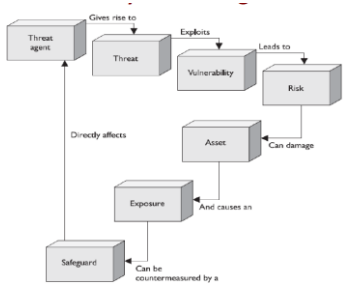
CS2107 Cheatsheet

Intro to Info Security

Graphical passwords - password where users input the sequence etc of how they chose the images as a form of authentication
Covert Channel - using channels for data transfer that was not intended for original use (unauthorized)
Covert channel attack - an attack where communication is transmitted over a channel that was not intended for communication
Side channel attack - any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself.
End-to-end encryption - a system of communication where only the communicating users can read the messages.
Key escrow - an arrangement in which the keys needed to decrypt encrypted data are held in escrow so that, under certain circumstances, an authorized third party may gain access to those keys
ps aux/ps -ef – display list of running processes
cp src file dest_file – copy file from src to dest if there was alrd an a.txt, and we do cp a.txt b.txt, we will create a b.txt and copy the file contents inside

Assets, Threat, Vulnerability, Control
Assets: Object of value
Threat: Set of circumstances that has the potential to cause loss or harm
Vulnerability: a weakness in the system
Control: Security mechanism / countermeasure to counter threats



Types of controls

CIA Triad
Confidentiality: Ability to ensure an asset is viewed only by authorized parties
→ You have secure ciphertext = confidential
Integrity: An asset is modified only by authorized parties
→ You have some way to prove that an asset is not modified; digital signatures, ordering etc
Availability: ability to ensure an asset can be used by any authorized parties
→ Possible attacks include DDoS (distributed denial of service attacks) by flooding servers with http request.

Other requirements:
Authenticity: similar to integrity.
→ Logins, passwords
Accountability: non-repudiation of a prior commitment. Digital signatures

Challenges to Security
Security often not considered during early design stage, difficult to formulate security requirements, systems hard to implement, deployed system most vulnerable at weakest point

Security Tradeoffs
- Ease of use: Interferes with working patterns user originally familiar with
- Performance: consume more computing resources
- Cost: Expensive to develop
→ The more secure, the less usable/functional

Symmetric vs Asymmetric Cryptosystems
Symmetric: Same key is used for encryption / decryption
Asymmetric: Separate encryption and decryption keys

Encryption / Cipher Requirements
Correctness, Security, Performance
- **Correctness**: For any plaintext x and key k, $D_k(E_k(x)) = x$
- **Security**: Given the ciphertext, it is (computationally) difficult to derive useful information of the key k and plaintext x. Ciphertext should resemble a sequence of random bytes.
- **Performance/Efficiency:** Encryption & decryption processes can be efficiently computed.
→ **Encryption is ONLY for confidentiality**, not for integrity

Cryptography
- Study of techniques in securing communication in the presence of adversaries who have access to communication
→ Encryption, hashing, digital signatures
Key space – Set of all possible keys
Key space size – total number of possible keys
Key size/key length – number of bits required to represent a particular key (log2(key space size) for lower bound)
→ For e.g., substitution cipher key space size = 27!, key size = $\log_2(27!) = 94$ bits.
Other possible representations include 1 byte per symbol = 27 bytes = 216 bits

Substitution Cipher
Key: Substitution table S, which is a 1-1 mapping from plaintext m to cipher text c
- Ciphertext-only attack: frequency analysis of letters in English alphabet
- Known plaintext attack – very vulnerable
Attacks on a cipher
Attacker's goal: Find the key, obtain info about plaintext, ciphertext only attack (large number of ciphertext encrypted with same key), known plaintext attack (cipher text and plaintext pairs)

Possible Methods
- Exhaustive Search (Brute force)
→ Whether it is feasible or not depends on key length. 56 bits is breakable. This is brute forcing on the KEY
→ If plaintext is known, might be easier to be broken. For sub cipher, attacker can map plaintext to ciphertext and get some parts of map. Less things to permutate after when brute forcing
→Sub cipher is **insecure/can be broken** under known plaintext attack
→ Some messages such as emails/network protocols, attacker can know some plaintext, like the header etc

Shift cipher
- Each letter in the plaintext is shifted a certain number of places
- Caesar cipher – key represents the amount of shift(e.g. key = 3 = shift everything by 3)

Vigenere Cipher
- Polyalphabetic cipher
- Uses keyword instead of a single shift distance to represent shift
→ SOC – "18", "14", "2" shifts every 3 characters
Different occurrences have different cipher letter
- Still insecure against known-plaintext attack
- Ciphertext-only attack - If we know the length/period of the keyword, then we can apply frequency analysis again
→Determine period of the keyword by using kasiski method – look for blocks with same letter in cipher text to narrow down period length
Permutation Cipher (Transposition Cipher)
- Group plaintext into blocks of t characters, apply secret permutation by shuffling chars
- Key is the permutation, written as a sequence p = (p1, p2, p3, …, pt). (2,3,1) means shift 1st to 2, 2nd to 3 and 3rd to 1, with t = 3
- block size t could be part of key as well
- Known-plaintext attack – fails miserably
- Ciphertext-only attack – easily broken if plaintext is English

One Time Pad
Encryption: plaintext ⊕ key → ciphertext
Decryption: ciphertext ⊕ key → plaintext
XOR is:
- 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: $A \oplus A = 0$

- Attacker can derive the key with known plaintext attack, but for OTP, key is only used once so its useless anyway
- Leaks 0 information about plaintext except for length. Basically achieves perfect secrecy, impossible for attacker to learn anything about plaintext x given c
→ Caveats: Key must be as long as the message, must be new key

Definitions/Properties of Cryptosystems
Algorithms (G, E, D) over sets (K, M, C):
K: set of all keys (key space)
M: set of all plaintext (plaintext/msg space)
C: set of all ciphertext(ciphertext space)

G: generates $k \in K$, key-generation algorithm
E (Enc): K, M⟶ C, encryption algorithm
D (Dec): K, C⟶ M, decryption algorithm
→ Same requirements as encryption reqs.

Perfect Secrecy:
for all $m \in M$ and all $c \in C$ (with $Pr[C=c] > 0$)
$Pr[M=m|C=c] = Pr[M=m]$
Informally: Regardless of any prior information that attackers (with unlimited computational power) has about the plaintext, the ciphertext should leak no additional information about the plaintext

Computational Secrecy:
- More relaxed than perfect secrecy: It's ok with the cipher leaks some information with tiny probability for attackers to crack ciphertext

- We quantify security by its most efficient search. E.g 2048 bit RSA takes roughly 2^{112} searches to crack, so key strength is 2^{112} bits

We usually analyse cipher w.r.t:
Threat Model - Assumption about attacker's capabilities, black-box/gray-box models
Security guarantee/goal: Intended attacker's in-capabilities, what we do not want attker to accomplish

Black-box Threat Models – Attackers can see what goes in and out of cipher, can be encryption query or decryption query
→ Ciphertext-only attack, known-plaintext attack, chosen-plaintext/ciphertext attack by performing encryption/decryption queries

Attackers Model	Observe Ciphertexts	Know the Corresponding Plaintexts	Can Perform/ Influence Encryption Queries	Can Perform/ Influence Decryption Queries
Ciphertext-Only Attackers	Yes	No	No (passive)	No (passive)
Known-Plaintext Attackers	Yes	Yes	No (passive)	No (passive)
Chosen-Plaintext Attackers	Yes	Yes	Yes	No
Chosen-Ciphertext Attackers	Yes	Yes	Yes	Yes

Gray-Box Models – Attacker has access to cipher's implementation
→ Side-channel attacks – Observing/measuring characteristics of cryptosystem not related to the implementation of the crypto (e.g. execution time, acoustic noise..)
Invasive attacks: Physical manipulation of hardware to perform attack

Attacker's in-capabilities
- Find key – **total break**
- Recover plaintext from ciphertext – partial break

Modern Ciphers
- Known-plaintext attack, frequency analysis, other known attacks covered for
- DES, RC4, A5/1 AES, A5/3
However, DES key length too short, RC4 broken in some adoptions, A5/1 is vulnerable, AES believed to be secure. RC4 is a stream cipher

Block Cipher Vs Stream Cipher
- For block cipher, m encrypted in blocks, for stream cipher, m encrypted bitwise

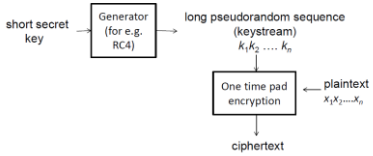
	Stream	Block
Advantages	• Speed of transformation • Low error propagation	• High diffusion • Immunity to insertion of symbol
Disadvantages	• Low diffusion • Susceptibility to malicious insertions and modifications	• Slowness of encryption • Padding • Error propagation

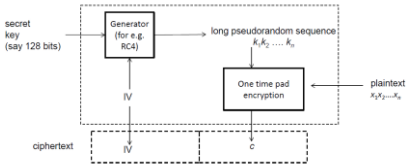
- For stream cipher, generator must be carefully designed to give cryptographically secure pseudorandom sequence, known as keystream
- stream cipher typically has IV (initial value, initialisation vector), chosen randomly. m using same key will output different c.
- Without an IV, attacker can even use xor 2 ciphertexts, c and c' to get some info of a message

Stream Cipher

- Stream cipher is inspired by one-time-pad: "pseudo/simulated OTP"
- 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***
- The pseudorandom sequence is sometimes also called ***keystream***

- Visually:





Role of IV Initial Value

- With different IVs, 2 pseudorandom sequences are different, ciphertexts are also different, making encryption non-deterministic / probabilistic

Pseudorandom Generator

- Security of stream cipher relies on used generator: known as PRG or PRNG

→ Maps seed space to output space deterministically and efficiently, output must look random, seed must be random

Block Cipher

→ have a block size, then we encrypt our plaintext by these blocks

→ Cannot be too small, <64 bits would not be secure (can perform codebook attack), where attacker compiles lookup table of all plaintext-ciphertext pairs observed under same key

→ n = block size, s = key size

→ DES: n = 64 bits, s = 56 bits

→ 3DES: n = 64 bits, s (up to) 168 bits, but effective security is lower, MITM attack

→ AES: n = 128 bits, s = 128,192,256 bits

→ Longer: more secure but slower performance

→ It is a permutation function: Maps 2^n plaintext to a 2^n ciphertext. So, to be a block cipher, we need a secure pseudorandom permutation function so that:

1. The permutation is determined by the key
2. Different keys must result into different permutations
3. The permutation looks random (indistinguishable)

→ Usually, block ciphers are not gigantic algorithms, but an iteration of rounds.

→ DES = 16 r, 3DES = 48 r, AES-128: 10 r

→ Two main techniques for each round: substitution-permutation network (AES) and Feistel scheme (DES)

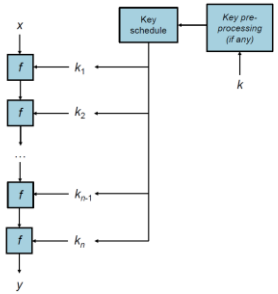
→ For a round, we have easy to specify/implement/analyze simple operations

→ A round function, which is basically the encryption part of the algo

→ Key pre-processing: Key expansion

→ Key schedule function: Produce a sequence of round keys (subkeys), so same round function with 2 different round keys behave differently

Encryptions in Block Ciphers: Rounds and Key Scheduling

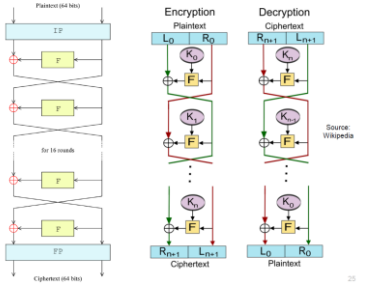


→ Output of 1 round is input for next round

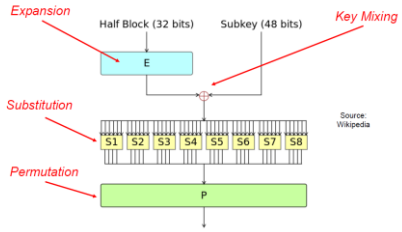
DES

- Block length: 64 bits
- Key length 56 bits (unsecure)
- 16 round using feistel function as round function, forming Feistel Network
- S-box (substitution): intro **confusion**
- P-box (permutation): intro **diffusion**

Feistel Network



→ Internally, we perform feistel function in DES in 16 rounds to get ciphertext. Don't need to know how its done



Exhaustive Search on DES

- Key length is 56 bits → can be brute forced

AES

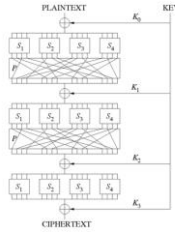
→ Block size: 128 bits

→ Key size: 128, 192, 256 bits (longer more secure but slower)

→ Using SPN (Substitution and Permutation Network), not Feistel

→ Substitution layer done using ByteSub operation, permutation using ShiftRow and MixColumn operations, not impt

Example of a SPN with Three Rounds



→ In each round for SPN, plaintext is split to do substitution operations, then do permutation operations, ciphertext will be used for next block

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

Stream vs Block Ciphers

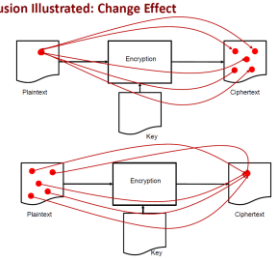
	Stream	Block
Advantages	• Speed of transformation • Low error propagation	• High diffusion • Immunity to insertion of symbol
Disadvantages	• Low diffusion • Susceptibility to malicious insertions and modifications	• Slowness of encryption • Padding • Error propagation

Diffusion: A change in plaintext affects many parts of cipher text.

→ This means info from plaintext is spread over entire ciphertext, transformations depends equally on all bits of the input

→ Cipher with good diffusion requires attacker to access much of ciphertext to infer encryption algorithm.

Diffusion Illustrated: Change Effect



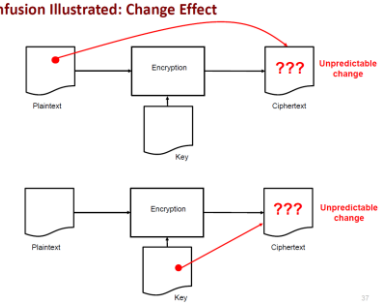
→ 1 bit of m affects many bits in c, 1 bit in c is affected by many bits in m

Confusion: Attacker should not be able to predict what will happen to ciphertext when one character in the plaintext or key changes

→ This means input (m and k) undergoes complex transformations during encryption

→ Cipher with good confusion has a complex functional relationship between m/k pair and c

Confusion Illustrated: Change Effect



Block Cipher – Modes of Operation

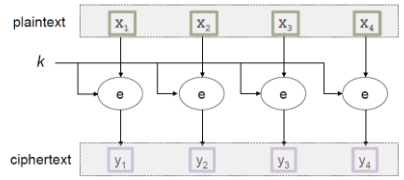
→ Encrypting a arbitrarily long message using block cipher, need to extend block cipher to m

→ Mode of operation: Method of encrypting messages of arbitrary size using block cipher

→ **There might be some security implications of simply extending encryption to multiple blocks!**

Electronic Code Block (ECB) (Insecure)

→ Divide m into blocks, apply block cipher to each block with same key, merge all c



→ There might be 2 identical segments of c, because they are encrypted with same key, so info abt m is leaked!

→ **Always take note in questions when we are passing same m and k (e.g. no IV). It will give the same c and that's dangerous**

→ In that sense, an encryption scheme is **deterministic**.

→ A probabilistic/randomized scheme produces different c even with same input given.

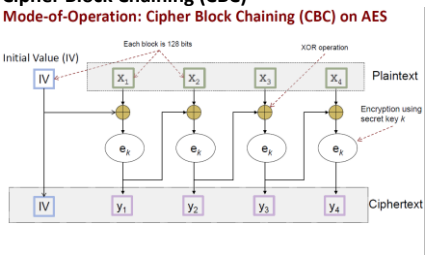
→ AES is deterministic, but we employ with randomly chosen IV, it is now probabilistic

→ We cannot simply choose a random IV for each block to achieve probabilistic encryption, because that will significantly increase the size of the final c!!

→ Use popular modes of operations: **CBC (Cipher Block Chaining), CTR (counter) modes**

Cipher Block Chaining (CBC)

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

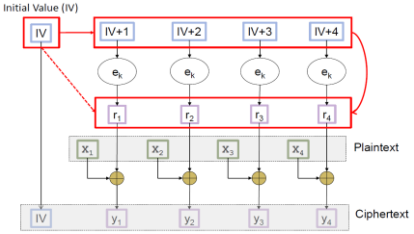


→ XOR first block of m with IV, encrypt to produce first block of c, y1. We use the intermediate value continuously (E(Xn XOR IV)) for the next round of encryption

CBC Decryption

- Refer to figure in padding oracle attack, we take cipher text, decrypt, then use IV XOR intermediate state to get plaintext

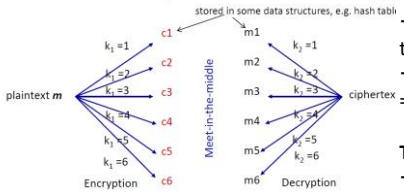
Counter (CTR) on AES



→ First define an IV. Then for each IV, we increment it by 1 for each increasing block. We run our AES round function with the key on the IV, get our intermediate random value, then xor with plaintext to get ciphertext. So basically, our IV increments in blocks, giving random intermediate values (r1, r2… rn), that acts like a 'keystream'
→ This is why we say CTR turns a block cipher into something like a stream cipher

Attacks on Block Ciphers
→ Meet-in-the-middle attack & double DES
→ Padding oracle attack (CBC AES)

Meet-in-the-Middle Attack & Double DES
→ Since DES is not secure, we encrypt using DES multiple times using different keys.
→ So, we do double DES. Now key length is 112bits (56 * 2)
→ However, key strength is not 112 bits because of MITM attack



→ known-plaintext attack, find 2 keys used
→ Exhaustively brute force all possible keys
→ We create 2 maps, C and M, C is map of key to ciphertext output of plaintext m, and M is map of the key to the intermediate plaintext input when we decrypt c with all keys.
→ Now we just need to match the common element in C and M, then we know the 2 keys used, so if they are 56 bit keys, then we just need $2 * 2^{56}$ to crack the 2 keys

Solution? 3DES - $E_{k3}(D_{k2}(E_{k1}(x)))$
Variants – 3TDEA (triple-length keys), with 3 independent keys, k1 k2 k3
- 2TDEA (double length keys), with 2 independent keys k1, and k2, and k3 = k1
- All 3 keys identical is another possible variant
Runtime – 3 times slower than DES

- **Encryption options**:
  (a) $E_{k_3}(E_{k_2}(E_{k_1}(x)))$ or
  (b) $E_{k_3}(D_{k_2}(E_{k_1}(x)))$
- Both options are believed to have the same level of security*

→ An advantage of using b is that if you put k1 = k2 then it's the same as normal DES (backward compatible)
→ 3 DES is still used today, but is less efficient than AES: Lower performance, can only encrypt 64-bit blocks at a time
→ Less secure than AES as well

Padding Oracle Attack
Encryption Oracle: Query containing plaintext m, oracle outputs ciphertext $E_k(x)$, where k is key
Decryption Oracle: Query containing ciphertext c, oracle outputs plaintext $D_k(c)$, where k is key
→ Attacker can send multiple queries

- In padding oracle attack, attacker has (IV, c), and wants to get plaintext of (IV, C)
- Padding Oracle knows k, attacker does not and does not need to. Padding Oracle will tell attacker if the padding of a ciphertext is in the correct padding format or not

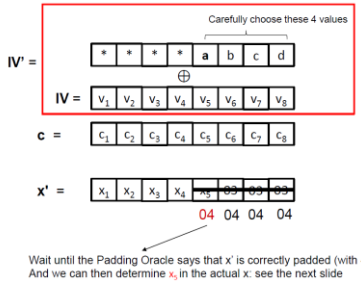
→ In a message with say 200 bits, we know that the second block might be padded with 56 bits. This padding follows a certain standard (e.g. PKCS #7)

- Suppose the block size is **8 bytes**, and the last block has only **5 bytes** (thus **3 extra padding bytes** required), the padding is done as follow:

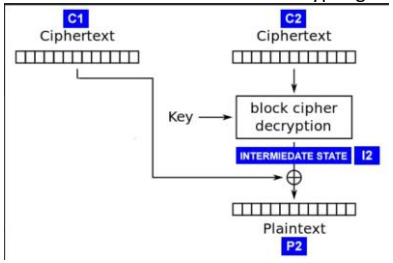
DD DD DD DD DD DD DD DD | DD DD DD DD DD 03 03 03

→ If last block is full, i.e. has 8 bytes (64 bits), then an extra block of all zeros are added
→ Take note this is for block size = 8, if block size = 16 then 13 byte string requires 3 padding bytes

The Attack
→ Attacker has (IV || C). We brute force the last byte of IV, from range 0 to 256 (ascii encoded hex). When the oracle tells us that we have a correct padding of in this case "01", then we know that last byte of the plaintext



From example above, we know that with (IV || c), to perform decryption, the IV will be XOR-ed with the intermediate value of decrypting c.



→ So, we pick a carefully chosen IV (or C1), that will be XOR-ed with I2 to give us the plaintext. Once we get the correct last byte of C1 which XORs to /x01, then we get the last byte of that ciphertext! When we try to find the next byte of IV, we use the value that we found, XOR with /x02 next, so that we know that byte will always have valid padding. Then continue byte by byte to get IV

→ Mainly, padding oracle works because the intermediate never changes, and if we keep c the same, the intermediate value is the same and we leverage on that

→ Padding oracles are frequently present in web applications, because we return explicit error messages like 500 error

Cryptographic Pitfalls
→ Reusing IV, wrong choice of IV & key in OTP
→ Predictable secret key generation
→ Designing your own cipher
→ Disregarding Kerckhoff's principle and relying on obscurity as a means of security
→ Using encryption for wrong purposes, like ensuring message integrity

Reusing IV/Wrong choices of IV & OTP Key
→ Some applications overlook IV generation (e.g. Using file name to derive IV)
→ Using AES under CBC mode, IV has to be unpredictable to prevent attack (vulnerable to choose IV as 1,2,3..)

Predictable Secret Key Generation
Getting random numbers to generate temporary secret key (e.g. Using java.util.Random vs java.security.SecureRandom, or #include <time.h> vs #include <stdlib.h> in C
→ java.util.Random gives a 48 bit random number, which is also generated using a mathematical algorithm that is hardly secure enough to use as a key.
Java.security.SecureRandom can have up to 128 bits, using random values as seed.
Java.util.random uses system clock as seed

Designing your own cipher
→ Never design your own cryptosystem, or even make slight modifications to existing scheme, unless you have indepth knowledge of the topic
→ You will fail

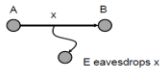
Kerckhoffs' Principle vs Security through Obscurity
- "A system should be secure even if everything about the system, except the secret key, is a public knowledge.(It can be stolen by the enemy without causing trouble.)"
- This principle is useful because:
1. Easier to keep secret key vs algo
2. Easier to change secret key vs algo
3. Standardized algo = easy deployment
4. Peer review & security validation: public will scrutinize on open algo
- versus security through obscurity, it can't be achieved as easily, and is just not a good way to keep your system secure

Supporting Obscurity
→ However, having more obscurity than less is still definitely helpful to keep a system secure
→ E.g. Usernames are not secrets but not exactly public
→ Location of firewall/firewall rules not secrets within organisation, but not public
→ Program used in a smart-card: not advisable to publish, because an adversary might be able to identify vulnerability that was previously unknown or perform side-channel attacks, or reverse-engineer code
→ Obscurity is simply one layer in a defense in depth strategy, can be used for some deterrence, but ineffective against experts/highly motivated individuals, which is why kerckoffs' principle is important!

Entity Authentication (Password)
Authentication: Assuring the communicating entity, or origin of a piece of information is one it claims to be
→ Entity Auth: For connection-oriented communication.
- Communicating entity is any entity involved in a connection.
- Mechanisms: Password, challenge/response, biometrics
→ Data-origin auth: For connectionless communications.
- Communicating entity is origin of a piece of info
- **Data-origin authenticity** implies **data integrity**
- Mechanisms: MAC or digital signature

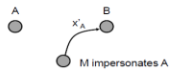
Illustration of threats to Confidentiality, Integrity, Authenticity
→ Not to be confused with CIA triad, where A = availability

- Confidentiality:



- Integrity:

- Authenticity:



Authenticity and Integrity
Authentic: Claimed entity/origin is assured by supporting evidence
Authenticity: Condition of being authentic
P("a message whose integrity is compromised")
→ Q("a message is not authentic")
Contraposition: (P → Q) <-> (~Q → ~P), i.e. A message is authentic = integrity is preserved
→ Data-origin authenticity implies data integrity, but **data integrity does not imply authenticity**
→ Authenticity is a stronger requirement than integrity

Examples of Problems Ensuring Authenticity
→ User logging in to luminus. Is she interacting with the authentic "LumiNUS", and is LumiNUS interacting with the authentic Alice?
→ Authenticity can also apply to other cases such as a user using a wifi access point, or receiving a phone call

Passwords (Weak Authentication)
- Stage 1: Bootstrapping
→ Server and user establishes common password
→ Server keeps track of file recording identity (username) and a corresponding password
- Stage 2: Authentication
→ Server authenticates an entity
→ If entity gives correct password corresponding to identity, entity is deemed authentic

- Identity does not need to be kept secret. E.g. Username in a computer system, bank account no, customer id, etc
- Password is a secret – Only authentic user and server know it. We base authentication on the fact that an entity knows password = they are either server or authentic user

Identification / Authentication / Authorization

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	-

- Identification: username
- Authentication: Password
- Authorization: Are you allowed to do this action? E.g. type of account, rights allowed

Stage 1: Bootstrapping
- Password to be established during bootstrapping, can be done by **either user chooses password, send to server (vice versa)** or **default password**
→ Basically, its just sign up

Stage 2: Password-based Authentication
- User logs in to server using normal login/logout
- User can also authenticate 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)

→ Password system classified as "weak authentication" system
→ Subjected to "replay attack": Information sniffed from communication channel can be used to impersonate user at later time
→ In contrast, with "strong authentication", info sniffed during process of authenticating cannot be used to impersonate user
→ Replay attack: attacker eavesdrops on secure network communication, intercepts it, and then fraudulently delays or resends it to misdirect the receiver into doing what the attacker wants.
→ Solution to replay attack: Use a **nonce**

Sniffing vs Spoofing
Sniffing: Eavesdropping, monitor data packets
Spoofing: Impersonation, introduce fake traffic

Attacks on Password System:
→ Attack bootstrapping
→ Searching for password: Guessing, dictionary attacks, exhaustive attacks
→ Steal password: Eavesdropping by sniffing network/using keylogger, spoofing login screen, phishing, password caching, insider attacks (the attacker is an insider of some organisation)

Bootstrap Attacks
→ Intercept password during bootstrapping
→ Use of default password. Shipping each access point with individual password is expensive
Attacks by Searching for Password
→ Guess password from social information: Gather social info about user, infer password (spouse name, hp number)
→ Types of guessing:
1. Online guessing: Attacker directly interacts with auth system
2. Offline guessing: Attacker steals password file from authentication system
3. Exhaustive search: try all combination. Probably not as feasible since key space size could be really huge
4. Dictionary attack: Restrict search space to a large collection of probably passwords: words from English dictionary, known compromised passwords etc

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

Char. Set Size	Character Types: Digits	Letters	Symbols	Other	Password Length: 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	$2*10^9$	$6*10^{18}$	$1*10^{28}$	$3*10^{37}$	$8*10^{46}$

Dictionary Attack + Exhaustive Search = Hybrid Attack:
- Try all combinations of 2 words from dictionary, exhaustively try all possible capitalizations of each word. E.g Substitute "a" by "@"

Attacks by Stealing Password
1. Shoulder surfing: look over shoulder attack
2. Sniffing: Listening/intercepting communication channel. FTP/Telnet/HTTP are protocols that send plaintext over public network unencrypted
→ **SFTP/SSH/HTTPS** are secure versions of them

3. Sniff wireless keyboard, other methods including using sound made by keyboard
4. Keylogger – capture keystrokes, by software or hardware
5. Login Spoofing: Attacker displays spoofed (fake) login screen
→ Prevention measures: Secure attention key or secure attention sequence (A special key or key combination that interacts directly with computer hardware, and kernel can know to start trusted login process. Makes spoofing impossible. Ctrl + alt+ dlt for windows)

6. Phishing: User tricked to voluntarily send password to attacker over network
7. Spear phishing: Phishing target to particular small group of users
→ Phishing is a type of social engineering attack, which refers to psychological manipulation of people to performing actions or divulging confidential information
→ Spear phishing is extremely effective
→ Phishing can be done through emails, phone calls (Phishing, Pharming, Vishing, Smishing)
→ Pharming: Use malicious program to redirect traffic from one site to another malicious one
→ User education is done to prevent phishing
→ Ask for help/clarification if not sure

8. Password Caching: When using shared workstation, information keyed in could be cached, next user can see cache. Must clear browser cache and close browser when using shared work station

9. Insider attack: Malicious sys admin who steals password file
→ An attack where sys admin's account is compromised (e.g. password stolen via phishing), leading to loss of password file is also considered insider attack

Preventive Measures
1. Use strong password
→ Randomly chosen, high entropy, but difficult to remember
→ Pbmbval! = pls be my valentine!
- **User selection**:
  - Mnemonic method: Pbmbval!
  - Altered passphrases: Dressed*2*tge*9z
  - Combining and altering word: B@nkC@mera

2. Password Protection
→ Limited Login attempts: Add delay into login session, add security questions, auto-lock account after a few failed attempts
→ Password Checker: Check for weak password when user registers/changes password
→ Password metering: Indicate weak/avg/strong pw
→ Password ageing: Users must regularly change password. However, some believe that frequent changes of passwords might lower security
→ Password usage policy: Rule set by organisation, such as password must be at least 10 chars

3. Protecting Password File
→ Password file which stores userid+pw can be leaked due to insider attack/accidental leakage/hacked system
→ Highly recoed to add additional layer of protection to pw file
→ **Hashed Passwords (impt!).** Passwords can be hashed, and during authentication, password entered is hashed, and compared with value in password file. **Note:** Hashed =/= encrypted. Passwords are hashed not encrypted. **Important:** Use a salt so that the same password will be hashed into 2 different values. This salt must also be saved

Security Questions
- Mechanism for fallback authentication or self service password reset
→ + Enhances usability: User can still login even if pw is lost
→ + Reduce cost: Reduce operating cost of helpdesk
→ - Weakens security: Attacks have another mean to obtain access
→ If secret questions are common, such as name your pet, aunt's middle name etc, its not really secret

Choices of Security Questions
1. Safe - Answers to security questions should not be something easily guessed/researched (e.g. a matter of public record)
2. Memorable: Users must be able to remember answers to their security questions else you achieved nothing
3. Nearly universal: Must apply to wide audience of people
4. Consistent: User's answer should not change over time. Shouldn't be something like "who's your significant other"

ATM Attacks
ATM Card: User presents a card and PIN for authentication
- Card contains magnetic stripe, which stores user account id
- Magnetic stripe simplifies input of account id into ATM system: Instead of keying it in, just insert the card
- PIN plays the role of password
- Data encoded into magnetic stripe using well-known standards, so attacker can easily read the info from the card and copy the card

1. ATM Skimmer
→ Steals victim's account id (username) and PIN (password). Skimmer consists of:
1. Card-reader attached on top of exiting ATM reader
2. Camera overlooking keypad, or spoofed key-pad above existing keypad
3. Some means to record and transmit info back to attacker
→ Attacker uses info obtained to spoof victim's ATM card or obtain the PIN
Preventive Measures for ATM Skimmer
→ Install anti-skimmer device, a device that prevents external card reader to be attached onto ATM, shield the keypad, user awareness, use newer chip-based (EMV) cards, which uses encryption

Biometrics
- Use unique physical characteristics of a person for authentication
- During enrolment, a reference template of a user's biometric data is constructing and stored (similar to bootstrapping in password system)
- Biometric sample data of person-in-question is captured and compared with template using a matching algorithm during **verification**
- Algo decides whether to accept/reject
- Useful for **verification**: 1:1 verification whether person is claimed person.
- **Identification**: 1:n comparison to identify person from a database of many persons

Biometrics vs Password

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

→ A password MUST match, but biometrics have a probability of error

Matching Algorithm: Similarity/Inexact Matching
- There are inevitable noises in capturing biometric data → leads to error in making matching decisions. FMR (false match rate) vs FNMR (False non-match rate)

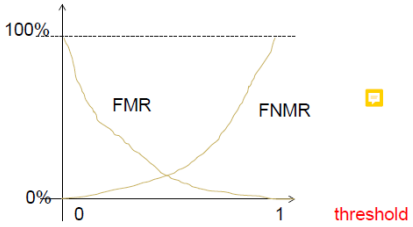
$$\text{FMR} = \frac{\text{number of successful false matches } (B)}{\text{number of attempted false matches } (B+D)}$$

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

Matching algorithm's output

Actual input	accepted/ match	rejected/ non-match
genuine attempt	*A*	*C*
false attempt	*B*	*D*

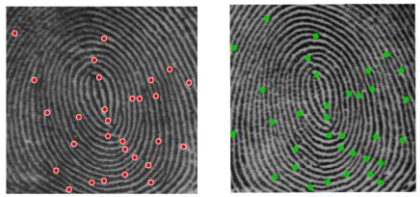
→ False match rate is much more serious, but depends on biometrics used for what context
→ C – False non match, B – False match
Threshold Value Selection
- Matching algorithm makes decision based on some adjustable threshold
- FMR and FNMR adjusted by adjusting threshold
→ Lower threshold → more relaxed in accepting
→ Higher threshold → more stringent in accept
→ The values are arbitrary, not really taught in lecture



→ Threshold set depend on application. Entering a database centre (high threshold) vs into a gym (low threshold)

→ If your threshold is high (close to 1) then any attempt will be very hard to match, i.e. the false non match rate is super high
→ If your threshold is low, you accept everything, so your false match rate is very high

Other types of errors:
- Equal error rate (EER): rate when FNMR = FMR
- Failure-to-enroll rate (FER): Some users' biometric data can't be captured for enrolment due to past injury
- Failure-to-capture rate (FTC): User's biometric data may fail to be captured during authentication (Dry/dirty fingers etc)

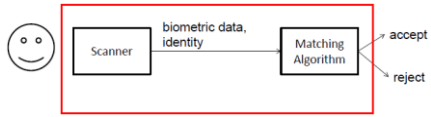


The set of feature points
(known as *minutiae* for fingerprint)

- Finding set of feature points for authentication
- Base off your threshold to see if you want to accept, since the scan may have similar results but NOT the exact same
→ Other forms of biometrics: Palm print, palm veins, face, iris, retina, dna

Security of Biometric System

- It is assumed that no tampering is possible, although some biometric data could be spoofed
- Liveness detection – verify entity scanned by scanner is indeed live instead of spoofed from a photograph



n-Factor Authentication (2FA)

- At least 2 factors

- Commonly-used factors:
  1. What you **know**: password, PIN
  2. What you **have**: smart card, ATM card, mobile phone, mobile SIM card, security/OTP token
  3. Who you **are**: biometrics
- Other possible factors [Gollmann]:
  1. Where you **are**: geolocation
  2. What you do
- **Recently-used factors**:
  1. Who (some trusted system) can confirm you: PKI certificate, single sign-on (SSO)
  2. Whom you know: social authentication (in social media)

What you have: OTP Token
- Hardware that generates OTP, each token and server share some secret

2 types:
1. Time based – based on shared secret and current time interval, password K is generated
- Server and user has a common password K
2. Sequence-based: An event (user pressing some button) triggers change of password

OTP Registration (Password + OTP):
- Server issues OTP token to user, which contains some "secret key" server knows
- User sets password

OTP Authentication (Password + OTP):
- User "presses" token, which computes and displays OTP
- User sends username, password, OTP to server
- Since server has secret key, server can also compute OTP
- Server verifies both OTP and password are correct

OTP Registration (Password + SMS):
- User gives server his mobile phone number and password

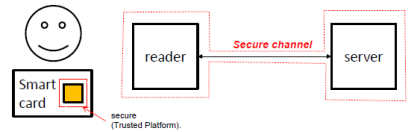
OTP Authentication (Password + SMS):
- User sends password and username to server
- Server verifies password, sends OTP to user via SMS
- User recvs SMS, enters, OTP
- server verifies that OTP is correct
→ Singpass, Internet Banking

SMS OTP Security
- It is not secure due to various threats:
→ Interception of cellular networks' channel, SMS stored as plaintext by Short Message Service Centre (SMSC), Malware/trojan on smartphones.
→ Expert opinion is that it is still better than just userid + password

OTP Registration (Smartcard + Fingerprint):
- Server issues smartcard to user (which contains a secret key K), user enrolls his/her fingerprint

OTP Authentication (Smartcard + Fingerprint):
- User inserts smartcard to reader, reader obtains user identity, verifies whether smartcard is authentic. If so, continue
- User presents fingerprint to reader. Reader performs matching to verify authenticity. If yes, open door

→ Usually, info on user identity, secret K, fingerprint template not stored in reader. Reader has secure communication channel to server that stores these info. We assume the reader and server are secure and attackers cannot access them

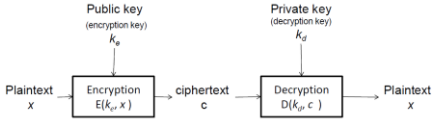


Security Requirements

Some additional 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?

Public Key Cryptography (PKC)

→ Uses 2 different keys for encryption and decryption



→ Decryption key often consists of 2 parts <k1, k2>, where k1 is part of the encryption/public key, and k2 is kept secret

Why is it called Public Key?
→ Alice tells everyone her public key
→ Another person has plaintext m for Alice, p encrypts it using Alice's public key, sends ciphertext to Alice, Alice decrypt with secret private key

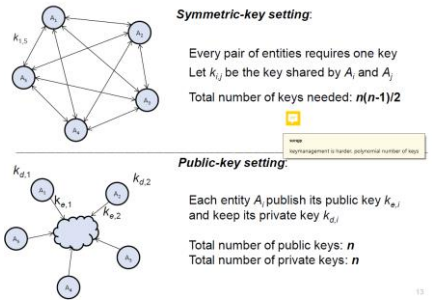
Public-Key Encryption Scheme Definition

Can be more formally defined as **algorithms** (**G**, **E**, **D**) over **sets** (***K***, ***M***, ***C***):
- $K = K_e \times K_d$: set of all keys (= key space)
- ***M***: set of all plaintexts (= plaintext/message space)
- ***C***: set of all ciphertexts (= ciphertext space)
- G (key-generation algorithm): generates a ***key pair*** (k_e, k_d), where k_e is **publicly known** whereas k_d is **kept secret**
- E (encryption algorithm): $K_e \times M \rightarrow C$
- D (decryption algorithm): $K_d \times C \rightarrow M$

Public Key Encryption Scheme Requirements
- **Requirements:**
  - **Correctness**: For all $m \in M$ and all $(k_e, k_d) \in K_e \times K_d$ outputted by G: $D(k_d, E(k_e, m)) = m$
  - **Efficiency**: G, E, and D (with the knowledge of k_d) are fast, i.e. they run in polynomial time
  - ***Trapdoor function*** **E()**:
    - E() can be **efficiently evaluated** using the publicly-known k_e
    - But its inverse D() is **computationally infeasible** *without* the corresponding private key k_d (as the ***trapdoor information***)
    - That is, without the trapdoor k_d, there's an **asymmetric computational requirement** between E() and its inverse D()
  - **Security**: For all k_e and all $c \in C$, the plaintext m corresponding to c **can be recovered only** with the k_d corresponding to k_e
- Public-key encryption scheme as a secure ***trapdoor permutation***

→ **Trapdoor function**
→ Plaintext can be recovered only with the availability of the private key kd
Security Requirement: Given public key and ciphertext, but not private key, it is difficult to determine the plaintext
→ Or, without knowledge of private key, ciphertext resembles sequence of random values
→ It must be difficult to get private key from public key

Advatanges of Public Key Scheme
→ If we use SKC, any 2 entities must share a secret key. Many keys are required especially if no. of entities is large.
→ Hard to establish all the secret keys



Integer Representations for PKC systems

- Many PKC systems represent the data (plaintext, ciphertext, key) as **integers**, and the algorithms are **arithmetic operations** on integers
- The integers can be represented using their **binary representations**
- When we say that a **key is 1024 bits**, we mean that it can be represented using 1024 bits **under binary representation**:
  e.g. a 3-bit integer is a value from 0 to 7
- Note that the total number of 1024-bit integers is 2^{1024}:
  an algorithm that exhaustively searches 1024-bit numbers is **infeasible**

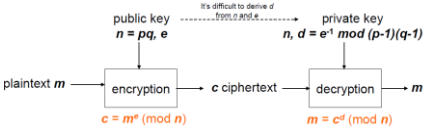
RSA - Setup

$\Phi(n) = (p-1)(q-1)$ – Euler's totient function

1. The owner randomly chooses 2 large primes ***p***, ***q*** and computes $n = pq$ as the public ***composite modulus*** (Note that the values of p and q must **not** be revealed to the public)
2. The owner randomly chooses an ***encryption exponent*** ***e*** s.t. $\gcd(e, \Phi(n)) = 1$, i.e. $e < \Phi(n)$ and e is relatively prime to $\Phi(n)$
   (The term $\Phi(n) = (p-1)(q-1)$ is the **Euler's totient function**, which is the number of integers < n that are relatively prime to n.)
3. The owner determines the ***decryption exponent*** ***d***, where: ***d*** **e** ≡ 1 (mod $\Phi(n)$), i.e. $d = e^{-1} \pmod{\Phi(n)}$
   (There is an **efficient algorithm** to find d when given e, p and q, but we won't get into the details see: http://shell.cas.usf.edu/~wclark/elem_num_th_book.pdf.
   After determining d, the owner doesn't need to keep p and q.)
4. The owner publishes (***n***, ***e***) as her public key, and keep (***n***, ***d***) as her private key

Question: Why can't an attacker derive d from the public key (n, e)?
Answer: The (prime) "***factorization problem***": the difficulty of factoring the product of two large primes

→ Can find d when given e, p, q → can attack
→ (n, e) as public key, (n, d) as private key
→ Encryption: Given a message m, the ciphertext c is: c= $m^e \pmod n$
→ Decryption: Given a ciphertext c, the plaintext mis: m = $c^d \pmod n$



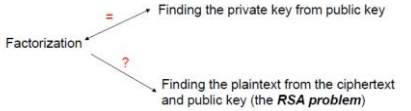
Correctness of RSA
- Note that for any positive $m < n$, and any pair of e and d:
  Decrypt (d, Encrypt (e, m)) = m
- That is:
  $(m^e)^d = m \pmod n$

RSA Example
- $p = 5$, $q = 11$, $n = 5 \times 11 = 55$
- $\Phi(n) = (p-1)(q-1) = 40$
- Suppose $e = 3$, which is co-prime with 40
- Then $d = 27$
  [you can check that $3 \times 27 = 81 = 1 \pmod{40}$]
  There is an efficient algorithm to compute d from p, q, e (details are omitted)
- Suppose $m = 9$
- The encryption derives:
  $c = m^e \pmod n$
  $= 9^3 = 14 \pmod{55}$
- The decryption derives:
  $m = c^d \pmod n$
  $= 14^{27} = 14^{16+8+2+1} = 9 \pmod{55}$
  There is an efficient algorithm that computes modular exponentiation (details are omitted)

→ Since we are dealing with mods here, e and d can be anything that fulfils the criteria (I think)
→ **For RSA, d can be used to encrypt** and **e and be used to decrypt** as well.

RSA Efficiency
→ Finding a random prime (for p and q), randomly pick number, primality testing can be done efficiently. **You must find "strong" primes**
→ Extended Euclidean algorithm – efficiently compute d from e, p, q
→ Performing decryption and encryption operations – we need to perform modular exponentiation operations, which can be efficiently done
→ In practice, **either one of e or d is intentionally chosen to be small (both cant be small!!!!)** so that encryption or decryption (either one) is fast
- How to easily perform the **encryption** and **decryption** operations?
  - We need to perform **modular exponentiation** operations: $m^e \pmod n$, $c^d \pmod n$
  - There is an **efficient algorithm** to compute a modular exponentiation

Security of RSA (RSA problem)
→ The problem of **getting the RSA private key from public key** is as difficult as the problem of factorizing n
→ However, it is not known whether the problem of getting the m from c and public key is as difficult as factorization (i.e. it could be easier)



→ **Homomorphic Property (RSA vulnerability)**
E(m1 * m2) can be directly evaluated by using E(m1) and E(m2) without decrypting m1 * m2
→ Useful in applications such as blind signature, encrypted domain processing
→ But can lead to attacks and information leakage

Post-Quantum Cryptography
→ Quantum computer can factorize and perform "discrete log" in polynomial time
→ RSA and "discrete log" based PKC will be broken with a quantum computer in the future
Other types of cryptography:

- A quantum computer can factorize and perform "*discrete log*" in **polynomial time**:
  - In 2001, a 7-*qubit* quantum computer was built to factor 15, carried out by IBM using NMR (See: http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20011219_quantum.html)
- Hence, both RSA and "discrete log" based PKC will be **broken** with a quantum computer:
  - See Shor's algorithm (https://en.wikipedia.org/wiki/Shor%27s_algorithm)
- ***Post-Quantum Cryptography***:
  PKC schemes that are secure against quantum computer
- ***Lattice-based cryptography***:
  - Based on this hard problem: given the "basis" of lattice, it is computational hard to find a short lattice point
  - No good candidate yet
- ***Multivariate cryptography***:
  - Given a multivariate polynomial, it is difficult to find the solution (under modulo *p*)
  - No secure construction yet

RSA Padding
→ IV is used in RSA so that encryption is randomized (actual mechanism is optional)
→ Because of homomorphic property, "optimal padding" is added to RSA to destroy homomorphic property

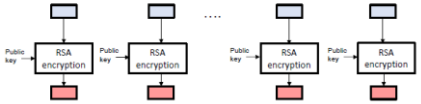
Improper RSA Usage
→ Using RSA as symmetric key
→ Consider follow scenario:

Consider a sample scenario:

"For his Final Year Project, Bob is tasked with writing 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

→ To encrypt a large image, Bob divides file into chunks of 128 bytes, applying RSA to each chunk



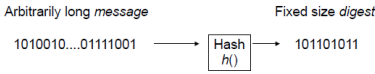
Issue 1: RSA Efficiency/Performance
→ RSA is significantly slower than AES – 128-bit AES ≈ 3072-bit RSA

Solution: Using "Hybrid Encryption"
→ When large file is to be encrypted under public key setting, we do this instead:
1. Randomly choose AES key k
2. Encrypt F using AES with k as key, producing C
3. Encrypt k using RSA to produce y (encrypted AES key)
4. Final cipher text is (y, C)
→ Decrypt? Use private key to turn y into k, then decrypt C using k

Issue 2: ECB with RSA
→ If we divide plaintext into blocks, encrypting with same key, since the output is deterministic, any 2 plaintext blocks that are the same will be encrypted into the same ciphertext

Issue 3: RSA Security
→ Not necessarily more secure than AES. We don't know about the RSA problem (search notes). No rigorous proof that RSA is "secure" in protecting ciphertext. This is similar to AES
→ Must modify "classroom" RSA for different encryptions of same plaintext to produce different ciphertexts (padding, IV)
→ Factorization can be done efficiently by quantum computer
PKC Strengths
- Main strength of RSA and PKC is public-key setting, which allows an entity in the public to perform an encryption without a pre-established pair-wise secret key
- Also useful for providing authentication (digital signatures)
- Rarely used to encrypt large data file

Unkeyed Hash and Keyed-Hash
→ We will be following Kerckhoff's principle
- A(cryptographic) hash is a function that takes an arbitrarily long message as input, and outputs a fixed-size (say 160 bits) digest



→ No key/secret
→ Different from non-cryptographic hash functions (CRC checksum, taking selected bits from data)

Cryptographic Hash: formal definition
- A hash function takes a message *m* and produces ***n*-bit digest**: $H: \{0,1\}^* \rightarrow \{0,1\}^n$

→ * means arbitrary length, n is length n

Cryptographic Hash Requirements
- Requirements:
  - **Efficiency**: Given *m*, it is computationally efficient to compute $y = h(m)$
  - **Pre-image resistance ("one-way")**: Given *y*, it is computationally infeasible to find a *m* such that $h(m) = y$
  - **Collision-resistance**: It is computationally infeasible to find *any* pair of messages (m_1, m_2) where $m_1 \neq m_2$, such that $h(m_1) = h(m_2)$, i.e. both messages have the same digest
- Collision resistance → pre-image resistance:
  - ¬ pre-image resistance → ¬ collision resistance
  - This also means: If we have an algorithm to attack the pre-image problem, then we can produce collisions
  - See Tutorial 4

→ Note the stronger requirement of collision resistance than preimage resistance
→ Some examples: SHA1 with 160-bit digest
→ Hashing 2 sentences with just 1 character difference causes hash to completely change (diffusion?)
→ MD5 (Popular but obsolete), do not use
→ SHA-0, SHA-1, SHA-2, SHA-3
→ SHA-1 is popular standard
C

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.

Base 64 encoding
→ Binary-to-text encoding to represent binary data in ASCII string
→ 6 bits because that's 64 numbers, 3 hex numbers (12 bits) = 2 base 64 characters (count the bits)

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

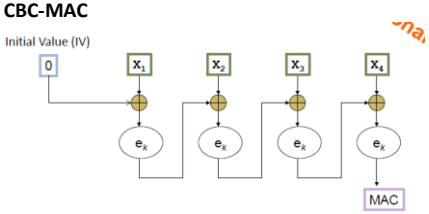
Source	Text (ASCII)	M	a	n
	Octets	77 (0x4d)	97 (0x61)	110 (0x6e)
	Bits	0 1 0 0 1 1 0 1	0 1 1 0 0 0 0 1	0 1 1 0 1 1 1 0
Base64 encoded	Sextets	19	22	5 46
	Character	T	W	F u
	Octets	84 (0x54)	87 (0x57)	70 (0x46) 117 (0x75)

Source	Text (ASCII)	M	a	
	Octets	77 (0x4d)	97 (0x61)	
	Bits	0 1 0 0 1 1 0 1	0 1 1 0 0 0 0 1	0 0
Base64 encoded	Sextets	19	22	4 Padding
	Character	T	W	E =
	Octets	84 (0x54)	87 (0x57)	69 (0x45) 61 (0x3D)

Keyed Hash (HMAC)
cb – Message Authentication Code (a.k.a keyed hash, with a secret key involved)
→ Function that takes an arbitrarily long message and a secret key as input, and outputs a fixed size (e.g. 160 bits) MAC

→ Correct MAC can only be generated by someone who knows the key
→ CBC-MAC (based on AES operated under CBC mode), HMAC (Based on iterative cryptographic hash function (SHA1/SHA2), or hash-based mac)

CBC-MAC



HMAC

$HMAC_k(x) = \text{SHA-1}((K \oplus opad) \,||\, \text{SHA-1}((K \oplus ipad) \,||\, x))$
where:
opad = 3636...36 (outer pad)
ipad = 5c5c...5c (inner pad)

(***Note***: the values above are in hexadecimal)

MAC: formal definition / correctness req
- A MAC is defined by (**G**, **Mac**, **Verify**):
  - **G**: Outputs a key *k*
  - **Mac**: Takes as input key *k* and message $m \in \{0,1\}^*$; and outputs tag $t = \text{Mac}(k, m)$
  - **Verify**: Takes key *k*, message *m*, and tag *t* as input; outputs 1 ("true"/"accept") if $\text{Mac}(k, m) = t$, or 0 ("false"/"reject") otherwise
- **Correctness** requirement: for all *k* outputted by G and all *m*, $\text{Verify}(k, m, \text{Mac}(k, m)) = 1$

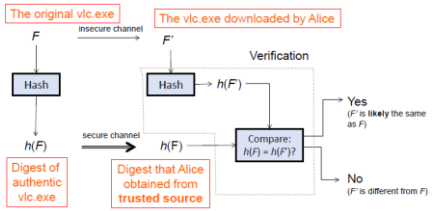
MAC: Security requirement
- **Security** requirement:
  - **Threat model**: adaptive chosen message attack
  - **Security goal**: "***existential forgery***", i.e. attacker cannot forge a valid tag on *any* message not authenticated by the sender
  - That is, after seeing multiple valid pairs of messages and their corresponding MACs, it is difficult for the attacker to forge the MAC of an *unseen* message (The precise formulation is quite involved, and with many variants. A higher level crypto module CS4236 would cover this.)
- How about *replay attacks*? Are they prevented by MAC?

→ Does not protect against replay attack
The problem is that MACs do not incorporate any notion of state in their verification algorithms. Thus, every time a valid pair(m,t) is presented to Verify(k) it returns the same answer

Data Integrity: Hash (Without secret keys)
→ How do we ensure that a piece of ciphertext received is authentic?

Using unkeyed hash for Integrity Protection
→ Assuming there is a secure channel to send a short piece of info:
- 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

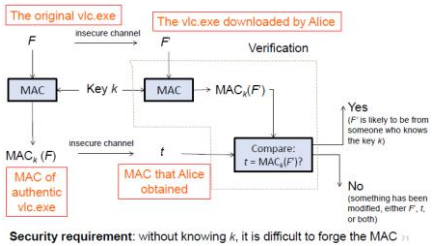
→ In short, Alice receives a file, and the hash of F, she just compares if h(file) = hash of file, if true then can say F = F' with high confidence else integrity compromised



Attacks against unkeyed hash for integrity protection
→ Attack would try and find second pre-image, another F' that is not F such that h(F') = h(F)
→ **Note:** The digest posted on webpage in the first place must be through secure channel like HTTPS. Else it both hash and file could have been modified (no integrity)
→ Provide (data-origin) authenticity by using **MAC or digital signatures**
→ Without secret keys, we often still refer to the problem as an "integrity" issue

Data-Origin Authenticity
→ MAC and Signature
- If a server hosting webpage is compromising, despite hosting file with message digest on website, the digest could have been forged, or in the first place, the website was spoofed
→ If we don't have secure channel, we can protect digest with MAC (symmetric key setting) or digital signatures (public-key setting)

Data Origin Authenticity: MAC (Message Authentication Code)
→ MAC might also be modified by attacker, **but it can be detected with high probability.** This is assuming the file itself was not modified



→Attacker will try to forge valid file pair (file, MAC)

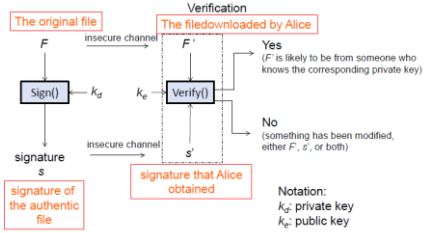
- **Other remarks:** Take note that in this current problem, confidentiality is not a requirement, so we can send file F in clear. Typically, MAC is appended to F, stored as single file and transmitted together. MAC is also called the authentication tag in this case

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

Data Origin Authenticity: Digital Signature

→ Essentially asymmetric version of MAC

Digital Signature Requirement: without knowing decryption key kd, it is difficult to forget s, the signature



Digital Signature formal definition

Digital Signature correctness:

- **Correctness requirement:**
  For all $(k_e, k_d) \in \boldsymbol{K_e} \times \boldsymbol{K_d}$ outputted by G and all $m \in \boldsymbol{M}$, Verify(k_e, m, Sign(k_d, m)) = 1

→ (m, e) to decrypt signature, (d, m) to sign

→ Kd, is also private key (double function)

→ Signature is computed using private key & m

→ Computed signature is typically appended to F and stored as single file or transmitted tgt

- 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

→ Signatures are based on the premise that only the sender (Alice) has the private key, so the message must be authentic

Digital Signature security requirement

- **Security requirement**:
  - Security goal: *"existential forgery"*, i.e. attacker **cannot** forge a valid signature on *any* message not signed by the sender
  - In other words, for all k_e and all $m \in \boldsymbol{M}$, the valid signature *s* of *m* can be generated only with k_d corresponding to k_e used in the verification
- The security requirement implies that:
  although Verify() can be efficiently evaluated using a public key k_e, it is difficult to evaluate Sign() *without* the corresponding k_d

Digital signature vs MAC

→ Digital signature is like a handwritten signature in a legal document, which achieves non-repudiation: assurance that someone cannot deny his/her previous commitments or actions

→ Data-origin authenticity != signature! Signature provides non-repudiation, a stronger requirement (I think, in a sense?)

Non-Repudiation (MAC vs digital signature)

→ MAC does not achieve non-repudiation

→ MAC uses a symmetric, shared key by Alice and Bob. If someone accuses alice of sending message, she can claim Bob generated the message

→ Digital Signatures achieve non-repudiation

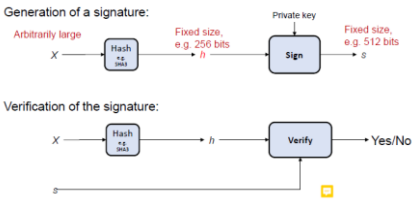
→ Only she has her own private key, which is proof she is the sender

Design of signature scheme

→ Most have 2 components: Unkeyed hash, sign/verify algorithms

→ The hash is to hash the original message, which is then signed with signature scheme.

→ We can verify by computing the hash of the message sent over, and verify the decrypted signature (e.g. decrypt with public key) is equal

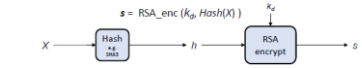


RSA-Based signatures

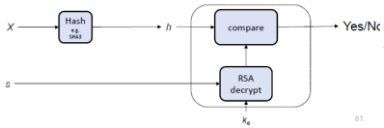
→ signature is the encrypted version of the message digest (which is the message passed through a hash)

- When RSA is used for signing and verification:
  - The **signature** is essentially the **"encrypted" digest** done using the private key
  - During verification, decrypt using the public key to obtain the digest and compare
  - *A messy notation issue: we previously used the public key to encrypt, but here we use the private key to encrypt; recall that for RSA, we can flip the role of both keys*

$s = \text{RSA\_enc}(k_d, Hash(X))$



If $Hash(X) = \text{RSA\_dec}(k_e, s)$ then accept, else reject



→ Take note of the notation, kd is used to 'encrypt' to give a signature, usually we use kd to decrypt

→ sign() – encryption using private key

→ Verify – decryption using public key

Attacks/Pitfalls of Hashes

1. Birthday Attack on Hash

→ Remember that hashes are designed to be collision resistant

→ However, they are subjected to birthday attack

Illustration of Birthday Problem

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 = 23×22/2 = 253 comparisons

→ **TODO: COPY FORMULAS FROM TUTORIALS**

Birthday Attack on Hash Functions

→ Supposed we have M messages and each message is tagged with value randomly chosen from {1,2,3,…,T}

→ If $M > 1.17T^{0.5}$, then with probability > 0.5, there is a pair of messages tagged with same value

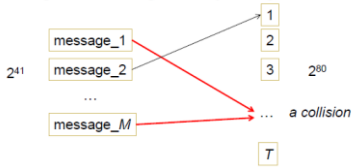
→ Supposed the digest of a hash is 80 bits: T= 2^{80}

→ If attacker randomly generates 2^{41} messages, ($M=2^{41}=2^{1+80*0.5}$), then $M > 1.17\ T^{0.5}$

→ If $M > 1.17\ T^{0.5}$, with probability > 0.5, two of them give same digest

→ In general, probability that a collision occurs ≈ 1 - exp ($-M^2/(2T)$)

among the 2^{41} messages, ***two of them*** *give the same digest*!



Birthday Attack implications on digest length

→ In previous example, the work factor in breaking a cipher is no longer 2^{80} but 2^{41}. When the key length for a symmetric key is 112, the corresponding recommended digest length is at least 224.

Birthday Attack Variant

- Birthday Attack: How many messages must an attacker generate for a probability of > 0.5 that 2 of them have the same digest, the attack variant is calculating the probability that one of the chosen strings is in the bigger set S
- Birthday Attack Variant: With a set of k distinct elements in S(each element is a n-bit binary string), randomly select m n-bit binary strings into a set M, (M smaller than S, |M| < |S|)

- It can be shown that, the probability that at least one of the randomly chosen strings is in *S* is (larger than):

$$1 - 2.7^{-km2^{-n}}$$

- Notice that the set *S* and the set *M* are *different*

Using encryption for authenticity

Encryption Misuse:

→ Encryption scheme may provide false sense of security

→ Suppose we have the following example, where a mobile phone and server share some secret 256-bit secret key k, server sends instructions to mobile phone via SMS (insecure channel)

→ Supposed we have message format of below:

- Suppose **the format** of the instruction is:
  X P
  where: *X* is a **8-bit** string specifying the type of operation, and *P* is a **248-bit** string specifying the parameter.
- If an operation doesn't take in parameter, *P* will be ignored

→ Suppose we have 16 valid instructions, where one of them is brick the phone

- 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.

→ Attacker can easily send a randomly chosen message. If the first 8 bits match, the phone will self-destruct. **Encryption provides confidentiality, NOT authenticity!!**

→ What is the probability of the first 8 bits matching? I think its $2^{248} / 2^{256} = 1 / 256$

→ Encryption provides **confidentiality. Not authenticity.** Use a secure design like MAC/digital signatures for authenticity

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

→ Some argue that encryption is "sufficient" to protect a message, as attacker doesn't know plaintext and obtains only ciphertext, and any change in c will result in a change in m.

→ However, there are still many things an attacker can do to an encrypted message, mainly regarding modifying the ciphertext

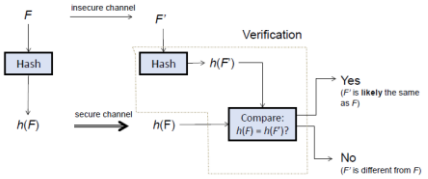
- ***Ciphertext blocks re-ordering*** say on CBC: *See Tutorial 4*
- Integrity attack on **One-Time-Pad** (*recall Tutorial 2*): what happen if **1 bit** of the ciphertext gets flipped?
  → Modification has a predictable impact on the plaintext!
  → Modification to the ciphertext can be **undetected**

→ "Non-malleability" not met: A cipher is malleable if it is possible to modify a ciphertext and cause predictable change to plaintext

Hash Digital Signatures Summary

Digest (Hash Value)

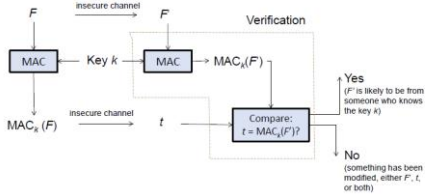
The digest must be sent over a **secure channel**



Security requirement: It is difficult to find a *F* that gives the same digest *h(F)*

MAC (Message Authentication Code)

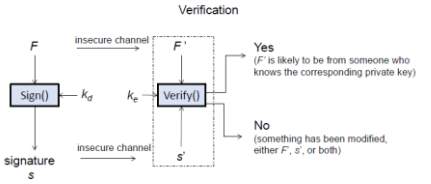
The MAC can be sent over an **insecure channel**



Security requirement: Without knowing *k*, it is difficult to forge the MAC

Digital Signature

The signature can be sent over an **insecure channel**
The signer and verifier **use two different keys**: k_d and k_e



Security requirement: Without knowing k_d, it is difficult to forge the signature s

PKI + Channel Security

Motivation: Any Mallory can impersonate a Bob and when Alice requests a public key from Bob, Mallory sends her public key instead
→ Need a secure channel to securely distribute/broadcast public keys. We can use public keys for **encryption** (confidentiality) and signature verification (authenticity)

Secure channel req for Public-key Setting

→ Public key setting doesn't require a secure channel to send a secret key from alice to bob, just a public key, which is much 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)

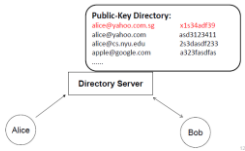
→ 3 methods: public announcement, publicly available directory, public key infrastructure (PKI)
→ PGP – pretty good public key

Public Announcement

→ Owner broadcasts his/her public key, publishing on website, blog, send to friends via email etc
Limitations:
- Not very standardized, no systematic way to verify public key
- (impt) Still need to trust the 'entity' distributing the public key. i.e. need to trust the website, email website

Publicly-Available Directory

→ Bob searches public directory by querying a server to find the public key associated with some email (e.g. alice@abc.com)



Potential Issues:

- Anyone can post his/her public key into server
- Not easy to have a secure public directory. When a server receives request to post a public key, it cannot really verify that the info is correct
- Eventually we still need to trust some entity, like the website
- Even if the user trusts the website, the website might be spoofed, i.e. user does not know website visited is indeed as what is claimed to be

PKI (Public Key Infrastructure + Certificate)

- Standardized system that distributes pub keys
- Objectives:
  - 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

→ Centred around 2 components/notions: **Certificate** and **Certification Authority (CA)**
- PKI provide mechanism for trust to be extended in a distributed manner, starting from the "root" CA

Certification Authority (CA)

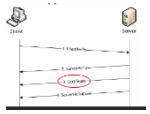
→ Issues and signs digital certificates
→ Keeps a directory of public keys
→ Has its own public-private key pair: We assume that it has been securely distributed to all entities involved
→ Most OSes and browsers have a few pre-loaded CAs' public keys: Knows as "root" CAs
→ There are stringent operational requirements for a CA. For eg, it must pass WebTrust audit

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, …)

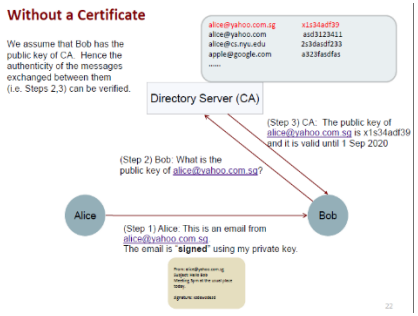
→ Widely used by internet applications: SSL/TLS, S/MIME, SSH
→ Used in SSL/TLS handshake protocol



Role of Certificate

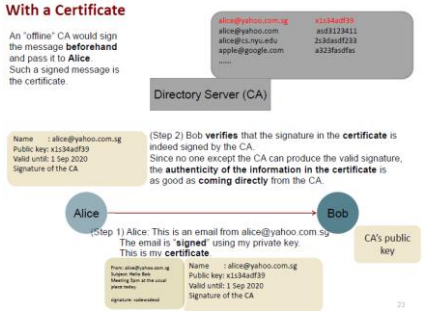
→ Can a certificate-based PKI work without a publicly-available directory server? Yes!
→ Supposed we treat CA as directory server
→ If we always have to retrieve public key from directory server, Bob needs to have online access to CA at verification point and CA becomes bottleneck

Without a Certificate



→ With a certificate, an "offline" CA signs a certificate beforehand and pass to Alice.
When alice sends message to bob, she sends signed email (signed with her private key) along with signed certificate (cert shows what's alice's public key, and signature of CA).
→ Proves ownership of a public key

With a Certificate



→ Bob already has CA's public key, he can verify the CA is valid, and he knows that the public key belonging to Alice issued in the cert is legit.
→ All these can be done without querying the server repeatedly

Role of Certificate: No required directory server

- CA binds an entity with her/her public key prior to verification point
- With cert, Bob can obtain Alice's public key and verify authenticity without connecting to a CA
→ We still need to check that the certificate has not been revoked: Online CRL Dist. Point(s) / OCSP Responder

X.509 Digital Cert Standard

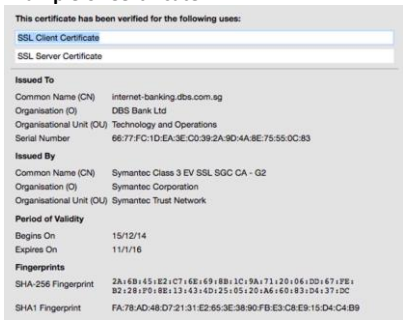
- Standardization **bodies**:
  - ITU-T X.509:
    Specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm
  - The Public-Key Infrastructure (X.509) Working Group (PKIX):
    IETF working group that creates Internet standards on issues related PKI based on X.509 certificates

Structure of an X.509 v3 digital cert

→ Serial Number uniquely identifies certificate
- Certificate:
  - Version Number
  - Serial Number
  - Signature Algorithm ID *(Note Signature Algorithm below too)*
  - Issuer Name
  - Validity period: Not Before, Not After
  - 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 entity (e.g. issuer and subject names of a cert):
- Common attribute types:
  Country (C), State (S), Locale (L), Organization name (O), Organizational unit name (OU), ***Common name*** (CN)
- ***Common name*** (CN):
  can be an individual user or any other entity, e.g. a web server

Example of Certificate:



Certificate Authority & Trust Relationship

- CA is responsible to verify information is correct, besides issuing certificate
- E.g. CA should check if applicant indeed owns domain name before issuing a certificate for nus.edu.sg
- Checking might be done manually, thus could be costly, especially for EV SSL (most stringent checks done, activates padlock/green addr bar)

What is checked by CA before Cert Issuance?

DV Cert (Domain validation) SSL Cert:
- 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)

OV Cert (Organisation Validation) SSL Cert:
- Issued if the purchaser additionally has an organization's actual existence as a legal entity

EV Cert (Extended Validation) SSL Cert:
- Issued if the purchaser can persuade the cert provider of its legal identity, including manual verification checks by a human

TLS Certificate Level Summaries

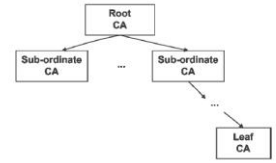
Certificate type	HTTPS encrypted?	Padlock displayed?	Domain validated?	Address validated?	Identity validation	Green address bar?
DV	Yes	Yes	Yes	No	None	No
OV	Yes	Yes	Yes	Yes	Good	No
EV	Yes	Yes	Yes	Yes	Strong	Yes

Different Browser UI Indicators

→ DBS has green address bar and padlock
→ Amazon only has padlock

Types of CA

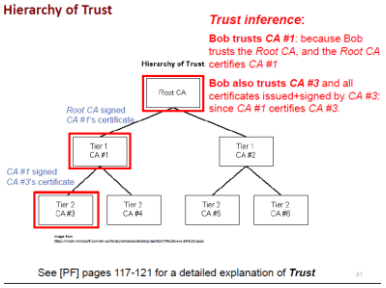
→ Root CA, Subordinate/intermediate CA, Leaf CA
- There are **3 different types** of CA:
  - ***Root CA***: whose certificate is self-signed
  - ***Sub-ordinate/intermediate CA***: Tier 1, 2, …
  - ***Leaf CA***: which issues certificates to end entities*



* **Note:** Some people/books *do not* distinguish between intermediate CA and leaf CA, i.e. a leaf CA is an intermediate CA

Hierarchy of Trust

→ If a person trusts the root CA and the root CA certifies some CA#1, then Bob trusts CA#1, and because of that, it will trust a CA#3 signed by CA#1



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

Certificate Chain/Path Verification Example

- If Bob trusts root CA, root CA trusts CA1, Alice's cert issued by CA1, Alice will issue her message with message + message cert (issued by CA1). Bob verifies message cert using root CA's public key, verifies message using the public key on the message cert.
  - Suppose Alice's certificate is issued & signed by $\mathbf{CA_1}$, which is a **tier-1 *intermediate* CA**
  - Bob **doesn't have** the public key of CA_1
  - ***Question: 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

- Illustration:

- In our example, CA_1's certificate clearly indicates that CA_1 is a CA that can issue certificate
- Without that "Note" portion, the certificate owner can't issue other certificates

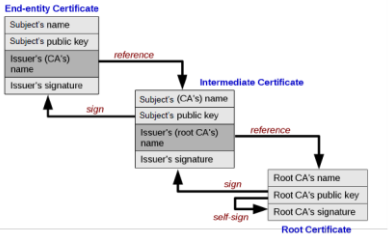
→ It must be clearly indicated that a cert is a CA that can issue certs

Certification Chain/Path Definition

- 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 CA 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 **private key** of **the next certificate** in the list
- The last certificate in the list, i.e. the root CA's, is the **trust anchor**

Certificate chain verification

- How does a certificate chain get verified?



→ Intermediate Certificate signs and writes signature in child cert using its own private key
Which we can verify using their public key
→ Root CA is self signed

Self signed certificate

A "**self-signed certificate**" is a certificate that is signed by the stated entity's private key.
Stated entity refers to the entity within the certificate
i.e. I sign my own certificate with my private key

Who typically uses one?

It is used by a root CA.
It is also quite commonly used by developers during the early stage of software development period when a valid certificate of a relevant host is not available yet.

Certificate Revocation

Non-expired certificates can be revoked for:
1. Private key was compromised
2. Issuing CA was compromised
3. Entity left an organization
4. Business entity closed

→ A verifier needs to check if cert in question is still valid, although the cert is not expired yet

Approaches to cert revocation:
→CRL (Certificate Revocation List): CA periodically signs and publishes a revocation list
→ OCSP(Online Certificate Status Protocol): OCSP responder validates a cert in question
→ Online CRL Distribution Point / OCSP Responder is needed
→ CRL deprecated for OCSP (On Mozilla Firefox 28)

Some OCSP Problems

- Privacy: OCSP Responder knows certificates you are validating. No privacy for user
- Soft-fail validation: Some browsers proceed if there is no reply to an OCSP request (no reply = 'good' reply)

Solutions:
→ OCSP Stapling: Allow 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

Limitations/Attacks on PKI

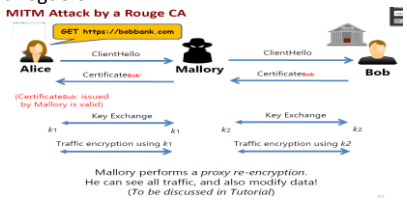
→ Compromised CAs (DigiNotar, Turktrust)
→ Abuse by CAs – There are so many CAs, some of them could be malicious. A **rogue CA** can forge any certificate

- **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

Weak Browser Trust Model

- Trust Browser Model: A pre-loaded list of widely-used root CAs compiled by web browser developers
- A form of certificate trust list (CTL) approach, list of CAs' certificates compiled by a 'trusted' authority

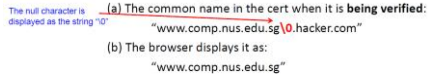
Security Issue: Certification is only as strong as the weakest root CA in the trust anchor (the union of all root CAs), since we don't know which root CA the browser uses from the root-CA list
→ This allows a Mallory to be a MITM if there is a rogue CA



→ Mallory can issue a fake certificate, which Alice's browser will accept, and then proxy all request from alice to bob. **Can see all traffic and also modify data.**
→ Session Key Exchange happens between Alice and Mallory & Mallory and Bob.

Browser Implementation Bugs: Null-byte Injection Attack

→ Some browsers ignore substrings in the entity's identity/name field after null characters when displaying on address bar, but it is included when verifying the certificate



→ User thinks he is connected to the correct website, but in fact to some insecure site!

CVE(Common vulnerabilities and Exposures)

- Database of publicly disclosed information security issues.

Social Engineering

→ Malicious hackers carry out **typosquatting** (registers domain name that is very similar to actual website)

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`.

Secure Channel, TLS/SSL, Misc Crypto Topics

Strong Authentication

- Password is a weak auth system; eavesdropper can get password and replay it. The issue is that the shared secret is exchanged between Alice & Bob, but can Alice prove to Bob she knows the secret without revealing the secret itself?

Cryptographic Challenge (Symmetric Key based)

- Suppose Alice and Bob have a shared secret key k, both agree on an encryption scheme (e.g. AES), and Bob wants to authenticate Alice
→ Alice sends Bob message to initialize communication, Bob randomly picks a challenge m, sends to alice y=Ek(m), Alice proves she has key by decrypting y with k to get m, sends m back to Bob and verifies the message received is indeed m
→ Eve cant get the secret key k even if she has all their communication messages
→ She cant replay the response because the challenge m is randomly chosen and likely to be different in the next authentication session → m ensures **freshness** of the authentication process.
→ Note: the protocol only authenticates Alice: unilateral authentication (there are also protocols to verify both parties i.e. mutual authentication)

Cryptographic Challenge (Public Key Based)

→ Alice authenticates herself to Bob using PKC; Initiates conversation by sending message to Bob, Bob chooses random number r, sends to Alice (Compared to SKC based, he doesn't encrypt it first), Alice uses her private key to sign r, sends back to Bob sign(r) + Alice's cert, Bob verifies cert, extract Alice's public key from the cert, then verifies the signature is correct.
→ Eve cannot derive Alice's private key and replay response because challenge r likely to be different. R is known as cryptographic **nonce.**
→ Nonce ensures freshness

Is Authentication enough for secure communication?

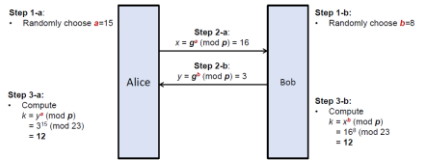
→ From the above, strong authentication assumes that Mallory cannot interrupt the session. However, it might be possible after authentication that the session is interrupted; Mallory is MITM.
=> Outcome of this auth process must be a new secret k(session key) established by Alice and Bob, which they can use to communicate, secured using the session key
→ Process of establishing secret is called **Key exchange / Key Establishment / Key Agreement**

Key Exchange / Authenticated Key Exchange

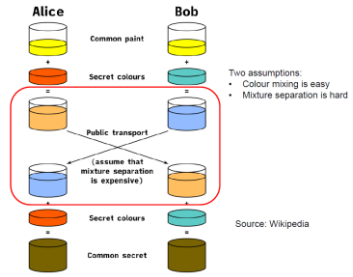
Diffie-Hellman Key Exchange Protocol

→ Assume that Alice and Bob have agreed on 2 publicly-known parameters: generator g and large (~1000 bit) prime p

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



Diffie-Hellman Key Exchange: Analogy/Visualization

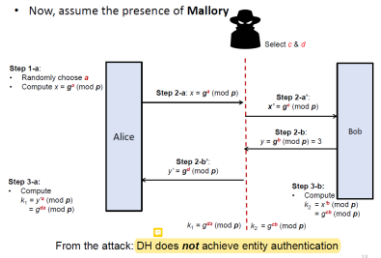


→(Optional) Diffie-Hellman works because we assume Discrete Log problem and Computational Diffie-Hellman problems are computationally hard.

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

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

→ **Unauthenticated DH Key Exchange does not achieve entity authentication.**
- A Mallory can sit as the MITM (Alice thinks she is communicating with Bob but communicating with Mallory, Bob also communicating w Mallory unknowingly). She just needs to select a c and d to achieve DH key exchange with both parties, then translate these messages (encrypted by different keys)



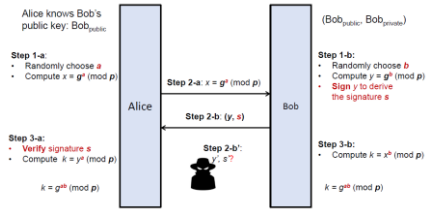
From the attack: DH does ***not*** achieve entity authentication

→ Decrypt message with Alice using k1, re-encrypt with k2 and send to Bob => She can **see and modify** the message

Solution? **Incorporate key exchange process with authentication**

Station-To-Station (STS) Protocol (Provides mutual key and entity authentication)

- Consider **unilateral** authentication: Alice wants to authenticate Bob. We add signature to DH



→ Alice also probably needs to authenticate Bob's private key using his certificate

=> Mutual Authentication? Make Alice sign her message in message 2a.

Requirements for authenticated key exchange:

Alice and Bob must have a way to know each other's public key (using PKI)

Unilateral – Only party that is being authenticated needs to have public/private key pair

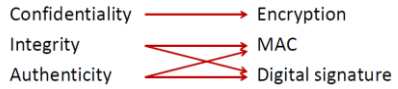
→ Actually with this derived 'session key', it is more like a secret value that can be used to derive a set of session keys, using a Key Derivation Function (KDF)

- After the protocol has completed, **a set of session keys** is established using a ***Key Derivation Function (KDF)*** like HKDF (based on HMAC) https://en.wikipedia.org/wiki/Key_derivation_function, **e.g.**:
  - 1 key for encryption & 1 key for MAC
  - Even more: 1 key for client-encryption, 1 key for server-encryption, 1 key for client-MAC, 1 key for server-MAC

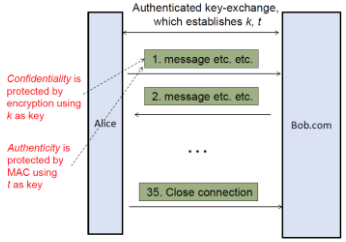
Securing a Communication Channel

→ Internet is a communication channel subjected to **sniffing and spoofing**

→ We need to achieve confidentiality, integrity, and authenticity



→ Do Authenticated key-exchange which establishes k, t (k for encryption, t for authenticity)



Question: Why do we need the sequence number?

→ Sequence number prevents a message-reordering attack

Secure channel between Alice and Bob.com

- From previous diagram, Alice and Bob.com carried out unilateral authenticated key exchange using Bob's private/public key.
- After authentication, both Bob and Alice know the randomly selected session keys k, t.
- Subsequent communication between Alice and Bob.com will be protected by k,t and a sequence number i. **MAC of encrypted message is concatenated to encrypted message**

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

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

where: i is the sequence number,

|| refers to concatenation

Note: The technique above is known as "**encrypt-then-MAC**". There are other variants of **authenticated encryption** called "MAC-then-encrypt" and "MAC-and-encrypt": *see later in this lecture*

→ Of course, with a public/private key, we will have PKI (certificates) to authenticate the public key

Putting All Together: TLS/SSL

- HTTPS (Ensures confidentiality) is widely used to secure web traffic, built on top of SSL/TLS

=> HTTPS = HTTP + SSL/TLS

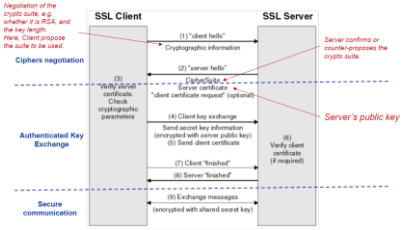
- TLS secures communication using cryptographic means, SSL is predecessor of TLS

TLS: High Level

1. Cipher suite negotiation
2. Authenticated Key Exchange: Exchange session key, authenticates identities of parties involved
3. Symmetric-key based secure communication
4. Re-negotiation (if needed)

5. 2 sub-protocols: Handshake Protocol for 1,2,4; Record protocol for 3, a lower-layered protocol

TLS Handshake (Ciphers Negotiation & Authenticated Key Exchange)



→ Difference between TLS and DH key exchange is in the way the key exchange is done. TLS simply uses PKC encryption, client generate string of random bytes 'premaster-secret', encrypts it with public key of server. With client nonce, server nonce and PMS, client and server generates the Master Secret

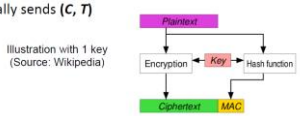
5.5. **Authenticated Encryption**

- Symmetric encryption that returns both ciphertext and authentication tag
- Combine cipher and MAC: ensuring message confidentiality (cipher) and authenticity (MAC)
- Authenticated Encryption: AE(Kab, M) = (C,T)
- Decryption: AD(Kab, C, T) = M only if T is valid

→ Can use Encrypt and Mac (E&M), Mac then Encrypt, Encrypt then Mac, specialized authenticated cipher

Encrypt and MAC (E&M)

- Uses 2 separate keys, K1 and K2, one for encryption and one for MAC
- It performs **encryption**, e.g. using 2 keys K_{1AB} and K_{2AB} as follows:
  - $C = E(K_{1AB}, M)$
  - $T = \text{MAC}(K_{2AB}, M)$
- It finally sends **(C, T)**



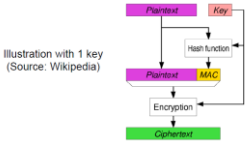
→ Note: Illustration only has 1 key!!!!

→ Used in SSH (Using strong MAC like HMAC-SHA-256)

→ Issue: T might not be random looking and could leak info

MAC-then-Encrypt (MtE)

- Generate tag T, then generate cipher text C, sends C
- The sender first computes the **tag** $T = \text{MAC}(K_{2AB}, M)$
- It then **generates** the ciphertext $C = E(K_{1AB}, M \,||\, T)$
- It finally sends **C**



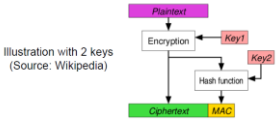
→ Encrypt message concat with tag using K1ab

→ Used in SSL/TLS up to version 1.2, v1.3 TLS uses authenticated ciphers (AES-GCM)

→ Issue: Decryption still needed on corrupted message

Encrypt then MAC (EtM)

- Generate ciphertext C, then generate tag T, then sends (C,T)



- It is used in **IPsec**

- Feature: Decryption not performed on corrupted message

Summary of the 3 methods:

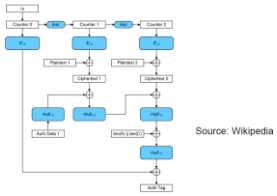
→ We always start by performing encryption / hashing with a key with message (could be either to produce tag or cipher text, depending it is MACed or Encrypted)

→ The other component is computed, depending on scheme

→ For MtE, since only one C is sent, we are forced to decrypt first before we can compare ciphertext with MAC(K2ab, message or C), so extra computation

Authenticated Cipher

- Returns authentication tag together with ciphertext (Don't need to know details)
  - One authenticated cipher in TLS version 1.3
  - The most widely used authenticated cipher



Source: Wikipedia

Security Reduction

→ Mathematical Proof that some cryptographic primitive/protocol is secure

We want to prove:

For any system *S*, there exists an attack on *S* that breaks requirement *A*	⇒	there exists an attack on *S* that breaks requirement *B*
For any system *S*, there does not exist attack on *S* that breaks requirement *A*	⇐	there does not exist attack on *S* that breaks requirement *B*
For any system *S*, the system *S* achieves requirement *A*	⇐	the system *S* achieves requirement *B*

From our Tutorial 4's question:

- *S* = hash function *h*()
- Requirement *A* = one way
- Requirement *B* = collision-resistant
- Objective: to prove the following **3 equivalent statements**

For any hash function *h*(), There exists an attack that invert *h*()	⇒	There exists an attack that finds collision of *h*()
For any hash function *h*(), There does not exist an efficient way to invert *h*()	⇐	There does not exist an efficient way to find a collision
For any hash function *h*(), *h*() is one-way	⇐	*h*() is collision resistant

→ Its just about what we need to prove when 2 requirements are linked

→ For many security protocol/implementations, there is no definite mathematical proof that it works, so we often say that 'we believe' there is 'no known attacks'

Other types of encryption

- Format preserving encryption
  - A basic cipher **does not** care if, for instance, a plaintext is an image
  - The ciphertext is **not** a viewable image
  - A format-preserving encryption solves this issue: ciphertexts have ***the same format*** as the plaintexts
  - Other possible target **plaintext types**: IP address, ZIP code, credit card numbers
- Fully-homomorphic encryption (evaluate another cipher text without decrypting the original ciphertext C)
  - Preserves the mathematical structure between plaintext & ciphertext spaces
  - Enables its user to replace a ciphertext $C = E_K(M)$ with another ciphertext $C' = E_K(F(M))$, where $F()$ is as a function of M, **without ever decrypting** the initial ciphertext C
  - Example: M is a text document, $F()$ is a modification of part of the text
  - It's very useful for a **cloud provider**: it doesn't know the plaintext/data, but **can change** the data as requested by the data owner (on the owner's behalf)
  - It is still very **slow**: a basic operation needs an unacceptably long time!

Network Security

→ Besides establishing secure channel, other security requirements (availability, anonymity) present.

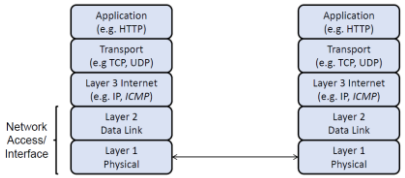
1. Preserving the **confidentiality**, **integrity**, and **authenticity** of messages from communicating entities
2. Maintaining the **availability** of the network and network services
3. **Controlling** and **monitoring** network traffic flow

OSI (Open Systems Interconnection) 7-Layer Model

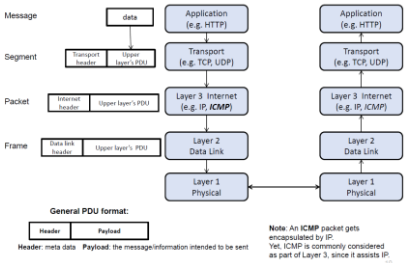
The ***Open Systems Interconnection (OSI) model***: a conceptual/reference model that standardizes the *communication functions* of a telecommunication/computing system

7 – Application	7 – Application
6 – Presentation	6 – Presentation
5 – Session	5 – Session
4 – Transport	4 – Transport
3 – Network	3 – Network
2 – Data Link	2 – Data Link
1 – Physical	1 – Physical

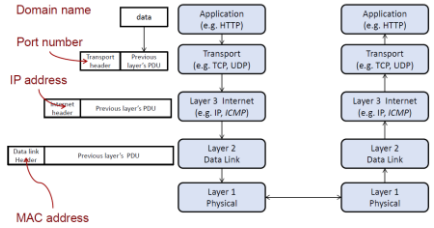
Internet (TCP/IP) Reference Model



Network Message Encapsulation



Network addressing schemes



Router: up to network/internet layer, switch: up to data link layer, repeater: up to physical layer, hub: up to physical layer

UDP vs TCP

- UDP: Connectionless-oriented and unreliable communication

- If an application wants to send a **UDP datagram**, the library call is typically of the following form:

```
DatagramSend (65533, "99.123.4.2",53, message)
```

src ip is implicit and thus doesn't need to be specified | src port | dest ip | dest port

- The library call would construct the corresponding *IP packet* to be sent using data-link layer protocol:

src ip	dest ip	src port	dest port	message

- There is limit on the size of message: at most ~65,000 bytes
- The library call does not return a result indicating whether the destination has indeed received the packet:
  - There is a possibility that the packet is lost!
  - This is the property of UDP protocol (i.e. how it works): ***connectionless-oriented*** *and* ***unreliable communication***

TCP: Connection-oriented, reliable.

→ 3 way handshake

- In contrast, TCP/IP is ***connection-oriented***
- An application would typically make the library calls of the following form and in the specified order:

```
P = open_connect (65533, "99.123.4.2", 80)
    send (P, out_message)
    read (P, in_message)
close_connection(P)
```

There could be multiple rounds of send/read

- `open_connect` would carry out a TCP 3-way handshake between the two nodes
- `send` would construct the corresponding *IP packet* to be sent using data-link layer protocol:

src ip	dest ip	src port	dest port	message

- If the message is too long, multiple IP packets will be formed
- TCP is ***reliable***: it has mechanisms to **re-transmit, re-order, acknowledge packets** so that the destination can receive the sent messages in the correct order

→ TCP is **reliable, but not secure**

→ Intermediate nodes can still read/modify data in header/payload of packets. This is address by TLS (HTTPS)

Resolution Protocols

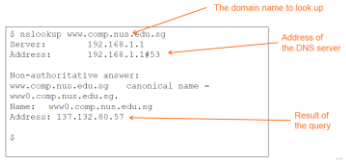
DNS: Map domain name to IP address. Can be attacked

ARP: Associate/Map IP Address to MAC address. Attacker can target association

DNS Example

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***

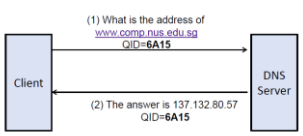


Authentication of DNS query

→ DNS Query: 16-bit number known as Query ID (QID)

→ Every query has a QID, response from the name server (DNS server) must also contain QID, if QID in response don't match QID in query, client rejects answer

→ DNS queries have no encryption/MAC 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

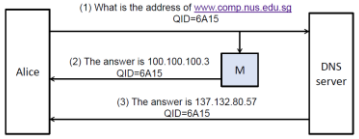
Premise: Alice is using café's WiFi (w/o protection) to surf web, visits website, making query to DNS server

Mallory: can sniff/spoof data from/into communication channel

→ But cannot modify or remove data already sent by Alice. Attacker also owns a webserver, a spoofed website that alice is visiting

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.



→ Basically intercept message, and give wrong ip address

Remarks about DNS:

- Operates at application layer. DNS can be 'single point-of-failure' for the network.

→ DoS attack attacking the DNS server instead of webserver

Denial of Service Attacks

→ Attack on **availability**

→ Local and Remote based DoS attacks

- Types of **DoS attacks**:

	Stopping Service	Exhausting Resources
Local Attack	• Process killing • Process crashing • System reconfiguring	• Spawning processes • Filling up file system
Remote Attack	Sending malformed packet attacks	***Packet flooding***

→ Local attacks more easily tracked

→ Malformed packet attacks does not usually work on updated OSes

Packet flooding attacks:

→Remotely flood victims with overwhelming requests/data

→ Attack can amplify small traffic to obtain large traffic, using public servers such as DNS, NTP, CharGen

ICMP/Smurf Flood Attack

1. Attacker sends "ICMP PING" request to router, router broadcasts this request to all local nodes. We spoof the source IP address to be the **victim's** IP address

2. Once the router broadcasts this echo request, all entities who received this request will reply to it with an "Echo Reply". Victim gets overwhelmed with replies from entire network due to the **amplification effect**

→ **AAF(Attack Amplification Factor):** Ratio between the sizes of the response and request.

→ Attack is no longer effective nowadays because most routers are configured not to broadcast request.

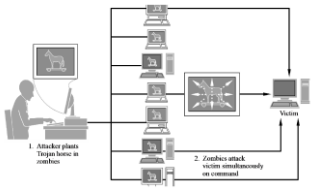
Application-Layer DOS attack (Http GET)

→ Flood a web server with HTTP Requests

→ Large number of attackers are required for this attack to be effective, since attacker can only send requests at low rate

→ **Distributed Denial of Service (DDoS):** DoS carried out by a large number of attackers

Distributed Denial of Service (DDoS)



Botnet

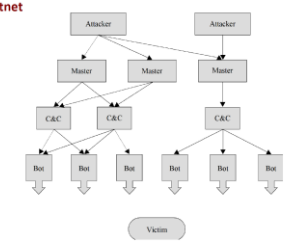
→ A bot/zombie is a compromised machine

→ Botnet (zombie army) is a large collection of connected bots, communicating via covert channels

→ **Command & Control mechanism:** can be controlled by 1 individual to carry out DDoS

→ Usages of Botnet: DDoS flooding, vulnerability scanning, anonymizing HTTP proxy, email address harvesting, cipher breaking

Botnet



UDP Based Attacks

→ BAF (Bandwidth Amplification Factor) – number of UDP payload bytes an amplifier sends to answer a request, compared to the number of UDP payload bytes of request

- For UDP-based attacks, ***Bandwidth Amplification Factor (BAF)*** is the number of UDP payload bytes that an amplifier sends to answer a request, compared to the number of UDP payload bytes of the request
- See the table on the right for **different BAFs**
- What was the **largest DDoS attacks**?
  → an unnamed customer of the US-based service provider Arbor Networks, reaching a peak of about **1.7 terabits per second**

UDP-based Amplification Attacks

Protocol	Bandwidth Amplification Factor
Memcache	50000
NTP	556.9
CharGen	358.8
DNS	up to 179 [50]
QOTD	140.3
Quake Network Protocol	63.9
BitTorrent	4.0 - 54.3 [51]
SSDP	30.8
Kad	16.3
SNMPv2	6.3
Steam Protocol	5.5
NetBIOS	3.8

https://en.wikipedia.org/wiki/Denial-of-service_attack

Nmap (Port Scanner)

→ Port scanning: Determine which ports are open on hosts in a network

→ Attacker and network admin can use to scan for vulnerabilities

Security Protocols at Different Layers

→ TSL/SSL, WPA, IPSEC protects 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

→ This slide doesn't really say much but will just put for good measure

→ 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: When we implement a security protocol at layer k we only protect from layer k **onwards**, so anything before that is not protected

→ For e.g. by encrypting payload with TLS, we are protecting app layer data, but IP header is in plaintext, so an **attacker in the physical layer** can know that a user is visiting google.com but wont know the contents of the user's upload to google.com

→ But **attacker in application layer** will be able to see Alice's report because it is residing in layer before TLS

WPA2 (WiFi Protected Access II): Protocol employed in home WiFi access point, more secure than WEP, WPA, ensures confidentiality

→ Provides Layer 2 (link) and layer 1 (physical) protection. **Note: Not all layer 2 information in protected**

IPSec – provides integrity/authenticity protection of IP addresses. **Does not provide confidentiality**

→ Attackers cannot spoof source IP address

→ Attackers still can learn the source and destination IP addresses of the sniffed packets

→ Works at the internet layer of the internet protocol suite

→ Secures IP communications by authenticating and encrypting each IP packet of a communication session

Firewall

→ SSL/TLS or WPA2 cannot prevent attacks such as DoS. DNS spoofing is not protected by SSL/TLS (DNS queries are in plaintext)

→ Control flow of traffic between networks, especially between untrusted public network and trusted internal network

→ Divide internal network into different network segments to **deny unnecessary access: Principle of least privilege, compartmentalization**

→ Looks at addresses/services of traffic

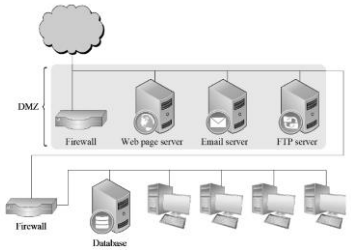
→ Ingress (who enters network) / egress(who leaves network) filtering

- **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)

→ Small subnetwork that exposes organization's external service to untrusted internet

Demilitarized Zone (DMZ)



Sample Firewall Configuration

Sample Firewall Configuration

Rule No	Protocol Type	Source Address	Destination Address	Source Port	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
n	*	*	*	*	*	Deny

Matching condition

* (means "any"), which matches **any** values

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*

→ Prefer whitelists over blacklists! Deny anything else that doesn't fit whitelist rules

Types of Firewalls

1. ***(Traditional) packet filters***:
   - Filters packets based on information in **packet headers**
2. ***Stateful inspection***:
   - 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

Lecture 7: Access Control

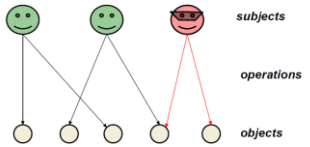
- Computer systems has own processes and data, security concern regarding access to the processes & memory/storage
- Concerns: Data confidentiality, data integrity, process integrity: whether it deviates from original execution path

- Upper layer in app layer (upper layer) hard to detect bug in OS layer (lower layer)

- An important **design consideration** of the security mechanism: **how to *prevent* attacker** from getting access to a *layer inside* the boundary
  - 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; so that 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**

Access Control

- Selective restriction of access to a place or other resource
- Specifies & enforces restriction of operations on objects by principals/subjects



→ A principal (or subject) wants to access an object with some operation. **Reference monitor** either grants/denies this access

Principals vs subjects

Principals vs subjects:

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

Access types:

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

→ Basically read,write,execute

1. Discretionary Access Control (DAC): Owner of object decides the rights. UNIX uses this.

2. **Mandatory Access Control (MAC):** System-wide policy decides the rights, which must be followed by everyone in system

Access Control Representation

1. **Access Control Matrix (ACM):** A matrix of principals vs objects

R=read, w=write, x=execute, o=owner, s=execute as file owner

- ACM uses a **matrix/table** to specific the access rights of a particular principal to a particular object

principals \ objects	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}	{}

→ Issues: Table will be very large and difficult to manage, hardly used

Access Control List / Capabilities

→ ACM is basically an ACL and capabilities combined. Object centred vs subject centred

- ***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")

→ UNIX uses ACL.

Access Control List (ACL)

- Each object has a list of access rights

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}) → (Alice, {r,w,o})

Issues: Difficult to get an overview of which objects that a particular subject has access rights to (we need to traverse through all files just to find where all the subject's name appears)

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})

Issues: Difficult to get an overview of which subjects a particular objects allows its rights to (must traverse through all users to find where all the file name appears)

Drawbacks of ACL / Capabilities

- Size of lists can still be too large
- Need model that is fine-grained but easy to manage

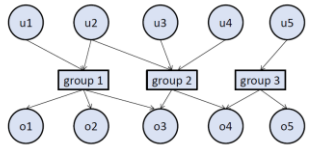
→ Solution? Group subjects/objects and define access rights on defined groups, aka intermediate control

Intermediate Access Control

- **Group in UNIX**: Specify the rights for the following user classes: user (the owner), group (the owner's group), others (world)
- Non- owner subjects in same group have same group 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`

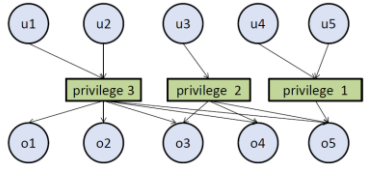
Users and Groups: Illustration



→ In UNIX, groups can only be created by root. Extra groups info is stored in /etc/groups

Privileges

→ Describes the access rights of a user. Is an intermediate control



Role-based Access Control (RBAC)

- Grouping can be determined by the 'role' of the subject. Role is associated with a set of procedures, and access rights to certain objects are granted to carry out these procedures

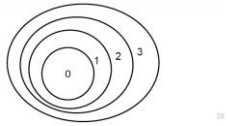
Least privileged principle

least privilege principle: access rights that are *not* required to complete the role will *not* be assigned

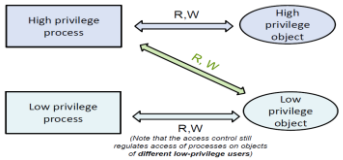
Intermediate Control: Protection Rings

- The smaller your number the higher your privilege, can access all higher numbers but cannot access smaller number than you

If a **process** is assigned a number *i*: it runs **in ring *i***
A subject **cannot** access (read/write) an object with smaller ring number

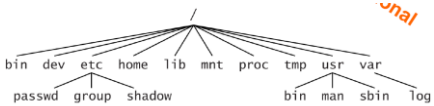


→ UNIX has 2 rights: Superuser and (normal/non-root) user



Access Control in UNIX/Linux

File system:



/etc/shadow: hashed passwd file

Unix/Linux Objects

- files, directories, memory devices, I/O devices
- All resources treated as a file in UNIX

UNIX/Linux User and Groups

Each user:

1. Has unique user/login name
2. Has numeric user identifier (uid) stored in /etc/passwd
3. Belong to one or more groups: first group stored in /etc/passwd, additional group(s) are stored in /etc/group

Each group:

1. Has unique group name
2. Has a numeric group identifier (GID)
   - Main **purposes** of UIDs and GIDs *(more on this later)*:
     - To determine **ownership** of various system resources, e.g. files
     - To determine the **credentials** of running processes
     - To control the **permissions** granted to processes that wish to access the resources

UNIX/Linux Principals

- Principals: User identities (UIDs) and group identities (GIDs)
- Superuser has uid 0, and username usually root

→ All security checks turned off for root

Information of the user accounts are stored in the **password file** `/etc/passwd`

E.g. `root:*:0:0:System Administrator:/var/root:/bin/sh`

Username:password:userid:groupid:gecos field:path to users home dir:program that is started when user logs into system

→ * password means theres no password, disabled for login, cant login

→ gecos: some comments about user

Unix Password File: Remarks on Protection

- World-readable because some info in /etc/passwd is needed by non-root processes
- All users could access to hashed-password field in older versions of UNIX (instead of seeing *), allowing for **offline password guessing**
- Now password is replaced by *, actual password stored somewhere else, not world-readable
- hashed password stored in /etc/shadow instead

Shadow Password File

Fields of an entry:

- 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$bUdht9fglwJW0LduAxEJpcExtMfKokFMJoT8tGkKLx5xFGJk22/trPstOHXr4PdBlD0AV1xko5LfFVDwW.aJS.:17275:0:99999:7:::
```

Hashed password format:

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

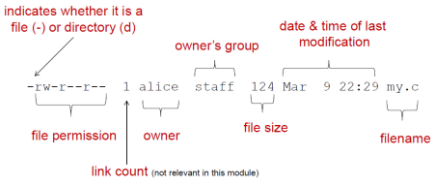
- **id**: ID of the hash-method used (5=SHA-256, 6= SHA-512, …)
- **salt**: up to 16 chars drawn from the set [a-zA-Z0-9./]
- **hashed-key**: hash of the password (e.g. 43 chars for SHA-256, 86 chars for SHA-512)

Unix/Linux Subjects

→ Processes are subjects in UNIX

→ Identified by PID

Files/File System Permission



→ first – indicates file or directory!!

→ File permission grouped into user, group, others with read, write, execute

'-' 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 (chmod)

- 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

Symbolic mode notation:

- *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
```

→ Only owner of a file or superuser can change the permission bits

→ For symbolic, multiple specs separated by comma

Octal mode notation:

→ 3 – 4 octal digits, 3 rightmost digits refer to permissions for: user, group others

- **Octal mode** notation:
  - 3-4 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	---

→ If 4 digits, **first digit** specifies (setuid, setgid, stickybit)

SetUID/SetGID/Sticky Bit

→ If set-UID=1, the process' effective user ID is the owner of the executable file (usually root), rather than user running executable (vulnerability!!)

→ If set-GID=1, process' effective id is the group owner of the executable file

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

Prevents a user from deleting other users' files in public directories (such as /tmp)

Permission-Checking Rules

→ Objects are files, each file is owned by a user and a group

- 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

Controlled Invocation & Privilege Escalation

→ Certain resources in unix/linux can only be accessed by superuser/root, but a non-root user needs those resources

"Elevated Privilege": Created by high-privilege users such as root, a set of predefined programs (bridges) where users can invoke these applications **with privilege of high-privilege users**

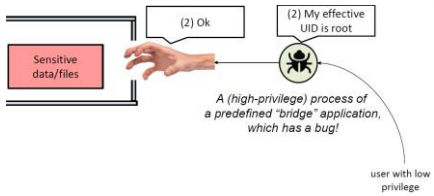
→ Can carry out limited operations only, these bridges need to be implemented correctly else can be abused by low-privileged users to perform unauthorized actions

→ Attacker can trick bridge to perform 'illegal' operations not expected by the programmer

→ Privilege Escalation Attacks

- Suppose a "bridge" is ***not*** implemented correctly, and contains **exploitable vulnerability**
- In some vulnerabilities, an attacker can trick the bridge to perform **"illegal" operations** not expected by the programmer/designer
- 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.)*



Controlled Invocation in UNIX/Linux

- Controlled-invocation provisioning in UNIX: Provide set of operations (programs) in a superuser mode, so that non-root user can run these operations with superuser mode

→ Using set-UID bit

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

- The **"s" (set-UID) bit** indicates that the privilege is **elevated** to the ***file owner*** (in this case `root`) while a user is running this process:
  - If the file owner is *not* `root` but `bob`, then the privilege is correspondingly elevated to `bob`'s instead

Process Credentials and Set-UID bit

→ process associated with process credentials: real UID and effective UID.

→ If set-UID is 1, then effective ID is inherited from uid of file owner. Else effective id and real uid always inherits from user executing process

- 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`

When a Process (Subject) wants to read a file (object)

→ Effective UID is treated as the 'subject' and checked against the file permission to decide whether it will be granted or denied access

- 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
- 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"
- 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`

Unix Search Path

→ We can execute a program by relative path. We might accidentally run a malicious program with a common name, that is stored in a different directory, invoking the malicious program instead!

→ **Specify full path. Don't use relative path.**

Web Security

- Subresources in HTML – images, multimedia, CSS, scripts

Web client and Server Components

Client-side: HTML, CSS, JS,PHP

Server Side Web server, DB

HTTP Request/Response

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, */*
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:

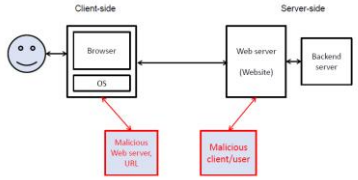
```
Content-Type: text/html
Content-Length: 35
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

  - An empty/blank line
  - An optional message body

Complications of Web Security: Browser's Operations

- Browsers run with same privileges as user. Browser can access user's files
- Access isolation among sites is required
- Browsers keep user's info and secrets (cookies)

Threat Model 1: Attacks as Another End Systems



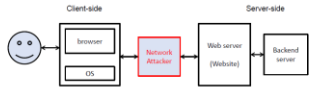
- In this scenario, the attackers are just another ***end systems***
- Examples:
  - A **malicious web server** that lures the victim to visit it
  - A **malicious web client** who has access to the targeted server

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, e.g.:
    - Via "Click Here to Get a Free iPad" link
    - Via an advertisement (no clicks needed)
  - **Can *not*** intercept/read traffic for other sites
  - The most **commonly-assumed** attacker type. *Why?*

Threat Model 2: Attackers as Network Attackers

Threat Model 2: Attackers as *Network Attackers*



- Here, the attacker has **access** to exchanged network packets (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, e.g. **MiTM**
  - 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?*

Attacks on Secure Communication Channel (SSL/TLS)

MiTM Attack:

- Attacker is a MiTM in between browser and web server
- Attacker is able to sniff & spoof packets at the TCP/IP layers

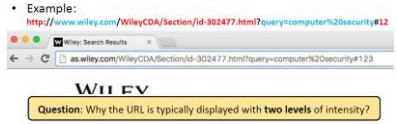
If connection is HTTPS, MiTM cannot compromise both confidentiality & authenticity unless **web user accepts a forged certificate or rogue CA**

→ If there are vulnerabilities in the protocol's design or implementation, above doesn't need to happen to compromise confidentiality & auth. E.g. Re-negotiation attack, FREAK attack, heartbleed (attack of protocol's implementation), BEAST attack

Misleading Web User (UI/Visual-based Attacks)

URL Components:

- 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:



Misleading Delimiter: A malicious website's hostname can contain the targeted hostname followed by a character resembling a /
(e.g. www.wiley.com.[wiley.in/Section/id-302477.html)

- Another example: nuslogin.789greeting.co.uk (from phishing email)
- The displayed different intensities could help user **spot** the attack

→ That's why there are different intensities so users can spot which is the domain and subdomain.

Address Bar Spoofing

Address is the only indicator of what url the page is actually rendering
→ What if it could 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
- Only could happen in poorly designed browsers

- In the early design of some browsers, a webpage 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

Cookies and Same Origin Policy

HTTP Cookie: A piece of textual data that is set by a web server, and sent in a HTTP response's "Set Cookie" header field
- Consists of name-value pair
- Can be used to indicate a user preference, server based session identifier
- Stored on user's browser while user is browsing
- Browser automatically sends all 'in-scope' cookies to server in all its HTTP request's "cookie" header field when user revisits website
  - (*Optional*) **In-scope** cookies: cookies set by the ***"same-cookie" origin***, i.e. the "origin" of the cookies
    (Note: the scheme/protocol checking *may* be optional, *see later*)

- HttpOnly: prevents client-side script from accessing cookie data, prevents XSS

Cookies: Usage

Types of cookies
1. Session Cookie: Deleted after browsing session ends
2. Persistent Cookie: Expires at a specific date or after a specific length of time
3. Secure cookie: Can only be transmitted over HTTPS
   - Note: the checking on scheme in the "same origin" for cookies is **optional**, except for **secure cookies** which strictly require HTTPS

→ HTTP is stateless. Cookies are used to keep track of state in a web session (e.g. Session ID in token based web authentication scheme)

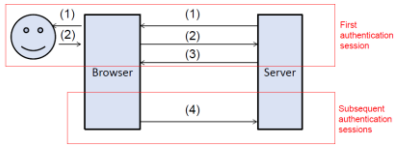
Token-based Authentication and Cookie

- Many websites use token-based authentication, web users do not have to repeatedly login

1. After a user *U* is **authenticated** (for e.g. **password verified**), the server sends a value ***t***, known as the **token**, to *U*
2. In subsequent connections, whoever presents the **correct token** is thus **accepted** as the authentic user *U*

→ A token can identify a session, known as session ID
→ Token typically has an expiry date for a limited session lifetime (cookie expiry)
→ Token often is stored in a cookie
→ Better approach than attached session ID as URL-encoded param or as a form field (more secure), but cookies has its own issues too!

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***

Choice of Session Token and Storage Requirement

- A token t needs to be random and sufficiently long
- If token t is a randomly chosen number → server must keep a table storing all issued tokens
- To avoid storing table:
  - (**Insecure**) The cookie is some meaningful information concatenated with a **predictable *sequence number***
    E.g. ***t*** = "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*** = "alicetan:16/04/2015:adc8213kjd891067ad9993a"
- For both methods, when the server finds out that the token is not in the correct format or 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 running in the browser could access cookies (and also webpage's objects)
- Due to security concern, browser employs same-origin policy access control: **A script in webpage A can only access cookies in a webpage B if both A and B have the same origin**.
Origin: protocol, hostname, port number

SOP: Some complications

Origin Determination Rules:

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

→ There are many exceptions, and the rules are not consistent. Microsoft IE does not include the port in the calculation of the origin, using **Security Zone** in its place

Cross Site Scripting (XSS) Attacks

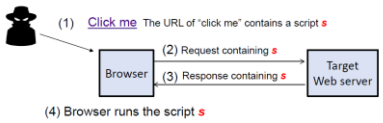
Reflected XSS (Non-Persistent)

In many websites, client can enter a string s in the browser, to be sent to the server. Server responds with a HTML that contains s, which is rendered and displayed by the client's browser
→ We can add a script inside the input field!
- Important question: what if the string ***s*** contains a **script**?
  - Example: http://www.comp.nus.edu.sg/<script>alert("hehehe!");</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: The 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***



→ Basically, 'inject' some script into some input instead of normal text, which without sanitization, will be run

Complications of Reflected XSS Attack

- Malicious script can steal cookies, deface original webpage
- **Can access cookies from web server,** since script is also coming from web server!!
- **Example of privilege escalation**: A malicious script coming from the attacker **has privilege of the web server** and read the latter's cookies
- Exploits client's trust of server: The browser believes the injected script is from the server

Stored XSS (Persistent)

- Script s is stored in the target web server
- E.g. stored in a forum page: Attacker is a malicious forum poster
- More dangerous than reflected XSS attacks: Malicious script is rendered automatically, without the need to lure target victims to a 3rd party website
- **Victim-to-script** ratio is many:1
  - 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))

→ E.g. Forum poster uploads script onto forum, script is automatically run everytime the forum page loads, since the script is on the webpage!

XSS 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"

→ Works by exploiting the victim's trust of the 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

XSS Defenses

- Mostly rely on mechanisms on the server-side: Server filters and then removes/sanitizes any malicious script in a HTTP request while constructing its response page or before its saved into forum database
Techniques: Script filtering, noscript region: Do not allow JS to appear in certain region of webpage, httponly cookie
→ Still not foolproof. Some browsers additionally employ client-side detection mechanism (XSS auditor) to detect reflected XSS attack

CSRF (Cross Site Request Forgery) Attacks

- "Sea surf" or session riding
- request forgery vs xss:
xss is about injecting some form of script to do bad action, request forgery is like doing something with the rights of alice, "forging a request"

- 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* e.g. in a phishing email, 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** by *S*

CSRF Attack: Without victim clicking on URL

- A web attacker can perform 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:
  <IMG SRC="www.bank.com/transfer?account=Alice&amount=1000&to=Bob" 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** by *S*

CSRF Attack Summary

- 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"

→ Disrupts integrity of victim user's session
→ Reverse of XSS: Exploits the server's trust of the client

CSRF Attack Defense

Cause of CSRF: Forged request is indeed issued by victim user's browser, but without user's real intention or knowledge
→ SID/auth-token cookie is insufficient, there should be an additional measure to tell server that the user really issues a transaction
- *Any good yet simple security measure?*
- How about ensuring that the user's transaction does come **from a transaction page** sent by the server to the user?
  (Note that the user must be previously authenticated)

- CSRF is relatively easier to prevent than XSS

Anti-CSRF Token:
Server issues an anti-CSRF token, which is extra dynamic information to a user's transaction page
- When submitting his/her transaction req, user must also include this token: indicates user transacts from the page
- Attack who didn't previously receive the transaction page from server cannot include the token in its forged requests

- In the considered attack scenario, Alice transaction's **HTTP request** must **include** the server's sent anti-CSRF token:
  - In the request's **URL**:
    www.bank.com/transfer?account=Alice&amount=1000&to=Bob&
    CSRFtoken=xxk34n890ad7casdf897e324
  - As a HTTP request **header field**
  - As a ***hidden form field*** (in a HTTP's POST request)

SQL Injection

- 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

Attack can pass the following as 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!

Software Security

- Program has to be correct, efficient and secure
→ Else, a program may behave beyond its intended functionality
- **Insecure Impl:** programs not implemented properly, allowing attacker to deviate from programmer's intents
- **Unanticipated Input:** Attacker may supply input in a form that is not anticipated by programmer resulting in: access to sensitive rsrces, execution of injected codes, deviation from original intended execution path, **elevating its privilege**

Buffer Overflow Examples

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)

SQL Injection Examples

- **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 Example

- **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

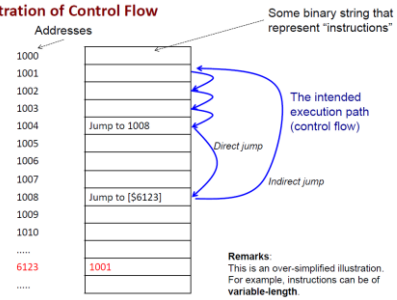
Code vs Data, Program Counter

- Code and data are stored together in memory, no clear distinction
- Programs may be tricked into treating input data as code

Control Flow

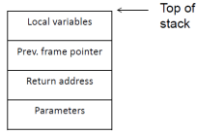
Program Counter/Instruction Pointer – stores address of next instruction, processor fetches next instruction from address stored in PC. PC typically increases by 1, but can be changed by **direct jump** or **indirect jump**

Illustration of Control Flow



Stack (Execution Stack, Call Stack)

- During program execution, stack keeps track of control flow information (return addresses), parameters passed to functions and local variables of functions



→ **Stack grows upward,** from higher addresses to lower addresses. When inputting in buffer, we input towards the return address as well!

Control Flow Integrity

Treating Code as Data: Security Implications

- Call stack stores a return address as data in memory. Attack can compromise a process' execution integrity by modifying the process' code or control flow
- 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**".

Integer Overflow Attack

- Integer arithmetic is 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 in **modulo 256**

→ By getting negative number, can use to access negative indices on arrays.

Buffer Overflow Attack

- C and C++ allows programmers to manage memory: pointer arithmetic, no array-bounds checking
- Data is written beyond buffer's boundary
- 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

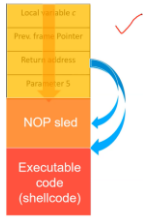
→ strcpy has no bounds checking on s2, will copy the entire string, leading to buffer overflows
→ use strncpy() instead

- The function `strncpy()` 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*

Examining the Requirements of Buffer Overflow

It's useful to look at the following BO requirements:

- **Existence of vulnerability**
- Overwriting important data
- Known location of injected code
- Executable code in input



Stack smashing (Stack overflow)

- Buffer overflow that targets a process' call stack
- Overflow stack such that return address is modified, changing the exection's control flow
- if setuid bit is set and owner is root, escalates privilege for attacker

Buffer Overflow Targets

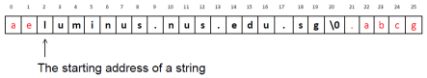
- Data in memory that controls program execution:
  - **Return address**
  - Function pointer
  - Virtual function table (vtable): in heap overflow
- **Important program variables** (program specific):
  - Credential-related variables: user-id, authentication status, secret key
  - Current balance of bank application
  - ...

Data/String Representation & Security

Different parts of program/system adopt different data representations

String Representations (printf)

- 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



→ **Not all systems adopt this convention:** NULL-termination vs non NULL-termination

Exploitable Vulnerability 1: NULL-byte Injection

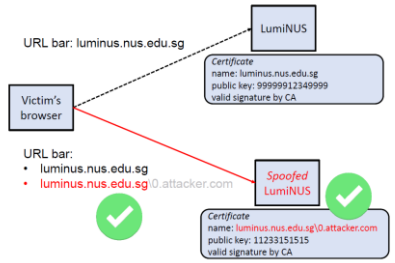
→ CA may accept host name containing NULL character

- 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 enter by user: based on the **NULL-termination** representation
- Now, there could be an **attack** as described on the next slide!

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

→ Cert is valid when CA does non-NULL termination representation. But on address bar, the string was terminated on null, so user cannot see that website is spoofed!

A Sample Attack (on LumiNUS): Illustration



→ If its just a normal web spoofing attack, when attacker redirects user to spoofed webserver, user can notice that address displayed in browser's address bar is not LumiNUS, or that the protocol rejects the connection (certificate is not trusted)
→ This attack is much more dangerous

Vulnerability 2: ASCII Character Encoding

ASCII encodes 128 characters into 7-bit integers.
In a byte, first ascii bit is 0.
95 printable characters: digits, letters, punctuation
33 non-printing (control) characters

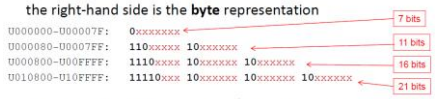
- **Extended ASCII** (EASCII or high ASCII) character encodings, which comprises:
  - The standard 7-bit ASCII characters
  - Plus ***additional characters***
  - See: https://en.wikipedia.org/wiki/Extended_ASCII

More about utf 8 utf-8

→ Unicode to encode more characters using 1 to 4 8 bit bytes
→ Variable length encoding: For more frequently occurring characters, we use the lower numerical values
→ First 128 characters are simply ascii. So ascii is unchanged on utf-8, providing backward compatibility with ascii

Exploitable Vulnerability 2: UTF-8 Variant Encoding Issue

- 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)**

→ With variable length encoding, we represent them using different prefixes in the first byte (leading byte) to differentiate them. The continuation bytes always have prefix of 10.
→ Representation of a utf-8 character should be unique, but many implementations accept

multiple and longer variants of a character! (e.g browsers)

Different representations of a UTF-8 character

→ '/' has 1,2,3,4 byte versions

- 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 '/'

→ There could be inconsistency in the character verification process, and the character usages: operations using the character

UTF-8 Variant Vulnerability: 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

→ What if attacker adds a ../ path in the string?

- 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

→ We try to add input validation to make sure '../' never appears as a substring. However, this may not be sufficient.

Now, further suppose that the **system call in Step 2**:

- 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`
- Uses **UTF-8**

→ Different encoding may be accepted. In fact, due to variant encoding in UTF-8, the checks we have in place may not be complete and miss 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`

→ Blackbox filtering could be incomplete due to flexible use of character encoding

Another example: IP Address blacklisting

→ 4 byte addresses usually written as a string, "132.127.8.16"

→ To check for blacklist, a function takes in 4 integers, check whether all 4 integers match. If yes, then we know this is a blacklisted IP.

- 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,` and `D[i]==d`

→ If not a blacklisted IP, program generates the IP. However there may be problems!

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 (***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?

→ There might be a different interpretation to the ip, and by carefully setting the 4 integers, we can get a calculated ip which would have been blacklisted, bypassing the blacklist

Lessons – Use Canonical Representation

- Never trust input from user, always convert them to a standard representation!!
- Don't rely on the verification check done in the application, try to use underlying system access control mechanisms

Undocumented Access Points (Easter Eggs)

→ Many programmers insert 'undocumented access points' to inspect states for debugging

- 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' (covert method of bypassing normal authentication) to attackers.
Such access points are known as Easter Eggs

Defense and Preventive Measures

- In general, it is difficult to analyze a program to ensure it is bug-free (halting problem). There is no fool proof method!

Using safer functions

- Using strncpy instead of strcpy() (even then, can still be vulnerable)

Input validation using filtering

- Reject input if it is not in some expected format
- Difficult to ensure that filtering is complete (hard to design filter that blocks all bad inputs and accepts all legitimate inputs)

Filtering

→ White list filtering vs blacklist filtering, allowing items known to be benign to pass (use regex), or using blacklist filtering (List of items known to be bad and to be rejected, e.g. input too long, contains control/meta characters, contains negative number)

→ Preventive measure is to always perform input validation/filttering whenever an input is obtained from user

Bounds Checking and Type safety

Some languages (e.g. java) perform bounds checking at runtime

→ Upper/lower bounds of an array, index/subscript of array , throw exception if checks fail

→ Reduce efficiency, but prevent buffer overflow

Type Safety

→ Type checking to ensure arguments have same type (8-bit vs 64 bit integer), can be done at runtime or compile time.

Program Analysis & Code Inspection

- Manual Checking – slow, tedious
- Automated Checking – Taint analysis
  - 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)

Memory Protection (Canaries, Randomization)

- Deals with overwriting important data of buffer overflow attacks

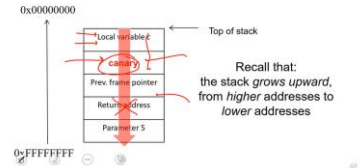
Canaries

"Secret" values inserted at carefully selected memory locations at runtime, checks are carried out at runtime to make sure values are not being modified

- Canaries help detect overflow (especially **stack overflow)**
  - In a typical buffer overflow, **consecutive** memory locations have to be over-ran: the canaries would be **modified**
- **Important to keep values 'secret'**. If attacker knows value, it may be able to write the secret value to the canary while over-running it
- **Turn off gcc canary-based stack protection:**
  - (***Optional***) In Linux, you can **turn off** gcc canary-based stack protection by supplying this flag when invoking gcc: `-fno-stack-protector`

Example

Canaries



→ Place canary in between local variables and frame pointer, return address info in stack. Canary can detect when value is being overwritten (stack smashing detected)

- If compiled with stack protector:

```
./program-too-wsp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Function is exiting
*** stack smashing detected ***: ./program-too-wsp
terminated
Aborted (core dumped)
```

- If compiled *without* stack protector:

```
./program-too-wsp
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Your supplied argument is
:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Function is exiting
Segmentation fault (core dumped)
```

→ Checking of canary is done AFTER function exits, not right after a strcpy! **Cannot prevent** buffer overflow, can only detect BO.

Memory Randomization

- Deals with known location of injected code of buffer overflow attacks
- When data and code are always stored in same locations in memory, easy for attacker to exploit

ASLR (Address space layout randomization) is a prevention technique that can help decrease the attacker's chance of identifying the location

- 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`

Testing

- Vulnerability can be discovered via testing
- Whitebox testing (tester has access to app's source code), blackbox testing (no access to source code), graybox testing (combination of above)
- Attempts to discover intentional attack. Use **Fuzzing,** sending malformed inputs to discover vulnerabilities. Can be automated or semi automated

Principle of least privilege

- Be conversative in elevating privilege. Do not give users more access rights than necessary, do not activate unnecessary options

- 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.

Patching (Vulnerability lifecycle)

- When patch is announced, vulnerability can be made known to attackers

- **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

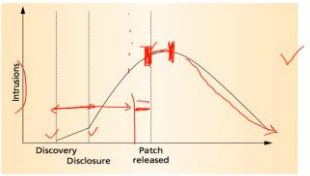


Figure 1. Intuitive life cycle of a system-security vulnerability. Intrusions increase once users discover a vulnerability, and the rate continues to increase until the system administrator releases a patch or workaround.

→ Patching may not be that easy

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

Software Security Summary

Summary: Defense Mechanisms and Stages of SDLC

Adopt various counter measures at **different *stages of SDLC***:

- **Development** stage:
  - Using safer functions
  - Filtering (input validation)
  - Bounds checking and type safety
  - Program analysis (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**: address randomization
  - The principle of least privilege*: disable unnecessary features
  - Patching

* Applicable to *multiple* stages

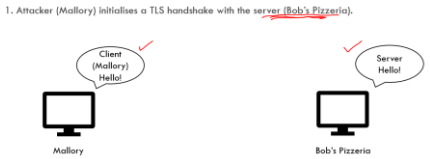
107

TLS Renegotiation attack

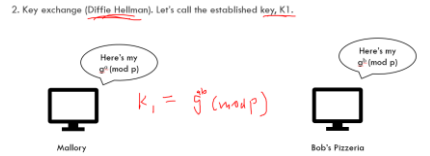
- Renegotiation allows the client to reconnect to the server faster that re-establishing connection
- Allows client to use previously established session keys to renegotiate new keys, and preserves session established by client

How it works:

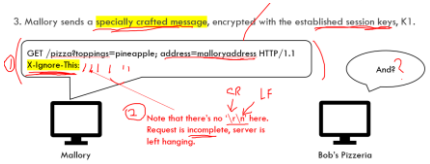
1. Mallory initializes TLS handshake with server



2. Mallory performs key exchange with server (DH, TLS key exchange etc)

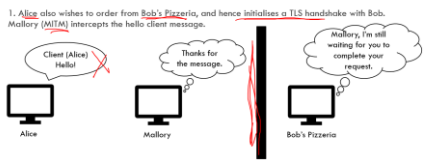


3. Mallory sends his message with an x-ignore-this header, without \r\n (incomplete req)



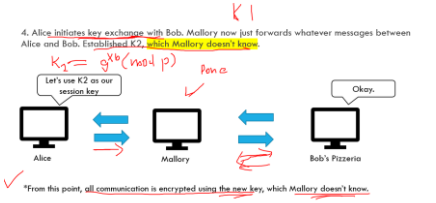
1. Alice tries to establish TLS handshake with Bob, Mallory intercepts it. Bob is still left hanging from Mallory's incomplete request

THE ATTACK



2. Mallory encrypts Alice's ClientHello (its in plaintext) with the k1 Mallory and Bob established. Bob will reply with server hello. Mallory decrypts the server hello and sends it to alice.

3. Alice will initiate key exchange. Mallory **will not know the key,** since he doesn't have the b value to calculate the new session key!



4. So now, Mallory will need to receive and send the requests from alice to bob and the other way round, acting as a man in the middle.

5. When Alice sends her encrypted request, since Bob is actually connected to Mallory, and Mallory has a hanging request, Alice's request will be ignored, but Mallory can send a request with Alice's cookies!

Full request that Bob received:

GET /pizza?toppings=pineapple; address=malloryaddress HTTP/1.1
X-Ignore-This:GET /pizza?toppings=pepperoni; address=aliceaddress HTTP/1.1
Cookie: alicecookie

Which is effectively:

GET /pizza?toppings=pineapple; address=malloryaddress HTTP/1.1
Cookie: alicecookie

** Mallory just earned herself a FREE pizza!!

Possible things to look out for some of the encryption schemes they might ask

- AES needs securely generated key and an randomly generated IV to be secure. If none are these are met = insecure
- In CBC decryption, an altered/corrupted ciphertext block affects the corresponding plaintext block and the following plaintext block, but the rest of the blocks remain intact, i.e. the effect of a block re-ordering attack is localized
→ Add some extra info about the block order so that reordering attack can be easily detected. Attack can go undetected if the plaintext represents an image or video, where a small localized modification may not be sufficiently noticeable by the receiver
- keystream for stream cipher needs a different iv for the key else same key stream

Birthday Paradox Question

(Birthday paradox:) The are about 100,000 hair glands on a human's scalp, and different persons have different numbers of hair glands. Suppose there are 1,000 undergraduate students in SoC. Are the chances high that there exist two SoC students with the same number of hair glands?

For 2 students to have the same number of hair glands, we compare every 1000 undergrad with another undergrad, so (1000 * 999) / 2 =499500 comparisons.
Suppose there are 100 000 possible number of hair glands, the probability of 2 people having different number of hair glands is

1 - (1 / 100 000) = 0.99999.
After 499500 comparisons, there is a 0.677% chance 2 people dont have same number of hair glands, i.e. there is a 99.3% chance 2 people have same number of hair glands

Probability that 2 students share the same # of hair glands

$= 1 - P(\textbf{All}\ Different\ number\ of\ hairglands)$

$= 1 - P(Different\ number\ of\ hairglands)^{Number\ of\ Pairs}$

→ M is number of students, T is number of hair glands

Probability that 2 students share the same # of hair glands

$= 1 - P(Different\ number\ of\ hairglands)^{\frac{M(M-1)}{2}}$

$= 1 - (1 - \frac{1}{T})^{\frac{M(M-1)}{2}}$ // Exact Value

$\approx 1 - e^{-\frac{M^2}{2T}}$

→ Supposed the digest of a hash is 80 bits: T= 2^{80}
→ If attacker randomly generates 2^{41} messages, (M=2^{41}=$2^{1+80*0.5}$), then M > 1.17 $T^{0.5}$
→ If M > 1.17 $T^{0.5}$, with probability > 0.5, two of them give same digest

Just apply this formula!

$M > 1.17\sqrt{T}$

Having trouble remembering which is M and T?
- The bigger number usually goes to T
- The smaller number usually goes to M
- Why?

What happens when we have more students (M) than number of hair glands (T)?
Pigeon Hole principle!!

Unit measurements

(*Hint*: For simplicity, you can take 1 year ≈ 2^{25} seconds. 1K = 2^{10}, 1M = 2^{20}, 1G = 2^{30}.)

A session key is a single-use symmetric key used for encrypting all messages in one communication session.
Prove that collision-resistant is a stronger requirement than one way

8. (*Security requirements of a cryptographic hash function:*) Lecture 3 states two security requirements of a cryptographic hash function $h()$, namely *collision resistant* and *preimage resistant (one way)*. The former says that it is difficult to find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$. The latter says that, given x, it is computationally difficult to find a m such that $h(m) = x$.

Show that if a hash function h is collision resistant, then it is also one-way (i.e. *collision-resistant(h)* ⇒ *one-way(h)*).

(*Hint:* You can prove the implication statement above by showing that its contrapositive is true. That is, ¬ *one-way(h)* ⇒ ¬ *collision-resistant(h)*. This contrapositive basically says that if it is easy to invert $h()$, then it is also easy to find a collision. This can be established as follows. Suppose there exists a fast algorithm $\mathcal{A}$ that, when given x, successfully finds a m such that $h(m) = x$. Given $\mathcal{A}$, then there also exists another fast algorithm $\bar{\mathcal{A}}$ that can successfully find a collision with high probability, i.e. $\bar{\mathcal{A}}$ can find m_1, m_2 such that $h(m_1) = h(m_2)$ and $m_1 \neq m_2$. Now, suggest how we can construct $\bar{\mathcal{A}}$ from $\mathcal{A}$!)

i. *Website defacement* is an attack on a website that changes the visual appearance of the site or a webpage. Website defacement compromises the _____ of the webpage.

ii. A student plugged a USB hardware keylogger into the common desktop in the classrooms of two professors to capture their login details. In this example, the keylogger compromises the _____ of the passwords.

iii. A(n) _____ service provides assurance of the origin or delivery of data in order to protect the sender against false denial by the recipient that the data has been received, or to protect the recipient against false denial by the sender that the data has been sent.

iv. Twitter was inaccessible for several hours on Thursday morning, followed by a period of slowness and sporadic time-outs (and more outright downtime). The company is blaming an "ongoing" _____ attack but has not said anything further.

v. Instead of sending out bulk emails, _____ is by no means random – it is a highly-targeted operation.

vi. A CA issues digital certificates to entities and individuals after verifying their identity. It signs these certificates using its _____.

vii. The reason that _____ are used is that people tend to choose the same passwords, and not at all randomly.

viii. A session key is a single-use _____ used for encrypting all messages in one communication session.

ix. A(n) _____ protects against message forgery by anyone who doesn't know the secret key (shared by sender and receiver).

x. Question 2(b) serves a good example to illustrates that stream cipher in general does not preserve _____.

xi. Somebody owning a Tor exit node can sniff and modify any traffic going through it, and hence can carry out _____ attack.

xii. In the case of Enigma, Alan Turing determined that the word "one" ("eins" in German) was the most common string in the plaintext. He formulated a process, which exhaustively checked each position in the ciphertext, assuming that the plaintext was "eins". This was a classic example of _____ attack.

xiii. Nowadays, almost every website requires some form of authentication to access its features and content. With the number of websites and services rising, a centralized login system has become a necessity. _____ is now required more than ever.

xiv. A(n) _____ is a attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy.

xv. DNS poisoning/spoofing could render web server unreachable. Hence, attack on the DNS server can be viewed as a(n) _____ on the web services.

3.
(i) integrity
(ii) confidentiality
(iii) signature
(iv) Denial of Service
(v) spear phishing
(vi) private key
(vii) salt
(viii) symmetric key
(ix) MAC
(x) integrity
(xi) man-in-the-middle
(xii) frequency analysis
(xiii) Single-sign-on
(xiv) covert channel
(xv) Denial of Service

A botnet is a number of Internet-connected devices, each of which is running one or more bots. Botnets can be used to perform Distributed Denial-of-Service attacks, steal data, **send spam**, and allow the attacker to access the device and its connection.

A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. It often uses a computer network to spread itself, relying on security failures on the target computer to access it. It will use this machine as a host to scan and infect other computers

A port scan is a common technique hackers use to discover open doors or weak points in a network. A port scan attack helps cyber criminals find open ports and figure out whether they are receiving or sending data. It can also reveal whether active security devices like firewalls are being used by an organization.

a nonce is an arbitrary number that can be used just once in a cryptographic communication. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks.

Phishing attacks

Pharming - cyberattack intended to redirect a website's traffic to another fake site by installing a malicious program on the computer. Pharming can be conducted either by changing the hosts file on a victim's computer or by exploitation of a vulnerability in DNS server software.

Smishing is a phishing cybersecurity attack carried out over mobile text messaging, also known as SMS phishing.

Vishing is a cybercrime that uses the phone to steal personal confidential information from victims. Often referred to as voice phishing

Scanning is a set of procedures for identifying live hosts, ports, and services, discovering Operating system and architecture of target system, Identifying vulnerabilities and threats in the network. Network scanning is used to create a profile of the target organization.

Scanning refers to collecting more information using complex and aggressive reconnaissance techniques.

Typosquatting, also called URL hijacking, a sting site, or a fake URL, is a form of cybersquatting, and possibly brandjacking which relies on mistakes such as typos made by Internet users when inputting a website address into a web browser.

Zero day vulnerabilities - A zero-day is a computer-software vulnerability either unknown to those who should be interested in its mitigation or known and a patch has not been developed. Until the vulnerability is mitigated, hackers can exploit it to adversely affect programs, data, additional computers or a network.

Second preimage resistance: it is computationally infeasible to find any second input which has the same output as that of a specified input; i.e., given x, it is difficult to find a second preimage x′ ≠ x such that h(x) = h(x′). (collision resistant is still stronger req)

NIST – US organisation, has huge influence on cybersec scene, lays guidelines I think
NSA – National Security Agency, national-level intelligence agency of the United States Department of Defense
CVE - The Common Vulnerabilities and Exposures system provides a reference-method for publicly known information-security vulnerabilities and exposures.
OWASP- Open Web Application Security Project is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.

Cryptography backdoor - typically covert method of bypassing normal authentication or encryption in a computer, product, embedded device (e.g. a home router), or its embodiment,

Backdoors are most often used for securing remote access to a computer, or obtaining access to plaintext in cryptographic systems. From there it may be used to gain access to privileged information like passwords, corrupt or delete data on hard drives, or transfer information within autoschediastic networks.

Decryption order – ordered by legal parties to decrypt your personal info

Cryptology – Study of both encryption and decryption
Cryptography – study of encryption
Cryptanalysis - the study of analysing information systems in order to study the hidden aspects of the systems. Cryptanalysis is used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown

Mallory – Malicious attacker
Trent – neutral third party
Eve – eavesdropper

Graphical passwords – use image rather than letters for authentication
End to End encryption - a system of communication where only the communicating users can read the messages

SSO (single sign on) - authentication scheme that allows a user to log in with a single ID and password to any of several related, yet independent, software systems. True single sign-on allows the user to log in once and access services without re-entering authentication factors

Quantum rng – can produce truly random numbers
Retina scans are 70x more accurate than iris scans

Man in the middle attack - a cyberattack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other

90-bit digest, in total there are 2^{90} digests in the entire space, there are 2^{15} such digests, probability of no clash is $(1 - 1/(2^{90}))^{\text{number of hashes}}$, M is 2^{75}. So its like doing exhaustive search for 2^{75} digests, when we go more than that, we should have found a collision

Vulnerability scanning: process of identifying security weaknesses and flaws in systems and software running on them

Anonymous proxy: An anonymizer or an anonymous proxy is a tool that attempts to make activity on the Internet untraceable. It is a proxy server computer that acts as an intermediary and privacy shield between a client computer and the rest of the Internet. It accesses the Internet on the user's behalf, protecting personal information of the user by hiding the client computer's identifying information.

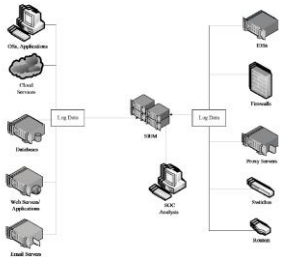
SoC (Security Operations Centre):

- A centralized unit in an organization that monitors the IT systems and deals with security issues

SIEM (Security information and Event Management):

- 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)



Privilege escalation - 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

Drive-by download - A drive-by download attack refers to the unintentional download of malicious code to your computer or mobile device that leaves you open to a cyberattack. You don't have to click on anything, press download, or open a malicious email attachment to become infected.

A drive-by download can take advantage of an app, operating system, or web browser that contains security flaws due to unsuccessful updates or lack of updates. Unlike many other types of cyberattack, a drive-by doesn't rely on the user to do anything to actively enable the attack.

Drive by downloads are designed to breach your device for one or more of the following:

1. **Hijack your device** — to build a botnet, infect other devices, or breach yours further.
2. **Spy on your activity** — to steal your online credentials, financial info, or identity.
3. **Ruin data or disable your device** — to simply cause trouble or personally harm you.

Web bug
A Web bug, also known as a Web beacon, is a file object that is placed on a Web page or in an e-mail message to monitor user behavior

Clickjacking
Clickjacking is an attack that tricks a user into clicking a webpage element which is invisible or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online

Click fraud
Click fraud is the act of illegally clicking on pay-per-click (PPC) ads to increase site revenue or to exhaust a company's advertising budget

nslookup is a network administration command-line tool for querying the Domain Name System to obtain the mapping between domain name and IP address, or other DNS records.

Wireshark – for network packet capturing, network packet analysis

traceroute and **tracert** are computer network diagnostic commands for displaying possible routes and measuring transit delays of packets across an Internet Protocol network

John the ripper - free password cracking software tool. Originally developed for the Unix operating system, it can run on fifteen different platforms

Nmap – port scanning
CVE – common vulnerabilities and exposures
Zero-day vulnerability - A zero-day is a computer-software vulnerability either unknown to those who should be interested in its mitigation or known and a patch has not been developed. Until the vulnerability is mitigated, hackers can exploit it to adversely affect programs, data, additional computers or a network

Logic bomb - A logic bomb is a piece of code intentionally inserted into a software system that will set off a malicious function when specified conditions are met
Easter egg - a joke, a message or a feature that hasn't been documented
Backdoors - malware type that negates normal authentication procedures to access a system
Insider threat - malicious threat to an organization that comes from people within the organization, such as employees, who have inside information concerning the organization's security practices, data and computer systems

Blacklist – list of users denied
Whitelist – list of users allowed
Blackhat - hackers with malicious intentions
Whitehat – hackers with good intentions
Spamhaus – company supplying realtime highly accurate threat intelligence to the Internet's major networks
CERT – Computer Emergency Response Team
SingCert – Cyber security agency of singapore

In computing, **hardening** is usually the process of securing a system by reducing its surface of vulnerability, which is larger when a system performs more functions; in principle a single-function system is more secure than a multipurpose one

Fuzzing or **fuzz testing** is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks

Even if we can invoke a set-UID-root shell, we will not get a root privilege since,
as a security protection measure, bash automatically downgrades its privilege.
This is possible, since if bash detects that it is executed in a set-UID process,
then it will immediately change its e ective user ID to the process' real user ID, thus essentially dropping the escalated privilege.

Allow non root user to have uid 0:

Thus, instead of copying `/bin/sh`, as the root, Bob can just compile `getroot.c` to crate an executable named `/games/pokemon/scores/removescores`, and enable its set-UID bit. As you can see, `getroot.c` first turns the current *set-UID* process into a *real root* process by invoking, among others, `setuid(0)`. As explained in `https://www.gnu.org/software/libc/manual/html_node/Setting-User-ID.html#Setting-User-ID`, if the calling process is privileged, `setuid(0)` sets both the real and effective user IDs of the process to 0. Once both real and effective user IDs are set to 0, a shell `/bin/sh` is subsequently invoked. As a result, when later Bob executes the planted executable as a non-root local user, he will obtain a shell running with a root privilege as planned!

Additional Notes:
Notice that the technique above is just one way of creating a backdoor. There can be other ways, such as: setting a non-root local user to have UID 0, including a non-root local user into the `sudo` group, etc.

Strcat strncat
- strncat is safe if n < the remaining characters available on the buffer. Buffer must be large enough to contain the concatenated resulting string **plus an additional null character**

strncat <cstring>

```
char * strncat ( char * destination, const char * source, size_t num );
```

Append characters from string
Appends the first *num* characters of *source* to *destination*, plus a terminating null-character.

If the length of the C string in *source* is less than *num*, only the content up to the terminating null-character is copied.

3. *(Safe/unsafe functions)*: Find out more about the following C library functions. Which usages should be avoided, and why?

```
(a) strcat (dest, source);
(b) strncat (dest, source, n);
(c) memcpy (dest, source, n);
(d) strncpy (dest, source, strlen(source));
(e) sprintf (str, f);
(f) printf ("Please key in your name:  "); gets (str);
(g) scanf ("%s", str);
(h) scanf ("%20s", str);
```

Solution

(a) Unsafe. The buffer `dest` can get overflowed since there is no limit to the number of characters concatenated into it.

(b) Safe if `n` < the remaining characters (bytes) available on the `dest` buffer. Note that the `dest` buffer must be large enough to contain the concatenated resulting string and also *an additional null character*. See `https://linux.die.net/man/3/strncat` for details.

(c) Safe if `n` ≤ the sizes of the `dest` and `src` buffers. Note that `memcpy` performs a binary-copying operation. See `http://www.cplusplus.com/reference/cstring/memcpy/` for details.

(d) Unsafe. The buffer `dest` can get overflowed since `strlen(source)` can be greater than `dest`'s length.

(e) Unsafe. The buffer `str` can get overflowed since the formatted string output can be longer than `str`'s length.

(f) Unsafe. The buffer `str` can get overflowed since there is no limit to the number of characters read and stored into it.

(g) Unsafe. The buffer `str` can get overflowed since there is no limit to the number of characters read and stored into it.

(h) Safe if the size of `str` > 20 since at most 20 characters are stored by `scanf()` into `str`. Note that a *terminating null character* is added at the end of `str`. The specified maximum input length (i.e. 20) does not include this additional terminator character. As such, `str` must be at least one character longer than the specified maximum input length. See also `https://en.wikipedia.org/wiki/Scanf_format_string`.

SOC2 is an auditing procedure that ensures your service providers securely manage your data to protect the interests of your organization and the privacy of its clients. For security-conscious businesses, SOC 2 compliance is a minimal requirement when considering a SaaS provider

A **rainbow table** is a precomputed table for caching the output of cryptographic hash functions, usually for cracking password hashes. Tables are usually used in recovering a key derivation function up to a certain length consisting of a limited set of characters

ARP Poisoning Attack / ARP attack / ARP spoofing attack

An ARP spoofing, also known as ARP poisoning, is a Man in the Middle (MitM) attack that allows attackers to intercept communication between network devices. The attack works as follows:

1. The attacker must have access to the network. They scan the network to determine the IP addresses of at least two devices—let's say these are a workstation and a router.
2. The attacker uses a spoofing tool, such as Arpspoof or Driftnet, to send out forged ARP responses.
3. The forged responses advertise that the correct MAC address for both IP addresses, belonging to the router and workstation, is the attacker's MAC address. This fools both router and workstation to connect to the attacker's machine, instead of to each other.
4. The two devices update their ARP cache entries and from that point onwards, communicate with the attacker instead of directly with each other.
5. The attacker is now secretly in the middle of all communications.

→ MITM of connection between a client and router
→ Just need to poison mac to ip mapping on all clients by spoofing as the subnet's router, once clients cache the wrong mapping, the attacker starts receiving requests, which he can view and modify. Layer 2 (Data link) attack

Format String Attack
- Attacker can use this to execute code, read the stack, or cause seg fault in the running app

The attack could be executed when the application doesn't properly validate the submitted input. In this case, if a Format String parameter, like %x, is inserted into the posted data, the string is parsed by the Format Function, and the conversion specified in the parameters is executed. However, the Format Function is expecting more arguments as input, and if these arguments are not supplied, the function could read or write the stack.

In this way, it is possible to define a well-crafted input that could change the behavior of the format function, permitting the attacker to cause denial of service or to execute arbitrary commands.

```
#include  <stdio.h>
void main(int argc, char **argv)
{
    // This line is safe
    printf("%s\n", argv[1]);

    // This line is vulnerable
    printf(argv[1]);
}
```

Safe Code

The line `printf("%s", argv[1]);` in the example is safe, if you compile the program and run it:

```
./example "Hello World %s%s%s%s%s%s"
```

The `printf` in the first line will not interpret the "%s%s%s%s%s%s%s" in the input string, and the output will be: "Hello World %s%s%s%s%s%s"

Vulnerable Code

The line `printf(argv[1]);` in the example is vulnerable, if you compile the program and run it:

```
./example "Hello World %s%s%s%s%s%s"
```

The `printf` in the second line will interpret the %s%s%s%s%s%s in the input string as a reference to string pointers, so it will try to interpret every %s as a pointer to a string, starting from the location of the buffer (probably on the Stack). At some point, it will get to an invalid address, and attempting to access it will cause the program to crash.

Firewall Example

No	*Source IP*	*Source Port*	*Dest IP*	*Dest Port*	*Protocol*	*Direction*	*Action*
1	Attackers	*	Web-server	HTTP, HTTPS	TCP	IN	Deny
2	*	*	Web-server	HTTP, HTTPS	TCP	IN	Allow
3	Web-server	HTTP, HTTPS	*	*	TCP	OUT	Allow
4	Internal	*	Banned-sites	HTTP, HTTPS	TCP	OUT	Deny
5	Internal	*	*	HTTP, HTTPS	TCP	OUT	Allow
6	*	HTTP, HTTPS	Internal	*	TCP	IN	Allow
7	*	-	*	-	ICMP	OUT	Allow
8	*	-	Internal	-	ICMP	IN	Allow
9	*	*	*	*	*	*	Deny

Proxy re-encryption schemes are cryptosystems which allow third parties to alter a ciphertext which has been encrypted for one party, so that it may be decrypted by another

Strlen() specification

(**Hint**: The `strlen()` function takes a string as an argument, and returns its length. The length is of the unsigned integer type, which can take 32 bits on a 32-bit system, and 64 bits on a 64-bit one. Hence, the type's possible values can be greater than 255.)

Hash VS Mac

The main difference is conceptual: while **hashes** are used to guarantee the integrity of data, a **MAC** guarantees integrity AND authentication.

This means that a hashcode is blindly generated from the message without any kind of external input: what you obtain is something that can be used to check if the message got any alteration during its travel.

A MAC instead uses a private key as the seed to the hash function it uses when generating the code: this should assure the receiver that, not only the message hasn't been modified, but also wh sent it is what we were expecting: otherwise an attacker couldn't know the private key used to generate the code.

LFI (Local File Inclusion)

Local file inclusion (also known as LFI) is the process of including files that are already locally present on the server, through the exploitation of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included, and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP, and others.

Nmap example

```
root@wks01:/home/vivek# nmap --top-ports 10 192.168.1.1

Starting Nmap 5.00 ( http://nmap.org ) at 2012-11-27 03:30 IST
Interesting ports on 192.168.1.1:
PORT    STATE  SERVICE
21/tcp   closed ftp
22/tcp   open   ssh
23/tcp   closed telnet
25/tcp   closed smtp
80/tcp   open   http
110/tcp  closed pop3
139/tcp  closed netbios-ssn
443/tcp  closed https
445/tcp  closed microsoft-ds
3389/tcp closed ms-term-serv
MAC Address: BC:AE:C5:C3:16:93 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.58 seconds
```

Nslookup Example

```
fabio@Computer:~$ nslookup wikipedia.com
Server:         209.222.18.222
Address:        209.222.18.222#53

Non-authoritative answer:
Name:   wikipedia.com
Address: 208.80.154.224
```

Reasons of Invalid TLS/SSL Certificate Error

- Misconfiguration of certificate. ...
- Domain mismatch. ...
- Break in the chain of trust. ...
- Incorrect date/time on your computer. ...
- Broken certificate structure. ...
- Only SHA-1. ...
- Certificate revoked. ...
- Check the date on your computer.

Forward secrecy, also known as perfect forward secrecy, is a feature of specific key agreement protocols that gives assurances that session keys will not be compromised even if long-term secrets used in the session key exchange are compromised

IPSec

In computing, **Internet Protocol Security** (**IPsec**) is a secure network protocol suite that authenticates and encrypts the packets of data to provide secure encrypted communication between two computers over an Internet Protocol network. It is used in virtual private networks (VPNs).

strlen: Returns the length of the given byte string not including null terminator

%s format string behaviour

```c
#include <stdio.h>

int main() {
  char arr1[] = {'1', '2', '3', '4', 0x00, '5'};
  char arr2[] = {'1', '2', '4', '5', 0x00};
  char arr3[] = {'1', '2', '5', 0x00, '6'};
  char arr4[] = {'1', '2', '3', '4'};
  printf("hello: %s\n", arr1);
  printf("hello: %s\n", arr2);
  printf("hello: %s\n", arr4);
  return 0;
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
hello: 1234
hello: 1245
hello: 1234125
>
```

ASCII printable code chart

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	~