topmost layer
- A **concern** of data confidentiality and integrity



Compared to Layers in Communication Network

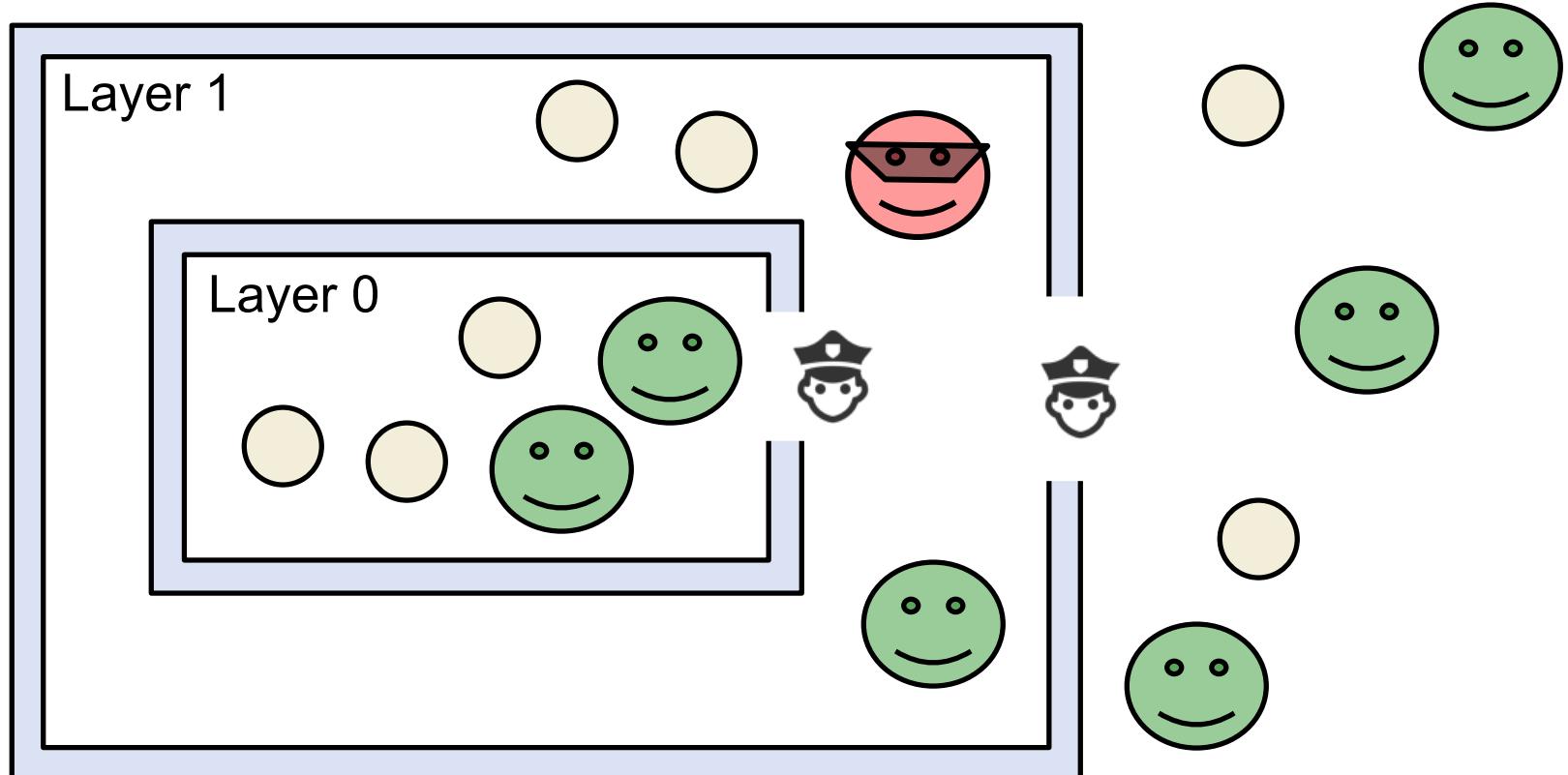
System:

- The **boundary** is *less* well-defined
- Every layer has its own “**processes**” and “**data**”
(although ultimately, the raw processes/data are handled by hardware)
- The main security **concern** is about access to the processes & memory/storage
- Hence, besides data confidentiality & integrity (e.g. password file), there is also a concern of **process “integrity”**: whether it deviates from its original execution path



Attackers and System's Layers

Layer 2



- Suppose an attacker sneaks into Layer 1
- The attacker must *not* be able to directly manipulate objects/processes in (more privileged) Layer 0
- Note that it is very difficult to ensure this requirement (due to possible implementation errors, overlooks in the design, etc.)

Security Mechanism

- It is insightful to figure out at *what layer* a security mechanism/attack resides
- A (layered-based) *security mechanism* should have a well-defined ***security perimeter/boundary***:
 - See [Gollmann] Section 3.5: “The parts of the system that can malfunction without compromising the protection mechanism *lie outside* this perimeter. The parts of the system that can be used to disable the protection mechanism *lie within* this perimeter”.
- Nevertheless, quite often, it is difficult to determine the boundary
- An important **design consideration** of the security mechanism is: on *how to prevent attacker from* getting access to a layer inside the boundary

Security Mechanism

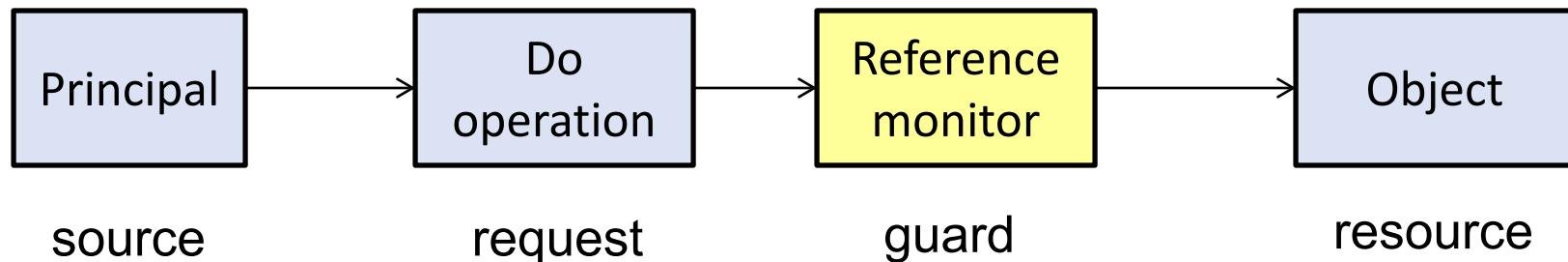
- For example: **SQL injection attacks** target at the SQL DBMS. The **OS password management** (which is in a layer below) *should remain intact* even if an SQL injection attack has been successfully carried out.
- It is also possible that an application takes care of *its own security* (i.e. self-secure itself):
 - For example: if an application **always encrypt** its data before writing them to the file system, even if the access control of the file system is compromised (e.g. malicious user can read someone else files), the *confidentiality* of the data will still be preserved.

6.2 Access Control Model

Why Access Control?

- Access control is required in computer system, information system, and physical system
- Different application domains have different requirements /interpretation
- Generally, **access control** is about: “*selective restriction of access to a place or other resource*” (Wiki)
- The access control system/model gives a way to **specify and enforce** such restriction on the subjects, objects and actions
- Examples of application domains:
 - OS
 - Social media (e.g. Facebook)
 - Documents in an organization (document can be classified as “restricted”, “confidential”, “secret”, etc.)
 - Physical access to different part of the building

Principal/Subject, Operation, Object



- A *principal* (or *subject*) wants to access an *object* with some *operation*
- The *reference monitor* either grants or denies the access
- Examples:
 - LumiNUS: a **student** wants to **submit** a **forum post**
 - LumiNUS: a **student** wants to **read** the **grade of another student**
 - File system: a **user** wants to **delete** a **file**
 - File system: a **user Alice** wants to **change** a **file's mode** so that it can be **read** by **another user named Bob**

Principal/Subject, Operation, Object

Principals vs subjects:

- **Principals:** the human users
- **Subjects:** the entities in the system that operate on behalf of the principals, e.g. processes, requests

Accesses to objects can be classified to the following:

- **Observe:** e.g. reading a file
(In LumiNUS: downloading a file from File/Workbin)
- **Alter:** e.g. writing a file, deleting a file, changing properties
(In LumiNUS: uploading a file to the File/Workbin)
- **Action:** e.g. executing a program

Object Access Rights and Ownership

Who decides the access rights to an object?

There are two approaches:

1. ***Discretionary Access Control (DAC):***
the **owner** of the object decides the rights
2. ***Mandatory Access Control (MAC):***
a **system-wide policy** decides the rights, which must be followed by *everyone* in the system must follow

6.3 Access Control Matrix

Access Control Matrix (See [PF] pg. 79)

- How do we **specify the access rights** of a particular principal to a particular object? We can use a table called ***access control matrix***

| | my.c | mysh.sh | sudo | a.txt |
|-------|---------|---------|---------|---------|
| root | {r,w} | {r,x} | {r,s,o} | {r,w} |
| Alice | {r,w} | {r,x,o} | {r,s} | {r,w,o} |
| Bob | {r,w,o} | {} | {r,s} | {} |

Note: r: read, w: write, x: execute, s: execute as owner, o: owner

- Although the above access control matrix can specify the access rights for **all pairs** of principals and objects, the table can be **very large**, and is thus **difficult to manage**
- Hence, it is often treated as an **abstract concept** only, and seldom explicitly deployed (see the next slide)

Access Control List (ACL) and Capabilities

- The access control matrix can be represented in two different ways: ***Access Control List (ACL)*** or ***capabilities***
- **Access Control List (ACL):**
stores the *access rights* to a **particular object** as a list
- **Capabilities:**
 - **A subject** is given a list of *capabilities*, where each capability is the access rights to an object
 - “*A capability*: is an unforgeable token that gives the possessor certain rights to an object”
(see [PG] pg. 82 on the description of “capability”)

(Question: Does UNIX file system adopt ACL or capabilities?)

ACL and Capabilities: Examples

- **Access control matrix:**

| | my.c | mysh.sh | sudo | a.txt |
|-------|---------|---------|---------|---------|
| root | {r,w} | {r,x} | {r,s,o} | {r,w} |
| Alice | {} | {r,x,o} | {r,s} | {r,w,o} |
| Bob | {r,w,o} | {} | {r,s} | {} |

- **ACL:** each object has a list of access rights to it

| | |
|---------|--|
| my.c | → (root, {r,w}) → (Bob, {r,w,o}) |
| mysh.sh | → (root, {r,x}) → (Alice, {r,x,o}) |
| sudo | → (root, {r,s,o}) → (Alice, {r,s}) → (Bob, {r,s}) |
| a.txt | → (root, {r,w}) → (root, {r,w,o}) |

- **Capabilities:** each subject has a list of capabilities to objects

| | |
|-------|---|
| root | → (my.c, {r,w}) → (mysh.sh, {r,x}) → (sudo, {r,s,o}) → (a.txt, {r,w}) |
| Alice | → (mysh.sh, {r,x,o}) → (sudo, {r,s}) → (a.txt, {r,w,o}) |
| Bob | → (my.c, {r,w,o}) → (sudo, {r,s}) |

Drawbacks of ACL and Capabilities

- Drawback of ACL:
 - It is difficult to get an overview of the objects that **a particular subject** has access rights to
 - I.e. it is hard to answer: “which objects can *a particular subject* access?”
 - As an illustration: in UNIX, suppose the system admin wants to generate the list of file that the user `alice012` has “r” access to. How to quickly generate this list?
- Drawbacks of capabilities:
 - It is difficult to get an overview of the subjects who have access rights to **a particular object**
 - I.e. it is hard to answer: “which subjects can access *a particular object*?”

Drawbacks of ACL and Capabilities

- Drawbacks of **both** methods:
 - The size of the lists can still be **too large** to manage
 - Hence, we need some ways to **simplify** the representation
 - One simple method is to “*group*” the subjects/objects and define the access rights on **the defined groups**
 - We need an ***intermediate control***

6.4 Intermediate Control

Intermediate Control: Group

In Unix file permission, the ACL specifies the rights for the:

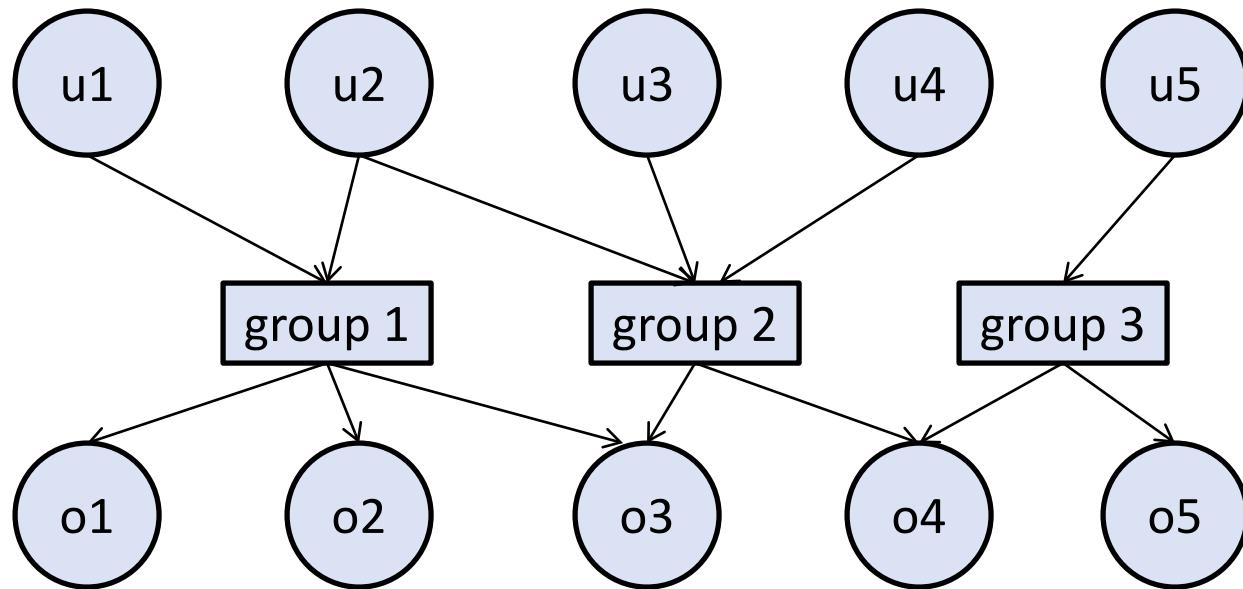
- **owner (user)**
 - **group**
 - **world (others)**
-
- Subjects in a same group: have the same access rights
 - Some systems demand that a subject is in a single group, but some systems don't put such a restriction

Question: Is it possible in UNIX that an owner **does not** have a read access, but others have?

Answer: Strangely, yes. See Alice's file temp below:

```
--w-r--r--    1 alice  staff      3 Mar 13 00:27 temp
```

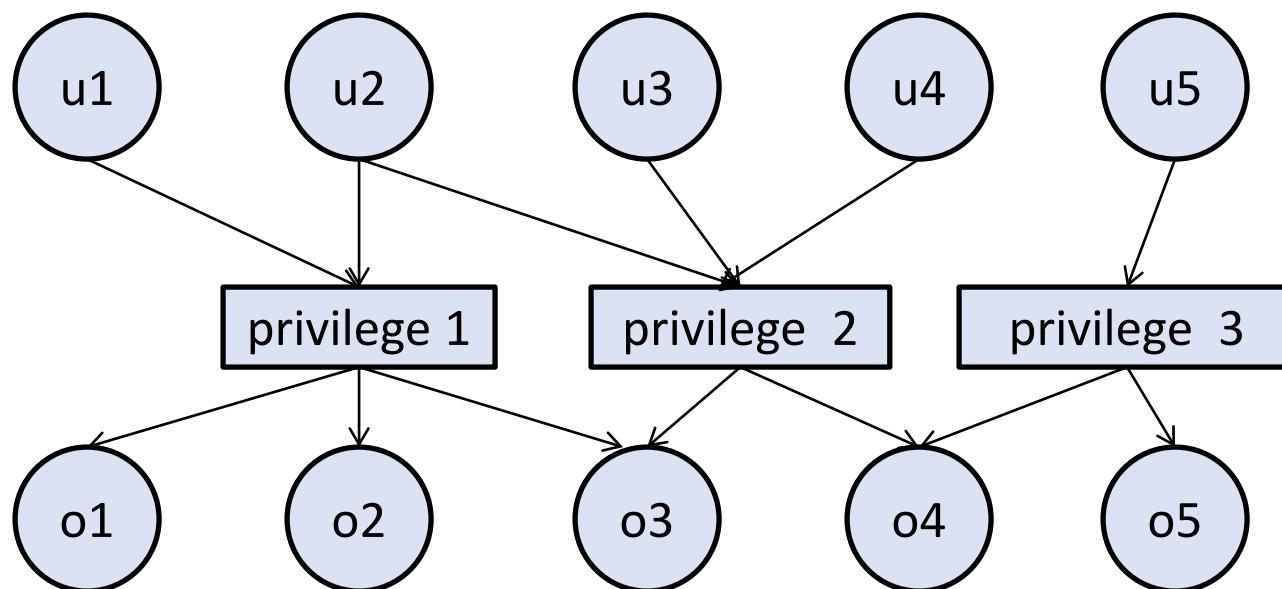
Users and Groups: Illustration



- In LumiNUS, *project groups* can be created.
Objects created in a group can only be read by members of the group + *the lecturer(s)*.
- In UNIX, groups can only be created by root.
The groups information is stored in the file /etc/group.

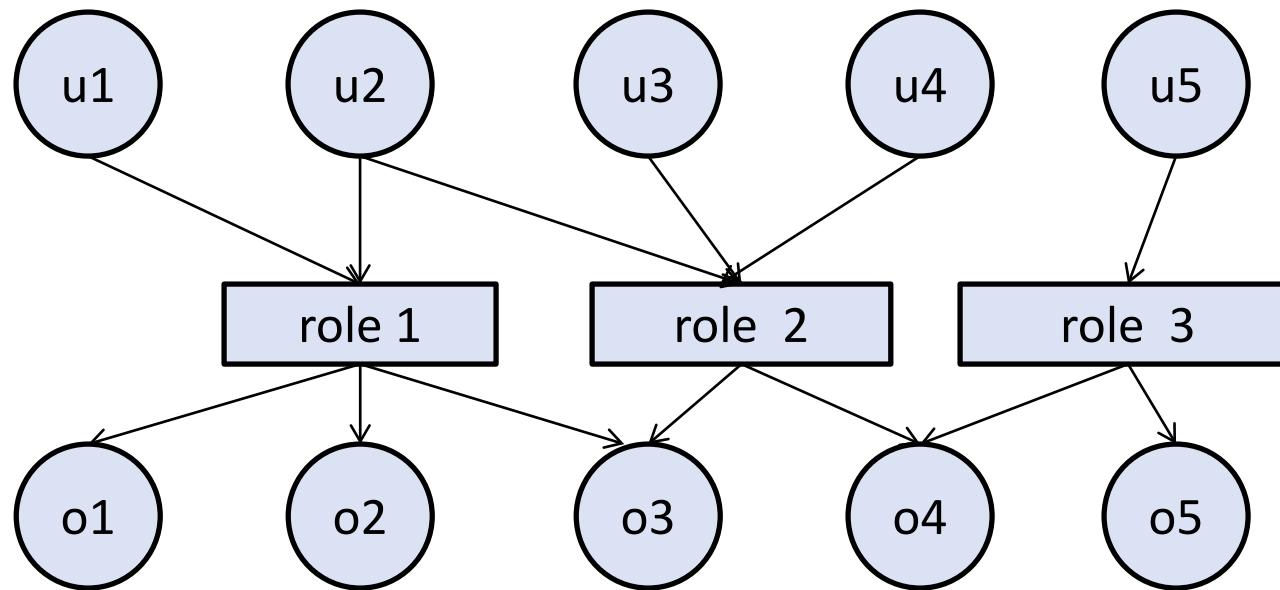
Intermediate Control: Privileges

- We often use the term ***privilege*** for:
the access right to execute a process
- Privilege can also be viewed as an intermediate control



Intermediate Control: Role-based Access Control (RBAC)

- The grouping can be determined by the “*role*” of the subjects
- A **role** associates with a **collection of procedures**
- In order to carry out these procedures, **access rights** to certain objects are required

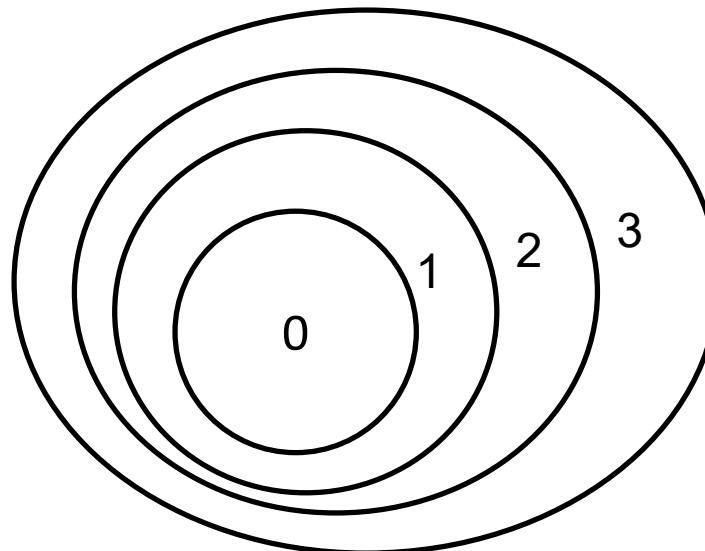


Intermediate Control: Role-based Access Control (RBAC)

- If a subject is assigned a particular role, the access rights to the objects can be determined based on the *least privilege principle*
- The **least privilege principle**: only access rights that are **required** to complete the role will be assigned
- Examples on **LumiNUS's gradebook**:
 - The roles of a TA include entering the grade for all students.
So the TA also **has** a “**write**” access on the grades.
 - Should we also give a TA an access right to *change names*?
This right is “irrelevant” as it is not related to the roles of the TA.
 - Since this change-names access right is **not** required for the TA to complete his/her tasks, by the least privilege principle,
the TA is **not** given the access right.

Intermediate Control: Protection Rings

- Here, each object (data) and subject (process) are assigned a **number**:
 - If a process is assigned a number i , we say that the process runs **in ring i**
 - Objects with **smaller number** are **more important**
 - Very often, we call processes with lower ring number as having “***higher privilege***”



Intermediate Control: Protection Rings

- Whether a subject can access an object is determined by their respective its **assigned number**:
 - A subject *cannot* access (i.e. both read/write) an object with smaller ring number
 - It can do so if its privilege gets “escalated”
- Some remarks:
 - UNIX has only **2** rings: **superuser** and **user**
 - We can also view this as a special case of RBAC in the sense that the ring number is the “role”

Examples of Intermediate Control: Bell-LaPadula vs Biba

- In protection rings, the subjects can access objects that are classified with **the same or lower** privilege.
Are there reasonable *alternatives*?
- Here are two well-known models: **Bell-LaPadula** and **Biba**: although they are rarely implemented as-it-is in computer system, they serve as a good guideline.
- In both models, objects and subjects are divided into **linear levels**, such as: level 0, level 1, level 2, ...
- **Higher level** corresponds to higher “security” (the opposite of protection rings).
Example: security clearance: unclassified < confidential < secret < top secret.
- **Multilevel security**: information with incompatible classifications (i.e. at different security levels).



Bell-LaPadula Model (for Confidentiality)

Read https://en.wikipedia.org/wiki/Bell-LaPadula_model.

Focus on **confidentiality**

Restrictions imposed by the Bell-LaPadula model:

- **No read up:**
 - A subject has only ***read access*** to objects whose security level is **below** the subject's current clearance level
 - This **prevents** a subject from getting access to information available in security levels **higher than** its current clearance level



Bell-LaPadula Model (for Confidentiality)

- **No write down:**
 - A subject has ***append access*** to objects whose security level is **higher** than its current clearance level
 - This **prevents** a subject from passing information to levels **lower than** its current level
 - Example: in order to prevent **information leakage**, a clerk working in the *highly-classified department* should not gossip with other staff from *lower security-level department*

Potential issues:

- Notice that a subject can append to objects at higher security level. Is there a concern of **integrity**?
Is it possible that appending to an object would **distort** its original content?

Biba Model (for Integrity)

Focus on Integrity

Restrictions imposed by the Biba Model:

- **No write up:**
 - A subject has only ***write access*** to objects whose security level is **below** the subject's current clearance level.
 - This **prevents** a subject from compromising the integrity of objects with security levels **higher than** its current clearance level.



Biba Model (for Integrity)

- **No read down:**
 - A subject has ***only read*** access to objects whose security level is **higher** than its current clearance level
 - This **prevents** a subject from reading forged information from levels **lower than** its current level

Remark:

- In a model that imposed **both** Biba and Bell-LaPadula models, subjects then can only read/write to objects **in the same level**

6.4 Access Control in UNIX/Linux

Notes:

- An easy way to get UNIX-like environment on Windows is **Cygwin** (<https://www.cygwin.com>).
- Another way is by installing a **hypervisor** or **virtual machine monitor** (VMM): **VirtualBox** (<https://www.virtualbox.org>), or **VMWare Player/Workstation** (<https://www.vmware.com>). Then install Linux (e.g. **Ubuntu desktop**).
Note: For VirtualBox, perform these additional installation steps as well:
install VirtualBox Extension Pack and Guest Additions.
- Yet, another method: **Bash shell in Window 10**.
(<https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/>).

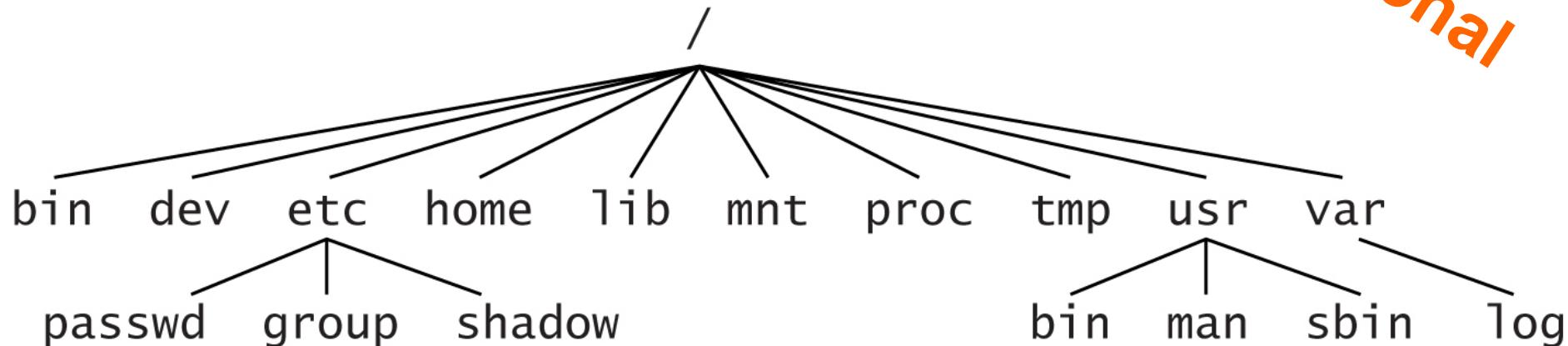
UNIX/Linux: Some Background

Optional

- History from 1970s
- Many versions:
Solaris, AIX, Linux, Android, OS X + iOS
- Linux is open source (<http://www.kernel.org>)
- Many available tools (usually also open source)
- Many Linux distributions (distros):
 - Vary in setup, administration, kernel.
 - A popular choice: Ubuntu desktop

UNIX/Linux File System Structure

Optional



/etc/passwd: user database, password file

/etc/shadow: user database containing hashed user passwords, shadow password file

/etc/group: group database

/bin/ls

man

UNIX/Linux Manual Pages

Optional

- UNIX documentation using the **man** command

- **man** is your friend!

- Note: small variations in **man** with different UNIX

```
$ man ls
```

```
$ man man
```

- Organized in sections

```
$ man printf
```

```
$ man 1 printf
```

```
$ man 3 printf
```

- A free good resource to learning Linux commands:

W. Shotts, “*The Linux Command Line*”, <http://linuxcommand.org>

UNIX/Linux Access Control

- In Unix, **objects** of access control include:
files, directories, memory devices, and I/O devices.
- All these resources are treated as **file**!
(a notion of “*universality of I/O*”, read also tis Wiki:
http://en.wikipedia.org/wiki/File_system_permissions).

```
%ls -al
-r-s--x--x 1 root wheel 164560 Sep 10 2014 sudo
-rwxr-xr-x 2 root wheel 18608 Nov 7 06:32 sum
-rw-r--r-- 1 alice staff 124 Mar 9 22:29 myprog.c
lr-xr-xr-x 1 root wheel 0 Mar 12 16:29 stdin
```

Question: What are the files in the directory /dev?

UNIX/Linux User and Groups

- Each **user**:
 - Has a unique **user/login name**
 - Has a numeric user identifier (**UID**): stored in /etc/passwd
 - Can belong to one or more groups:
the **first group** is stored in /etc/passwd,
additional group(s) are stored in /etc/group
- Each **group**:
 - Has a unique **group name**
 - Has a numeric group identifier (**GID**)
- Main purposes of UIDs and GIDs:
 - To determine **ownership** of various system resources
 - To determine the **credentials** of running processes
 - To control the **permissions** granted to processes
that wish to access the resources (*more on this later*)

UNIX/Linux Principals & Subjects

- **Principals:** *user identities* (UIDs) and *group identities* (GIDs)
- Information of the user accounts are stored in the **password file** /etc/passwd
 - E.g. root:*****:0:0:System Administrator:/var/root:/bin/sh
(Read Wiki page for details of these fields: <https://en.wikipedia.org/wiki/Passwd>)
- A special user is the *superuser*, with UID 0, and usually the username `root`:
 - All security checks are turned off for `root`.
(Recall that UNIX's protection rings consists of only 2 rings: superuser, users).
- **Subjects:** *processes*.
Each process has a process ID (PID): use the command `ps aux` or `ps -ef` to display a list of running processes

Remarks on the Password File Protection

- The file is made **world readable** because some information in `/etc/passwd` is needed by **non-root processes**
- Note that in the **older versions** of UNIX, the location of "*" was the hashed password $H(pw)$.
As a result, all users can have access to this hashed-password field.
- The availability of the hashed password allows attackers to carry out an **offline password guessing**.
- Since passwords are typically short, exhaustive search are able to get many passwords.
- To prevent this, it is now replaced by "*", and the actual password is stored somewhere else, and it is **not** world-readable.
The actual location depends on different versions of UNIX (e.g. the **shadow password file** `/etc/shadow`).

The Shadow Password File

The fields of an entry (separated by “:”):

- login name, **hashed password**, date of last password change, minimum password age, maximum password age, password warning period, password inactivity period, account expiration date, reserved field
- Example:

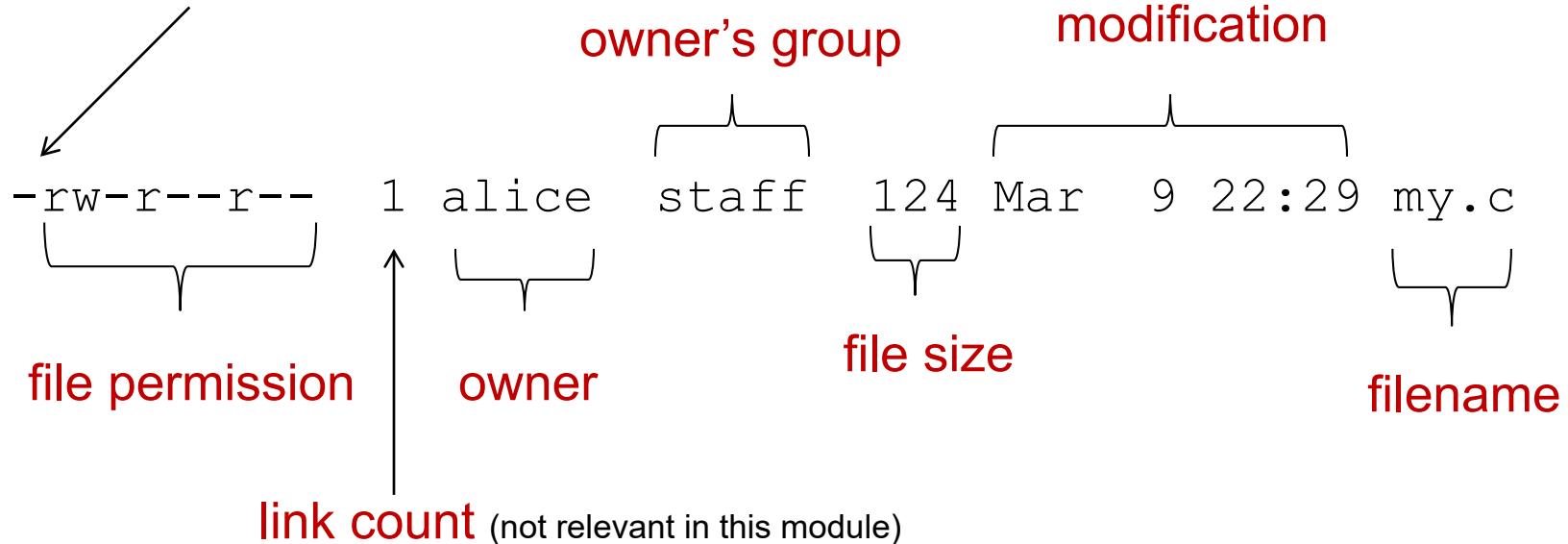
```
user1:$6$yonrs//S$bUdht9fglwJW0LduAxEJpcExtMfKokFMJoT8tGkKLx5  
xFGJk22/trPstOHXr4PdBID0AV1xko5LfFVDwW.aJS.:17275:0:99999:7:::
```

The **second** field (**hashed password**), which is shown in red, has the following format: **\$id\$salt\$hashed-key**

- **id**: ID of the hash-method used (5=SHA-256, 6= SHA-512, ...)
- **salt**: up to 16 chars drawn from the set [a-zA-Z0-9./]
- **hashed-key**: hash of the password (e.g. 43 chars for SHA-256, 86 chars for SHA-512)

File System Permission

indicates whether it is a file (-) or directory (d)



The **file permission** are grouped **into 3 triples**, that define the **read**, **write**, **execute** access for: **owner** (“user”), **group**, and **others** (the “world”):

‘-’ indicates access not granted

r: read

w: write (including delete)

x: execute (s: allow a user to execute with the permission of the *file owner*)

Changing File Permission Bits

- Use **chmod** command:
`chmod [options] mode[,mode] file1 [file2 ...]`
- Useful options:
 - `-R`: Recursive, i.e. include objects in subdirectories
 - `-f`: force processing to continue if errors occur
 - `-v`: verbose, show objects changed (unchanged objects are not shown)
- Two notations for mode:
 - *Symbolic mode* notation
 - *Octal mode* notation
- See also : <https://en.wikipedia.org/wiki/Chmod>

Changing File Permission Bits

- **Symbolic mode** notation:
 - **Syntax:** [references][operator][modes]
 - *Reference:* u (user), g (group), o (others), a (all)
 - *Operator:* + (add), - (remove), = (set)
 - *Mode:*
 - r (read), w (write), x (execute), s (setuid/gid), t (sticky)
 - Examples:
 - chmod g+w shared_dir
 - chmod ug=rw groupAgreements.txt
 - What are the file permission bits of shared_dir and groupAgreements.txt?
shared_dir: drwxr-xr-x → drwxrWxr-x

Changing File Permission Bits

- Octal mode notation:
 - 3 or 4 octal digits
 - 3 rightmost digits refer to permissions for: the file **user**, the **group**, and **others**
 - Optional leading digit, when 4 digits are given, specifies: **the special file permissions** (setuid, setgid and sticky bit)
 - Examples:

```
chmod 664 sharedFile
chmod 4755 setCtrls.sh
```
 - **-rw-rw-r-** and **-rwsr-xr-x**

| Octal | Binary | rwx |
|-------|--------|-----|
| 7 | 111 | rwx |
| 6 | 110 | rw- |
| 5 | 101 | r-x |
| 4 | 100 | r-- |
| 3 | 011 | -wx |
| 2 | 010 | -w- |
| 1 | 001 | --x |
| 0 | 000 | --- |

File System Permission: Additional Notes

- **Directory** permissions:
 - **r**: allows the contents of the directory to be **listed** if the **x** attribute is also set
 - **w**: allows files within the directory to be created, deleted, or renamed if the **x** attribute is also set
 - **x**: allows a directory to be **entered** (i.e. `cd dir`)
- **Special file** permissions:
 - **Set-UID**: the process' *effective user ID* of is the *owner* of the executable file (usually root), rather than the user running the executable

```
-r-sr-sr-x    3  root      sys          104580 Sep  
16 12:02 /usr/bin/passwd
```

File System Permission: Other Notes

- **Set-GID:** the process' *effective group ID* is the group owner of the executable file

```
-r-sr-Sr-x 3 root      sys          104580 Sep 16  
12:02 /usr/bin/passwd
```

- **Sticky bit:**

If a **directory** has the sticky bit set, its file can be deleted *only by* the owner of the file, the owner of the directory, or by root.

This prevents a user from deleting other users' files from public directories such as /tmp:

```
drwxrwxrwt 7  root    sys     400 Sep 3 13:37 tmp
```

- See also: <https://docs.oracle.com/cd/E19683-01/816-4883/secfile-69/index.html>

Objects and Access control

- Recall that the objects are **files**.
- Each file is owned by a **user** and a **group**.
Also recall that each file is associated with a **9-bit permission**.
- When a **non-root user** (subject) wants to access a file (object), the following are checked in order:
 1. If the user is the owner, the permission bits for **owner** decide the access rights
 2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for **group** decide the access rights
 3. If the user is not the owner, nor member of the group that own the file, then the permission bits for **other** decide
- The **owner** of a file or **superuser** can change the permission bits

Selected Issues: Search Path

- When a user types in the command to execute a program, say “su” without specifying the full path, which program would be executed:

/usr/bin/su or ./su?

- The program to be executed is searched through the directories specified in the **search path**.

Use a command `echo $PATH` to display the search path.

- When a program with the name is found in a directory, the search **stops** and the program will be **executed**.

- Suppose an attacker somehow stored a malicious program in the directory that appears in the **beginning** of the search path, and the malicious program has a **common name**, say “su”.
When a user executes “su”, the **malicious program** will be invoked instead.

- To prevent such attack, specify the **full path**.
- Also **avoid** putting the current directory (“.”) in the search path. *Why?*

Controlled Invocation

(More on this in next section)

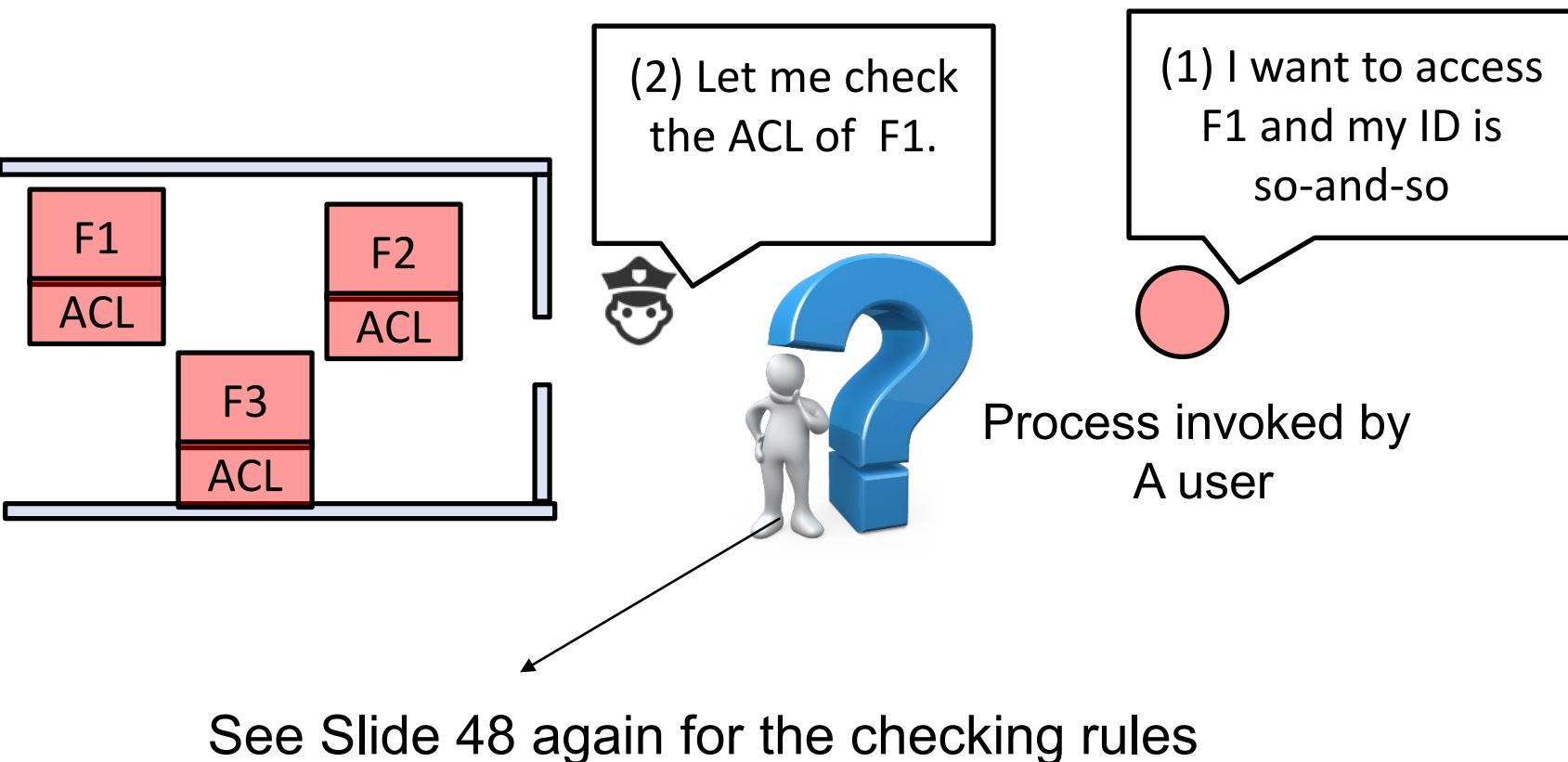
- Certain resources in UNIX/Linux can **only** be accessed by superuser: e.g. listening at the trusted port 0—1023, accessing /etc/shadow file
- Sometimes, a user **needs** those resources for certain operation: e.g. changing password
- It is **not** advisable to change the user status to superuser
- A **solution** is *controlled invocation*: the OS provides a *predefined set of operations (programs) in a superuser mode*, and the user can then invoke those operations with the superuser mode
- An example file:

```
-rws--x--x 3 root bin 59808 Nov 17 07:21  
/usr/bin/passwd
```

The “s” (set-UID) bit indicates that the privilege is escalated while a user is executing this process

6.6 UNIX/Linux: Privilege Escalation (Controlled Invocation)

Access Control and Reference Monitor



Process and Set User ID (Set-UID)

- A process is a subject: has an identification (PID)
- A new process can be created by executing a file or due to a “fork” by an existing process
- A process is associated with ***process credentials***: a **real UID** and an **effective UID**
- The **real UID** is inherited from the user who invokes the process. It identifies the **real owner** of the process.
E.g. if the user is `alice`, then the process’ real UID is `alice`.*
- For processes created by executing a file, there are 2 cases:
 - If the set-User-ID (set-UID) is disabled (the permission bit is displayed as “x”), then the process’ **effective UID** is same as **real UID**
 - If the set-User-ID (set-UID) is enabled (the permission bit is displayed as “s”), then the process’ **effective UID** is inherited from the UID of the **file’s owner**

Real UID and Effective UID: Examples

- If alice creates the process by executing the file:

```
-r-xr-xr-x 1 root staff 6 Mar 18 08:00 check
```

Then the process' real UID is alice,
whereas its effective UID is alice

- If the process is created by executing the following file:

```
-r-sr-xr-x 1 root staff 6 Mar 18 08:00 check1
```

Then the process' real UID is alice,
but its effective UID is root

This indicates that set-UID is enabled

When a Process (Subject) Wants to Read a File (Object)

- When a process wants to access a file, the **effective UID** of the process is treated as the “**subject**” and checked against the file permission to decide whether it will be granted or denied access
- Example:

Consider a file owned by the root:

```
-rw----- 1 root staff 6 Mar 18 08:00 sensitive.txt
```

- If the effective UID of a process is **alice**, then the process will be **denied** reading the file
- If the effective UID of a process is **root**, then the process will be **allowed** to read the file

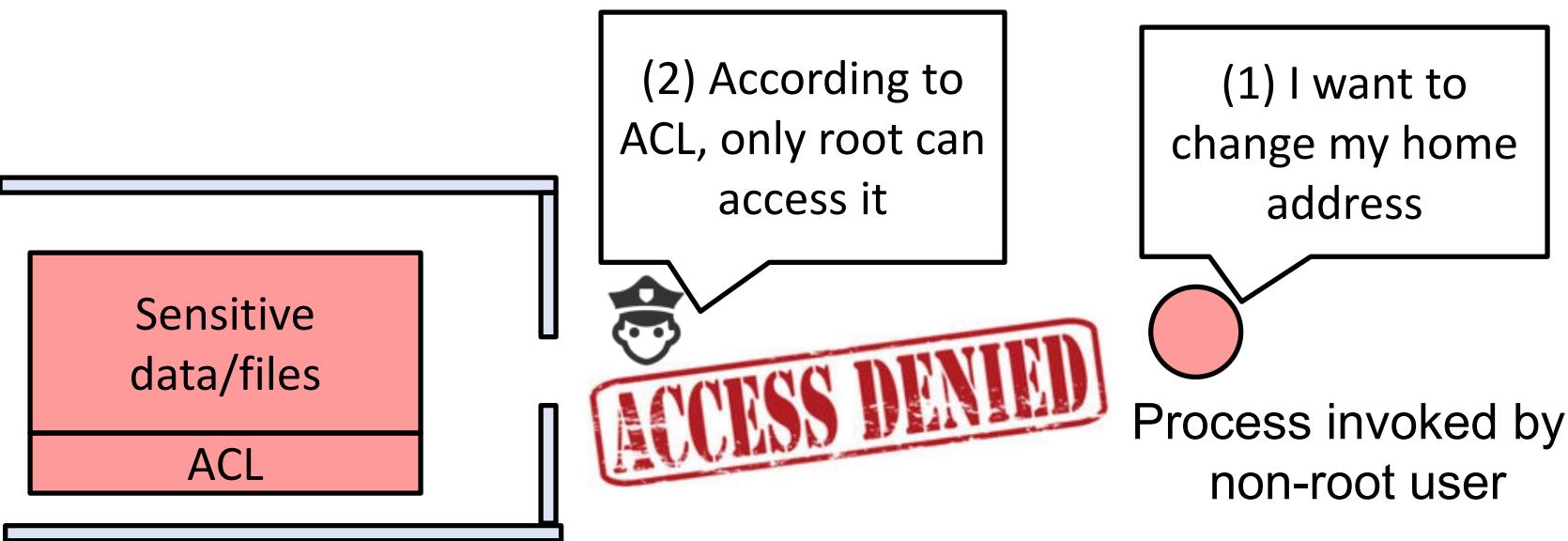
Use Case Scenario of “s” (Set-UID)

- Consider a scenario where the file `employee.txt` contains personal information of the users
- This is **sensitive** information, hence, the system administrator set it to **non-readable** except by root:

```
-rw----- 1 root staff 6 Mar 18 08:00 employee.txt
```

- However, users should be allowed to **self-view** and even **self-edit** some fields (for e.g. postal address) of their own profile
- Since the file permissible is set to “-” for all users (except root), a **process** created by any user (except root) **cannot** read/write it
- Now, we are stuck: there are data in the file that we want to **protect**, and data that we want the user to **access**
- *What can we do?*

Access Control and Reference Monitor



Solution

- Create an executable file `editprofile` owned by root:
`-r-sr-xr-x 1 root staff 6 Mar 18 08:00 editprofile`
- The program is made **world-executable**:
any user can execute it
- Furthermore, the **set-UID bit** is set (“s”):
when it is executed, its effective UID will be “root”
- This is an example of a ***set-UID-root*** program/executable
- Now, if `alice` executes the file, the process’ real UID is `alice`, but its effective UID is `root`:
this process now **can read/write** the file `employee.txt`



Comparison: When Set-UID is *Disabled*

- If the user `alice` invokes the executable, the process will have its **effective ID** as `alice`
- When this process wants to read the file `employee.txt`, the OS (reference monitor) will **deny** the access

Process info: name (`editprofile`) real ID (`alice`) effective ID (`alice`)



```
-rw----- 1 root staff 6 Mar 18 08:00 employee.txt
-r-xr-xr-x 1 root staff 6 Mar 18 08:00 editprofile
```

Two black arrows point from the text "effective ID (alice)" in the process info box above to the word "alice" in the "real ID" column and the "effective ID" column of the file listing below.

Comparison: When Set-UID is Enabled

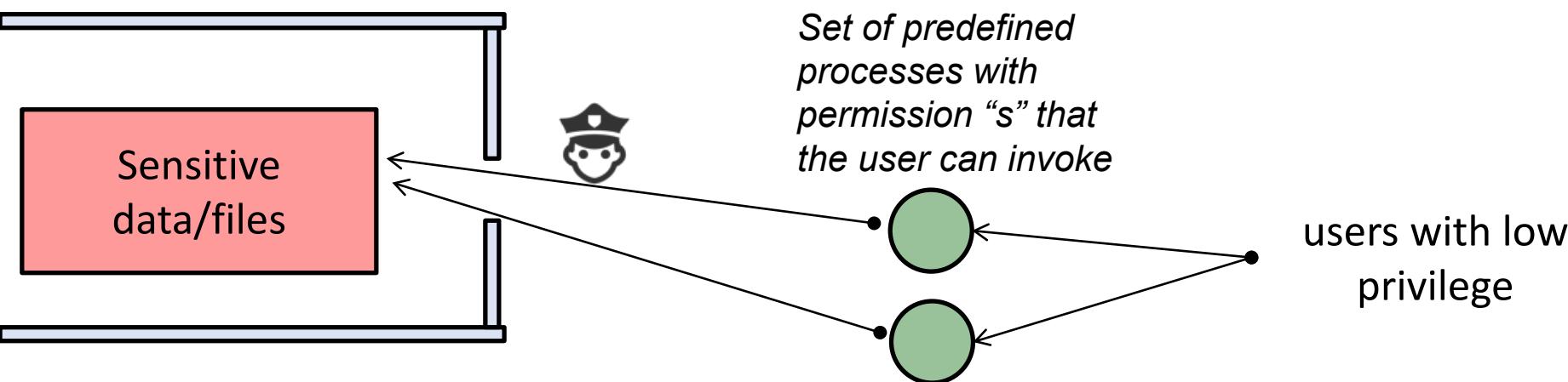
- But if the permission of the executable is “s” instead of “x”, then the invoked process will have `root` as its effective ID
- Hence the OS grants the process to read the file
- Now, the process invoked by `alice` can access `employee.txt`

Process info: name (`editprofile`) real ID (`alice`) effective ID (`root`)



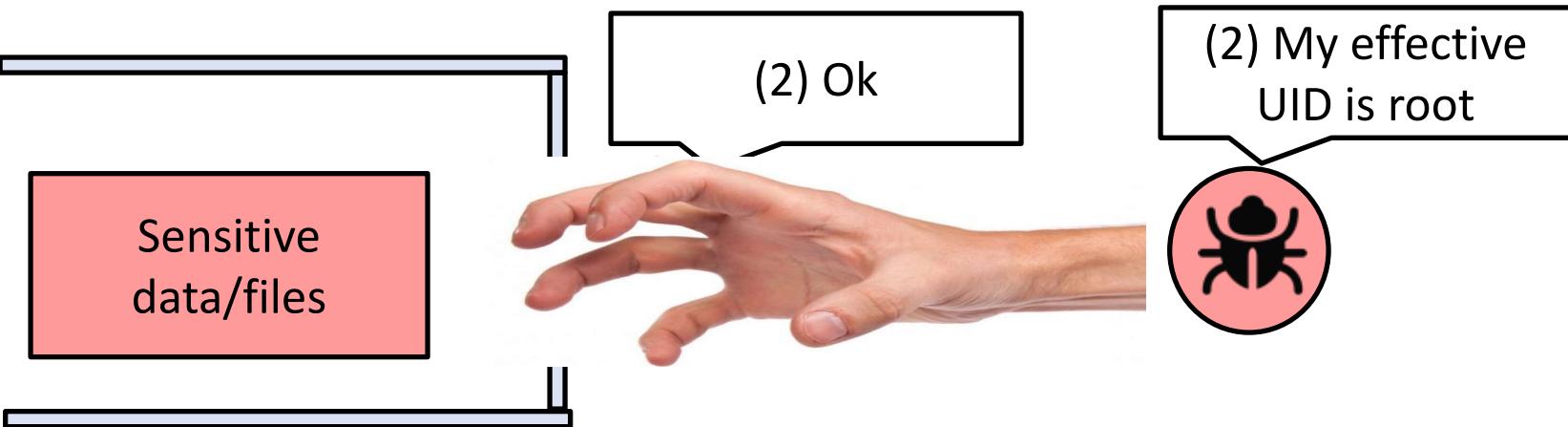
Elevated Privilege

- In this example, the process `editprofile` is temporarily **elevated** to superuser (i.e. root), so as to access the sensitive data
- We can view the elevated process as the **interfaces** where a user can access the “sensitive” information:
 - They are the predefined “**bridges**” for the user to access the data
 - Note that the “bridge” can only be built by **the root**
- So, it is important that these “bridges” are correctly implemented and do not leak more than required!



Privilege Escalation Go Wrong

- Suppose a “bridge” is *not* implemented correctly, and contains **exploitable vulnerability**
- An attacker, by feeding the bridge with a carefully-crafted input, can cause it to perform **malicious operations**
- This could have serious implication, since the process is now running with “**elevated privilege**”
- Attacks of such form also known as “**privilege escalation**” attacks
(Read https://en.wikipedia.org/wiki/Privilege_escalation on privilege escalation:
Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.)
- This leads us to another important security topic:
secure programming and **software security**



Program has permission “s”,
and the program has a “bug”!

Footnote: More Complications (Real UID, Effective UID, Saved UID)

- The OS actually maintains three IDs for a process:
real UID, effective UID, and **saved UID**
- **Saved UID** is like a “temp” container and is useful for a process running with elevated privileges to **drop privilege temporarily**:
a good set-UID programming practice
- A process removes its privileged user ID from its effective UID,
but stores it in its saved UID.
Later, the process may restore privilege by restoring the saved
privileged UID into its effective UID.
(See https://en.wikipedia.org/wiki/User_identifier#Saved_user_ID)
- The details may easily **confuse** many programmers
(Read <http://stackoverflow.com/questions/8499296/realuid-saved-uid-effective-uid-whats-going-on>)
- Different UNIX versions may have **different** behaviors:
complexity is bad for security!
(Optional: Chen et al., “Setuid Demystified”, USENIX Security, 2002)

Lecture 7: Software Security (Part I)

7.1 Overview of software security

7.2 Computer architecture background

 7.2.1 Code vs data, program counter

 7.2.2 Stack (aka execution stack, call stack)

 7.2.3 Control flow integrity

7.2 printf() and format string vulnerability

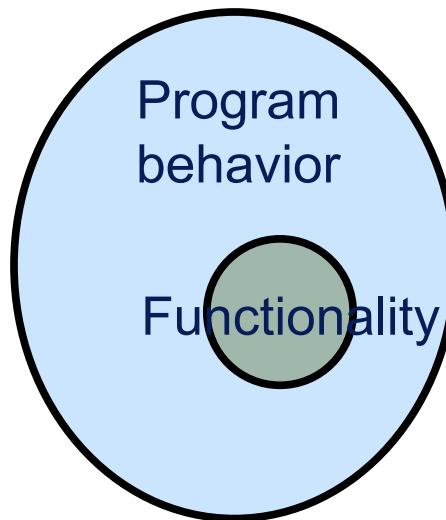
7.1 Overview of Software Security

Program: Requirements and Possible Behavior

Requirements of a program:

- A program has to be **correct**
- A program has to be **efficient**
- A program also has to be **secure**

Targeted program functionality vs possible behavior:



A program *may* behave **beyond** its intended functionality!

Possible Security Problems

- **Insecure implementation:**

Many programs are *not implemented properly*, allowing attacker (i.e. the person who invokes the process) to **deviate from the programmer's intents**

- **Unanticipated input:**

The attacker may supply **input** in a form that is *not anticipated* by the programmer, which can unintendedly cause the process to:

- Access sensitive resources;
- Execute some “injected” codes; or
- Deviate from the original intended execution path.

Either way, the attacker manages to **elevate its privilege**.

- In the next **3 lectures**, we will look at several **classes of insecure programs** and also **the reasons** behind their insecurity!

Some Sample Cases

Buffer Overflow:

- **Morris worm** (1988): exploited a Unix *finger* service to propagate itself over the Internet
- **Code Red worm** (2001): exploited Microsoft's IIS 5.0
- **SQL Slammer worm** (2003): compromised machines running Microsoft SQL Server 2000
- Various attacks on **game consoles** so that unlicensed software can run without the need for hardware modifications: **Xbox**, **PlayStation 2** (PS2 Independence Exploit), **Wii** (Twilight hack)
- ...

Some Sample Cases

SQL Injection:

- **Yahoo!** (2012): a hacker group was reported to have stolen **450,000** login credentials from Yahoo! by using a "union-based SQL injection technique"
- **British telco company TalkTalk** (2015): an attack exploiting a vulnerability in a legacy web portal was used to steal the personal details of **156,959** customers

Integer Overflow:

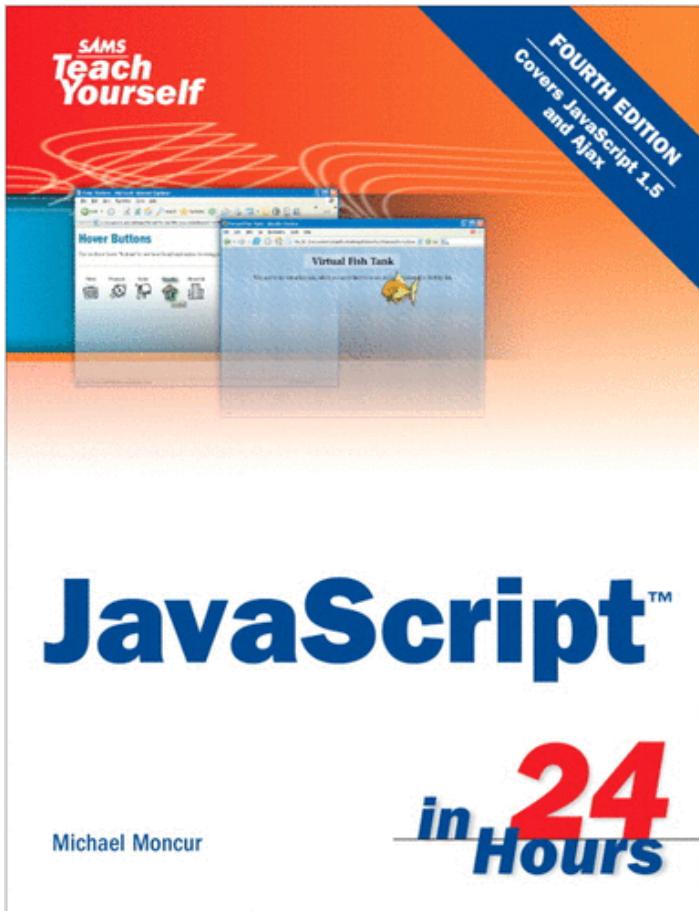
- **European Space Agency's Ariane 5 rocket** (1996): an unhandled arithmetic overflow in the engine steering software caused its crash, costing \$7 billion
- **Resorts World Casino** (2016): a casino machine printed a prize ticket of \$42,949,672.76

Root Causes of Security Problems

Why is there a **big security** challenge?

- **Functionality:** still the **primary concern** during design and implementation
 - Security is a **secondary goal**
 - Features pay the bills (typically)
- Unavoidable **human mistakes:**
 - (Lack of) awareness of security problems
 - Careless programmers
- **Complex modern computing systems:**
 - Many of the “bugs” are very *simple* and seem easy to prevent. But programs for complex systems are **large**, e.g. Window XP has 45 millions SLOC (source line of codes)
http://en.wikipedia.org/wiki/Source_lines_of_code.
 - Large **attack surface** as well

Programming can be Easy, but Good Programming Isn't So

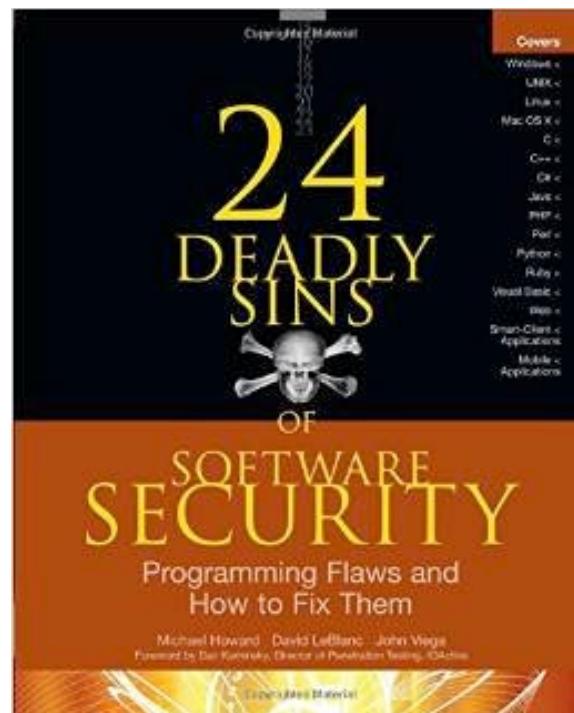


- Maybe enough for learning basic functionality
- **Never enough** for learning *subtle implications* of functionalities
- Result: programs can **do more** than you expect!

Recommended References for Secure Programming

Some well-known references:

- Michael Howard and David LeBlanc, *Writing Secure Code*, 2nd ed, Microsoft Press, 2002
- Michael Howard, David LeBlanc, and John Viega, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, McGraw-Hill, 2010



Optional

7.2 Computer Architecture Background

- 7.2.1 Code vs data, program counter
- 7.2.2 Stack (a.k.a execution stack, call stack)
- 7.2.3 Control flow integrity

7.2.1 Code vs Data, Program Counter

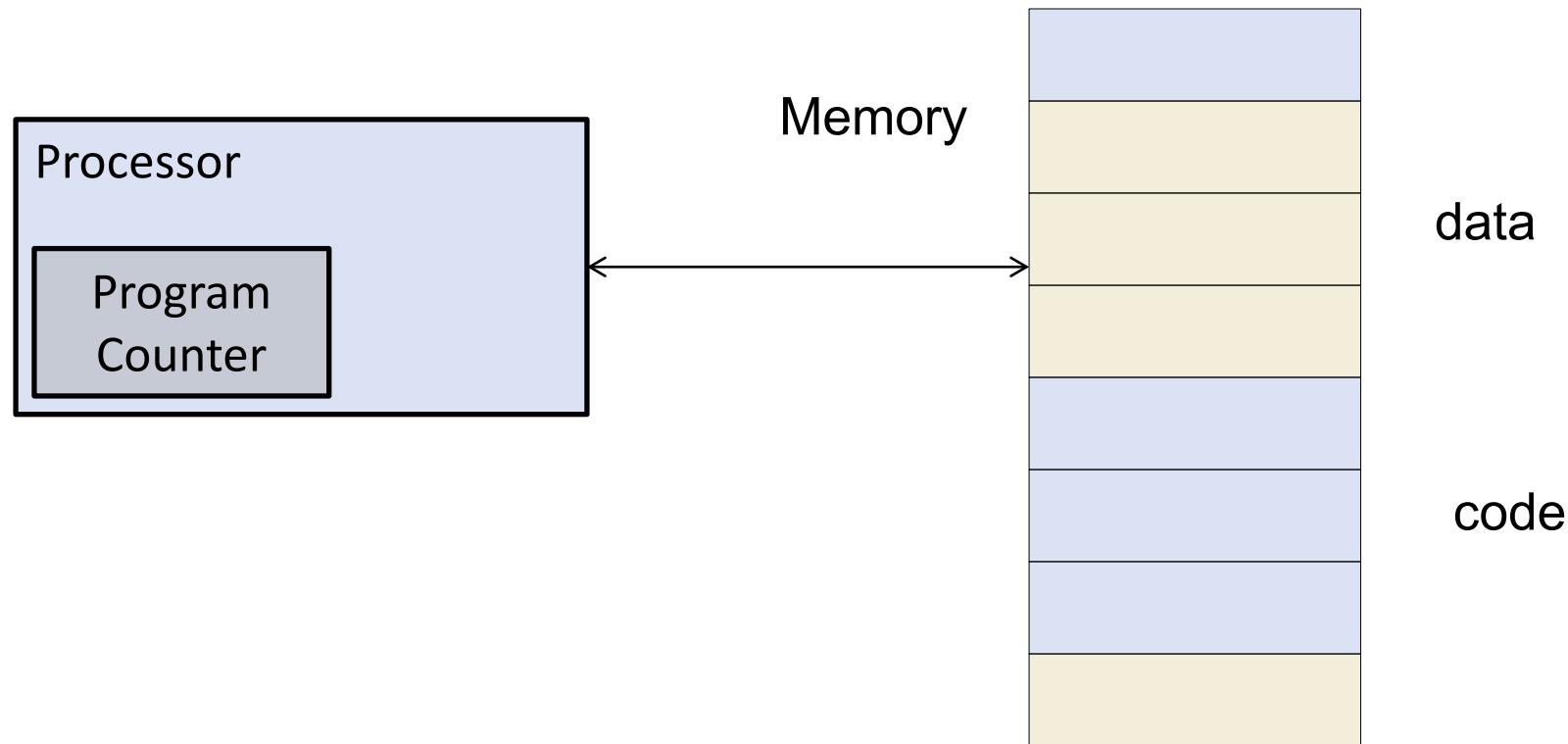
Code vs Data in Modern Computers

Modern computers are based on the

Von Neumann computer architecture:

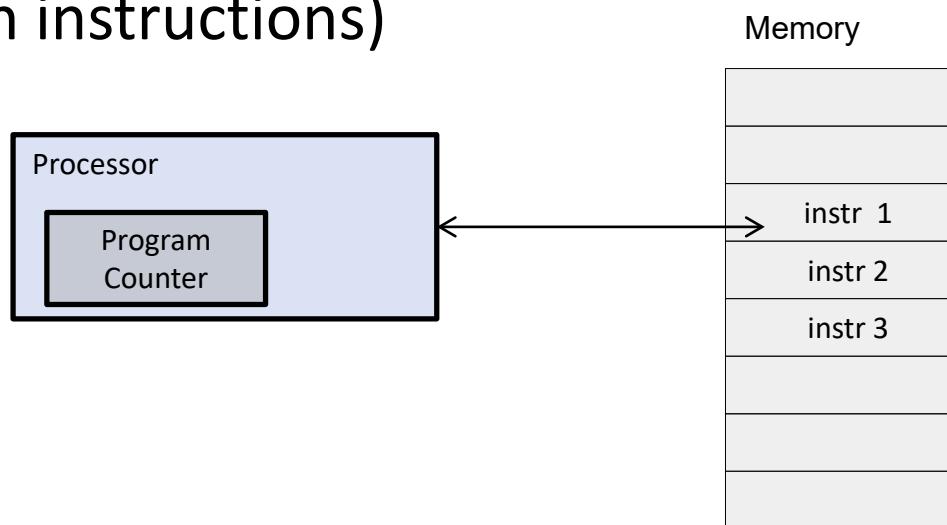
- The code and data are stored **together** in the memory
- There are **no clear distinction** of code and data.
- This is in contrast to the ***Harvard architecture***,
which has hardware components that separately store
code and data
- **Serious implication:** programs may be tricked into
treating input data as code: basis for all
code-injection attacks!

Code vs Data in Modern Computers



Control Flow

- The ***program counter*** (aka ***Instruction Pointer***):
a register (i.e. small & fast memory within the processor)
that stores the address of the next instruction
- After an instruction is completed, the processor **fetches**
the next instruction from the address stored in
the program counter
- After the next instruction is fetched, the program counter
automatically **increases** by 1 (assuming a system with
fixed-length instructions)



Control Flow

- During execution, besides getting **incremented**, the *program counter* can also be **changed**, for example*, by:

1. Direct jump:

Replaced with a *constant value* specified in the instruction

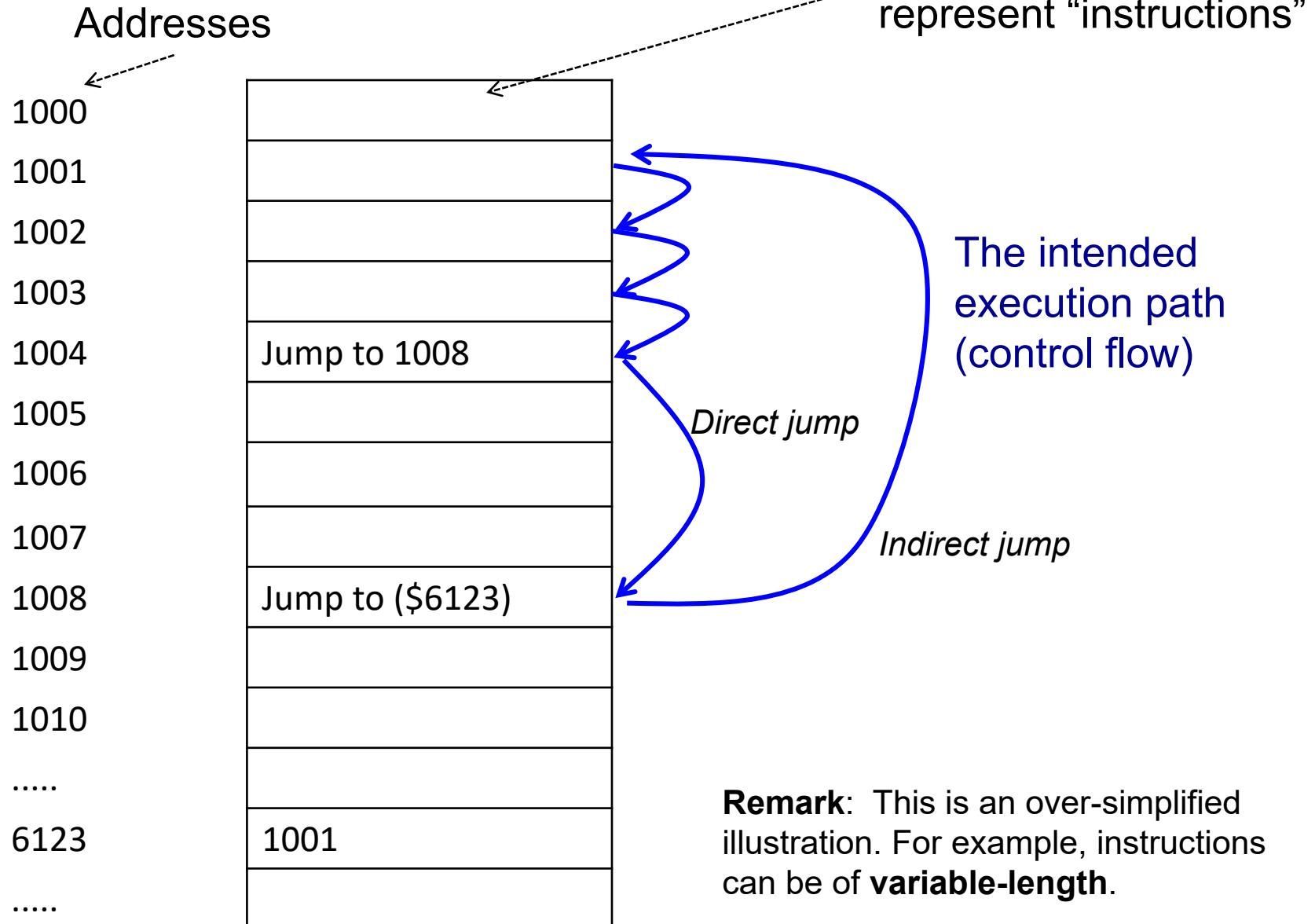
2. Indirect jump:

Replaced with a *value fetched from the memory*

(Note that there are many different forms of indirect jump)

*: For simplicity in this module, we omit conditional branch as well as call/return here

Illustration of Control Flow



7.2.2 Stack (aka Execution Stack, Call Stack)

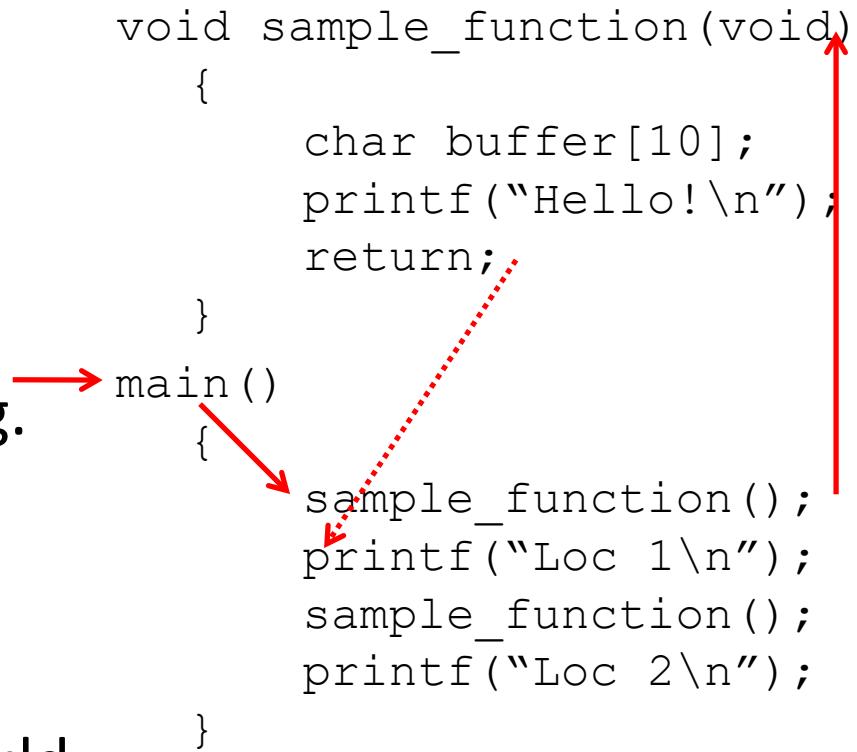
See: https://en.wikipedia.org/wiki/Call_stack

Functions and Their Executions

- **Functions** break code into smaller pieces:
 - Facilitate modular design and code reuse
- A function can be called in **many** program locations, e.g. 2, 10, 100, ... times (e.g. recursive function)
- **Question 1:** how does the program know where it should continue after it finishes?
- **Question 2:** where are the function's *arguments* and *local variables* stored?

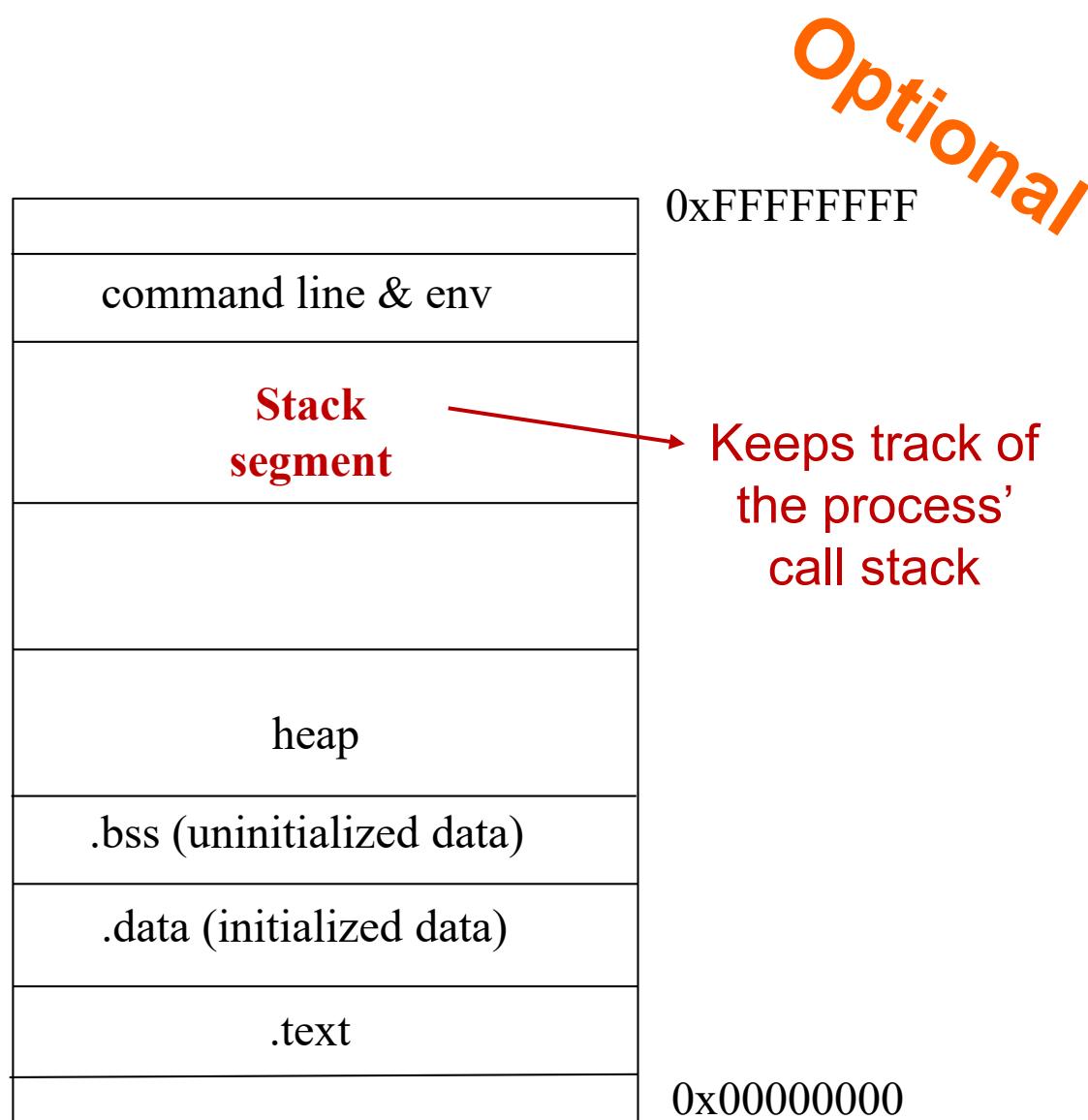
```
void sample_function(void)
{
    char buffer[10];
    printf("Hello!\n");
    return;
}

main()
{
    sample_function();
    printf("Loc 1\n");
    sample_function();
    printf("Loc 2\n");
}
```



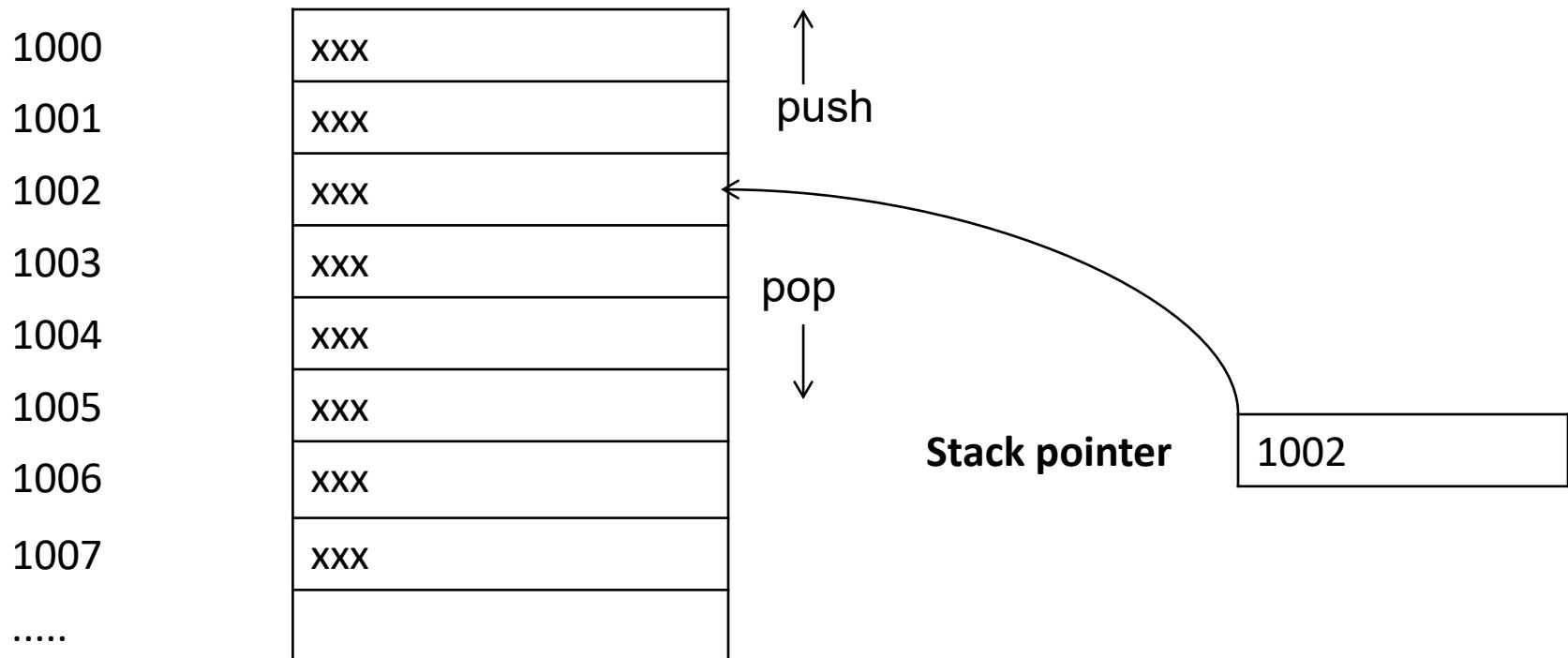
Process Memory Layout

- Simplified Linux process memory showing various *segments*:
- (Optional:
http://dirac.org/linux/gdb/02a-Memory_Layout_And_The_Stack.php)



Call Stack

- Call stack is a **data structure in the memory** (*not* in a separate hardware): stores important information of a running process
- **Last in, first out (LIFO)**: with push(), pop(), top() operations
- The location of the top element is referred to by the **stack pointer**



In this example, the stack grows “**upward**”, **but** from **high addresses to low addresses**. It is possible to have stack that grows downward.

Call Stack

- During a program execution, a stack is used to keep tracks of:
 - **Control flow information**: i.e. return addresses
 - **Parameters** passed to functions
 - **Local variables** of functions
- Each call of a function pushes an ***activation record (stack frame)*** to the stack, which contains:
passed parameters, return address, pointer to the previous stack frame, and function local variables
- **Note:** this stack is known as ***call stack***
In the context of system security, very often it is simply called the “***stack***”.
Sample expression: “smashing the *stack* for fun and profit”.

Illustration: A Function Call

When a function is called, the parameters, return address, and local variables are “pushed” into the stack

Consider the following C program segment:

```
int test(int a) {  
    int b = 1;  
    ...  
}  
  
int main() {  
    test(5);  
    ...  
}
```

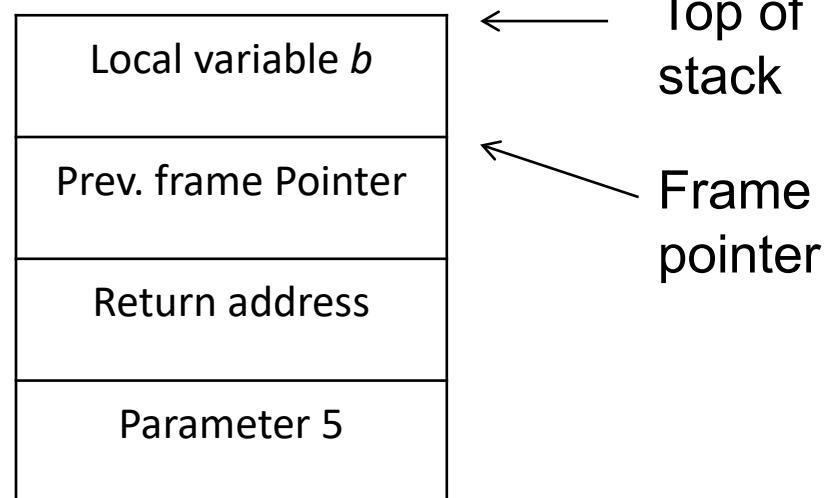
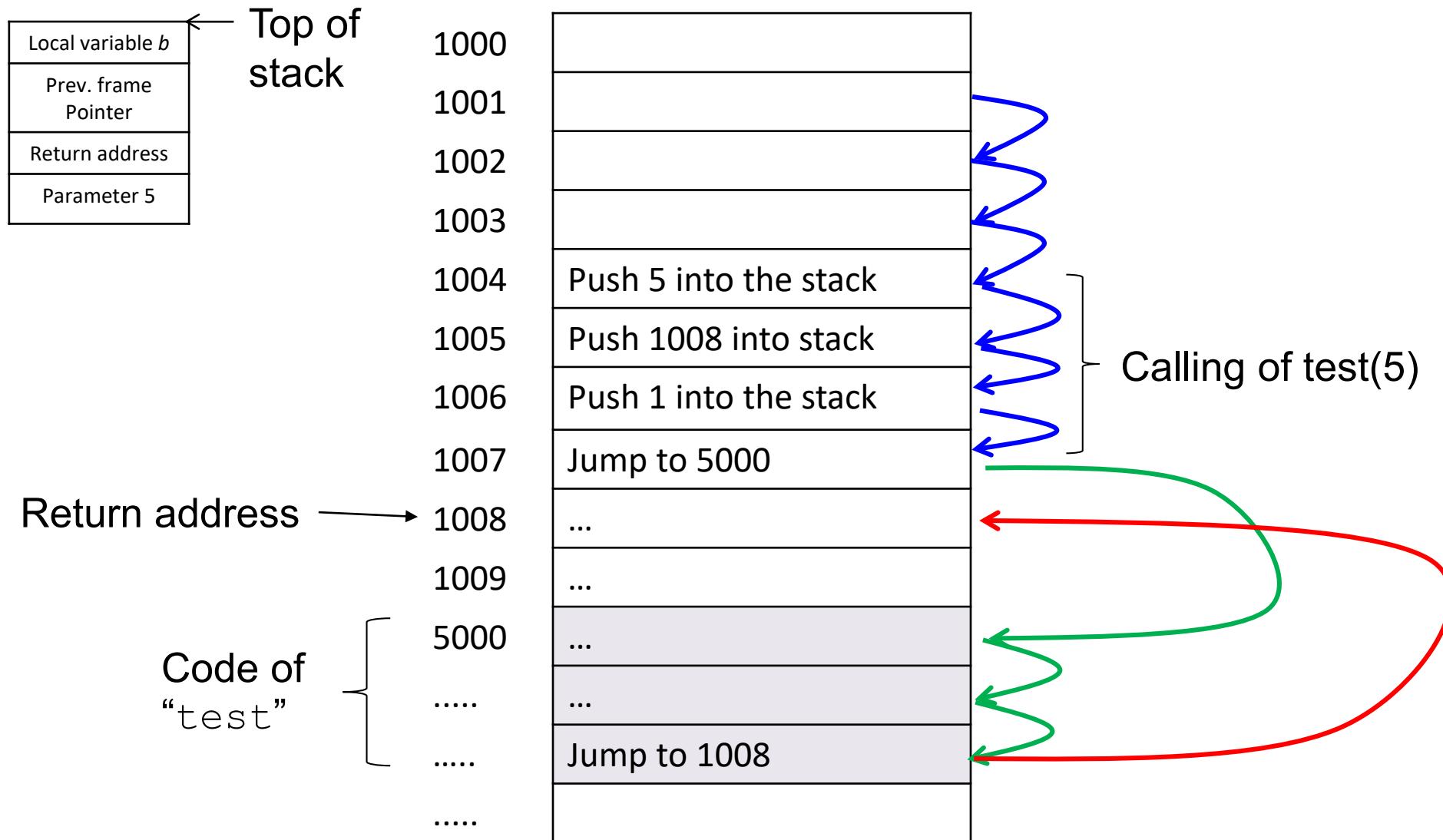


Illustration: A Function Call

When the function `test(5)` is invoked, the following are carried out:

- (1) Some values are pushed into the stack:
the parameter (i.e. 5), the return address, the previous frame pointer, and the value of the local variable *b* (i.e. 1)
- (2) The control flow jumps into the code of “`test`”
- (3) Execute “`test`”
- (4) After “`test`” is completed, the stack frame is popped from the stack
- (5) The control flow jumps into the restored return address

(Simplified) Illustration*



*: This slide gives a simplified view. Actual implementation includes “function return value”, and a “frame pointer”.
For more details, see: <http://www.tenouk.com/Bufferoverflowc/Bufferoverflow2a.html> or
https://en.wikipedia.org/wiki/Stack_buffer_overflow.

7.2.3 Control Flow Integrity

Treating Code as Data: Security Implications

- You have seen how the call stack stores a *return address* (*i.e. location of a to-be-executed instruction*) as **data** in the memory
- In fact, the **instruction** itself is stored as data in the memory
- The flexibility of treating code as data is useful, but it leads to **many security issues**
- Attacker could compromise a **process' execution integrity** by either modifying the process' **code** or the **control flow**
- It is difficult for the system to distinguish those malicious pieces of code from benign data

Compromising Memory Integrity

- In general, it is **not so easy** for an attacker to compromise memory integrity
- For **example**, consider an attacker who can only remotely communicate with the targeted Web server via HTTP.
How *can* he maliciously write to the web-server's memory?
- One way for the attacker to gain that capability is by:
exploiting some vulnerabilities so as to “trick” the victim process to **write to** some of its memory locations,
e.g. via a “**buffer overflow**” attack
- The above mechanisms typically have **some restrictions**:
for example, the attacker can only write to a small number of memory, or can only write a sequence of consecutive bytes. Hence, the attack has to be extremely “**surgical**”.

Possible Attack Mechanisms

Assuming that the attacker has the capability to **write** to some memory locations, the attacker could:

(Attack 1) Overwrite **existing execution code portion** with malicious code

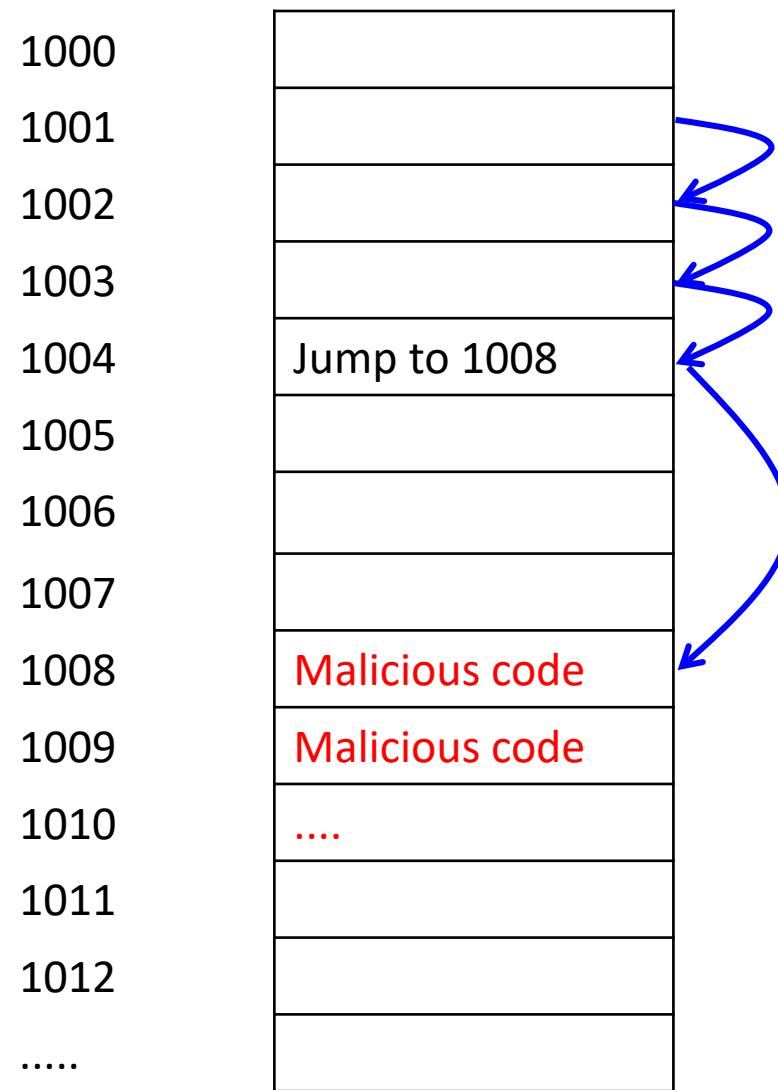
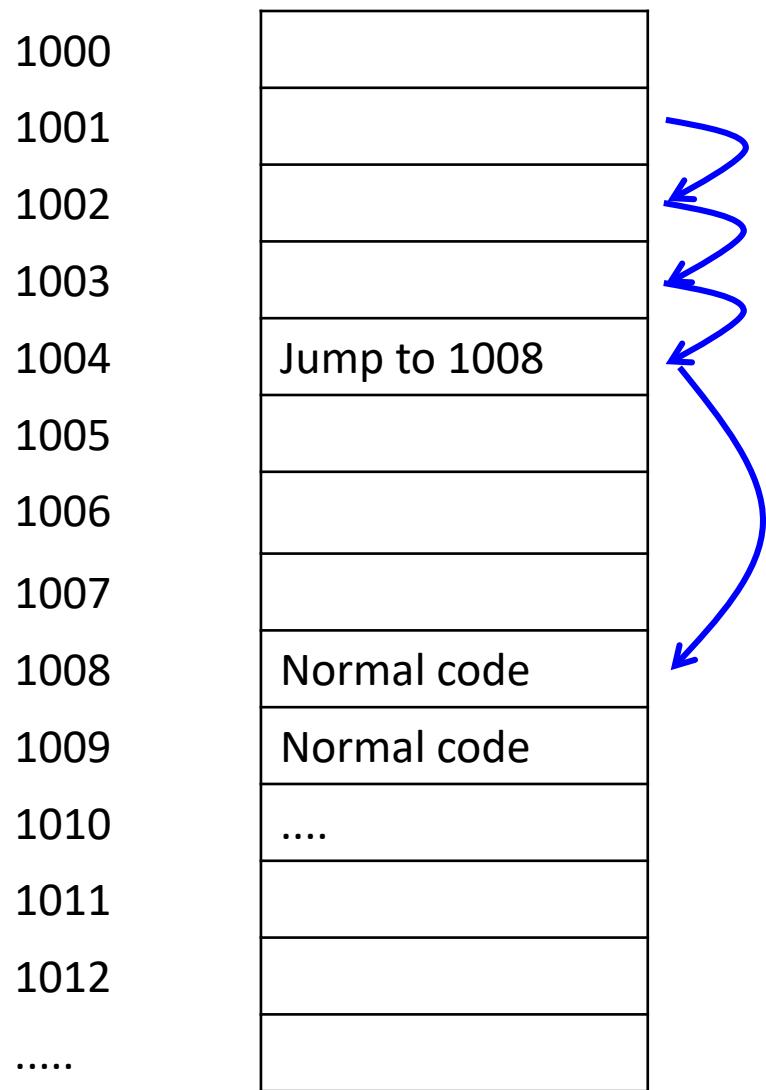
(Attack 2) Overwrite a piece of **control-flow information**:

(2a) Replace a **memory location** storing a code address that is used by a *direct jump*

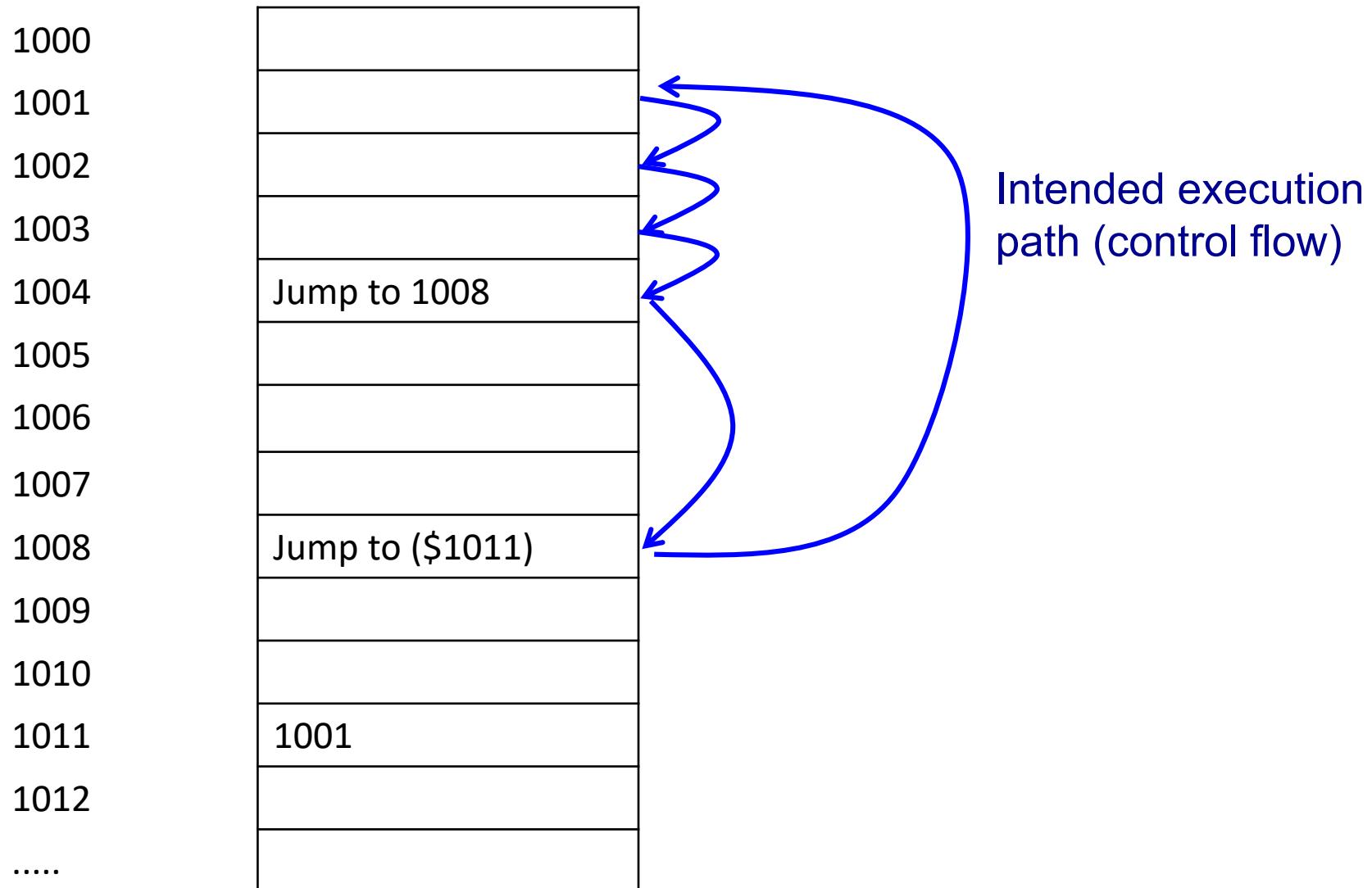
(2b) Replace a **memory location** storing a code address that is used by an *indirect jump*

The three attacks above are illustrated in the next few slides

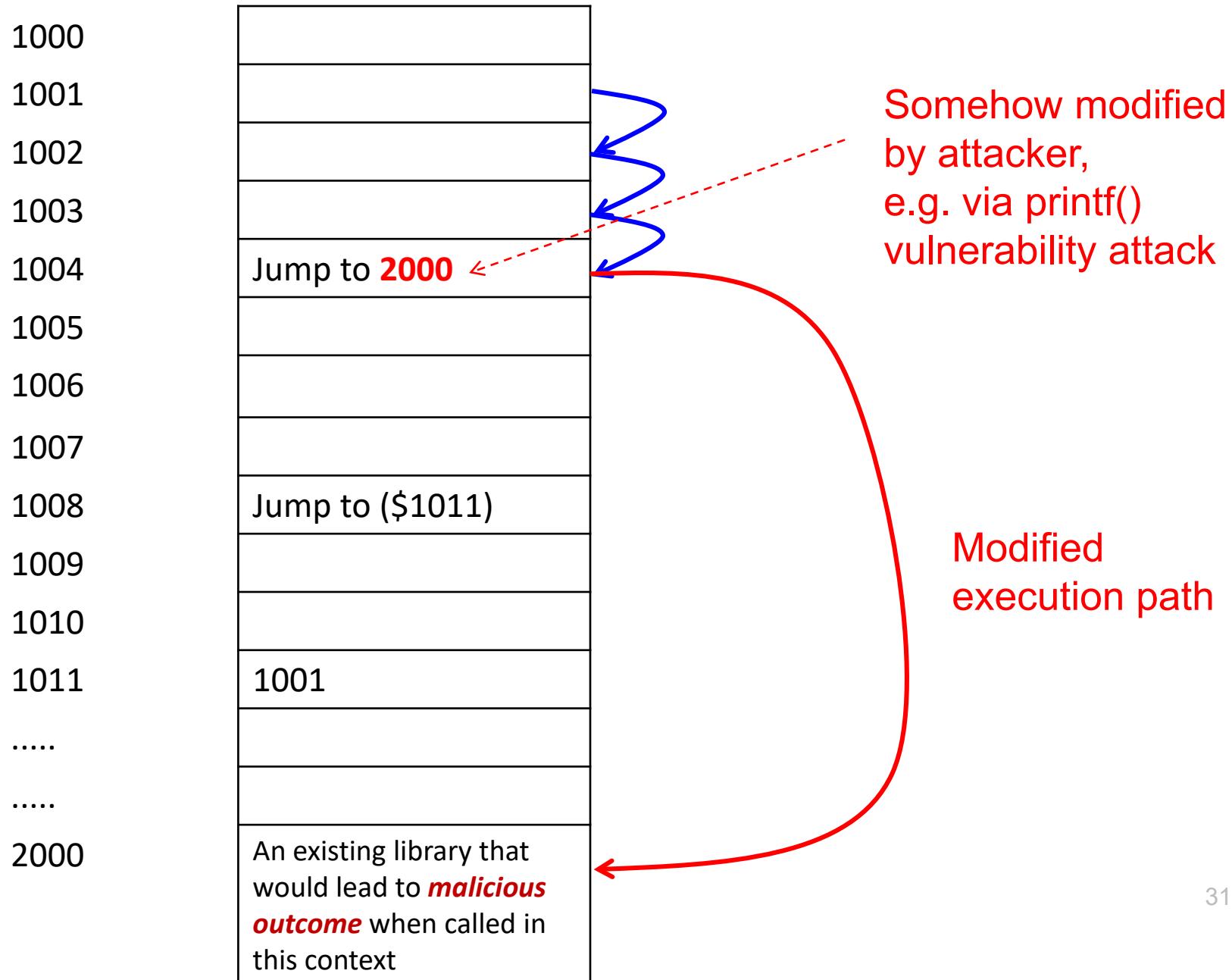
Attack 1 (Replace Existing Code)



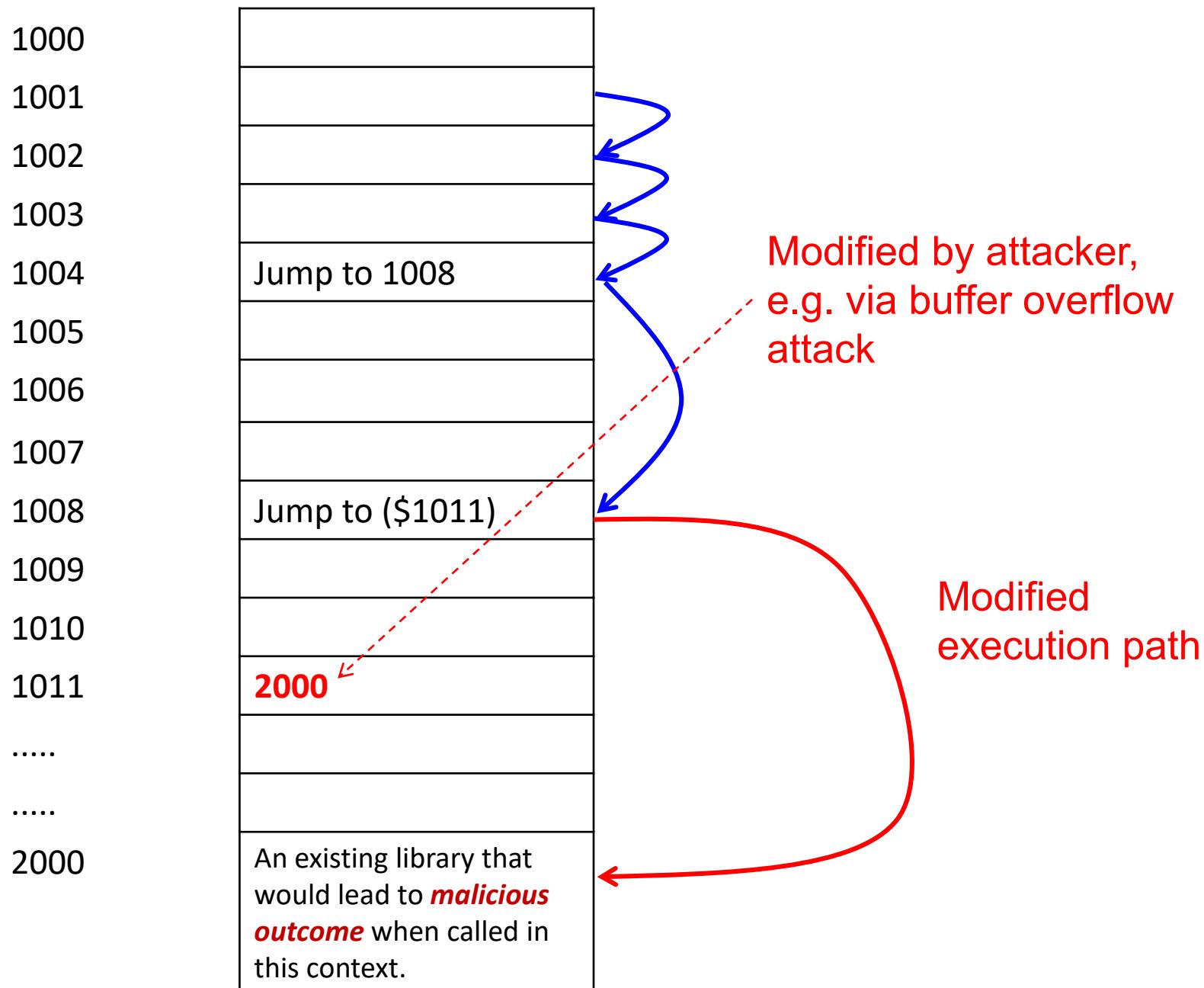
Attack 2a & 2b: Normal Control Flow before Being Attacked



Attack 2a (Replace Memory Location that Stores a Code Address)



Attack 2b (Replace Memory Location that Stores a Code Address)



7.3 printf() and Format String Vulnerability

Read Wiki http://en.wikipedia.org/wiki/Uncontrolled_format_string

Read https://www.owasp.org/index.php/Format_string_attack

For more details, see:

http://www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf

printf() Function

- printf () is a C function for **formatting output**
- It is special in that it can take in *any* number of arguments: one, two, or more arguments
- General format as written in its function definition:
`int printf(const char* format, ...);`
- Sample usage with two arguments:
`printf (format, s);`
where **format** specifies a format string, and
s is the variable to be displayed
- Hence, `printf ("The value of tmp is %d.\n", tmp);`
would display the following if tmp contains the value 100:
The value of tmp is 100.

printf() Format Specifiers

- The special symbol “%d” indicates the **type** of the variable
- Example:

```
printf ( "1st string is %s, 2nd string is %s", s1, s2);
```

Hence, printf () would:

1. Display “1st string is ”
2. Look up for the 2nd parameter **in the call stack** and display its value
3. Display “2nd string is ”
4. Look up for the 3rd parameter **in the call stack** and display its value

printf() Format Specifiers

- Some common format specifiers:
 - %d: decimal (int)
 - %u: unsigned decimal (unsigned int)
 - %x: hexadecimal (unsigned int)
 - %s: string ((const) (unsigned) char *)
 - %n: number of bytes written so far, (* int)
- Note that %s and %n are passed as a **reference**

The Case of Missing Argument(s)

- When only one parameter is supplied:

```
printf ("hello world");
```

Then, only "hello world" will be displayed.

- If there happens to have "%d" in the **first parameter**, e.g.:

```
printf ("hello world %d");
```

Then, printf() will **search for the 2nd parameter** in the stack to be displayed.

Then what being displayed could be:

hello world 15

- ***Any security implications??***

Example of a Vulnerable Program with Missing Argument(s)

- A format string variable **t** is supplied by the user (an attacker)
- The program calls `printf()` using only one parameter **t**
- The attacker **can get more information** by carefully designing the string **t**

```
#include <stdio.h>

int main()
{
    char t[100];
    scanf ("%s", t);
    printf (t);
    return 0;
}
```

Declare a string of 100 characters

Read in a string and
store it in *t*

Display the string *t*

How Such Vulnerability Can be Exploited

- If a program is vulnerable, the attacker might be able to:
 - Obtain **more information** of program's call stack (e.g. using "%d.%d.%d" or "%08x.%08x.%08x")
 - Cause the program to **crash** (e.g. using "%s%s%s%s%s%"):
 - %s will fetch a number from the stack, treat this number as **an address**, and print out the memory contents pointed by this address as **a string** (until a NULL char)
 - Typically one fetched number is **not an address**, and thus the memory pointed by this number does not exist, therefore the program will crash
 - **Modify** program's memory content (e.g. using %n):
not covered in this module

How Such Vulnerability Can be Exploited

- How the vulnerability can be exploited?
- In a **multi-user setting**:
if the program has an *elevated* privilege (i.e. set-UID),
a user (attacker) might be able to obtain ***system-level information***
- In a **client-server setting**:
if the server program is vulnerable, the client (attacker)
might be able to **submit a request** and ***obtain sensitive information*** (e.g. a secret key)

Simple Preventive Measures

Avoid taking a user input as format string:

- `printf (t)` Where `t` is supplied by the user, then it is potentially **insecure**
- `printf (f, t)` Where `f` is **not** supplied by the user, then it is generally **okay**

Many modern compilers can also **statically check** format strings and **produce warnings** for dangerous or suspect formats

E.g.: in GNU Compiler Collection, the relevant compiler flags are:

`-Wall, -Wformat, -Wno-format-extra-args,`
`-Wformat-security, -Wformat-nonliteral, and`
`-Wformat=2`

Heartbleed Bug: Graphical Illustration of Over-Read Request

Source: <http://xkcd.com/1354/>

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).



This page about books uses a secure connection using key "4538538374224". User Meg wants these 6 letters: POTATO. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435. User Eve (administrator) wants to set server's master key to "14835038534".



This page about books uses a secure connection using key "4538538374224". User Meg wants these 6 letters: POTATO. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435. User Eve (administrator) wants to set server's master key to "14835038534".



SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about bees in car why". Note: Files for IP 375.381. 83.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 34 connections open. User Brendan uploaded the file testfile (contents: 434ba9f2e0c8b0f7a91316f8)



HMM...



BIRD



SERVER, ARE YOU STILL THERE?
IF SO, REPLY "HAT" (500 LETTERS).



User Olivia from London wants pages about bees in car why". Note: Files for IP 375.381. 83.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 34 connections open. User Brendan uploaded the file testfile (contents: 434ba9f2e0c8b0f7a91316f8)



SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about bees in car why". Note: Files for IP 375.381. 83.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 34 connections open. User Brendan uploaded the file testfile (contents: 434ba9f2e0c8b0f7a91316f8)



HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "Colibriflame". Dear Weber requested new

User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "Colibriflame". Dear Weber requested new



*Get Exam Results
earlier via SMS
on 24 Dec 2019!*



For more
information



Subscribe via NUS EduRec System
by 10 Dec 2019.



<https://myportal.nus.edu.sg/studentportal/academics/all/scheduleoffresultsrelease.html>

Reading:
See [PF] pg. 131-166, or
See [Gollmann] 10.1, 10.2, 10.3

Lecture 8: Software Security (Part II)

8.1 Data Representation & Security

8.2 Buffer Overflow

8.3 Integer Overflow

8.4 Code/Script Injection

8.5 Undocumented Access Points

8.1 Data Representation & Security

Data Representation Problem

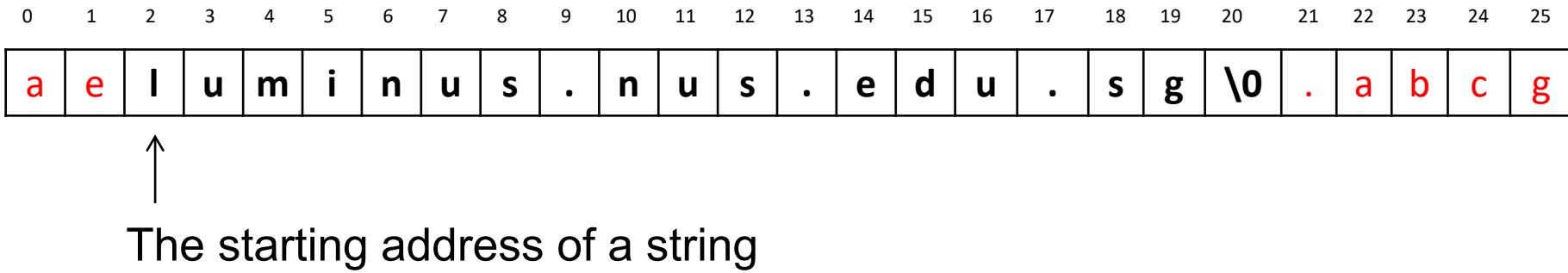
- **Different** parts of a program/system adopts **different** data representations
- Such **inconsistencies** could lead to vulnerability
- A sample vulnerability is CVE-2013-4073:
“Ruby’s SSL client implements **hostname identity check**, but it does not properly handle **hostnames in the certificate** that contain ***null bytes***.”

(Read <https://www.ruby-lang.org/en/news/2013/06/27/hostname-check-bypassing-vulnerability-in-openssl-client-cve-2013-4073/.>)

- **String** is a very important data representation type:
 - It has a **variable** length
 - How can we **represent** a string?

String Representations

- In C, `printf()` adopts an **efficient** representation:
 - The length is *not* explicitly stored
 - **The first occurrence of the null character** (i.e. byte with value 0) indicates the **end of the string**, thus *implicitly* giving the length



- Note that *not* all systems adopt this convention:
NULL-termination vs non NULL-termination representation

Exploitable Vulnerability 1: NULL-Byte Injection

- A CA **may accept** a host name containing null character
- For example: luminus.nus.edu.sg\0.attacker.com
- A verifier who uses **both** string-representation conventions to verify the certificate **could be** vulnerable
- Consider a browser implementation that does the following:
 1. Verify a certificate: based on **non NULL-termination** representation
 2. Compare the name in the certificate and the name entered by user: based on the **NULL-termination** representation
- Now, there could be an **attack** as described on the next slide!

A Sample Attack (on LumiNUS)

1. The attacker registered the following **domain name**, and purchased a **valid certificate** with the domain name from some CA:
luminus.nus.edu.sg\0.attacker.com
2. The attacker set up a **spoofed LumiNUS** website on another web server
3. The attacker **directed** a victim to the **spoofed web server** (e.g. by controlling the physical layer or social engineering)
4. When visiting the spoofed web server, the victim's browser:
 - Finds that the Web server in the certificate is **valid**: based on the ***non*** NULL-termination representation
 - Compares and displays the address as **luminus.nus.edu.sg**: based on NULL-termination representation
 -

Comparison: A Normal Web-Spoofing Attack (on LumiNUS)

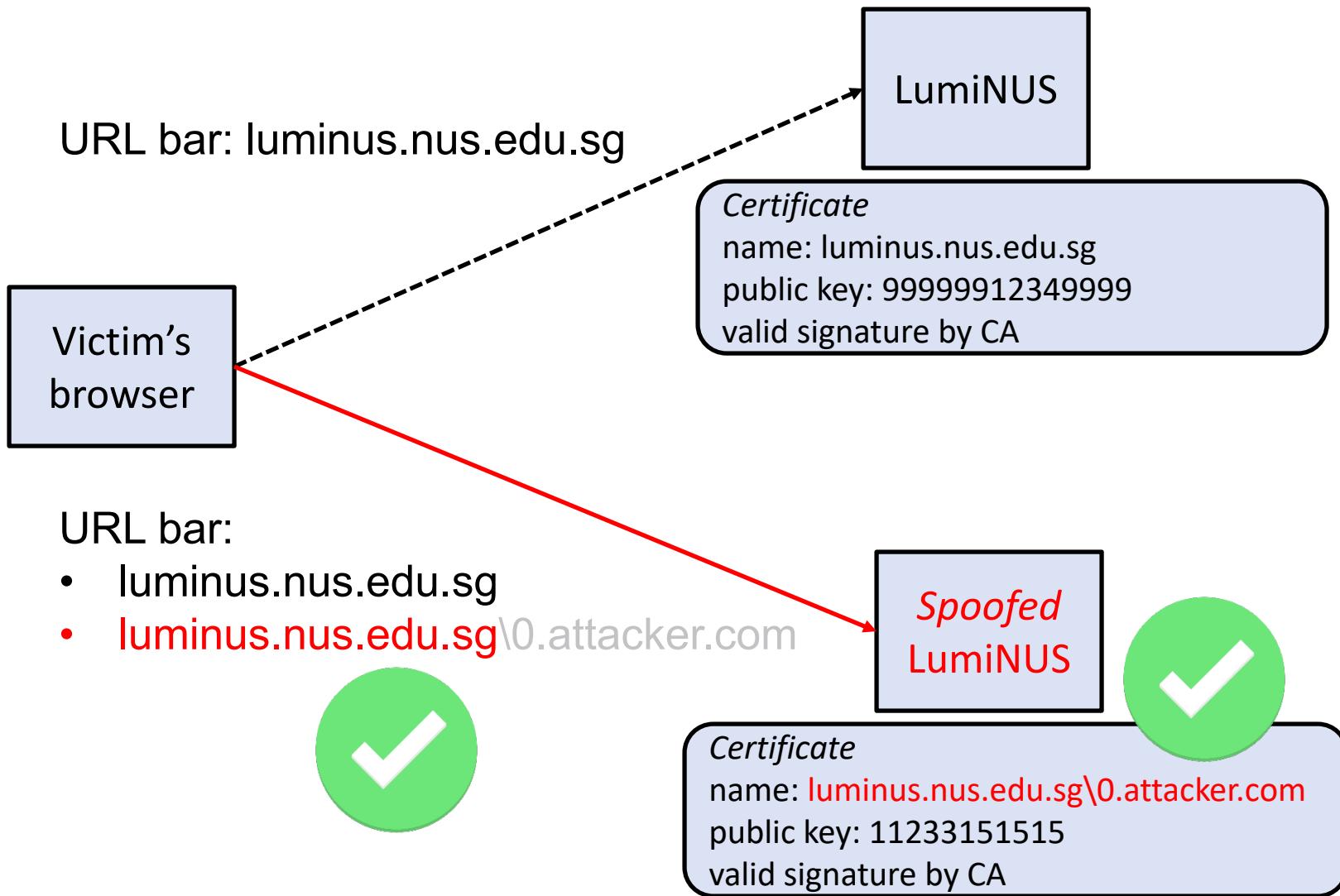
What if it is just a **normal web-spoofing** attack scenario?

Even if the attacker manages to redirect the victim to the spoofed web server (Step 3), a **careful** user would notice that *either*:

- The address displayed in the browser's address bar is **not** LumiNUS; or
- The address bar displays luminus.nus.edu.sg, but the TLS/SSL authentication protocol **rejects** the connection (i.e. “certificate is **not** trusted”)

Hence, the attack on the previous slide is **much more dangerous**: it can **trick** all browser users!

A Sample Attack (on LumiNUS): Illustration



CVE-2013-4073:

Hostname check bypassing vulnerability in SSL client (CVE-2013-4073)

Posted by nahi on 27 Jun 2013

A vulnerability in Ruby's SSL client that could allow man-in-the-middle attackers to spoof SSL servers via valid certificate issued by a trusted certification authority.

This vulnerability has been assigned the CVE identifier CVE-2013-4073.

Summary

Ruby's SSL client implements hostname identity check but it does not properly handle hostnames in the certificate that contain null bytes.

Details

`OpenSSL::SSL.verify_certificate_identity` implements RFC2818 Server Identity check for Ruby's SSL client but it does not properly handle hostnames in the subjectAltName X509 extension that contain null bytes.

Existing code in `lib/openssl/ssl.rb` uses `OpenSSL::X509::Extension#value` for extracting identity from subjectAltName. `Extension#value` depends on the OpenSSL function `X509V3_EXT_print()` and for dNSName of subjectAltName it utilizes `sprintf()` that is known as null byte unsafe. As a result `Extension#value` returns 'www.ruby-lang.org' if the subjectAltName is 'www.ruby-lang.org\0.example.com' and `OpenSSL::SSL.verify_certificate_identity` wrongly identifies the certificate as one for 'www.ruby-lang.org'.

When a CA that is trusted by an SSL client allows to issue a server certificate that has a null byte in subjectAltName, remote attackers can obtain the certificate for 'www.ruby-lang.org\0.example.com' from the CA to spoof 'www.ruby-lang.org' and do a man-in-the-middle attack between Ruby's SSL client and SSL servers.

What is **CVE**?

What is **zero-day vulnerability**?

What is an **exploit**?

Background: ASCII Character Encoding

- **ASCII** (American Standard Code for Information Interchange) **character encoding**: a character-encoding standard for electronic communication
- Encodes **128 characters** into **7-bit integers** (see the ASCII chart on the next slide):
 - 95 printable characters: digits, letters, punctuation symbols
 - 33 non-printing (control) characters
- **Extended ASCII** (EASCII or high ASCII) character encodings, which comprises:
 - The standard 7-bit ASCII characters
 - Plus *additional characters*
 - See: https://en.wikipedia.org/wiki/Extended_ASCII

ASCII Chart

ASCII printable code chart [\[edit\]](#)

| Binary | Oct | Dec | Hex | Glyph |
|----------|-----|-----|-----|---------|
| 010 0000 | 040 | 32 | 20 | (space) |
| 010 0001 | 041 | 33 | 21 | ! |
| 010 0010 | 042 | 34 | 22 | " |
| 010 0011 | 043 | 35 | 23 | # |
| 010 0100 | 044 | 36 | 24 | \$ |
| 010 0101 | 045 | 37 | 25 | % |
| 010 0110 | 046 | 38 | 26 | & |
| 010 0111 | 047 | 39 | 27 | ' |
| 010 1000 | 050 | 40 | 28 | (|
| 010 1001 | 051 | 41 | 29 |) |
| 010 1010 | 052 | 42 | 2A | * |
| 010 1011 | 053 | 43 | 2B | + |
| 010 1100 | 054 | 44 | 2C | , |
| 010 1101 | 055 | 45 | 2D | - |
| 010 1110 | 056 | 46 | 2E | . |
| 010 1111 | 057 | 47 | 2F | / |
| 011 0000 | 060 | 48 | 30 | 0 |
| 011 0001 | 061 | 49 | 31 | 1 |
| 011 0010 | 062 | 50 | 32 | 2 |
| 011 0011 | 063 | 51 | 33 | 3 |
| 011 0100 | 064 | 52 | 34 | 4 |
| 011 0101 | 065 | 53 | 35 | 5 |
| 011 0110 | 066 | 54 | 36 | 6 |
| 011 0111 | 067 | 55 | 37 | 7 |
| 011 1000 | 070 | 56 | 38 | 8 |
| 011 1001 | 071 | 57 | 39 | 9 |
| 011 1010 | 072 | 58 | 3A | : |
| 011 1011 | 073 | 59 | 3B | ; |
| 011 1100 | 074 | 60 | 3C | < |
| 011 1101 | 075 | 61 | 3D | = |
| 011 1110 | 076 | 62 | 3E | > |
| 011 1111 | 077 | 63 | 3F | ? |

| Binary | Oct | Dec | Hex | Glyph |
|----------|-----|-----|-----|-------|
| 100 0000 | 100 | 64 | 40 | @ |
| 100 0001 | 101 | 65 | 41 | A |
| 100 0010 | 102 | 66 | 42 | B |
| 100 0011 | 103 | 67 | 43 | C |
| 100 0100 | 104 | 68 | 44 | D |
| 100 0101 | 105 | 69 | 45 | E |
| 100 0110 | 106 | 70 | 46 | F |
| 100 0111 | 107 | 71 | 47 | G |
| 100 1000 | 110 | 72 | 48 | H |
| 100 1001 | 111 | 73 | 49 | I |
| 100 1010 | 112 | 74 | 4A | J |
| 100 1011 | 113 | 75 | 4B | K |
| 100 1100 | 114 | 76 | 4C | L |
| 100 1101 | 115 | 77 | 4D | M |
| 100 1110 | 116 | 78 | 4E | N |
| 100 1111 | 117 | 79 | 4F | O |
| 101 0000 | 120 | 80 | 50 | P |
| 101 0001 | 121 | 81 | 51 | Q |
| 101 0010 | 122 | 82 | 52 | R |
| 101 0011 | 123 | 83 | 53 | S |
| 101 0100 | 124 | 84 | 54 | T |
| 101 0101 | 125 | 85 | 55 | U |
| 101 0110 | 126 | 86 | 56 | V |
| 101 0111 | 127 | 87 | 57 | W |
| 101 1000 | 130 | 88 | 58 | X |
| 101 1001 | 131 | 89 | 59 | Y |
| 101 1010 | 132 | 90 | 5A | Z |
| 101 1011 | 133 | 91 | 5B | [|
| 101 1100 | 134 | 92 | 5C | \ |
| 101 1101 | 135 | 93 | 5D |] |
| 101 1110 | 136 | 94 | 5E | ^ |
| 101 1111 | 137 | 95 | 5F | _ |

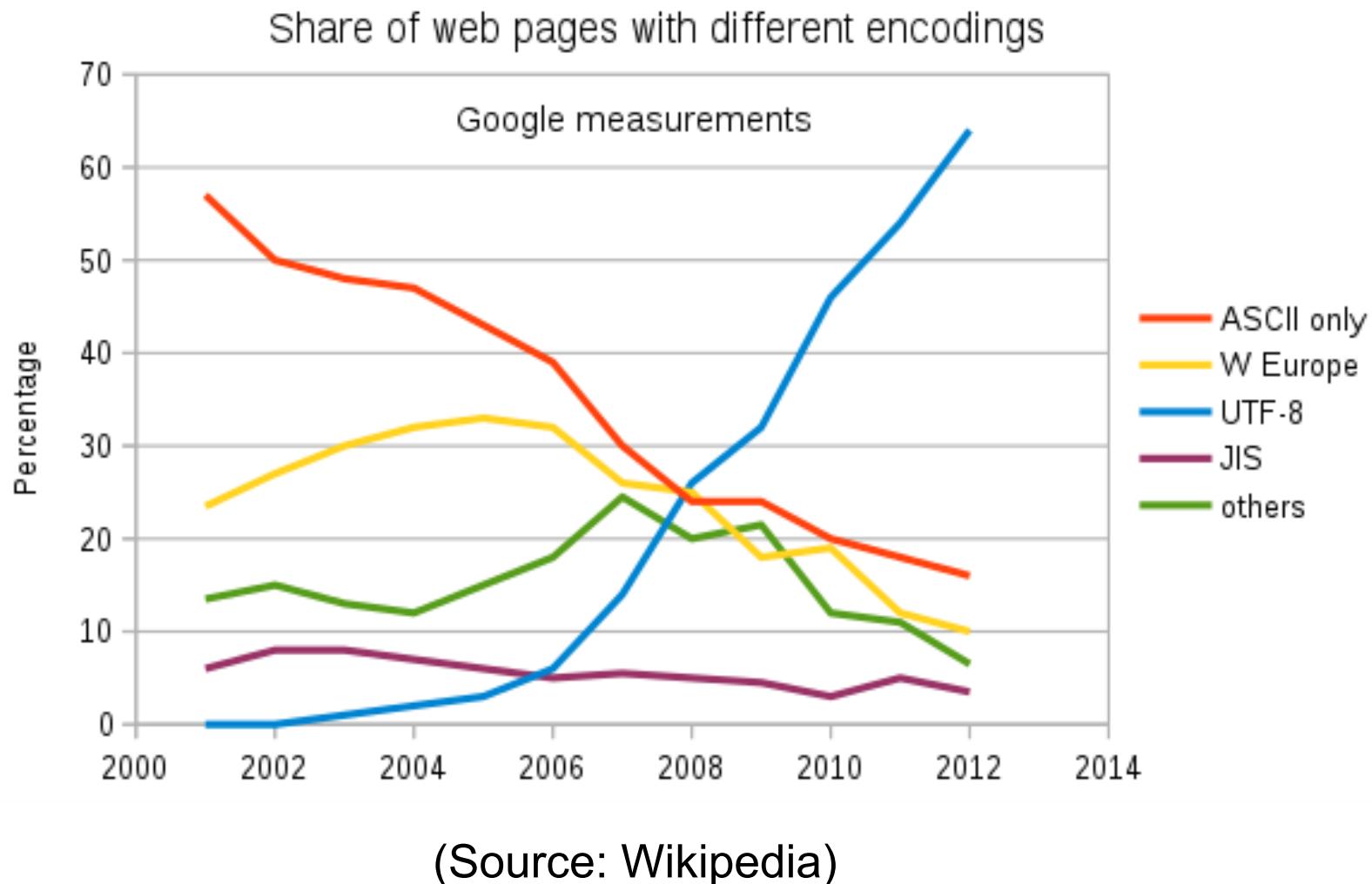
| Binary | Oct | Dec | Hex | Glyph |
|----------|-----|-----|-----|-------|
| 110 0000 | 140 | 96 | 60 | ` |
| 110 0001 | 141 | 97 | 61 | a |
| 110 0010 | 142 | 98 | 62 | b |
| 110 0011 | 143 | 99 | 63 | c |
| 110 0100 | 144 | 100 | 64 | d |
| 110 0101 | 145 | 101 | 65 | e |
| 110 0110 | 146 | 102 | 66 | f |
| 110 0111 | 147 | 103 | 67 | g |
| 110 1000 | 150 | 104 | 68 | h |
| 110 1001 | 151 | 105 | 69 | i |
| 110 1010 | 152 | 106 | 6A | j |
| 110 1011 | 153 | 107 | 6B | k |
| 110 1100 | 154 | 108 | 6C | l |
| 110 1101 | 155 | 109 | 6D | m |
| 110 1110 | 156 | 110 | 6E | n |
| 110 1111 | 157 | 111 | 6F | o |
| 111 0000 | 160 | 112 | 70 | p |
| 111 0001 | 161 | 113 | 71 | q |
| 111 0010 | 162 | 114 | 72 | r |
| 111 0011 | 163 | 115 | 73 | s |
| 111 0100 | 164 | 116 | 74 | t |
| 111 0101 | 165 | 117 | 75 | u |
| 111 0110 | 166 | 118 | 76 | v |
| 111 0111 | 167 | 119 | 77 | w |
| 111 1000 | 170 | 120 | 78 | x |
| 111 1001 | 171 | 121 | 79 | y |
| 111 1010 | 172 | 122 | 7A | z |
| 111 1011 | 173 | 123 | 7B | { |
| 111 1100 | 174 | 124 | 7C | |
| 111 1101 | 175 | 125 | 7D | } |
| 111 1110 | 176 | 126 | 7E | ~ |

Background: UTF-8 (Unicode Transformation Format 8-bit)

- **UTF-8**: a character encoding capable of encoding all **1,112,064** valid code points in **Unicode** using **one to four** 8-bit bytes
- **A *variable-length* encoding**: code points that tend to occur **more frequently** are encoded with **lower** numerical values, thus fewer bytes are used
- The **first 128 characters** of Unicode:
 - Correspond 1-to-1 with **ASCII**
 - Encoded using a **single octet** with the same binary value as ASCII:
Recall that there are 128 ASCII characters,
and each starts with the bit 0 in a single byte
- Hence, ASCII characters **remain *unchanged*** in UTF-8
- **Backward compatibility** with ASCII: UTF-8 encoding was defined for “Unicode” *on* systems that were designed for ASCII
- See: <https://en.wikipedia.org/wiki/UTF-8> for details

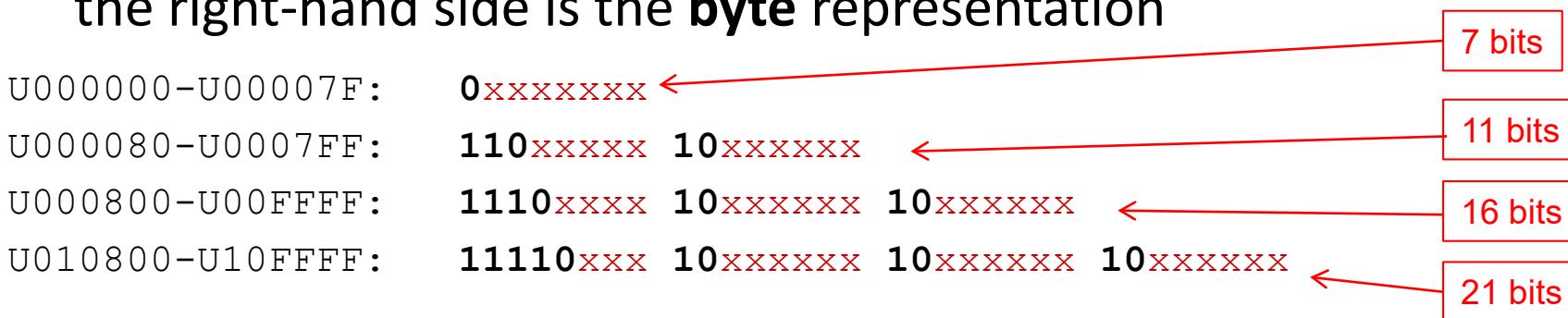
Background: UTF-8 Popularity

The **dominant** character encoding for the Web since 2009,
as of October 2019 accounts for **94.1%** of all Web pages



Exploitable Vulnerability 2: UTF-8 “Variant” Encoding Issue

- A **Unicode** character: referred to by "U+" & its hexadecimal digits
- The following are **byte representations** of Unicode characters:
the left-hand side is the **Unicode** representation,
the right-hand side is the **byte representation**



- Notice the **prefix bits** in the **first/leading byte** and **continuation byte(s)**
- The **xxx bits** are replaced by the **significant bits** of the code point of the respective Unicode character
- By the rules above, byte representation of a UTF-8 character is *unique*
- *However*, many implementations also accepts **multiple and longer “variants”** of a character! *Why is that so?*

Different Representations of the Same UTF-8 Character

- Consider the ASCII character ‘/’, whose ASCII code is:

0010 1111 = 0x2F

- Under UTF-8 definition, a **1-byte** 2F is a **unique** representation
- However, in many implementations, the following **longer variants** are also treated to be ‘/’:

(2-byte version) 11000000 10101111

(3-byte version) 11100000 10000000 10101111

(4-byte version) 11110000 10000000 10000000 10101111

- That is, all the above would be decoded to ‘/’
- Now, there could be an inconsistency between:
 - The **character verification** process; and
 - The **character usage(s)**: operations using the character

Potential Problem with UTF-8: A Sample Scenario

- In a typical file system, **files** are organized inside a **directory**
- Example: the **full path name** of a **file name** “`index.html`” is:
`/home/student/alice/public_html/index.html`
- Suppose a server-side program, upon receiving a string
`<file-name>` from a client, carry out the following steps:

Step 1: Append `<file-name>` to the **prefix (directory) string**:

`/home/student/alice/public_html/`
and take the concatenated string as string *F*

Step 2: Invoke a system call to **open** the file *F*,
and then **send** the file content to the client

Potential Problem with UTF-8: A Sample Scenario

- In the above example, the client can be any remote **public user** (similar to HTTP client)
- The original **intention**: the client can retrieve only files under the directory `public_html` → ***file-access containment***
- However, an attacker (the client) may send in this string:
`.../cs2107report.pdf`

Which file would be read and sent by the server?

- This is the file:
`/home/student/alice/public_html/.../cs2107report.pdf`
- This access **violates** the intended file-access containment
- To prevent this, the server may add an “**input validation**” step, making sure that “`.. /`” never appear as a substring in the input string: *is this check complete?*

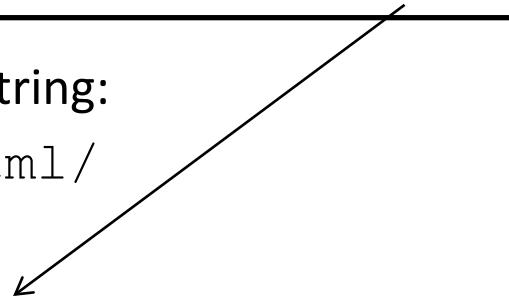
Added Input-Validation Step

Is this check “complete”?

Step 1: Append *<file-name>* to the prefix (directory) string:

/home/student/alice/public_html/

and take the concatenated string as string *F*



Step 1a: Checks that *<file-name>* does not contain the substring “*.. /*”;
Otherwise, quit

Step 2: Invoke a **system call** to open the file *F*,
and then send the file content to the client

Now, further suppose that the **system call in Step 2**:

1. Uses a convention that ‘%’ followed by two hexadecimal digits indicates **a single byte** (like **URL encoding**)
E.g.: In “/home/student/%61lice/”, %61 is to be replaced by **a**
2. Uses **UTF-8**

The Security Problem

- Then, the check carried out by Step 1a is ***incomplete***: it misses some cases!
- Any of the following string will **pass** the check in Step 1a, since it literally does not contain the substring “`../`”:

- (1) ..%2Fcs2107report.pdf
- (2) ..%C0%AFcs2107report.pdf
- (3) ..%E0%80%AFcs2107report.pdf
- (4) ..%F0%E0%80%AFcs2107report.pdf

- However, eventually, the filename will be **decoded** to:
`/home/student/alice/public_html/..../cs2107report.pdf`
- In general: a ***blacklisting-based filtering*** could be ***incomplete*** due to the “flexible use” of character encoding

Yet Another Example: IP Address

- Recall that the **4-byte IP address** is typically written as a string, e.g. “132.127.8.16”
- Consider a **blacklist** containing a lists of banned IP addresses, where each IP address is represented as 4 bytes
- A programmer wrote a **function BL()** :
 - Takes in 4 integers, where each integer is of the type “int” represented using **32 bits**
 - Checks whether the IP address represented by these 4 integers is in the black list
- In **C** language: `int BL(int a, int b, int c, int d)`
- **BL()** stores the blacklist as **4 arrays of integers** A, B, C, D:
Given the 4 input parameters a, b, c, d,
BL() simply searches for the existence of index *i* such that:
 $A[i] == a, B[i] == b, C[i] == c, \text{ and } D[i] == d$

Potential Problem

Now, a program that performs the **following checks** is vulnerable:

- (1) Get a string s from user
- (2) Extract 4 integers (each integer is of type **int**, i.e. 32-bits) from the string s , and let them be a, b, c, d :
If s does not follow the correct input format
(the correct format is 4 integers separated by ".") , then quit
- (3) Call `BL()` to check that that (a, b, c, d) is not in the black list;
Otherwise, quit
- (4) Let $ip = a*2^{24} + b*2^{16} + c*2^8 + d$, where ip is a 32-bit **integer**
- (5) Continue the rest of processing with the filtered address ip

Why is it vulnerable? Can you exploit it?

Security Guideline: Use Canonical Representation

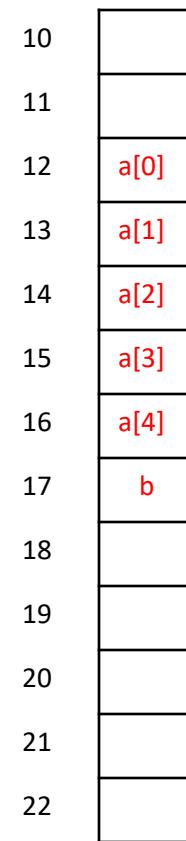
- Below are the important **lesson** and suggested **measures**
- **Never** trust the input from user
- Always convert them to a **standard** (i.e. *canonical*) **representation** immediately
- Preferably, *do not* rely on the verification check done in the **application**;
i.e. *do not* rely on the **application developers** to write the verification
- Rather, try to make use of the **underlying system access control** mechanism

8.2 Buffer Overflow

C/C++ and Memory Access

- C and C++ allows the programmers to **manage** the memory:
pointer arithmetic, no *array-bound checking*
- Such **flexibility** is useful, but **prone to bugs**,
which in turn leads to **vulnerability**
- Consider this simple program:

```
#include<stdio.h>
int a[5]; int b;
int main()
{
    b=0;
    printf("value of b is %d\n", b);
    a[5]=3;
    printf("value of b is %d\n", b);
}
```



Here, the value 3 is to be written to the cell a[5],
which is also the location of the **variable b**

Buffer Overflow/Overrun

- The previous example illustrates ***buffer overflow*** (a.k.a. **buffer overrun**)
- A ***data buffer*** (or just ***buffer***): “a contiguous region of memory used to temporarily store data, while it is being moved from one place to another”
- In general, a **buffer overflow** refers to a situation where data is written ***beyond*** a buffer’s boundary
- In the previous example, the array *a* is a buffer of size 5, and the location ***a[5]*** is beyond its boundary: hence, writing on it causes a “buffer overflow”
- A well-known function in C that is prone to buffer overflow is a string copying function: **`strcpy()`**

Strcpy() Function

- Consider this code segment:

```
{  
    char s1[10];  
    // .. get some input from user and store it in a string s2  
    strcpy(s1, s2);  
}
```

- In the above, the length of s2 can potentially be **more than 10**, since the length is determined by the first occurrence of null
- The strcpy () may copy the whole string of s2 to s1, **even if the length of s2 is more than 10**
- Since that the buffer size of s1 is only 10, the extra values will be **overflowed** and written to **other part** of the memory
- If s2 is **supplied by a malicious user**, a well-crafted input can overwrite important memory and modify the computation!

Secure Programming Defense/Practice

- Avoid using `strcpy()` !
- In secure programming practice,
use `strncpy()` instead
- The function `stcnncpy()` takes in **3** parameters:
`strncpy (s1, s2, n)`
- At **most** n characters are copied
- Note that improper usage of `strncpy()` could still
lead to vulnerability: *to be discussed in tutorial*

Stack Smashing (Stack Overflow)

- ***Stack smashing***: a special case of buffer overflow that targets a process' **call stack**
- Recall that when a function is invoked, information like parameters, *return address* will be pushed into the stack
- If the stack is being overflowed such that the **return address** is modified, the execution's control flow will *be changed*
- A **well-designed overflow** could also “inject” the attacker’s ***shellcode*** into the process’ memory, and then execute the shellcode
- What will happen if the target executable is **setUID-root**?
- Some defenses/counter-measures are available, such as: ***canary***, which will be discussed in the next lecture

Stack Smashing (Stack Overflow): Example

- Consider the following vulnerable segment of C program:

```
int foo(int a)
{
    char c[12];
    .....
    strcpy(c, bar); /* bar is a string input by user */
}

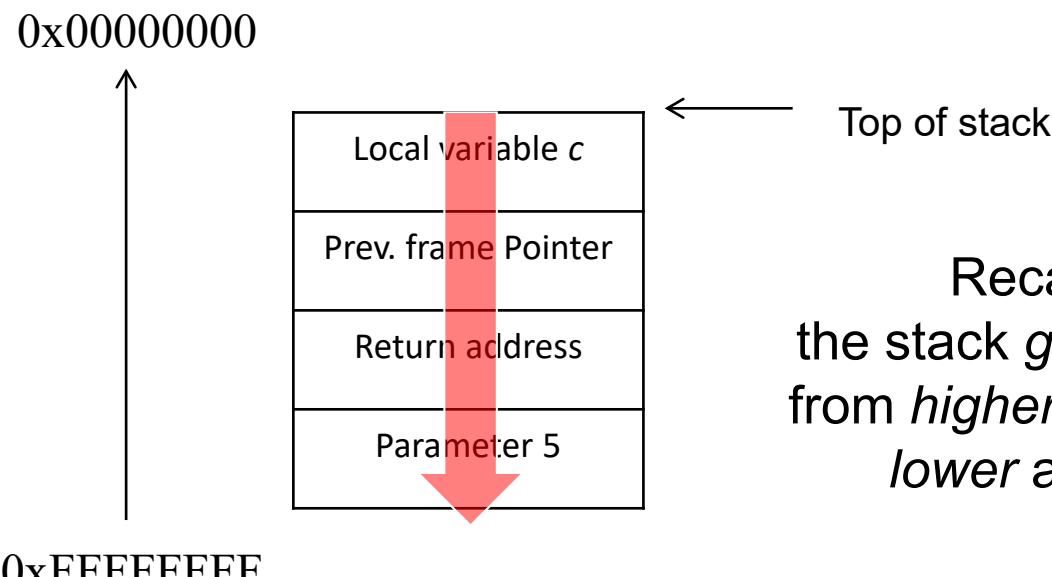
int main()
{
    foo(5);
}
```

Stack Smashing (Stack Overflow): Example

- After the `foo(5)` is invoked, a few values are pushed into the stack
- Important observation: the buffer `c` grows **toward return address!**
- If an attacker manages to modify the **return address**,
the control flow ***will jump*** to the address indicated by the attacker

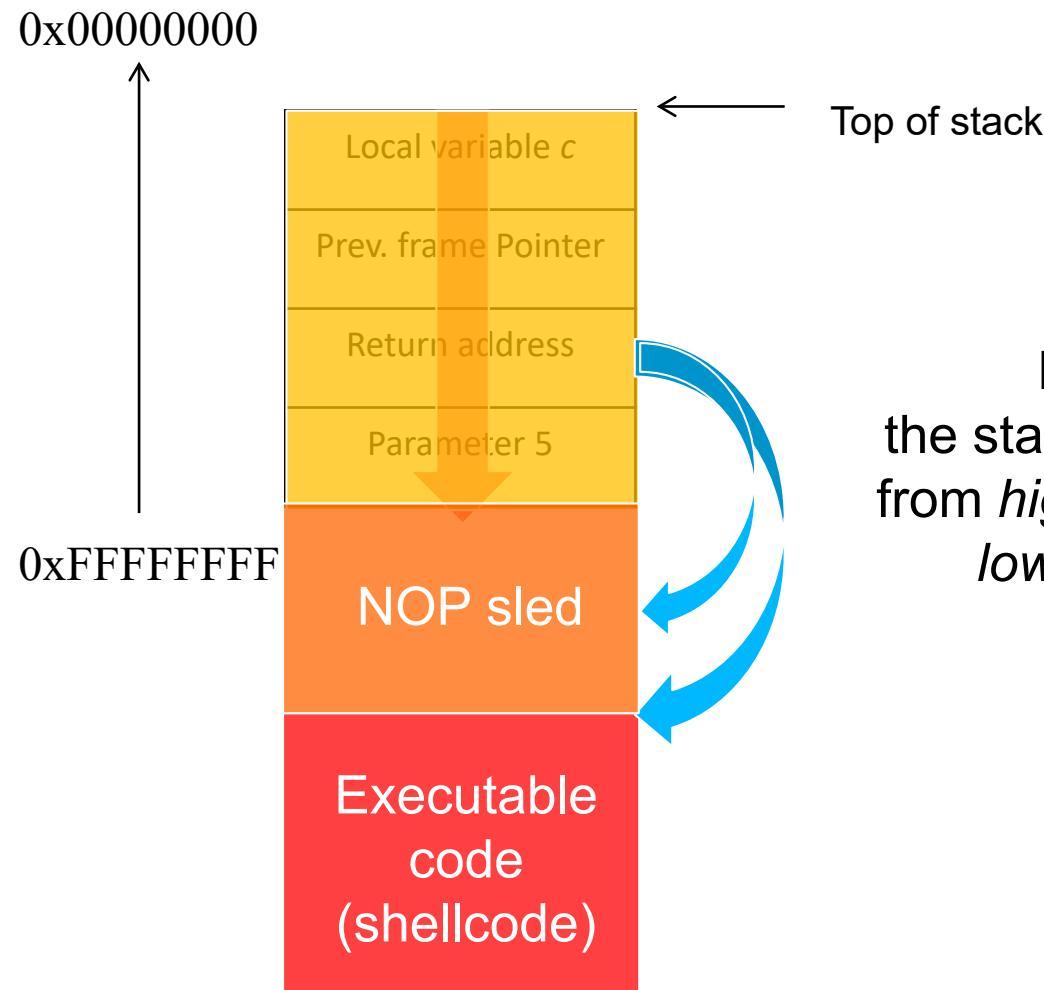
Read the *first* section: “Exploiting stack buffer overflows” of

https://en.wikipedia.org/wiki/Stack_buffer_overflow, other sections 2-4 are optional)



Recall that:
the stack *grows upward*,
from *higher* addresses to
lower addresses

Stack Smashing (Stack Overflow): Shellcode Illustration



Recall that:
the stack grows upward,
from *higher* addresses to
lower addresses

8.3 Integer Overflow

(Note: This is *not* to be confused with “buffer overflow”)

Integer Arithmetic and Overflow

- The **integer arithmetic** in many programming language are actually “*modulo arithmetic*”

- Suppose `a` is a single byte (i.e. 8-bit) **unsigned integer**. In the following C or Java statements, what would be the final value of `a`?

```
a = 254;
```

```
a = a+2;
```

- Its value is **0**, since the addition is done w.r.t./in **modulo 256**
- Hence, the following predicate is **not** necessarily always true!

$$(a < a+1)$$

- Yet, many programmers **do not** realize this, leading to possible vulnerability (see Tutorial 8)

8.4 Code/Script Injection

Scripting Language and Security

- A key concept in computer architecture is the **treatment of “code” (i.e. program) as “data”**
- In security, mixing “code” and “data” is potentially **unsafe**: many attacks inject malicious code as data, which then gets executed by the target system!
- We will consider a well-known ***SQL injection (SQLI) attack***
- **“Scripting” languages**: programming languages that can be **“interpreted”** by another program during runtime, instead of being compiled
- Well-known examples: JavaScript, Perl, PHP, **SQL**
- Many scripting languages allow the “script” to be **modified while being interpreted**: this opens up the possibility of injecting malicious code into the script!

SQL and Query

- SQL is a *database query language*
- Consider a database (which can be viewed as a **table**):
each **column/field** is associated with an *attribute*, e.g. “name”

| name | account | weight |
|--------------|---------|--------|
| bob12367 | 12333 | 56 |
| alice153315 | 4314 | 75 |
| eve3141451 | 111 | 45 |
| petter341614 | 312341 | 86 |

- This query script

```
SELECT * FROM client WHERE name = 'bob'
```

searches and returns the **rows** where the name matches ‘bob’

- The scripting language also allows **variable**:

e.g. a script may first get the user’s input and stores it in
the variable **\$userinput**, and subsequently runs:

```
SELECT * FROM client WHERE name = '$userinput'
```

SQL Injection: Example

- In this example, the database is designed such that the **user name is a secret**: hence, only the authentic entity who knows the name can get the record
- Now, an attacker can pass the following as the input:

Bob' OR 1=1 --

That is, the variable `$userinput` becomes

Bob' OR 1=1 --

- The interpreter, after seeing this script

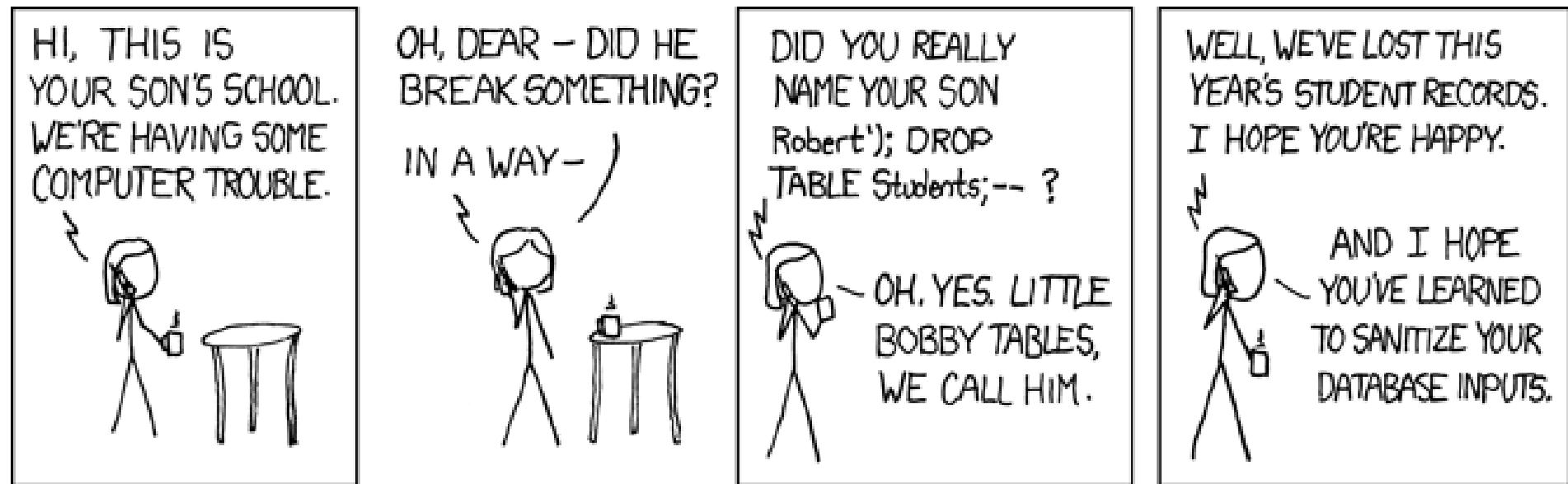
`SELECT * FROM client WHERE name = '$userinput'`

simply substitutes the above to get and execute:

`SELECT * FROM client WHERE name = 'Bob' OR 1=1 --'`

- Note: "--" is interpreted as **the start of a comment**
- The interpreter runs the above and return **all** the records!

SQL Injection a.k.a. “Bobby Tables”



Source:
<https://xkcd.com/327/>

Code Injection & Buffer Overflow

- Code injection does *not* limit to SQL injection
- It is possible to **exploit buffer overflow** by:
injecting malicious code, and then
transferring the process execution to the malicious code
- Details are omitted for this module
- For more details, see:
http://www.cis.syr.edu/~wedu/Teaching/IntrCompSec/LectureNotes_New/Buffer_Overflow.pdf

See fun and non-security related easter eggs:

www.pcadvisor.co.uk/feature/social-networks/11-best-easter-eggs-on-web-in-apps-3530683/

8.5 Undocumented Access Points (Easter Eggs)

Undocumented Access Points

- For debugging purposes, many programmers insert “***undocumented access point***” to inspect the states
- Examples:
 - By pressing **certain combination of keys**, the values of certain variables would be displayed
 - For certain **input strings**, the program would branch to some debugging mode
- These access points may mistakenly **remain** in the final production system, providing “***backdoors***” to the attackers
- A ***backdoor***: a covert method of bypassing normal authentication
- Such access points are also known as ***Easter eggs***

Undocumented Access Points

- Some Easter eggs are **benign** and intentionally planted by the developer for **fun or publicity**
- But, there are also known cases where unhappy/disgruntled programmer purposely **planted** the backdoors
- The backdoors can be accessed by the programmer, and also by **any** other users who knows/discovers them!
- ***Terminology:*** Logic bombs, Easter eggs, backdoors

Lecture 9: Software Security (Part III)

9.1 Time-of-Check Time-of-Use (TOCTOU) race condition

9.2 Defense and Preventive Measures:

 9.2.1 Filtering (input validation)

 9.2.2 Using safer functions

 9.2.3 Bounds checking and type safety

 9.2.4 Memory protection (randomization, canaries)

 9.2.5 Code inspection (taint analysis)

 9.2.6 Testing

 9.2.7 The principle of least privilege

 9.2.8 Patching (keeping up to date)

Read: <https://cwe.mitre.org/data/definitions/367.html>

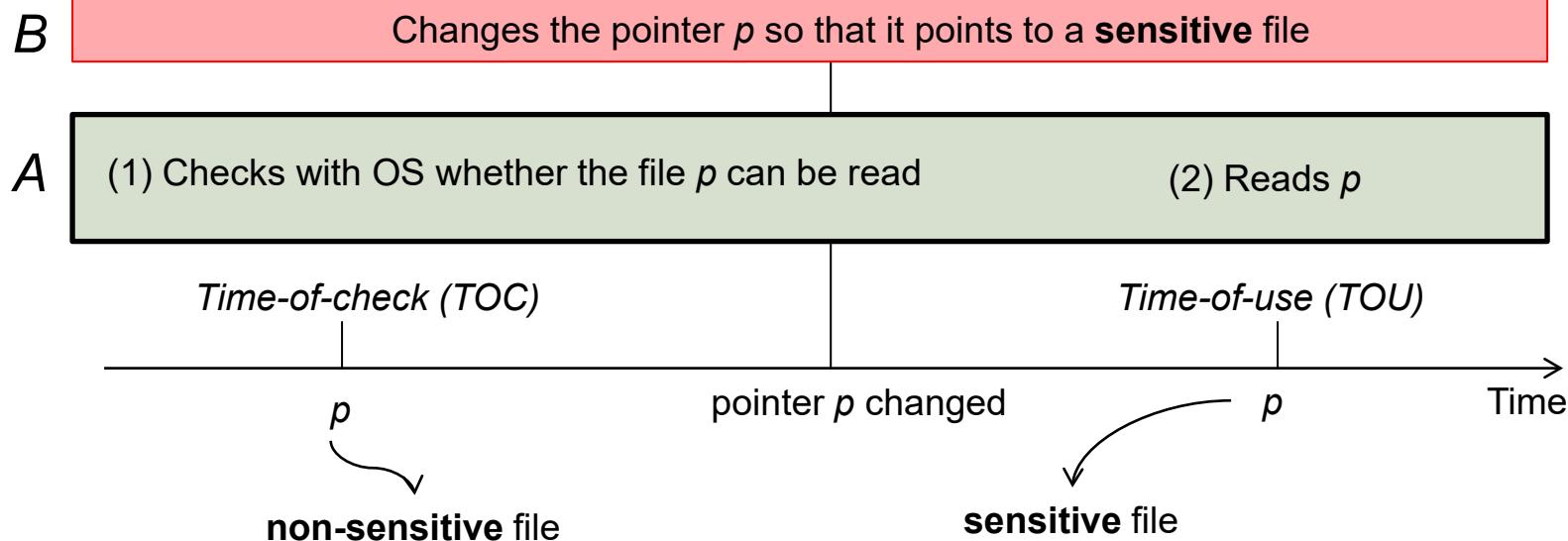
9.1 Time-of-Check Time-of-Use (TOCTOU) Race Condition

Race Condition and TOCTOU

- In general, a *race condition* occurs when **multiple processes** access a piece of **shared data** in such a way that the **final outcome** depend on the sequence of accesses
- In the context of security, the “**multiple processes**” typically refer to:
 1. A (**vulnerable**) process **A** that **checks/verifies** the permission to access a shared data, and subsequently **accesses** the data
 2. Another (**malicious**) process **B** that “**swaps**” the data
- Note that the two processes can be run by **one** malicious user in the system

Race Condition and TOCTOU

- There is a “**race**” between processes **A** and **B**:
If **B** manages to complete the swapping before **A** accesses the data, then the attack succeeds
- This scenario is also known as
Time-of-check time-of-use (TOCTOU)

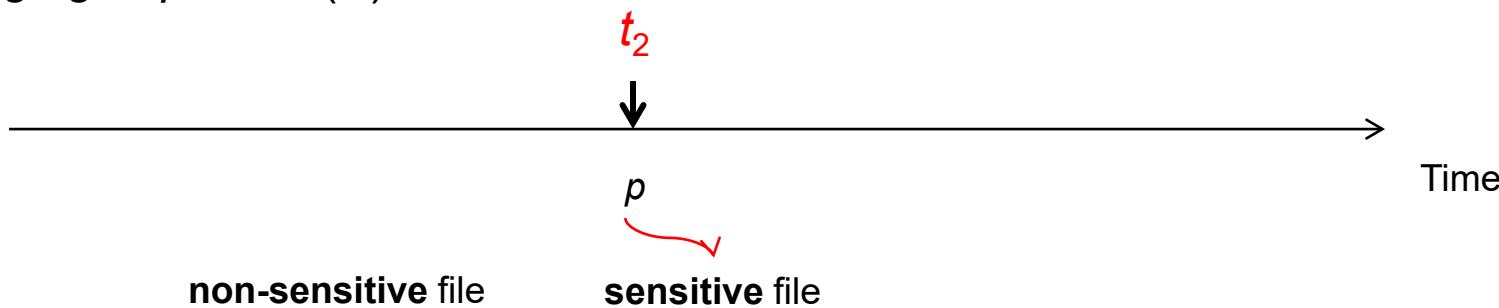


TOCTOU: Three Important Events/Actions

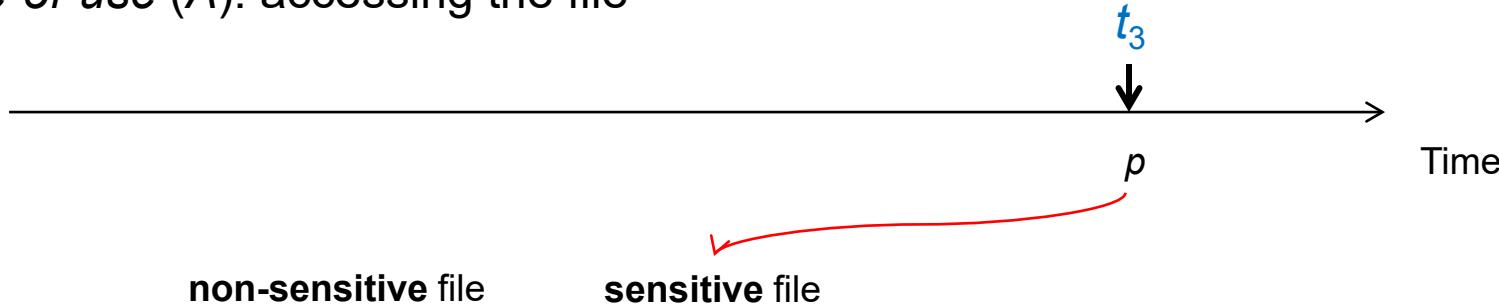
time-of-check (*A*): checking whether the process is authorized to access the file



changing of pointer (B)



time-of-use (A): accessing the file



A Sample Case: The Vulnerable Program

- A variant of Example 2 in
<https://cwe.mitre.org/data/definitions/367.html>
is shown next
- This setUID program is to be ran under an **elevated privilege** (i.e. setUID-root)
- The **access (f, W_OK)** system call:
 - Checks whether the user executing the program has the permission to **write** to the specified *filename f*
 - Returns **0** if the process **has** the permission
 - The check is done based on the process's **real UID**
- Because of the check on real UID, a malicious user needs to find **a way** to access a sensitive target file

A Sample Case: The Vulnerable Program (Continued)

TOC

```
// f is a string that contains the name of a file
// fd is the file descriptor

if(!access(f, W_OK))           // check whether the real UID has write permission to f
{
    fprintf (stderr, "permission to operate %s granted\n", f );
    fd = open(f,O_RDWR);      // open the file with read and write access
    OP(fd);                  // a routine that operates on the file. OP is not a system call
    ....
}
else
{
    fprintf(stderr,"Unable to open file %s.\n", f );
}
```

TOU

access() System Call

ACCESS(2)

Linux Programmer's Manual

ACCESS(2)

NAME

access, faccessat - check user's permissions for a file

SYNOPSIS

```
#include <unistd.h>
int access(const char *pathname, int mode);
...
```

DESCRIPTION

access() checks **whether the calling process can access the file pathname. If pathname is a symbolic link, it is dereferenced.**

The mode specifies the accessibility check(s) to be performed, and is either the value F_OK, or a mask consisting of the bitwise OR of one or more of R_OK, W_OK, and X_OK. F_OK tests for the existence of the file. **R_OK, W_OK, and X_OK test whether the file exists and grants read, write, and execute permissions, respectively.**

The check is done using the calling process's real UID and GID, rather than the effective IDs as is done when actually attempting an operation (e.g., open(2)) on the file. Similarly, for the root user, the check uses the set of permitted capabilities rather than the set of effective capabilities; and for non-root users, the check uses an empty set of capabilities.

...

A Sample Case: Two Files Involved

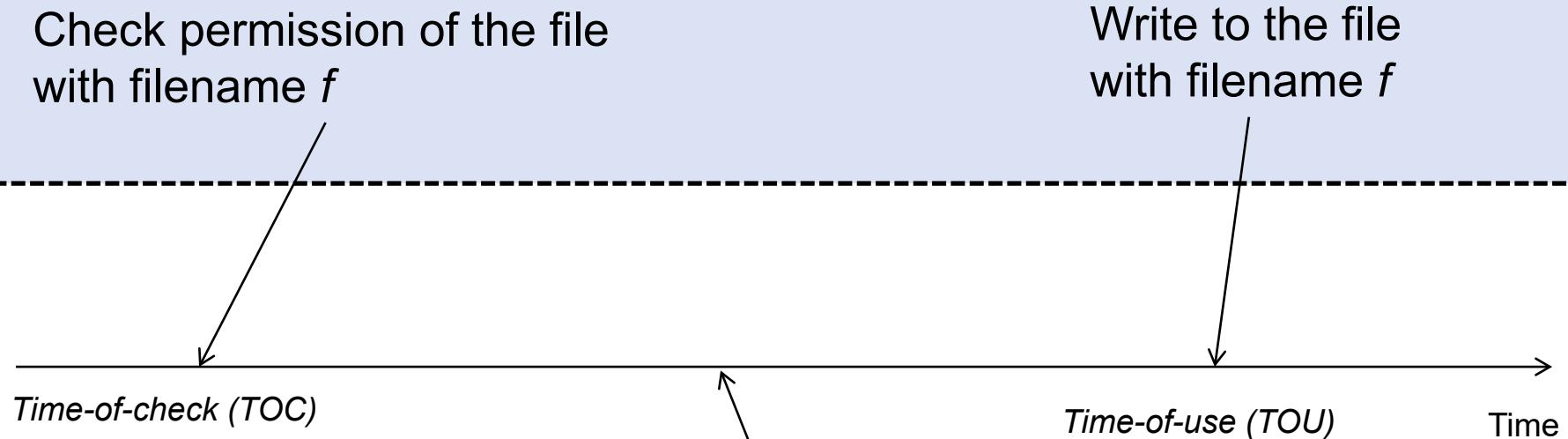
- Suppose the program has the “setUID-root” permission, and bob (a malicious user) invokes the program
- Thus the process’ **real UID** is **bob**, but its **effective UID** is **root**
- Furthermore, suppose that the filename `f` is
`/usr/year1/bob/list.txt`
which is a file **owned by bob**: bob has a permission to **change** it
- After bob has started the process, he immediately **replaces** the file
`/usr/year1/bob/list.txt`
with a symbolic link that points to a **sensitive system file**
- bob **does not** have a write permission to the sensitive system file, but he wants to change the file: *Can he do that?*

A Sample Case: Bob's Malicious Actions

- This can be done by running **a script** that carries out the following 2 steps:
 1. **Delete** /usr/year1/bob/list.txt
 2. Create **a symbolic link** with the filename *f* and points to a system file, by issuing this Unix command:

```
ln -s /usr/course/cz2017/grade.txt  
/usr/year1/bob/list.txt
```
- With ***some probability***, bob **wins** the race
- Hence the process operates on the system file
/usr/course/cz2017/grade.txt
instead of /usr/year1/bob/list.txt

A Sample Case: Timeline of Events



Delete the file with filename f ,
create a link with the name f ,
but points to a sensitive file f'

/usr/year1/bob/list.txt

/usr/course/cz2017/grade.txt

Avoiding TOCTOU on File Checking (C Programming): Approach 1

Defense approach 1:

- In your program, **avoid** using **separate system calls** that takes the **same filename** as input
- Instead, try to **open** the file **only once** (and thus **lock** it to **block** any further changes by other processes), and use the file handle/descriptor
- That is, in general, **avoid** using **access ()** system call
- Rather, open the file once using **open ()** system call, and then use **fstat ()** system call to check the permission

Safe Versions: Approach 1

```
// f is a string that contains the name of a file

fd = open(f, O_RDWR);           // open the file with read and write access

TOC → fstat(fd, &filestatus) ) // get the file status and store them to filestatus
if ↗ CP(filestatus) )         // check permission of the file (CP is not a system call)
{
    fprintf (stderr, "permission to operate %s granted\n", f );
    OP(fd);                  // a routine that operates the file (OP is not a system call)
    ....
}
else
{
    fprintf(stderr,"Unable to open file %s.\n", f );
}
```

TOU

Avoiding TOCTOU on File Checking (C Programming): Approach 2

Defense approach 2:

- A **better practice** would be:
 avoid writing your own access-control on files,
 and leave the checking to **the underlying OS**
 after appropriately setting your process credentials
- For instance, adopt the following in your program:
 set the process' **effective UID** to the **appropriate**
 (normal/non-root) user before **accessing** the file
- In this way, **the OS** checks the permission,
 and decide whether to grant or deny the access
- After it is done, **reset** the **effective UID** back to root
 if required

Safe Versions: Approach 2

```
// f is a string that contains the name of a file
// u is the UID of the appropriate user (i.e. Alice)
// r is the UID of root;
...
seteuid (u);                                // from now onward, the effective UID is u
fd = open (f, O_RDWR);
OP (fd);
...
seteuid (r);                                // from now onward, the effective UID is root
...
```

Elevated
Privilege

Lowered
privilege

Elevated
(root)
privilege
restored

9.2 Defense and Preventive Measures

Read http://en.wikipedia.org/wiki/Bounds_checking

General Comments

- As illustrated in previous examples, **many bugs and vulnerabilities** are due to **programmer's ignorance**
- In general, it is **difficult** to analysis a program to ensure that it is bug-free (recall the “halting-problem”)
- There is no “fool-proof” method
- However, various useful **counter measures** are available as discussed next

9.2.1 Input Validation using Filtering

Filtering

- In almost all examples (except TOCTOU) we have seen, the attack is carried out by feeding **carefully-crafted input** to the system
- Those inputs **do not** follow the “**expected**” format:
e.g. the input is too long, contains control/meta characters, contains negative number, etc.
- Hence, a preventive measure is to perform an **input validation/filtering** whenever an input is obtained from the user: if the input is not of the expected format, reject it

Problems with Filtering

- It is *difficult* to ensure that the filtering is “**complete**” (i.e. it doesn’t miss out any malicious strings), as illustrated in the example on UTF
- In that example on UTF, the input validation intend to detect the substring “`.. /`”
- Unfortunately, there are **multiple representations** of “`.. /`” that the programmer is not aware of
- A filter that completely **blocks all bad inputs** and **accepts all legitimate inputs** is *very difficult* to design

Filtering

- There are generally **two approaches** of filtering:
 1. ***White list:*** A list of items that are known to be **benign** and allowed to **pass**, which could be expressed using regular expression
However, some legitimate inputs may be blocked.
 2. ***Black list:*** A list of items that are known to be **bad** and to be **rejected**.
For example, to prevent SQL injection,
if the input contains meta characters, reject it.
However, some malicious input may be passed.
- *Which type of filtering is then more secure?*

9.2.2 Using “Safer” Functions

Safer Function Alternatives

- Completely **avoid** functions that are known to create problems
- Use the “**safer**” **versions** of the functions
- C/C++ have many of those:

strcpy ()

↔

strncpy ()

printf (*f*)

access ()

- Again, even if they are avoided, there **could** still be vulnerability: recall the example that uses a combination of strlen () and strncpy () in your **tutorial**

9.2.3 Bounds Checking and Type Safety

Bounds Checking

- Some programming languages (e.g. Java, Pascal) perform ***bounds checking*** at runtime
- That is, when an array is declared, its upper and lower bounds have to be declared
- At runtime, whenever a reference to an array location is made, the index/subscript is **checked** against the upper and lower bounds
- Hence, a simple assignment like:

`a[i] = 5;`

would **consists of** these steps:

1. Checks that `i` is \geq the lower bound;
2. Checks that `i` is \leq the upper bound; and
3. Assigns 5 to the memory location

Bounds Checking

- If the checks **fail**, the process will be **halted** (or an exception is to be thrown as in Java)
- The added first 2 steps reduce efficiency, but will **prevent** buffer overflow
- Many of the known vulnerabilities is due to buffer overflow that can be prevented by this **simple bounds checking**: visit <http://cve.mitre.org/cve/cve.html> to see how many entries contains “buffer overflow” as keywords
- The infamous C and C++ **do not** perform bounds checking!
- Yet, **many** pieces of software are written in C/C++!

Some Words of Wisdom

C. A. R. Hoare (1980 Turing Award winner) on his experience in the design of ALGOL 60, a language that **included** bounds checking:

“A consequence of this principle is that every occurrence of every subscript of every subscripted variable was on **every occasion checked at run time against both the upper and the lower declared bounds of the array**. Many years later we asked our customers whether they wished us to provide an option to switch off these checks in the interest of efficiency on production runs. Unanimously, they urged us not to—they already knew **how frequently subscript errors occur on production runs** where failure to detect them could be disastrous. I note with fear and horror that even in 1980, language designers and users have not learned this lesson. **In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law.**”

Type Safety

- Some programming languages carry out “**type checking**” to ensure that the arguments an operation get during execution are always correct
- For example: **a = b;**
if a is a 8-bit integer, b is a 64-bit integer, then the type is **wrong**
- The checking could be done at **runtime** (i.e. **dynamic type checking**), or when the program is being **compiled** (i.e. **static type checking**)
- Bounds checking can also be **considered** as one mechanism that ensures “type safety”
- For example in Pascal:

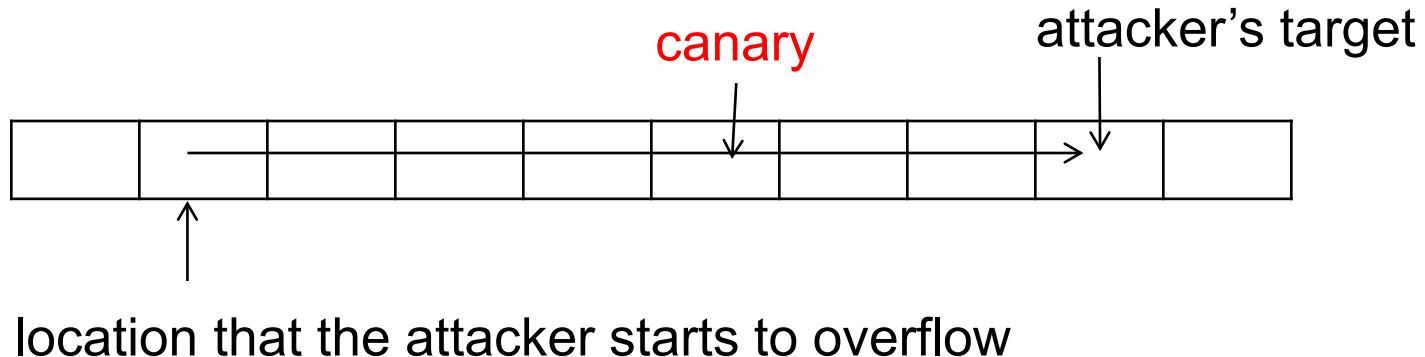
```
Type      indexrange = 1..10;  
Var       A: array [indexrange] of integer;  
Begin  
    A[0] = 5;  ← wrong type
```

9.2.4 Memory Protection (Randomization, Canaries)

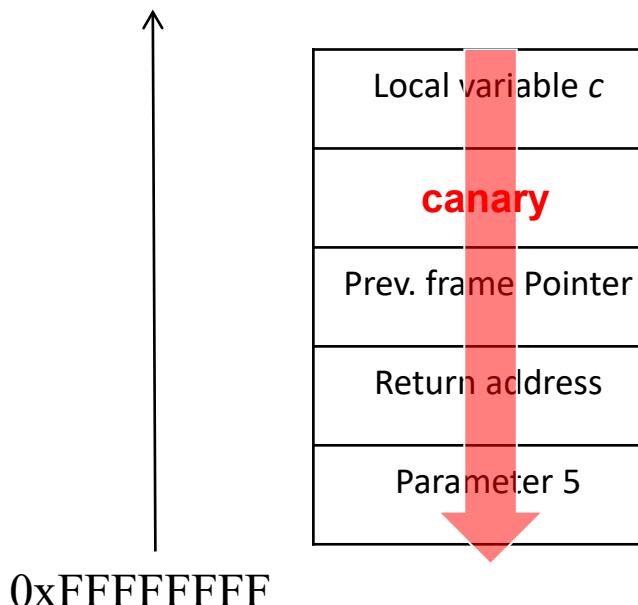
Canaries

- **Canaries** are “secret” values inserted at carefully selected memory locations at runtime
- Checks are carried out at **runtime** to make sure that the values are **not** being modified: if so, halts
- Canaries can help **detect** overflow, especially stack overflow:
 - In a typical buffer overflow, **consecutive** memory locations have to be over-ran: the canaries would be **modified**
- It is important to keep the values “**secret**”: if the attacker knows the value, it may able to write the secret value to the canary while over-running it!
- (*Optional*) In Linux, you can **turn off** gcc canary-based stack protection by supplying this flag when invoking gcc:
–fno-stack-protector

Canaries



0x00000000



Recall that:
the stack *grows upward*,
from *higher* addresses to
lower addresses

Stack Smashing Detected: Program 1

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char text[15];

    strcpy(text, argv[1]);
    printf("Your supplied argument is :%s\n", text);

    printf("main() is exiting\n");
    return 0;
}
```

Stack Smashing Detected: Program 1

- If compiled **with** stack protector:

```
./program-wspaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
Your supplied argument is  
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

main() is exiting

***** stack smashing detected ***: ./program-wsp
terminated**

Aborted (core dumped)

- If compiled **without** stack protector:

```
./program-wospaaaaaaaaaaaaaaaaaaaaaaa  
Your supplied argument is  
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

main() is exiting

Segmentation fault (core dumped)

Stack Smashing Detected: Program 2

```
#include <stdio.h>
#include <string.h>

void copy_this(char *arg1)
{
    char text[15];

    strcpy(text, arg1);
    printf("Your supplied argument is :%s\n", text);
    printf("Function is exiting\n");
}

int main(int argc, char *argv[])
{
    copy_this(argv[1]);

    printf("main() is exiting\n");
    return 0;
}
```

Stack Smashing Detected: Program 2

- If compiled with stack protector:

```
./program-too-wsp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Function is exiting

```
*** stack smashing detected ***: ./program-too-wsp
terminated
```

Aborted (core dumped)

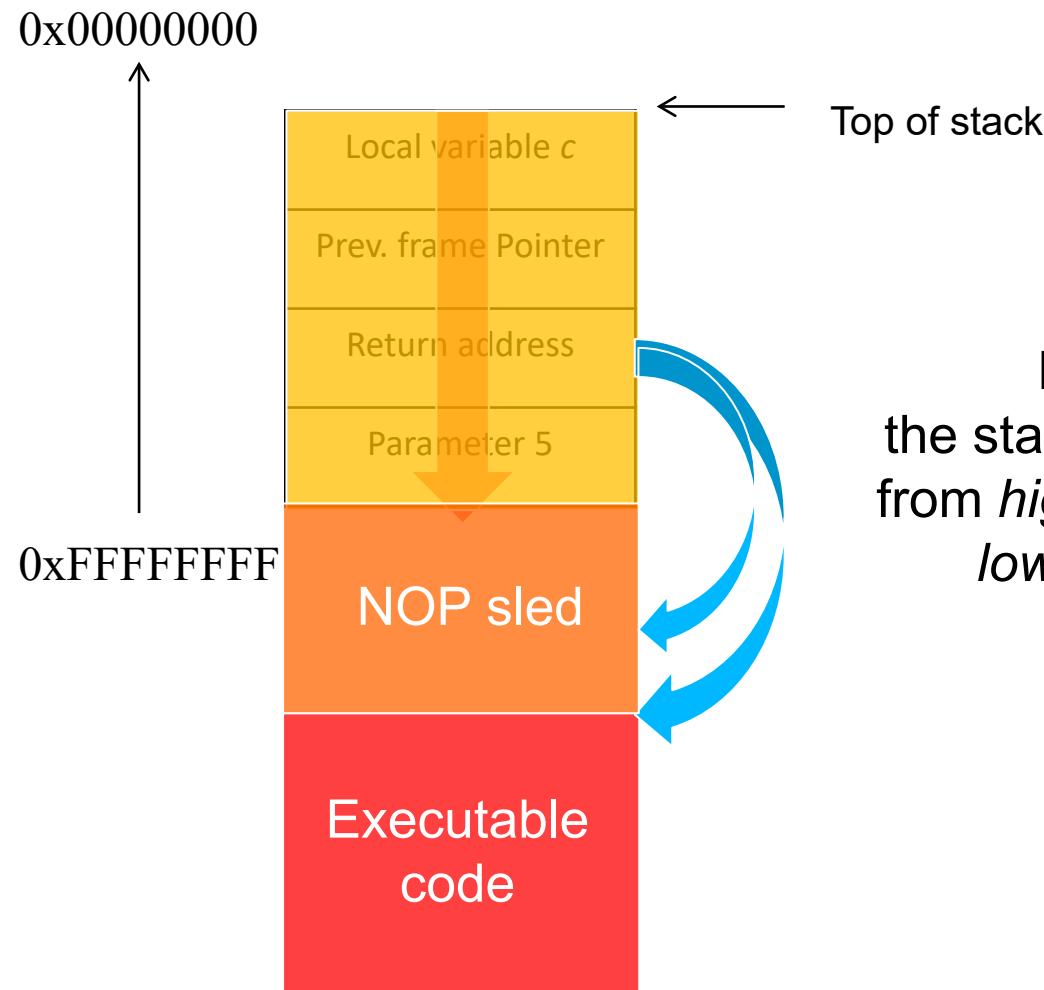
- If compiled *without* stack protector:

```
./program-too-wosp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Function is exiting

Segmentation fault (core dumped)

Stack Smashing (Stack Overflow): Illustration



Recall that:
the stack grows upward,
from *higher* addresses to
lower addresses

Memory Randomization

- It is to the attacker's advantage when the data and codes are always stored in the same locations in memory
- ***Address space layout randomization (ASLR)*** is a **prevention** technique that can help decrease the attacker's chance
- ASLR: **randomly** arranges the address space positions of **key data areas** of a process, including:
the base of the executable and the positions of the stack, heap and libraries
- *(Details are omitted in this module)*
- ***Optional:*** in Linux, you can turn off (disable) address randomization using:
`sudo sysctl -w kernel.randomize_va_space=0`

9.2.5 Code Inspection

Code Inspection

- **Manual checking:** manually checks the program, is tedious
- **Automated checking:** some automations and tools are possible
- An example is ***taint analysis***:
 - Variables that contain input from the (potential malicious) users are labeled as ***sources***
 - Critical functions are labeled as ***sinks***
 - Taint analysis **checks** whether any of the sink's arguments could potentially be affected (i.e. tainted) by a source
 - Example: sources = user input
sink: opening of system files, function evaluating a SQL command
 - If so, special check (e.g. manual inspection) would be carried out
 - Taint analysis can be **static** (i.e. checking the code without running/tracing it), or **dynamic** (i.e. running the code with some inputs)

9.2.6 Testing

Testing

- Vulnerability can be discovered via **testing**
- Types of testing:
 - ***White-box*** testing:
the tester has an access to the application's **source code**
 - ***Black-box*** testing:
the tester does not have an access to the source code
 - ***Grey-box*** testing:
A combination of the above, reverse-engineered
binary/executable

Testing

- Security testing attempts to discover **intentional attack**, and hence would test for inputs that are **rarely occurred** under normal circumstances
- Examples: very long names, names containing numeric values, string containing meta characters, etc.
- **Fuzzing** is a technique that sends **malformed inputs** to discover vulnerabilities:
 - There are techniques that are more effective than sending in random inputs
 - Fuzzing can be automated or semi-automated:
(the details are not required)
- **Terminology:** white list vs black list, white-box testing vs black-box testing, white hat vs black hat

9.2.7 The Principle of Least Privilege

The Principle of Least Privilege

Apply the “*Principle of Least Privilege*”:

- When writing a program, be **conservative** in elevating the privilege
- When deploying software system, do **not** give the users more access rights than necessary, and do **not** activate unnecessary options

The Principle of Least Privilege

- Example:
 - Software contain many **features**: e.g. a web-cam software could provide many features so that the user can remotely control it.
 - A user can choose to set which features to be on/off.
 - Suppose you are the developer of the software.
 - Should all features to be **switched on by default** when the software is shipped to your clients?
 - If so, it is **the clients' responsibility** to “**harden**” the system by selectively **switch off the unnecessary features**.
 - Your clients might not aware of the implications and thus at a higher risk.
 - Terminology: What does “*hardening*” mean?

9.2.8 Patching (Keeping up to Date)

Vulnerability Lifecycle

- **Life-cycle of a vulnerability:**
(1) a **vulnerability** is discovered → (2) affected code is fixed →
(3) the revised version is tested → (4) a **patch** is made public →
(5) patch is applied
- In some cases, the vulnerability could be announced (1?) **without the technical details** before a patch is released:
the vulnerability is likely to be known to only a small number
of attackers (even none) before it is announced
- When a **patch** is released (4), the patch can be **useful to attackers** too: they can inspect the patch and derive the vulnerability
- Hence, interestingly, the number of successful attacks
can **go up** after the vulnerability/patch is announced:
since more attackers would be aware of the exploit
(see the next slide)

Vulnerability Lifecycle

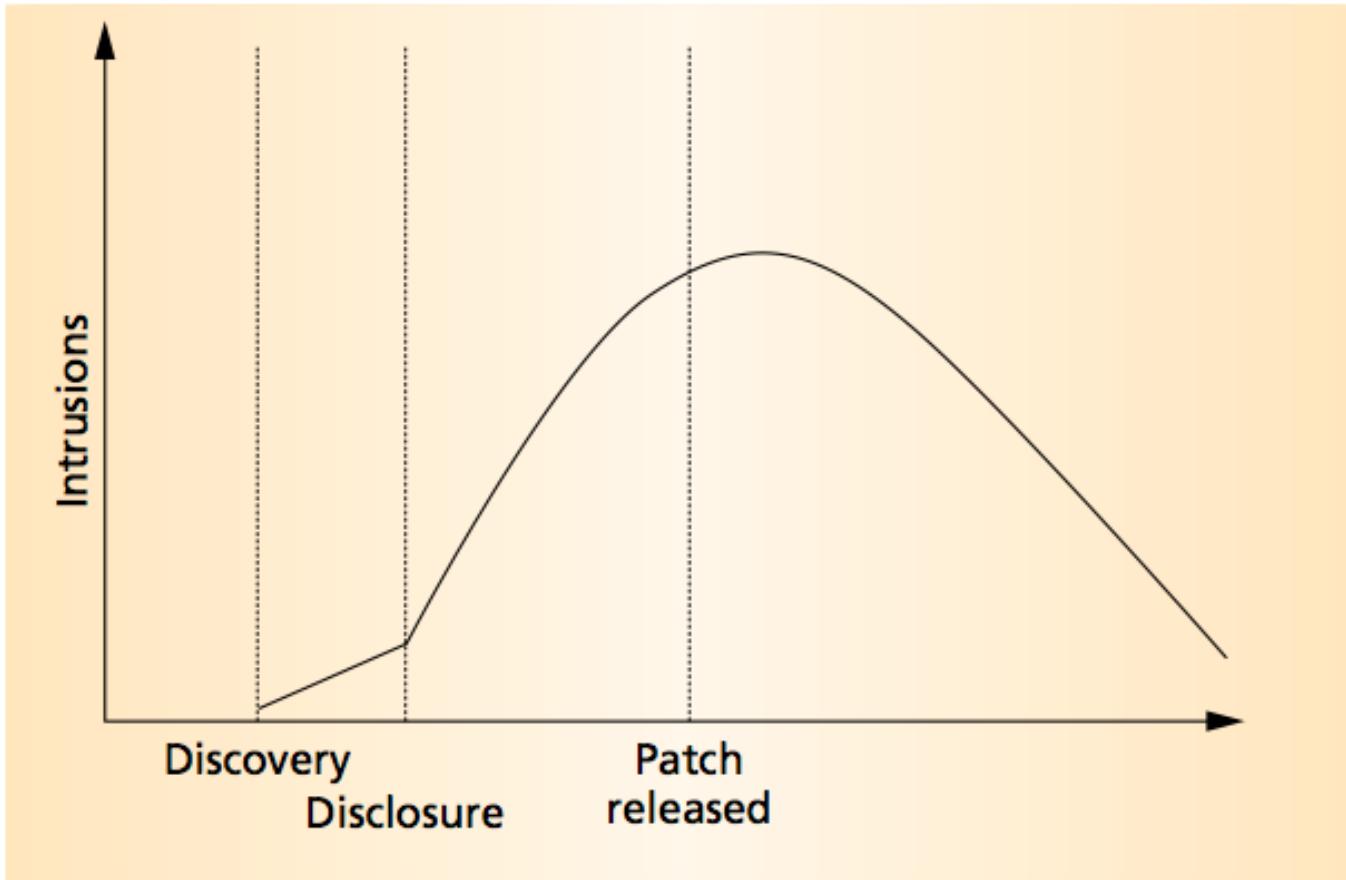


Figure 1. Intuitive life cycle of a system-security vulnerability. Intrusions increase once users discover a vulnerability, and the rate continues to increase until the system administrator releases a patch or workaround.

image obtained from

William A. Arbaugh et al. Windows of vulnerability: A case study analysis. IEEE Computer, 2000.

http://www.cs.umd.edu/~waa/pubs/Windows_of_Vulnerability.pdf

Patching

- It is crucial to apply the patch **timely**
- Although it seems easy, applying patches is **not** that straightforward:
 - For **critical systems**, it is not wise to apply the patch immediately before rigorous testing:
E.g. after a patch is applied, the train scheduling software crashes
 - Patches might **affect** the applications, and thus affect an organization **operation**:
E.g. after a student applied a patch on Adobe Flash, he couldn't upload a report to LumiNUS and thus missed a project deadline
- “**Patch management**” is a field of study
- See the guide on patch management issued by NIST:
“Guide to Enterprise Patch Management Technologies”, 2013,
<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r3.pdf>

Summary: Defense Mechanisms and Stages of SDLC

Adopt various counter measures at **different stages of SDLC**:

- **Development** stage:
 - Using safer functions
 - Bounds checking and type safety
 - Filtering (input validation)
 - Code inspection (taint analysis)
 - The principle of least privilege*
 - Executable generation with enabled memory protection*
- **Testing** stage:
 - Testing: fuzzing, penetration testing
- **Deployment** (including software execution) stage:
 - Runtime memory protection*: randomization
 - The principle of least privilege*: disable unnecessary features
 - Patching

* Applicable to *multiple* stages

Lecture 10: Web Security

10.1 Background

10.2 Security issues and threat models

10.3 Vulnerabilities in the “secure” communication channel (SSL/TLS)

10.4 Mislead the user

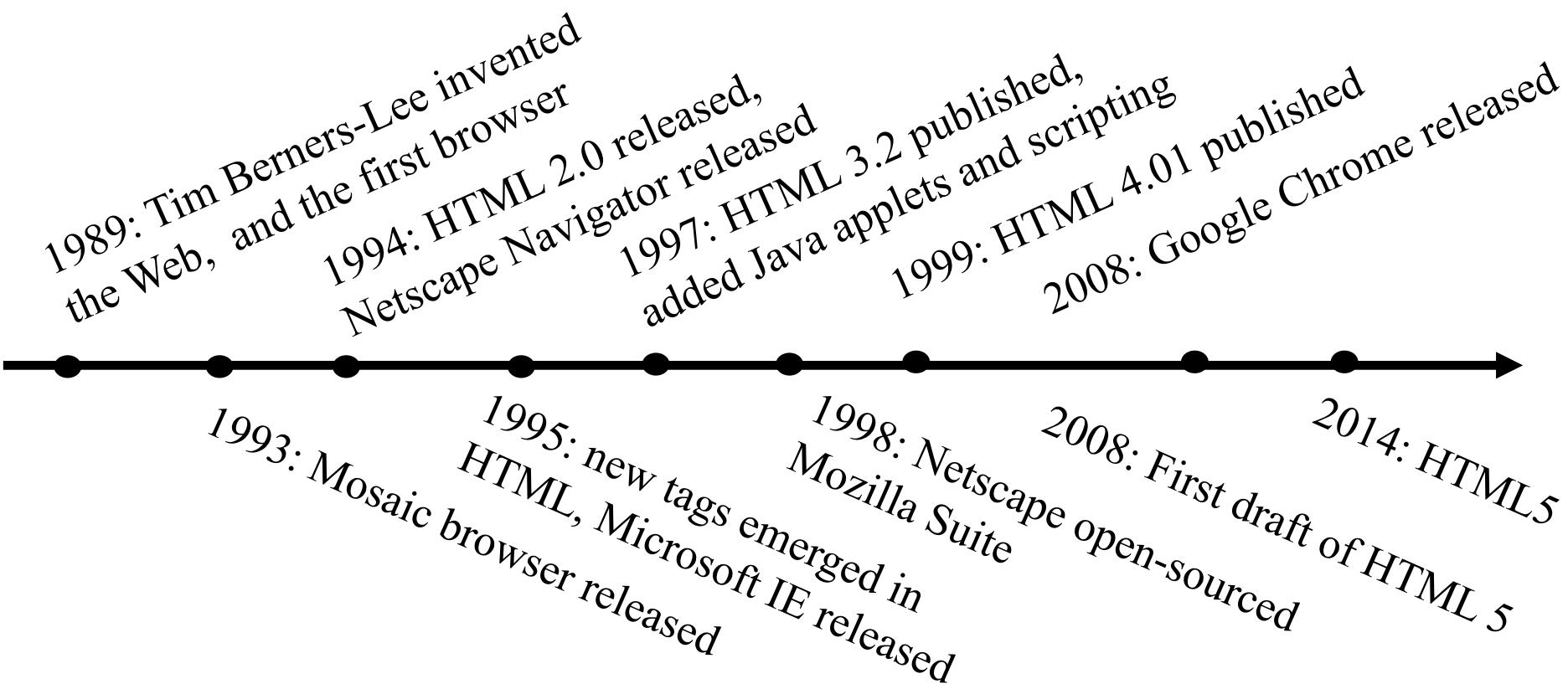
10.5 Cookies and the same-origin policy

10.6 Cross-site scripting (XSS) attacks

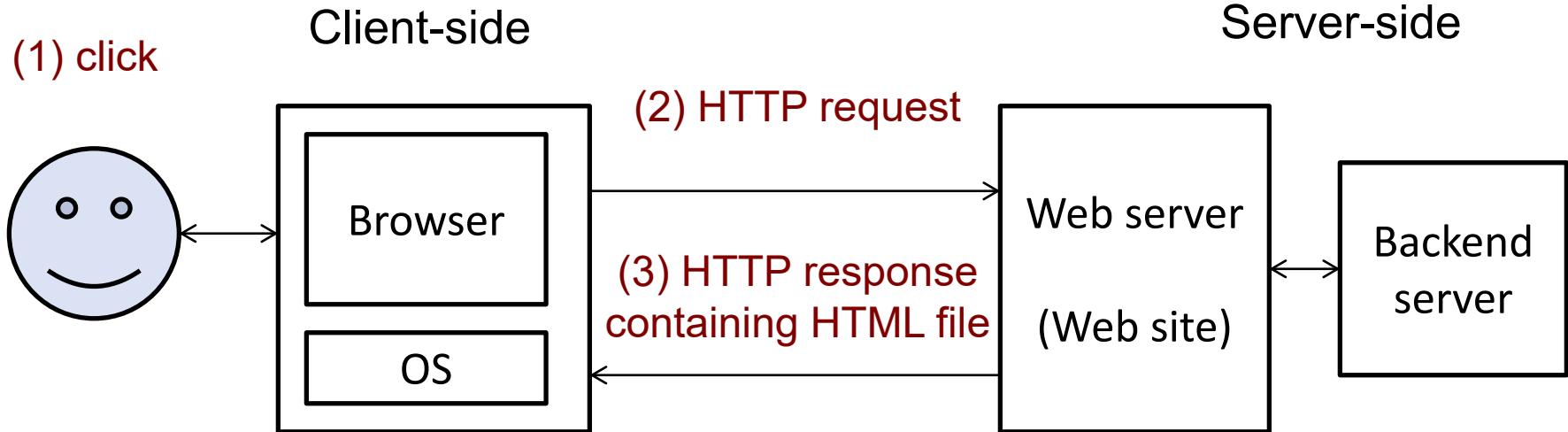
10.7 Cross-site request forgery (CSRF) attacks

10.1 Background

Evolution of the Web



Overview of HTTP: A Web-Page Access Process



(4) render (including running **some scripts** in the HTML file)

1. User clicks on a **URL “link”**, for example `luminus.nus.edu.sg/`
2. A **HTTP request** is sent to the server (with **cookies** if any)
3. Server constructs and include a **“HTML” file** inside its **HTTP response** to the browser, likely with **cookies**
4. The browser **renders the HTML file**, which describes the layout to be rendered and presented to the user, and the cookies are stored in the browser

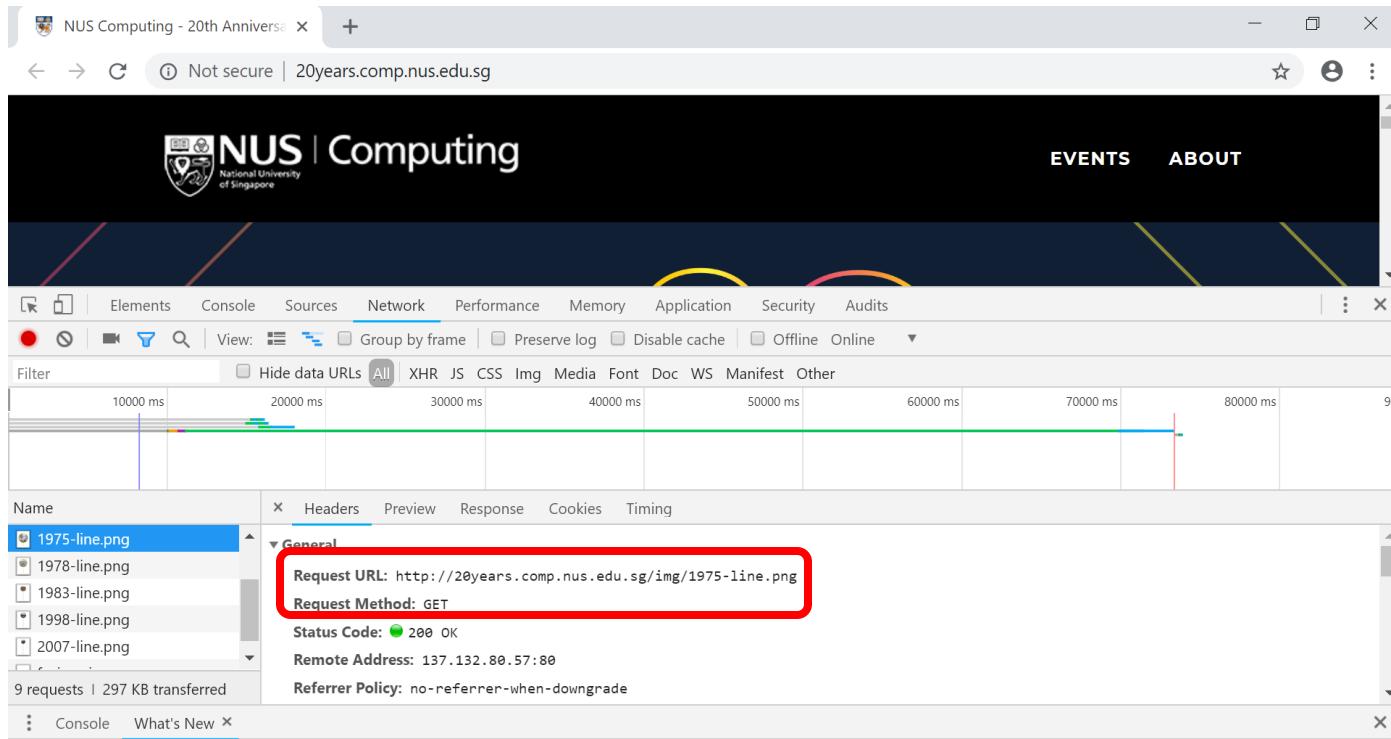
Notes: To view (**the raw form of**) the **HTML file** sent from the server to the browser:

right-click a page, choose **“View page source”** (in Firefox); View->Developer->**“View Source”** (in Chrome).

Note that there are many occurrences of the tag `<script>`, which marks the beginning of a **script**.⁴

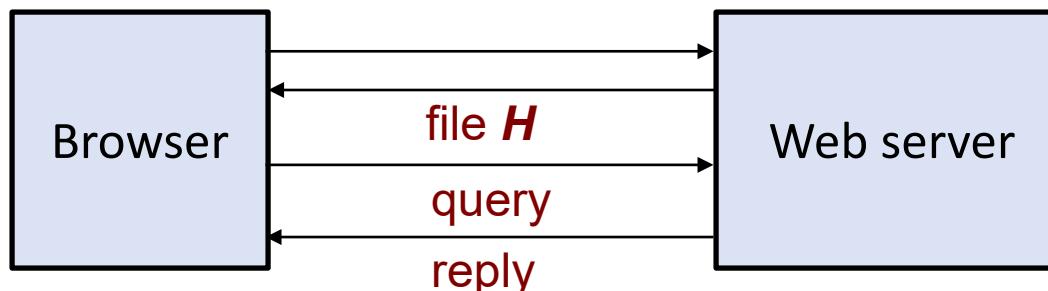
Sub-Resources of A Web Page

- A HTML page may contain **sub-resources** (e.g. images, multimedia files, CSS, scripts) including from **external/third-party** web sites
- When parsing a page with sub-resources, browser also contacts the **respective server** for each sub-resource
- A **separate HTTP request** for **every single file** on a page: since each file requires its own HTTP request



A Closer Look into an Interactive Query-Page Example

- 1) Browser visits **Google Search page** (`www.google.com.sg`).
A HTML file **H** is sent by the server to the browser.
The browser renders **H**.



- 2) Browser user enters the search **keywords** “CS2017 NUS”
- 3) The browser, by running *H*, constructs a **query**, for instance:
`https://www.google.com.sg/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=CS2107+NUS`
- 4) Note that additional information is added as **URL parameters**.
These info is useful for the server.
The info could even be in the form of **script**.
- 5) The server constructs a **reply**.
(Notice that in some cases, the reply **contain substrings** sent in Step 3.)

HTTP Request and Response Messages

- Note that various **request and response headers** are used



GET / HTTP 1.1
Host: www.example.com
User-Agent: Mozilla/...
...
HTTP/1.1 200 OK
Date: Thu, 13 Oct 2011
Server: Apache/1.3.41
Content-Type: text/html
...



HTTP Request and Response Formats

- **HTTP Request** contains:

- **Request line**, e.g.: GET /test.html HTTP/1.1
- **Request headers**, such as:
Accept: image/gif, image/jpeg, */*
Accept-Language: us-en, fr, cn
Cookie: theme=light; sessionToken=abc123;
- An empty/blank line
- An optional message body

- **HTTP Response** contains:

- **Status line** containing *status code & reason phrase*, e.g.:

HTTP/1.1 200 OK

HTTP/1.0 404 Not Found

- **Response headers**, such as:

Content-Type: text/html

Content-Length: 35

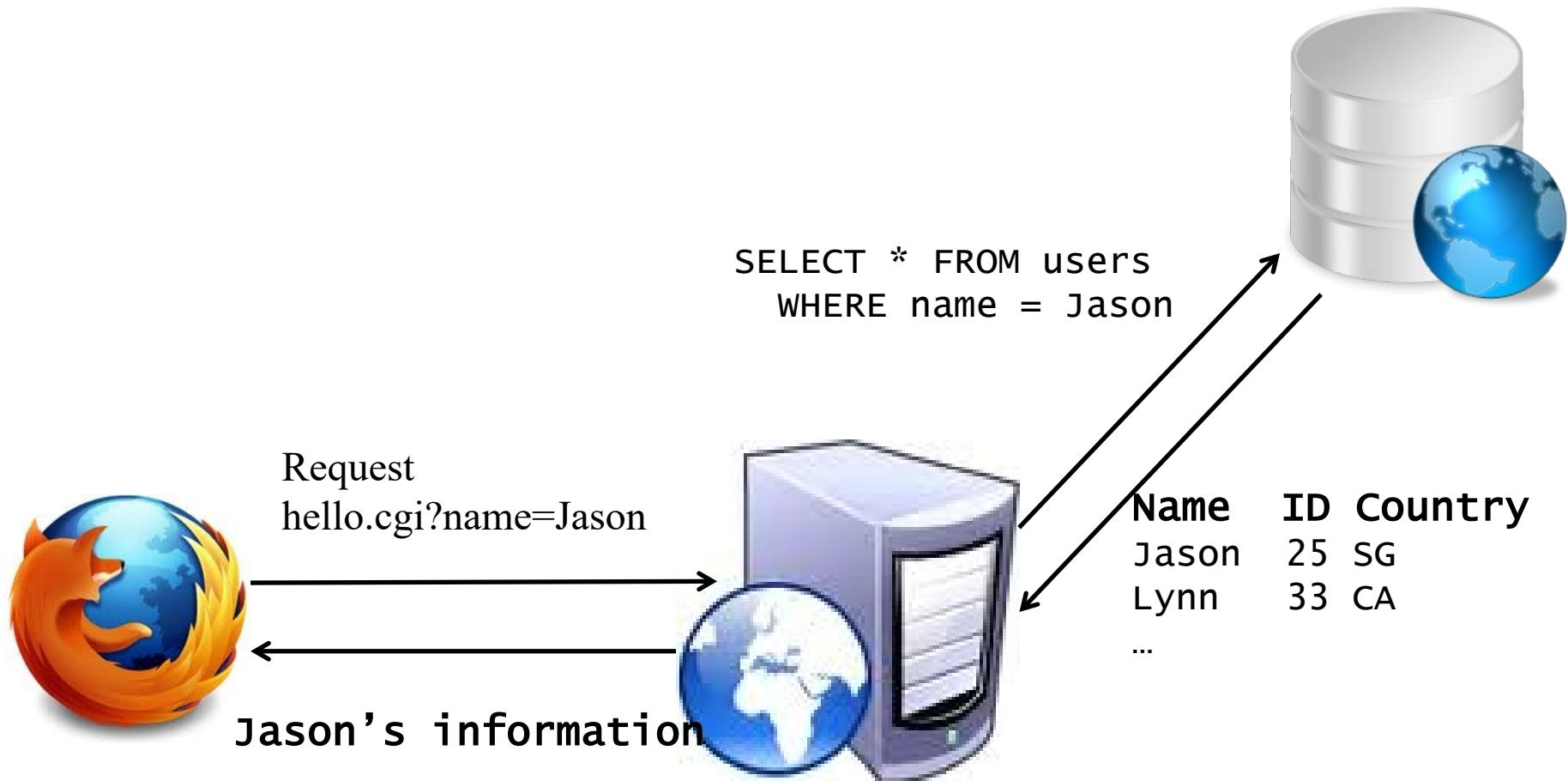
Set-Cookie: theme=light

Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT

Web Client and Server Components

- Client-side components:
 - Hypertext Markup Language (**HTML**): webpage **content**
 - Cascading Style Sheets (**CCS**): webpage **presentation**
 - **JavaScript**: webpage **behavior**, making pages “active” (interactive and responsive)
- Server-side components:
 - **Web server**: nowadays a scripting language is typically used as well, e.g. PHP
 - **Database server**

Three-tiered Web Applications with Database Server



JavaScript for “Active” Pages

- Example of JavaScript in HTML:

```
<script type="text/javascript"> document.write('Hello World!');  
</script>
```

- What can JavaScript do in a browser?

- Write a (variable) **text** into an HTML page:

```
document.write ("<h1>" + studentname + "</h1>")
```

- Read and change HTML elements:

```
var doc = document.childNodes[0];
```

- React to events, such as when a page has finished loading or when a user clicks on an HTML element:

```
<a href="someURL.html" onclick="alert('User just clicked me!')">
```

- Validate user data, e.g. form inputs

- Access cookies!

```
var doccookie = document.cookie;
```

- Interact with the server, e.g. using **AJAX (Asynchronous JavaScript And XML)**

PHP: A Popular Server-Side Scripting Language

- **PHP**: a widely used, free server scripting language for making **dynamic** web pages
- Sample PHP page:

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "My first PHP script!";
?>

</body>
</html>
```

10.2 Security Issues and Threat Models

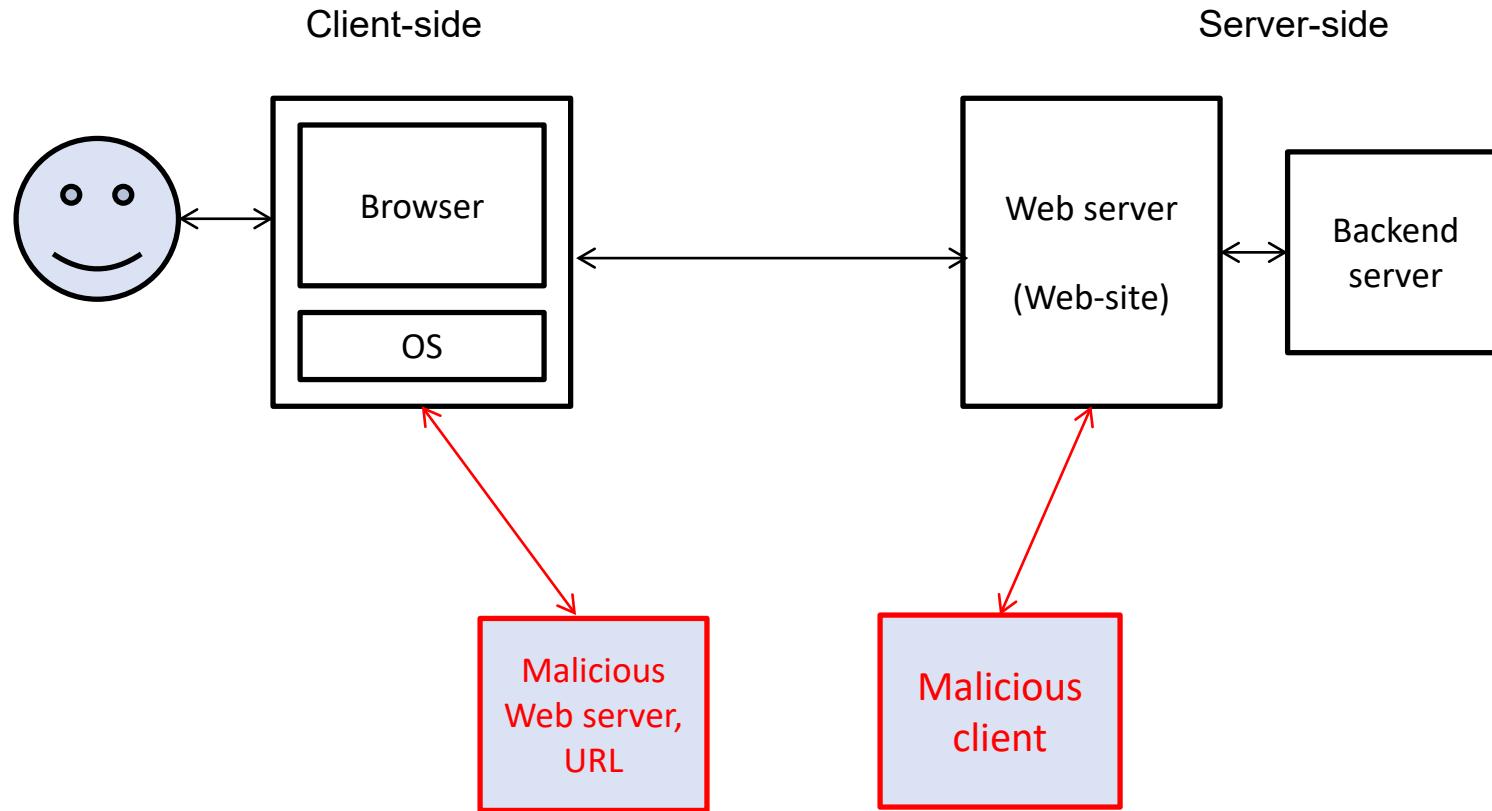
Complications of Web Security: Browsers

- Browsers run with the **same privileges** as the user: the browsers therefore can access the **user's files**
- At any particular instance, **multiple servers** (with different domain names) could provide the content: **access isolation** among sites is thus required
- Browsers support a **rich *command* set** and **controls** for content providers to render the **content**
- Browsers keep **user's info & secrets**: e.g. stored in cookies
- For enhanced functionality, many browsers support **plugins, add-ons, extensions** by third parties
(Note: the definitions and differences of plugins, add-ons, extensions may not be clear & depends on the developers.)

Complications of Web Security: Browser Usage

- Users could **update content** in the server:
e.g. forum, social media sites,
where names are to be displayed
- More and more users' **sensitive data** is stored
in the Web/cloud
- For PC, the browser is becoming the **main/super application**: in some sense, **the browser “is” the OS**

Threat Model 1: Attackers as Another *End Systems*



- In this scenario, the attackers are just another **end systems**
- Examples: a **malicious web server** that lures the victim to visit it; or a **malicious web client** who has access to the targeted server

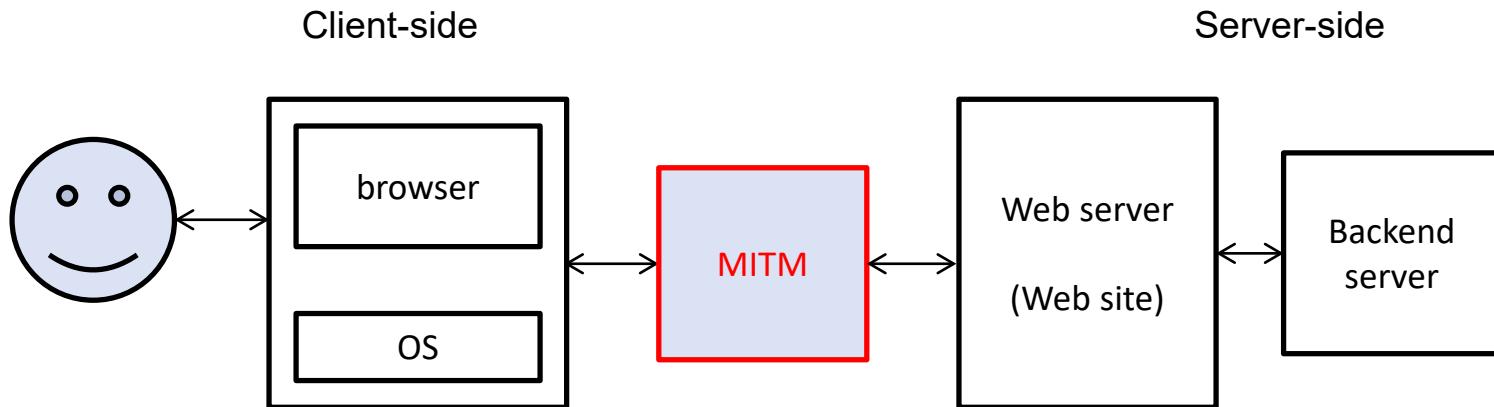
Attacker Types in Threat Model 1

- **1A: Forum poster:**
 - The weakest attacker type
 - A **user** of an existing web app
 - **Doesn't** register domains or host application content
- **1B: Web attacker:**
 - Owns a **valid domain & web server** with an SSL certificate
 - Can **entice** a victim to visit his site:
 - Say via “[Click Here to Get a Free iPad](#)” link
 - Or, via an advertisement (no clicks needed)
 - **Can't** intercept/read traffic for other sites
 - The most **commonly-assumed** attacker type. *Why?*

Attacker Types in Threat Model 1

- **1C: Related-domain attacker:**
 - A web attacker who is able to host content in a **related domain** of the target web application
 - Host on a **sibling or child domain** of the target app
 - E.g. attacker.target.com or attacker.app.target.com, which targets app.target.com
- **1D: Related-path attacker:**
 - A web attacker who is able to host an application on a **different path** than the target application, but **within the same origin**
 - E.g. www.comp.nus.edu.sg/~attacker, which targets www.comp.nus.edu.sg/~target

Threat Model 2: Attackers as a *MITM*



- Here, the attacker is a **Man-in-the-Middle** (at the IP layer)
- Example: a **malicious café-owner** who provides the free WiFi services in our previous examples

Attacker Types in Threat Model 2

- **2A: Passive network attacker:**
 - Eve who can **passively eavesdrop** on network traffic, but cannot manipulate or spoof traffic
 - Can **additionally act** as a web attacker
- **2B: Active network attacker:**
 - Mallory who can launch **active attacks** on a network
 - Can **additionally act** as a web attacker
 - The **most powerful** threat model
 - Yet, it is **not** generally considered to be capable of **presenting valid certificates** for HTTPS sites that are not under his control. *Why not?*

Web Attacks and Classification

- Yet, it can be **difficult** to clearly classify web attacks
- Many attacks uses a **combination** of other attacks
- This lecture describe some **web attacks** and relevant common **protection mechanisms**

10.3 Attacks on the “Secure” Communication Channel (SSL/TLS)

HTTPS

Review

- **HTTPS** protocol:
 - $\text{HTTPS} = \text{HTTP} + \text{TLS/SSL}$
 - Netscape SSL 2.0 [1993] ... TLS 1.3 [2018]
- Provisions a ***secure channel***, which establishes between 2 programs a **data channel** that has **confidentiality, integrity and authenticity**, against a computationally-bounded “network attacker”
- How does HTTPS work?
 - Ciphers negotiation
 - Authenticated key exchange (AKE)
 - Symmetric key encryption and MAC

Attacks on a Secure Channel by a MITM

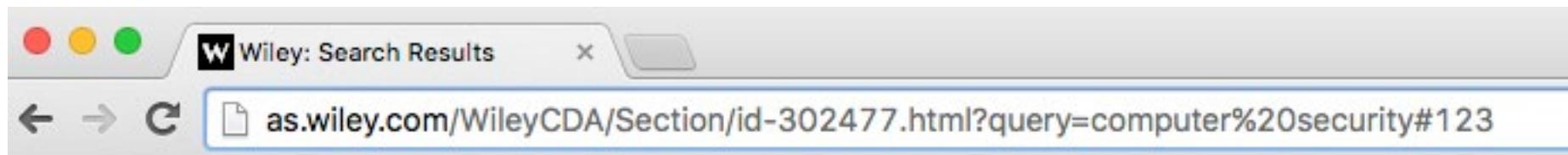
- Two **pre-conditions** of a MITM attack:
 - The attacker is a MITM in between the browser and web server
 - The attacker is able to sniff & spoof packets at the TCP/IP layers
- Note that if the connection is HTTPS, such MITM is **unable** to compromise both confidentiality & authenticity, **unless**:
 - Web user accepts a **forged certificate** or a **rouge CA**
 - Yet, this might not be the case when **there exist vulnerabilities** in the protocol or its **implementation**
- We have already covered some **HTTPS attack** examples:
FREAK attack, Superfish, Heartbleed, re-negotiation attack
(attack on protocol design)
- Other well-known attacks (*not required in this module*):
BEAST attacks (attack on cryptography)

10.4 Mislead the User

URL (Uniform Resource Locator)

- A URL consists of a few components (see https://en.wikipedia.org/wiki/Uniform_Resource_Locator):
 1. Scheme
 2. Authority (a.k.a the hostname)
 3. Path
 4. Query
 5. Fragment
- Example:

<http://www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security#123>



WILEY

Question: Why the URL is typically displayed with **two levels** of intensity?

URL: Possible Source of Confusion

- Suppose there is **no** clear visual distinction between the “hostname” and “path” of a URL
- The **delimiter** that separates hostname & the path can be a character **in the hostname or path**

www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security

The URL "www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security" is shown. A red bracket is placed under the first part "www.wiley.com", and another red bracket is placed under the second part "WileyCDA/Section/id-302477.html". Below the first bracket is the word "Hostname" and below the second bracket is the word "path".

- Example: a malicious Website whose hostname contains the targeted hostname followed by a character resembling the **delimiter** “/” (e.g. www.wiley.com.Iwiley.in/Section/id-302477.html)
- Another example: nuslogin.789greeting.co.uk (from phishing email)
- The displayed different intensities could help user **spot** the attack

Address Bar Spoofing

- **Address bar** is an important browser's component to protect: the **only indicator** of what URL the page is actually rendering
- *What if the address bar can be “modified” by a webpage?*
- An attacker could trick the user to **visit** a malicious URL X , while making the user **wrongly believe** that the URL is Y
- A poorly-designed browser may allow attacker to achieve the attack

Address Bar Spoofing: Example

- In the early design of some browsers, a web page could render objects/pop-ups in **an arbitrary location**
- This allows a malicious page to **overlay a spoofed address bar** on top of **the actual address bar**
- Current versions of popular browsers have mechanisms to prevent this issue
- Yet, a recent attack, e.g.: [Android Browser All Versions - Address Bar Spoofing Vulnerability - CVE-2015-3830](#)
(<https://www.rafaybaloch.com/2017/06/android-browser-all-versions-address.html>)



10.5 Cookies and the Same-Origin Policy

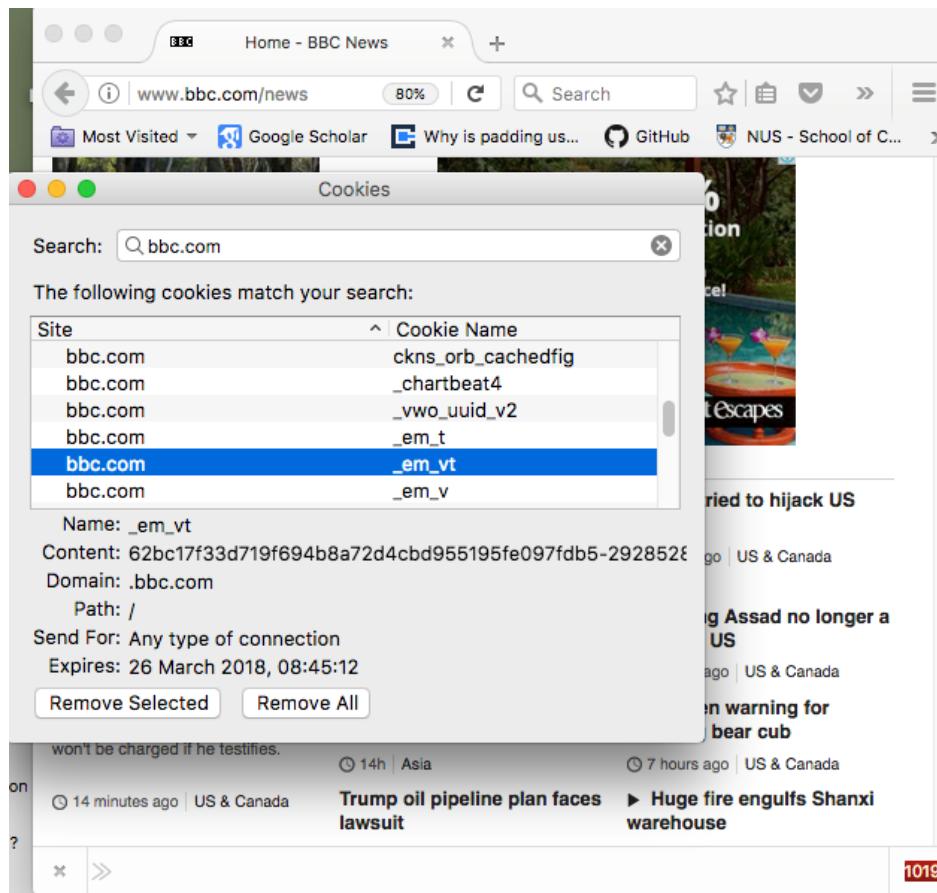
Remark:

The same-origin policy (SOP) is **not** an attack, but is a protection mechanism to protect cookies.

Cookies

- A **HTTP cookie**: a piece of textual data sent by a Web server and stored on the user's web browser while the user is browsing
- A cookie sent by the **web server**:
in an HTTP response's “**Set-Cookie**” header field
- A cookie consists of a **name-value pair**: can be used to indicate a **user preference**, shopping cart content, a server-based **session identifier**, etc
- Whenever a client revisits the Website (i.e. submit another HTTP request), the browser **automatically** sends all “**in-scope**” **cookies** back to the server in its HTTP request's “**Cookie**” header
- Note that cookies are sent back only to the “**same cookie origin**”: to the server that is the “origin” of the cookies
(Note: the scheme/protocol checking *may* be optional)

Viewing Cookies



On Firefox:

- Right-click → View Page Info → Security → View Cookies; or
- Tools → Web developer → Developer toolbar → Storage

On Chrome: chrome://settings/content/cookies

Cookies: Usage

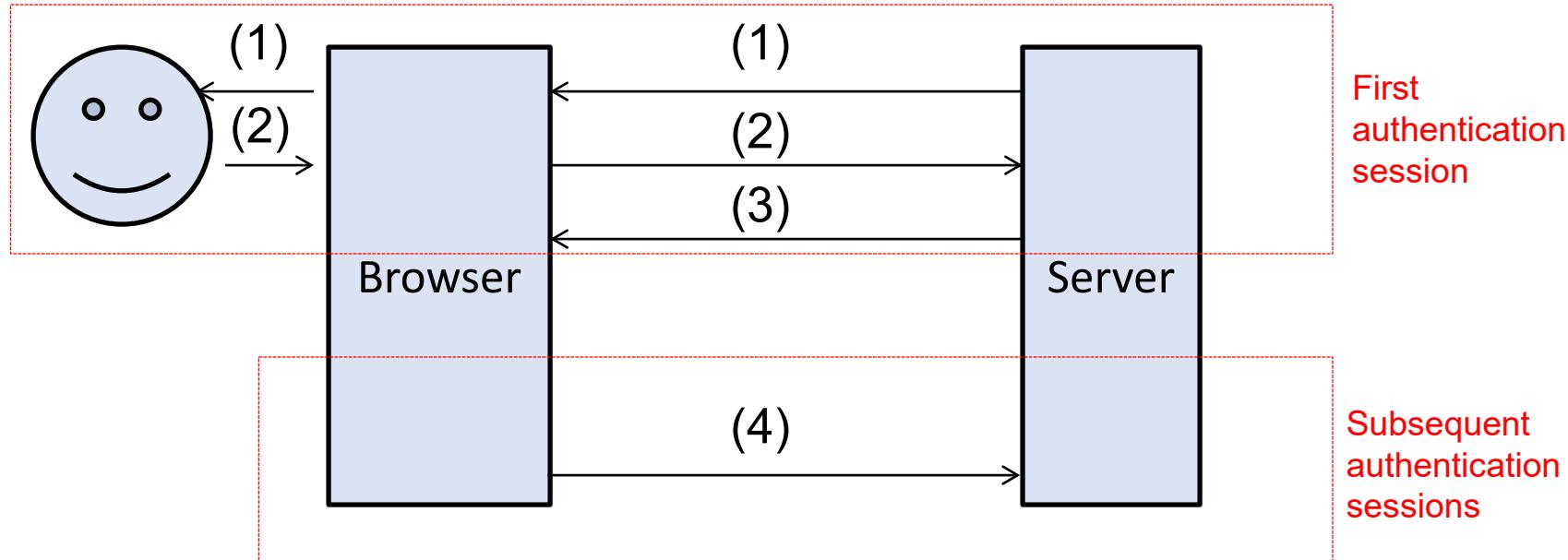


- There are **a few types** of cookie, such as:
 - **Session cookie**: deleted after the browsing session ends
 - **Persistent cookie**: expires at a specific date or after a specific length of time
 - **Secure cookie**: can only be transmitted over **HTTPS**
See https://en.wikipedia.org/wiki/HTTP_cookie#Terminology
- Note: the checking on scheme in the “same origin” for cookies is **optional**, except for **secure cookies** which strictly require HTTPS
- Since HTTP is **stateless**, there is a need to keep track of a **web session**
- Cookie is commonly used to **set** and **indicate** a **session ID**
- **Cookie** is a better approach than attaching the session ID as a **URL-encoded parameter** in the HTTP request or as a **form field**, but it has its own issues too

Token-based Authentication and Cookie

- To ease a web user's tedious task of repeated logins, many web sites use “**token-based**” authentication:
 1. After a user A is **authenticated** (for e.g. **password verified**), the server sends a value t , known as the **token**, to A
 2. In subsequent connections, whoever presents the **correct token** is thus **accepted** as the authentic user A
- **Remarks:**
 - A token typically has an **expiry date**
 - A token can identify a **session**: hence, the token is also called **session ID (SID)**
 - In web applications, a token is often stored in a **cookie**

Token-based Authentication: Diagram



- (1) **Authentication challenge** (e.g. asking for password)
- (2) **Authentication response** that involves the user
- (3) Server sends a token t , and Browser keeps the token t
- (4) Browser presents the token t with HTTP request,
and Server verifies the token t

Note: We assume that the **communication channel is secure**: it is done over HTTPS (with server being authenticated) and the HTTPS is free from vulnerabilities.

Storage Requirement and Choices of Token

- A token t needs to be **random** and sufficiently **long**
- Suppose token t is a **randomly chosen number**, then the server has to keep a **table** storing all issued tokens
- To **avoid** storing the table, one could use:
 - (**Insecure**) The cookie is some meaningful information concatenated with a **predictable** sequence number
 - E.g $t = \text{"alicetan:16/04/2015:128829"}$
 - (**Secure**) The cookie consists of **two parts**: a randomly chosen value or meaningful information like the expiry date; and concatenated with the **message authentication code (MAC)** computed using the server's secret key
 - E.g $t = \text{"alicetan:16/04/2015:adc8213kjd891067ad9993a"}$

Storage Requirement and Choices of Token

- For both methods, when the server finds out that the token is ***not*** in the correct format (or ***not*** the correct MAC), the server **rejects** the token
- The **first** method is **insecure**: an attacker, who knows how the token is generated (e.g. by observing its own token), can **forge it**
- This illustrates the weakness of “security by obscurity”: a wrong assumption that attackers don’t know the format
- The **second** method is **secure**: it relies on the security of MAC

Scripts & Same-Origin Policy (SOP): Browser Access Control

- A **script** that runs in the browser could access **cookies**
- Important question: **which scripts can access what cookies?**
- Due to security concern, browser employs the following **access control** mechanism
- The **script** in a web page *A* (identified by its URL) can access **cookies** stored by another web page *B* (identified by its URL), only if both *A* and *B* have the ***same origin***
- **Origin** is defined as the combination of:
protocol, hostname, and port number
- The above is simple and thus seemingly safe
- However, there are a number of possible **complications**

Same-Origin Policy (SOP): Some Complications

- Example of **origin determination rules**:

URLs with the same origin as <http://www.example.com>
(from http://en.wikipedia.org/wiki/Same-origin_policy)

| Compared URL | Outcome | Reason |
|--|---------|--|
| http://www.example.com/dir/page2.html | Success | Same protocol, host and port |
| http://www.example.com/dir2/other.html | Success | Same protocol, host and port |
| http://username:password@www.example.com /dir2/other.html | Success | Same protocol, host and port |
| http://www.example.com:81/dir/other.html | Failure | Same protocol and host but different port |
| https://www.example.com/dir/other.html | Failure | Different protocol |
| http://en.example.com/dir/other.html | Failure | Different host |
| http://example.com/dir/other.html | Failure | Different host (exact match required) |
| http://v2.www.example.com/dir/other.html | Failure | Different host (exact match required) |
| http://www.example.com:80/dir/other.html | Depends | Port explicit. Depends on implementation in browser. |

- Limitation:** there are many exceptions, and exceptions of exceptions: very confusing and thus prone to errors
- An example: unlike other browsers, **Microsoft IE** does *not* include the port in the calculation of the origin, using the **Security Zone** in its place
(See <https://blogs.msdn.microsoft.com/ieinternals/2009/08/28/same-origin-policy-part-1-no-peeking/>.)

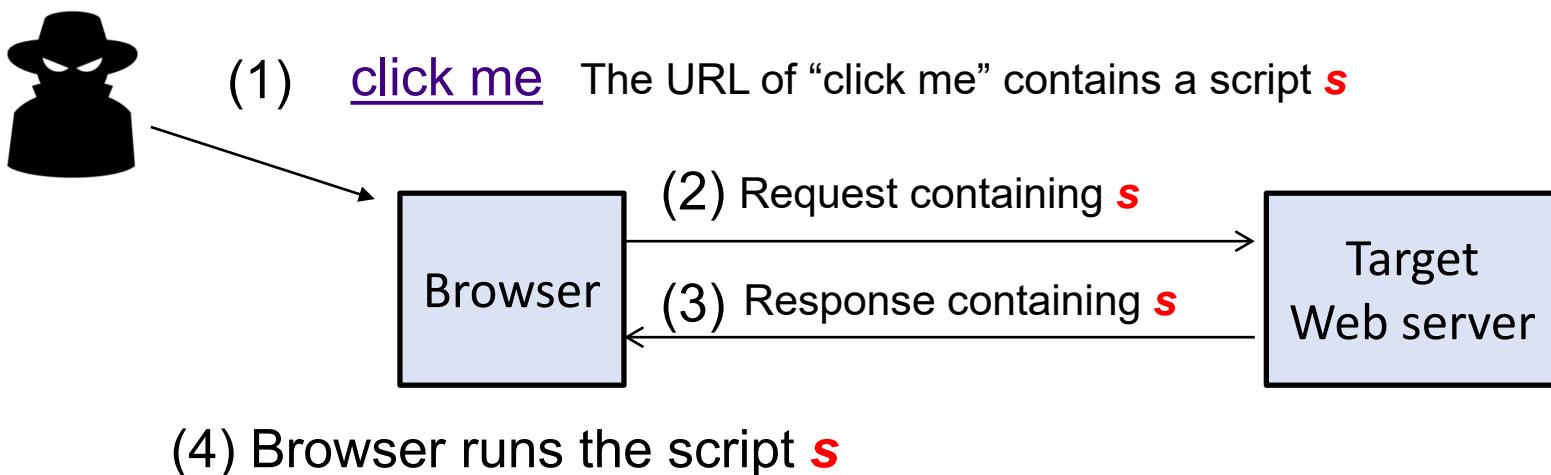
10.6 Cross Site Scripting (XSS) Attacks

Reflected (Non-Persistent) XSS Attack: Background

- In many web sites, the client can enter a string s in the browser, which is to be sent to the server
- The server then responds with a HTML **containing s** , which is then rendered and displayed by the client's browser
- Examples:
 - Enter a wrong address:
[http://www.comp.nus.edu.sg/nonsense test](http://www.comp.nus.edu.sg/nonsense_test)
 - Search for a book in library:
<http://nus.preview.summon.serialssolutions.com/#!/search?q=heeheeheee>
- Important question: what if the string s contains a **script**?
 - Example: [http://www.comp.nus.edu.sg/<script>alert\('heehee!'\);</script>](http://www.comp.nus.edu.sg/<script>alert('heehee!');</script>)
- Note that the attack above **won't** work if the server performs **HTML (entity) encoding**: replaces the special character "<" with <

Reflected (Non-Persistent) XSS Attack: Attack

1. The attacker tricks a user to **click** on a URL, which contains the target website and a malicious script **s** (For example, the link could be sent via email with “click me”, or a link in a malicious website.)
2. The request is **sent to the server**
3. The server constructs a **response HTML**: the server doesn’t check the request carefully, and its response **contains s**
4. The browser renders the HTML page, and **runs** the script **s**



Why is This an Attack?

- A script can be **benign**
- However, a **malicious** script could:
 1. Deface the original Webpage
 2. Steal cookies
- Recall the same-origin policy:
Because the script comes from the **target web server**, it can **access cookies** previously sent by the web server
- This is an example of **privilege escalation**:
a malicious script from the attacker has the privilege of the web server and read the cookies
- The attack above exploits the **client's trust of the server**:
the browser believes that the injected script is from the server

Stored (Persistent) XSS

- The script **s** is **stored** in the target web server
- For instance, it is stored in a **forum page**:
the attacker is a **malicious forum poster**
- Another example: **Samy XSS worms** on Myspace.com,
where Samy became a friend of 1M users in less than 20 hours!
(See [https://en.wikipedia.org/wiki/Samy_\(computer_worm\)](https://en.wikipedia.org/wiki/Samy_(computer_worm)))
- **More dangerous** than reflected XSS attacks:
 - The malicious script is **rendered automatically**,
without the need to lure target victims to a 3rd-party web site
 - The victim-to-script ratio is **many:1**

Stored (Persistent) XSS

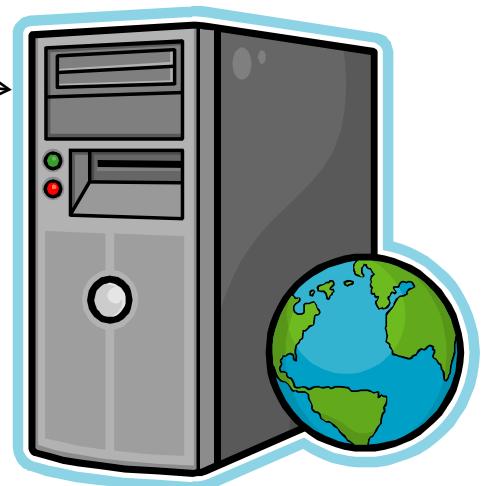


http://victim.com/store.cgi?<XSSCODE>

http://victim.com/view.cgi



Page with
<XSSCODE>



XSS Attacks: Summary

- What is XSS in short?
 - "A type of **injection attack** on web apps, whereby a **forum poster** or **web attacker** attacks **another web user** by causing the latter run a (malicious) script from the former in the **execution context** of a page from **an involved web server**, thus subverting the Same Origin Policy."
- The attack works by exploiting **the victim's trust of the involved server**:
 - In reflected XSS:
the web server that **returns** a page reflecting the injected script
 - In persistent XSS:
the web server that **stores** a page containing the injected script

Defenses

- Most defense rely on mechanisms carried out in the **server-side**:
 - The server **filters and removes** any malicious script in a **HTTP request** while constructing its response page
 - The server **filters and removes** any malicious script in a **user's post** before it is saved into the forum database
- Some example techniques:
 - Script filtering
 - Noscript region:
do not allow JavaScript to appear in certain region of a Web page.
- However, this defense is **not** a fool-proof method
- To additionally detect reflected XSS attack, some browsers employ a **client-side** detection mechanism: e.g. **XSS auditor**

10.7 Cross Site Request Forgery (CSRF) Attacks

CSRF Attack: *With the Victim Clicking on a URL*

A.k.a. “sea surf”, cross-site reference forgery, session riding

An attack **example** (with the victim clicking on a URL):

- Suppose a client Alice is **already** authenticated by a target website S , say www.bank.com, and S accepts an **authentication-token cookie**
- The attacker Bob tricks Alice to click on a **URL** of S , which maliciously **requests** for a service, say transferring \$1,000 to Bob:
www.bank.com/transfer?account=Alice&amount=1000&to=Bob
- Alice’s cookie will also be **automatically sent** to S , indicating that the request comes from already-authenticated Alice
- Hence, the transaction will be **carried out**

(For more details, see https://en.wikipedia.org/wiki/Cross-site_request_forgery.)

CSRF Attack: *Without* the Victim Clicking on a URL

A **web attacker** can also performs a CSRF attack **without** any victim user's UI actions

An attack **example** (**without** the victim clicking a URL):

- Again, suppose Alice is **already** authenticated by a target website S (www.bank.com), and S accepts an authentication-token cookie
- **Alice visits the attacker's site**, whose page contains the following:
`
WIDTH="1" HEIGHT="1" BORDER="0">`
- Alice's browser issues **another HTTP request** to obtain the image
- Alice's cookie will also be **automatically sent** to S
- Hence, the transaction will be **carried out**

CSRF Attacks

- What is the **CSRF** in short?

"A type of **authorization attack** on web apps, whereby a **web attacker** attacks **a web user** by issuing a **forged request** to **a vulnerable web server** 'on behalf' of the victim user."
- The attack disrupts the **integrity** of the target user's session
- This is, in a way, the reverse of XSS:
it exploits **the server's trust of the client**
(the server believes that the request is from the client)

Defenses

- Relatively easier to prevent compared to XSS
- The **SID/authentication-token cookie** automatically sent by the browser is *insufficient*: the server must issue and require an **extra information**, i.e. **anti-CSRF token**
- For example, the server **includes** a (dynamic) anti-CSRF token in its **money-transfer request** page
- The anti-CSRF token can be included in a **URL**:
`www.bank.com/transfer?account=Alice&amount=1000&to=Bob&
Token=xxk34n890ad7casdf897e324`
- It is also possible to include the anti-CSRF token inside a **HTTP request header** or a **hidden form field**

Other Web Attacks and Terminologies

- **Drive-by download**
- **Web bug** (aka Web beacon, tracking bug, tag, page tag).
- **Clickjacking** (User Interface redress attack)
See <https://www.owasp.org/index.php/Clickjacking>
- **CAPTCHA**
- **Click fraud**

Question: Could **merely visiting** a malicious web site
(for e.g. by clicking on a phishing email) make a web user
become a subject to web attacks?

CS2107 Review + Final Exam Tips

Teaching Mode

- **13** Lectures: including 2 guest lectures from industry & government
- **9** Tutorials
- Continual Assessment (50%):
 - 2 Assignments (25%): **Do submit A2 before its deadline**
 - 1 Mid-term quiz (15%)
 - 1 Group presentation on open-ended topic (5%)
 - **1 LumiNUS online quiz assessment (5%)**
- **Final Exam (50%)**: Open-book, Saturday 23 Nov, 09:00-11:00
Please double-check the timing & venue with CORS again!

Module Description

Objective

This module serves as an introductory module on information security. It **illustrates** the **fundamentals of how systems fail** due to malicious activities **and how they can be protected**. The module also places emphasis on the practices of secure programming and implementation. Topics covered include **classical/historical ciphers**, **introduction to modern ciphers** and cryptosystems, ethical, legal and organisational aspects, classic examples of direct attacks on computer systems such as **input validation vulnerability**, examples of other forms of attack such as **social engineering/phishing attacks**, and the **practice of secure programming**.

Outcomes

- Awareness of common and well-known attacks (e.g. phishing, XSS, SQLI, ...)
- Understand basic concepts of security (e.g. confidentiality, availability, ...)
- Understand basic mechanisms & practice of protections (e.g. crypto, PKI, access control, ...)
- Awareness of common pitfalls in implementation (Secure programming)

More Specific Intended Learning Outcome (ILO)

After completing the module, you will be expected to be able to:

1. Explain the *C-I-A security requirements* and recognize their breaches in recent security incident news
2. Describe *key concepts and basic mechanisms* of principal protection mechanisms in information security, such as encryption, authentication, and access control
3. Identify the *limitations of classical cryptographic schemes*, and recognize *well-known attacks* on vulnerable hosts, networks, and Web servers

More Specific Intended Learning Outcome

4. Utilize some *basic security tools* (e.g. OpenSSL, Wireshark) and security-related *Linux commands* to perform encryption, network traffic analysis, and file access control
5. Pinpoint flaws in programs due to *common insecure programming practices*, and suggest improvements using more secure practices instead

Some of the terminologies encountered in this modules

Secure channel, Alice, Bob, Eve, Encryption, Decryption, Key-space, Known-plaintext attack, Authenticity, Confidentiality, availability, Authentication protocol, man-in-the-middle, Passwords, Dictionary attack, random IV, Kerckhoff's principle.

Side-channel attack, timing attack, ATM skimmer, Social Engineering.

DDOS, Syn flood, WPA, SSL, Wireshark, Spoofing, Sniffing, Poisoning, Public Key Infrastructure, Digital Signature, RSA, Certificate, Tor.

Input validation, SQL injection, Secure Programming, buffer overflow, Stack smashing, Integer Overflow, TOCTOU, CVE.

Key getLogger, virus, worm, rootkit, botnet.

Access Control List, Capability, rwx, superuser, root, Least Privileges, Privilege escalation, Reference Monitor.

Completed Lectures

Lecture 1: Encryption

Security requirements, encryption/cryptography, key length, IV, Kerckhoffs's principle

Lecture 2: Authentication (weak)

Password, 2FA, confidentiality $\not\Rightarrow$ integrity, phishing

Lecture 3: Authentication (strong)

PKC, hash, MAC, signature, birthday paradox, strong authentication

Lecture 4: PKI, SSL

PKI, signature, certificate, CA, authentication protocol, key exchange, SSL/TLS

Lecture 5: Network Security

Layering, naming issue (DNS attack), DDoS, firewall

Lecture 6: Access Control

Access control matrix, UNIX access control, privilege escalation

Lecture 7 : Secure Programming (I)

Background on computer architecture, call stack, format string vulnerability

Lecture 8 : Secure Programming (II)

Various pitfalls, data representation, buffer overflow, integer overflow

Lecture 9 : Secure Programming (III)

Android security, TOCTOU, problem with scripting languages, counter measures

Lecture 10: Web Security

Completed Tutorials

- **Tutorial 1: Introduction & encryption**
Security requirement, key length requirement, tradeoff of usability & security
- **Tutorial 2: Encryption & password**
Password, security questions, 2FA, role of IV
- **Tutorial 3: Data-origin authentication**
Birthday attack, hash, secure random number generation, implementation issue on secret key generation (which illustrates that hash doesn't produce truly random sequence)
- **Tutorial 4: PKI, SSL and Birthday attack variant**
PKI, proxy-re-encryption, limitation of PKI, variant of birthday attack
- **Tutorial 5: Security protocol - renegotiation attack on SSL/TLS**
SSL/TLS, re-negotiation attack (which illustrates subtlety of protocol design)
- **Mid-term quiz discussion**
- **Tutorial 6: Network security + access control**
Firewall design (2-firewall setting, DMZ), access control (using LumiNUS as an example)
- **Tutorial 7: Privilege escalation**
SetUID, privilege escalation, how programming bug leads to security vulnerability
- **Tutorial 8: Software security**
Format string & buffer overflow vulnerabilities, safe/unsafe C functions, integer overflow
- **Group presentations (2 sessions)**

Assignments: CTF Style

- For hacking-challenge ***gamification***: phased hint releases, possible task-completion dependency, etc.
- For **automated** challenge-submission **marking**: real-time and scalable checking of submission attempts, *mark scoreboard*
- Assignment 1:
Cryptography, authentication
- Assignment 2:
Network, software and web security
- Additional **online quiz assessment** via LumiNUS:
for overall material review and final-exam practice

Ethical Use of Security Information

- We have discussed **vulnerabilities and attacks**
- Most vulnerabilities have been fixed, *but*:
 - **Do not** assume that all systems are patched/fixed
 - **Some attacks** may still cause harm!
- Purpose of our security modules:
 - Learn to prevent malicious **attacks**
 - Use your knowledge for **good** purposes

Hacking: It's Fun, Don't Cross the Yellow Line



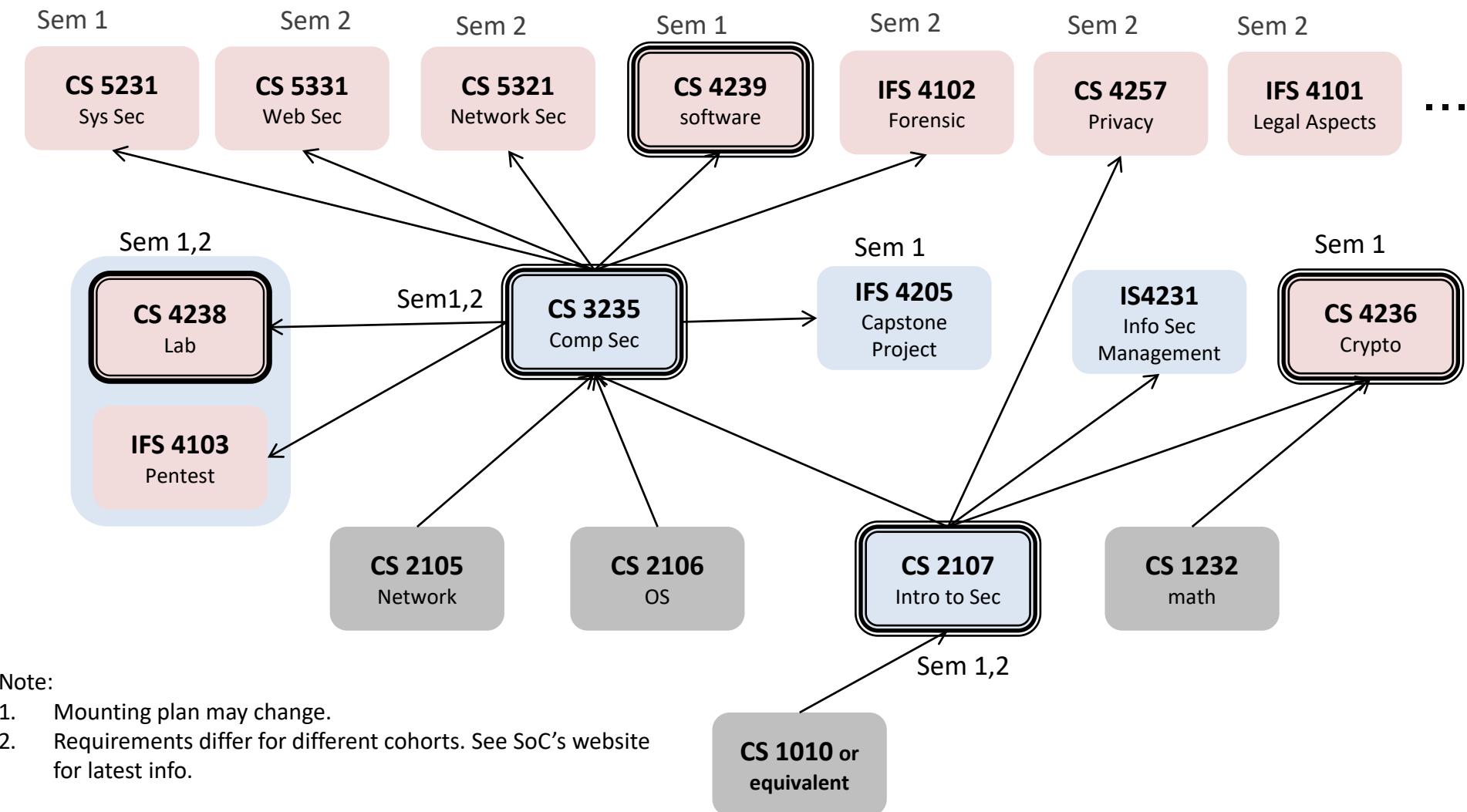
Next Steps

Security-Related Modules in SOC

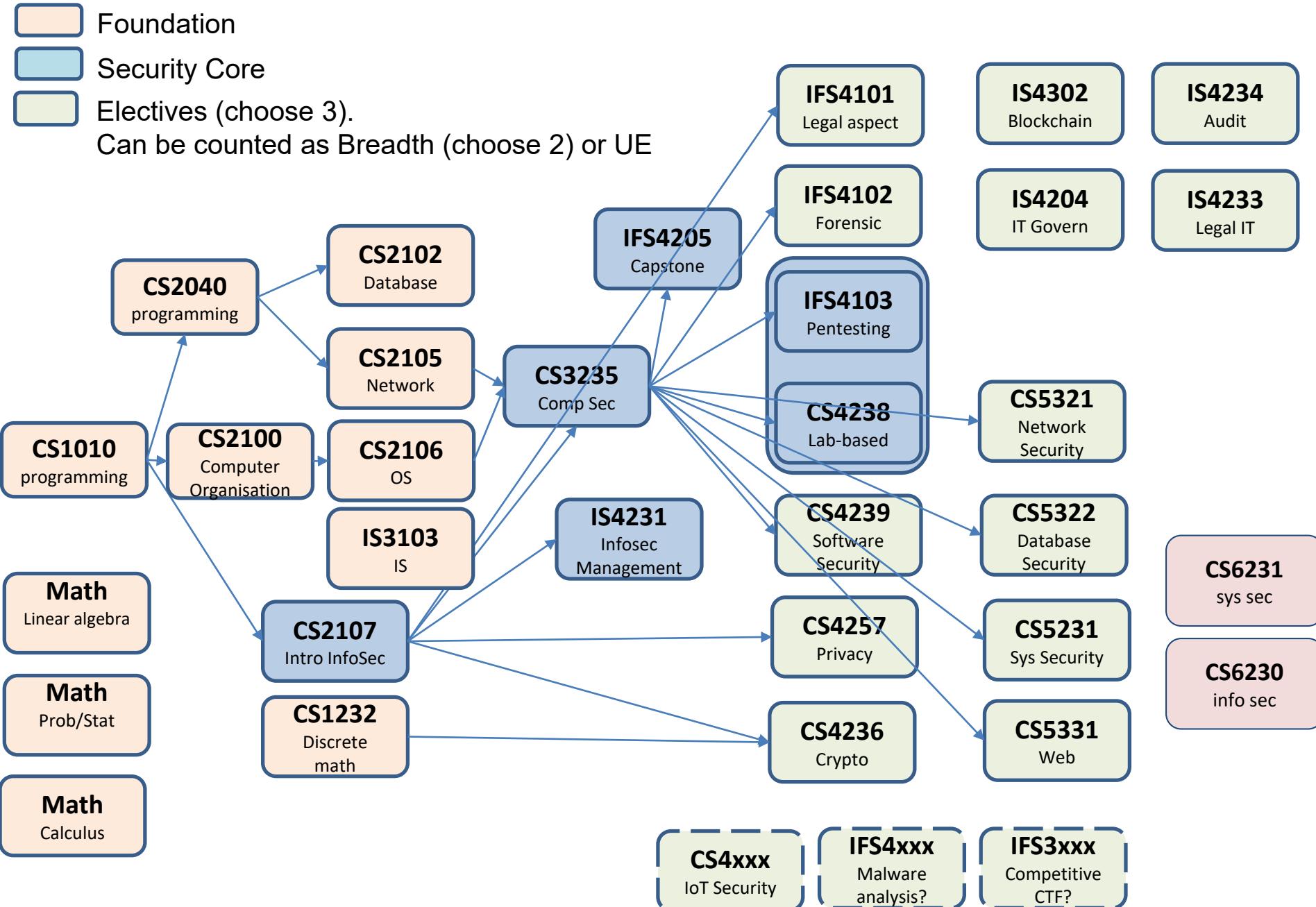
cores in InfoSec degree

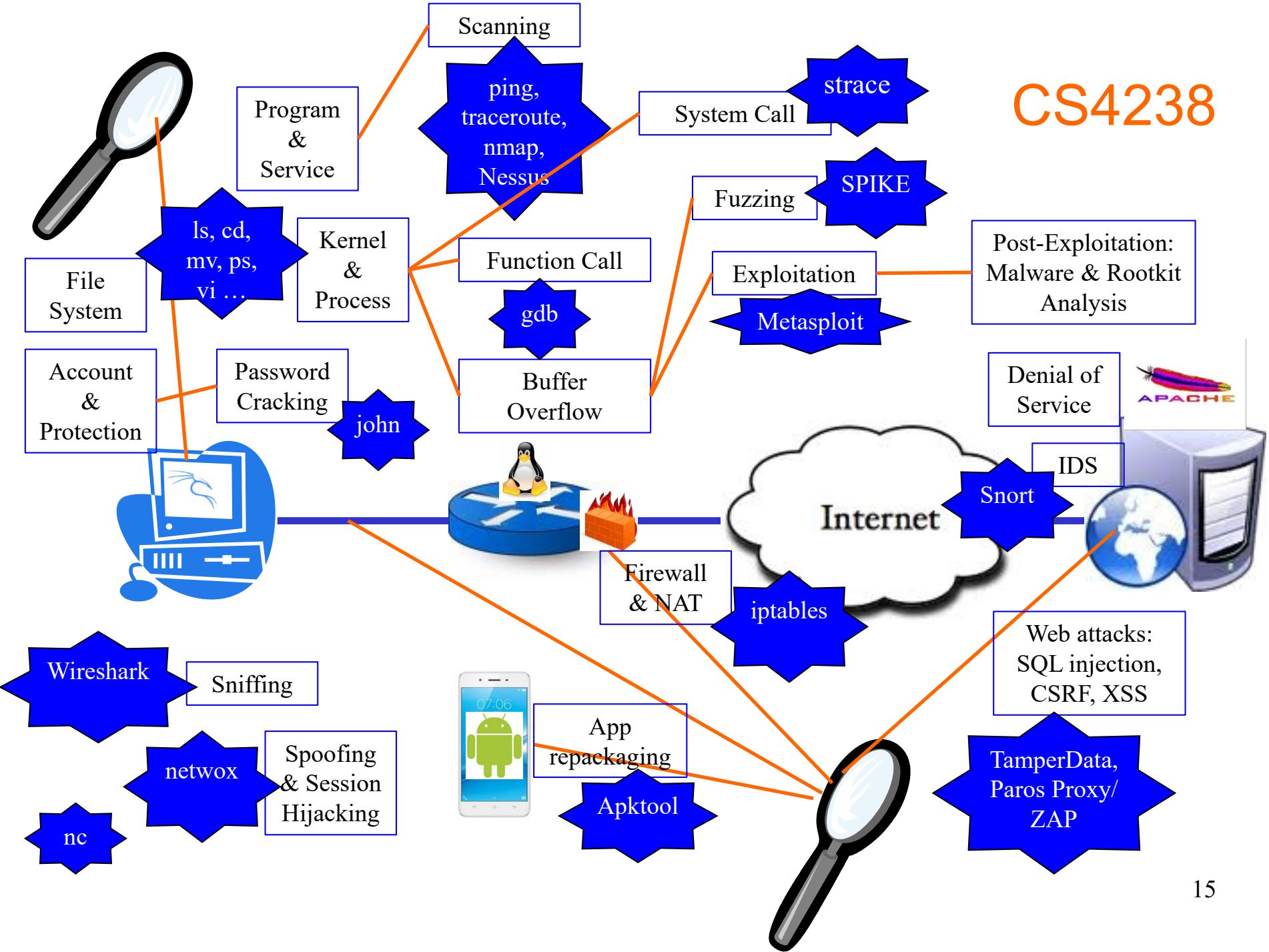
Electives in InfoSec degree
(choose 3)

Security Area Focus
(choose 3)



Security-Related Modules and BCOMP InfoSec Requirements





Recent News Items (Oct 2016)

NUS, Singtel launch \$43 million cyber security laboratory



The NUS-Singtel Cyber Security Research and Development Laboratory, hosted by the NUS School of Computing, is the 10th laboratory supported under the Laboratory@University scheme by the NRF. PHOTO: ST FILE

⌚ PUBLISHED 3 HOURS AGO | UPDATED 1 HOUR AGO

Irene Tham Tech Editor (<mailto:itham@sph.com.sg>)

Recent News Items (2017)

www.channelnewsasia.com/news/singapore/singapore-to-set-up-new-defence-cyber-organisation

Singapore to set up new Defence Cyber Organisation

National servicemen could also be selected for cyberdefence vocations as the army seeks to bolster itself against infocomm threats.

Posted 03 Mar 2017 12:44 Updated 03 Mar 2017 22:32



Photo illustration. (Photo: REUTERS/Kacper Pempel/Files)

REUTERS

Channel News Asia, Mar 3, 2017

1087 [f](#) [t](#) [in](#) [Email](#) [More](#)

CAPTION

SINGAPORE: A new Defence Cyber Organisation (DCO) will be set up to monitor and defend the Singapore Armed Forces' (SAF) networks around-the-clock from cyberthreats, Defence Minister Ng Eng Hen announced in Parliament on Friday (Mar 3).

Recent News Items (2017)

The screenshot shows a web browser window with the URL www.channelnewsasia.com/news/singapore/singapore-to-set-up-new-defence-unit. The page title is "Singapore to set up new defence unit". The main content discusses the creation of a Cyber Defence Group and the "WANTED: CYBERDEFENDERS" vocation.

The Cyber Defence Group consists of a security monitoring unit, an incident response and audit unit as well as the Cyber Defence Test and Evaluation Centre (CyTEC). Opened in 2015, CyTEC facilitates network security testing and conducts training, among others.

WANTED: CYBERDEFENDERS

The SAF has also created a new cyberdefence vocation for both full-time and operationally ready national servicemen. Those who have demonstrated their abilities at cyber competitions, as well as those currently working in the cybersecurity industry, may also be selected and identified to be "cyberdefenders".

"Our cyberdefenders will need to possess a high level of skill given the increasing frequency and complexity of cyberattacks," said Second Minister for Defence Ong Ye Kung. "They will be entering a very selective and demanding vocation, comparable to the commandos or naval divers."

In their vocation, which will be implemented from August, they are expected to perform roles such as monitoring networks and systems, responding to incidents and forensic analysis. As a pilot project, they may also be deployed to support the Cyber Security Agency to defend critical information infrastructure supporting Singapore's key networks.

MINDEF also announced that the Headquarters Signals and Command Systems, which includes the SAF training institute for cyberdefence, will sign a memorandum of understanding with Singapore Technologies Electronics (Info-Security) and Nanyang Polytechnic this month.

- CNA/jo

Channel
News Asia,
Mar 3, 2017

Recent News Items (Oct 2016)

THE STRAITS TIMES

Strengthening our cyber defences

Cyber security = job security for Singapore grads



From left: Mr Ang Yihan, 25, Mr Winwin Lim, 26, Mr Ian Yeo, 28, Mr Kelvin Tan, 28, and Mr Lee Wei Yan, 27, at the Kaspersky Lab headquarters in Moscow. The fresh graduates were in Russia for a one-year IT security attachment and training programme. PHOTO: KASPERSKY LAB

⌚ PUBLISHED OCT 23, 2016, 5:00 AM SGT

From Singapore to Moscow, such is the demand for professionals in this sector that the sky's the limit

The Straits Times,
Oct 23, 2016

The Rest of the Semester:

Final Exam

Final Exam

- Open book, **2** hours, NUS approved calculators, total: **50** marks
- Saturday, 23 Nov morning (*please double-check time & venue again!*)
- **Format:**
 - Q1: Terminology (10 marks)
 - Q2: MCQs (10 marks)
 - Q3: Short answer questions (10 marks): answer in 2-3 sentences or with a diagram
 - Q4: Scenario-based questions (20 marks)
- **Covered materials:** all lectures and tutorials, which also include:
 - Cryptography
 - Authentication
 - Network security
 - Firewall design
 - Access control
 - Secure programming
 - Web security

NATIONAL UNIVERSITY OF SINGAPORE

CS2107 — INTRODUCTION TO INFORMATION SECURITY

(Semester 1: AY2019/20)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. This assessment paper contains **FOUR** questions and comprises **SIXTEEN** printed pages.
3. Answer **ALL** questions.
4. Write your answer within the given box in each question on this question paper.
5. This is an **OPEN BOOK** assessment.
6. You may use **NUS APPROVED CALCULATORS**.

Nonetheless, you should be able to work out the answers without using a calculator.

Student Number: _____

This portion is for examiner's use only:

| Question | Full Marks | Marks | Remarks |
|----------|------------|-------|---------|
| Q1 | 10 | | |
| Q2 | 10 | | |
| Q3 | 10 | | |
| Q4 | 20 | | |
| Total | 50 | | |

Thanks! (And Please Congratulate Yourself Too!)

