CS2107 — INTRODUCTION TO INFORMATION SECURITY

PROF. CHANG EE-CHIEN

arsatis[*]

CONTENTS

[*]author: https://github.com/arsatis

1 INTRODUCTION

Lecture 1
12th January 2023

Security comprises of the **C-I-A triad**:

- **Confidentiality**: prevention of unauthorized disclosure of information.
- **Integrity**: prevention of unauthorized modification of information/processes.
- **Availability**: prevention of unauthorized withholding of info/resources.

E.g., **distributed denial of service (DDoS** attacks compromise *availability*.

Other requirements include, but are not limited to the following:

- **Anonymity** & **privacy**.
- **Non-repudiation**: assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.
- **Authenticity**.
- **Accountability**: e.g., through system logs.
- **Traitor-tracing**: e.g., having [hidden] watermarks in company printouts.
- **Black/White-listing**: e.g., firewalls.

Another notion in security is **Threat-Vulnerability-Control**: "a threat is blocked by control of a vulnerability".

- **Threat**: set of circumstances that has the potential to cause loss or harm.
- **Vulnerability**: a weakness in the system.
  - **Common vulnerabilities and exposures (CVE)** is a public repository containing discovered vulnerabilities.
  - **Zero-day vulnerabilities** are vulnerabilities which are discovered but not yet published.
- **Control**: a control, countermeasure, or security mechanism, is a means to counter threats.

1.1 *Challenges*

It is difficult to be secure because:

- **Security not considered**: many systems do not consider security during the early stages of design, where the main concerns are typically usability, cost, and performance.
- **Difficult to formulate requirements**: difficulty in scoping the appropriate security requirements, since designers may not be aware of all potential attack scenarios.

Adversarial thinking in design (i.e., assuming that there are attackers who will try to compromise the system) is often overlooked by engineers.

- **Difficult to design**: there may be many constraints that prevents a foolproof design.
- **Implementation bugs**: even if the design is secure, it is difficult to verify whether an implementation is correct.
- **Difficult to manage**: issues arising from human-in-the-loop.

A significant portion of reported vulnerabilities on CVE are considered as "implementation bugs".

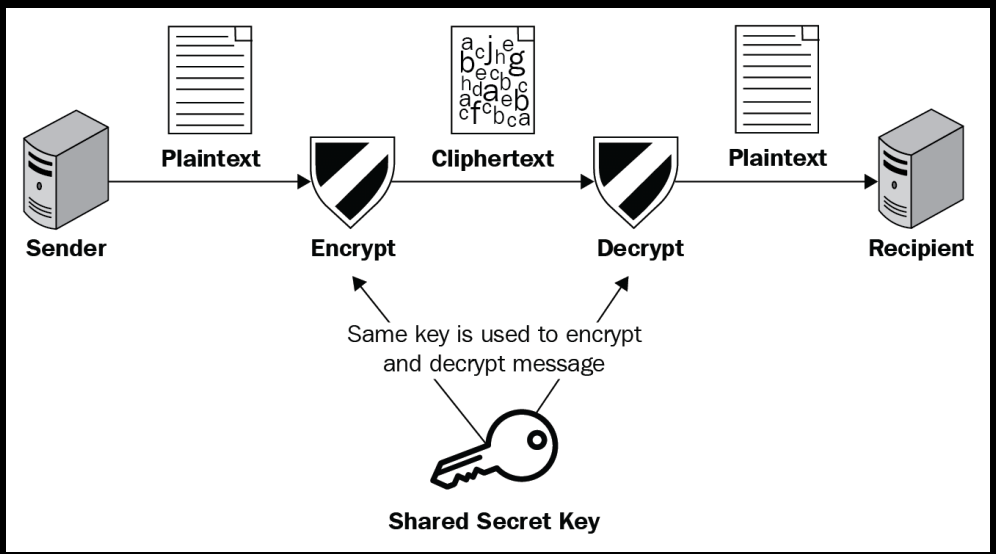
There is a trade-off of security with ease-of-use, performance, and cost.

- **Ease-of-use/Usability**: security mechanisms interfere with the working patterns which users were originally familiar with.
- **Performance**: security mechanisms consume more computing resources.
- **Cost**: security mechanisms are expensive to develop.

2 ENCRYPTION

Cryptography is the study of techniques in securing communication in the presence of adversaries who have access to the communication. **Encryption** schemes are one of the primitives in cryptography; they designed for confidentiality, and must be *secure* and *correct*.

A **symmetric-key** encryption scheme (i.e., **cipher**) consists of two algorithms: **encryption** $\mathrm{E}_k(x)$ and **decryption** $\mathrm{D}_k(x)$.



- **Correctness**: for any plaintext x and key k, $\mathrm{D}_k(\mathrm{E}_k(x)) = x$.
- **Security**: the ciphertext resembles sequences of random bytes, and it is difficult to derive useful information of the key k and the plaintext x.

2.1 *Security models*

A class of attacks is described by:

- the attacker's **knowledge** (i.e., information given) and **computing resources**.
- the attacker's **goal**.

Unless otherwise stated, we assume that the adversaries are polynomial-time machines w.r.t. the security parameter.

This is also known as an **attack model**, **adversary model**, or **security model**.

Attack models are *application-dependent*. If some attacks are successful on system S_1, whereas system S_2 can prevent all possible attacks, then S_2 is said to be *more secure than* S_1 *w.r.t. the attack model*.

An adversary may have different levels of information:

1. **Ciphertext-only attack**: adversary is given a collection of ciphertext, and may know some properties of the plaintext (e.g., an English sentence).
2. **Known plaintext attack**: adversary is given a collection of plaintext and their corresponding ciphertext.
3. **Chosen plaintext attack (CPA)**: adversary has access to an **encryption oracle**; he can feed any plaintext to the oracle and obtain the corresponding ciphertext.
4. **Chosen ciphertext attack (CCA2)**: adversary has access to a **decryption oracle**; he can choose any ciphertext and the oracle outputs the corresponding plaintext.

A **padding oracle** is a weaker form of a decryption oracle.

An adversary may have different goals:

1. **Total break**: attacker wants to find the key.
2. **Partial break**: attacker is satisfied with some minimum break (e.g., decrypting a ciphertext, determining some coarse information about the plaintext).
3. **Indistinguishability (IND)**: attacker is satisfied with the distinguishability of ciphertext (i.e., with some non-negligible probability > 0.5, the attacker is able to distinguish the ciphertext of a given plaintext from the ciphertext of another given plaintext).

2.2 *Classical ciphers*

2.2.1 Substitution cipher

- **Plaintext** & **ciphertext**: strings over a set of symbols U.
- **Key**: a **substitution table** S representing a 1-to-1 onto function from U to U.
  - **Key space**: set of all possible keys.
  - **Key space size**: total number of possible keys (in this case, 27!).
  - **Key size/length**: number of bits required to represent a key (in this case, $\log_2(27!) \approx 94$ bits).

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

Figure 1: An example of a substitution table.

- **Encryption**: given a plaintext string $X = x_1 x_2 ... x_m$ and a key S, the output is a ciphertext $E_S(X) = S(x_1)S(x_2)...S(x_m)$.
- **Decryption**: given a ciphertext string $C = c_1 c_2 ... c_n$ and a key S, the output is a plaintext $D_S(C) = S^{-1}(c_1)S^{-1}(c_2)...S^{-1}(c_n)$.

Security:

- **Exhaustive search**: in the worst case, exhaustive search needs to carry out 27! loops, which is infeasible using current compute power.
- For *known plaintext* attacks, the attacker can figure out the entries in the key by performing 1-to-1 plaintext-ciphertext matching. Therefore, the scheme is **broken/not secure under known plaintext attack**.
- For *ciphertext-only* scenarios, the scheme is vulnerable to **frequency analysis** attacks.
  - The frequencies of letters used in English are not uniform (e.g., "e" is more commonly used than "z").
  - Given a sufficiently long ciphertext, the adversary may correctly guess the plaintext by mapping the frequently occurred letters in the ciphertext to the frequent letters used in English.

  Therefore, the scheme is **not secure under ciphertext-only attack**.

For a cipher to be secure, exhaustive search must be computationally infeasible.

2.2.2 Permutation cipher

- **Plaintext** & **ciphertext**: strings over the same set of symbols.

E.g., $p = (1, 3, 5, 2, 4)$

- **Key**: a permutation $p = (p_1, p_2, ..., p_t)$.
- **Encryption**:
  1. First, group the plaintext into blocks of t characters.
  2. A secret *permutation* (i.e., the key) is then applied to each block by shifting the character at position i to position p_i.

Security:

- Under *known plaintext* attacks, permutation cipher fails miserably.
- Under *ciphertext-only* attacks, permutation cipher is also easily broken if the plaintext is an English text.

2.2.3 One time pad

- **Plaintext** & **ciphertext**: n-bit strings of the same length.
- **Key**: binary string $k_1, k_2, ..., k_n$ with n-bits.
- **Encryption**: given an n-bit plaintext $x_1, x_2, ..., x_n$ and an n-bit key $k_1, k_2, ..., k_n$, the output is a ciphertext $C = (x_1 \oplus k_1), (x_2 \oplus k_2), ..., (x_n \oplus k_n)$.
- **Decryption**: given an n-bit ciphertext $c_1, c_2, ..., c_n$ and an n-bit key $k_1, k_2, ..., k_n$, the output is a plaintext $C = (c_1 \oplus k_1), (c_2 \oplus k_2), ..., (c_n \oplus k_n)$.

Correctness:

- For any x, k, $(x \oplus k) \oplus k = x \oplus (k \oplus k) = x \oplus 0 = x$.
- Properties of XOR:
  - **Commutativity**: $A \oplus B = B \oplus A$
  - **Associativity**: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
  - **Identity**: $A \oplus 0 = A$
  - **Self-inverse**: $A \oplus A = 0$

Notice that an encryption scheme using XOR will have $E(k, p) = E(p, k)$.

Security:

- Under *known plaintext* attacks, the attacker can derive the key, but such a key is useless since it will no longer be used.
- Under *exhaustive search*, one-time-pad leaks no information of the plaintext even if the attacker has arbitrary running time → perfect secrecy.

However, the length of the key is the same as the length of the plaintext, which makes it useless in most applications.

A cryptosystem has **perfect secrecy** if for any distribution X, $\forall x, y, Pr\{X = x | Y = y\} = Pr\{X = x\}$.

In other words, for any ciphertext y and plaintext x, the chance that an attacker correctly predicts x before knowing y and after knowing y is the same.

2.3 *Modern ciphers*

Lecture 2
19th January 2023

Modern ciphers are generally designed with considerations of various attacks (e.g., known plaintext attack) and frequency analysis. Some examples include:

- **DES** (Data Encryption Standard).
- **RC4** (Rivest's Cipher 4).
- **A5/1**: used in GSM.
- **AES** (Advanced Encryption Standard).

We can quantify the security of an encryption scheme by the length of the key (e.g., DES's key length (56 bits) is too short, and is less secure w.r.t. exhaustive search as compared to other ciphers).

AES is designed for 128 bits input/output.

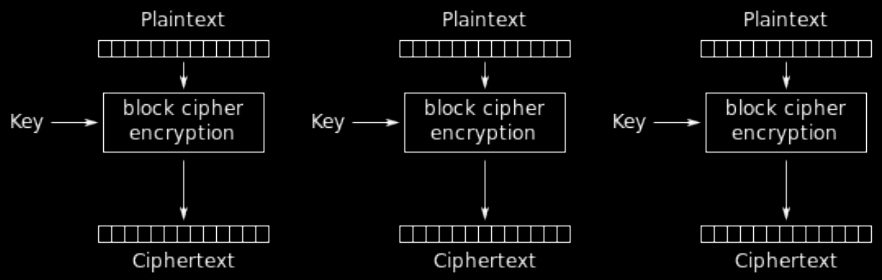
2.3.1 Block ciphers

Block ciphers (e.g., DES and AES) are designed for some fixed size input/output.

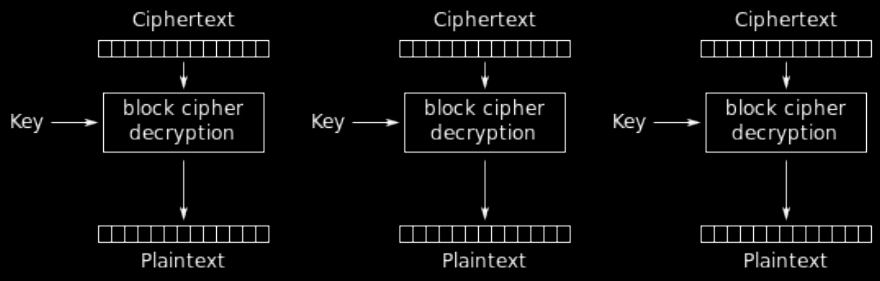
- For small plaintext (i.e., smaller than a single block), padding is used.
- For large plaintext, it is first divided into blocks before the block cipher is applied.
  - The **mode-of-operation** describes the method of extending encryption from a single block to multiple blocks.
  - Some examples of mode-of-operation include: ECB, CBC, CTR, and GCM.

2.3.2 ECB mode

Electronic Code Book (ECB) divides plaintext into blocks, and then applies the block cipher to each block, all with the same key.



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

However, ECB leaks information, since it is *deterministic*. Specifically, any two blocks which are the same (e.g., portions of an image which are entirely white) will be encrypted to the same ciphertext (e.g., the same RGB values).

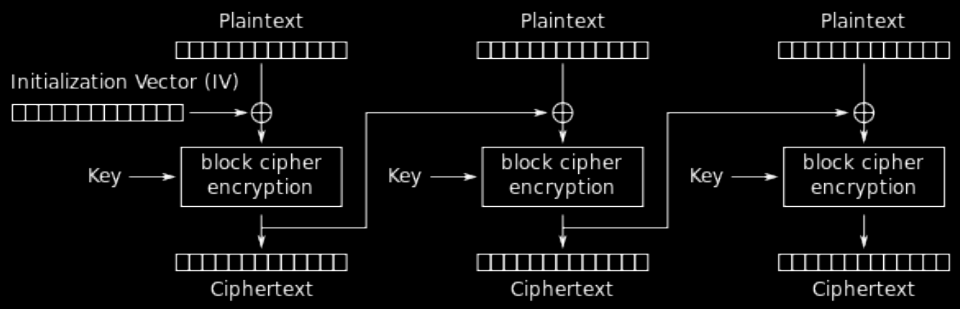
To prevent information leakage, additional mechanisms are required, e.g.:

This is rarely adopted since it would double the size of the final ciphertext.

- Use an initialization value (IV) for each block.
- Link the blocks so that two blocks with the same content will give different ciphertext.
- Stream ciphers.

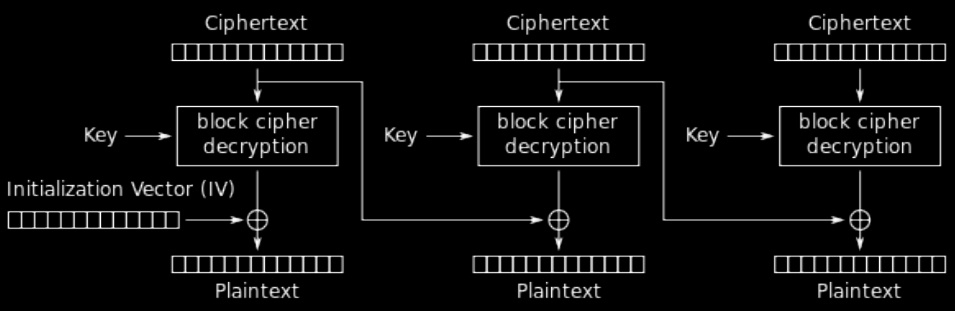
2.3.3 CBC mode

Cipher Block Chaining (CBC) chains preceding blocks along with an **initialization vector (IV)**, before applying the block cipher to each block (with the same key).



Cipher Block Chaining (CBC) mode encryption

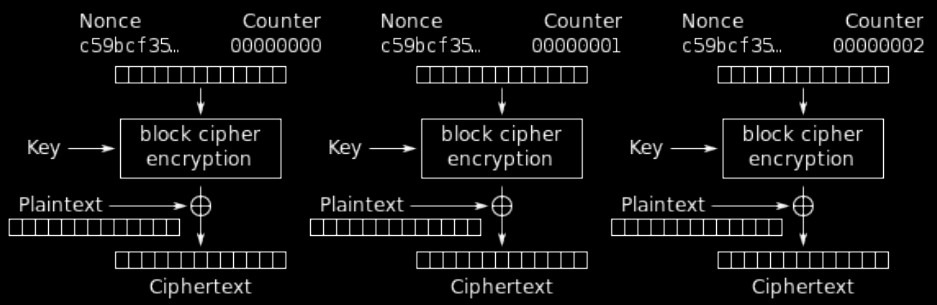
The initialization vector (IV) is an arbitrary value chosen during encryption (either randomly, using a counter, or from some other information/metadata), and is included as part of the resulting ciphertext. CBC decryption works by reversing the steps of encryption:



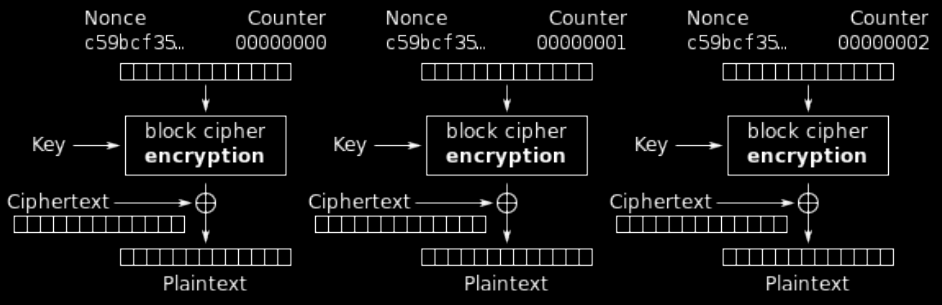
Cipher Block Chaining (CBC) mode decryption

2.3.4 CTR mode

Counter (CTR) utilizes an IV (a.k.a. nonce) combined with the counter using any invertible operation (concatenation, addition, or XOR) to produce the actual unique counter block for encryption.



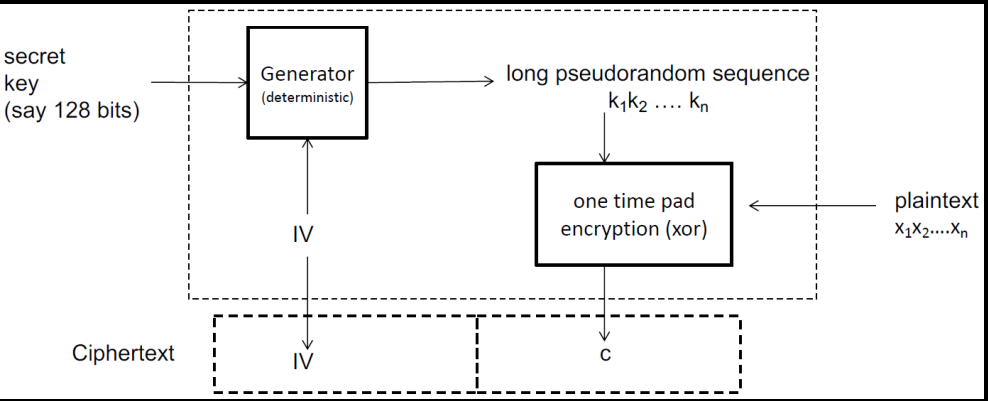
Counter (CTR) mode encryption



Counter (CTR) mode decryption

2.3.5 Stream ciphers

In a **stream cipher**, a long pseudorandom sequence is generated from the secret key together with the IV. The final ciphertext contains the IV followed by the output of the one-time-pad encryption.



For decryption, the IV is extracted from the ciphertext, and the same pseudorandom sequence can be generated from the IV and key, thus obtaining the plaintext.

The IV prevents information leakage by making an encryption probabilistic. Suppose there isn't an IV (or the IV isn't probabilistic), and the attacker obtained two ciphertexts U, V. Then, the attacker can compute:

$K = k_1k_2...k_n$ rep. the pseudorandom sequence.

$$\begin{aligned} U \oplus V &= (X \oplus K) \oplus (Y \oplus K) && \text{encryption using OTP} \\ &= (X \oplus Y) \oplus (K \oplus K) \\ &= X \oplus Y \end{aligned}$$

and obtain information about $X \oplus Y$ (which leaks a substantial amount of information about the original plaintexts X and Y).

Thus, the IV is required to be different for two different processes of encryption, in order to generate different pseudorandom sequences (and thus XOR'ing the two ciphertexts would not cancel out the pseudorandom sequences).

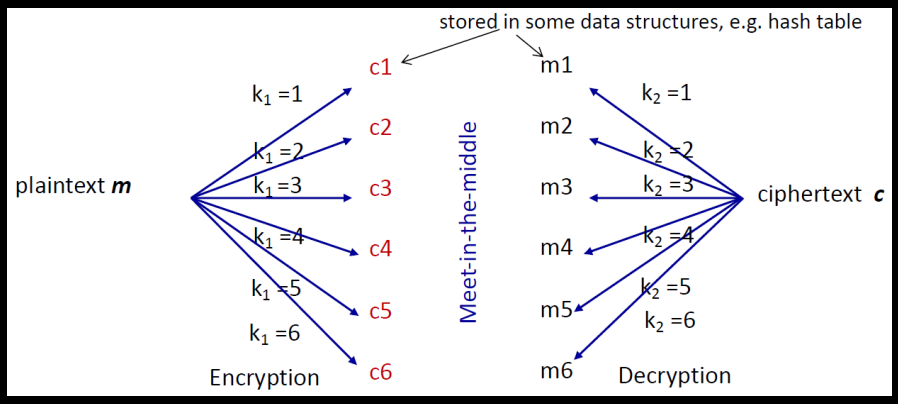
2.4 *Other attacks*

2.4.1 Meet-in-the-middle attack

While DES is not secure w.r.t. today's computing power, double DES is also insecure due to the **meet-in-the-middle attack**.

Suppose that an attacker has a pair $\{m, c = \mathrm{E}_{k1}(\mathrm{E}_{k2}(m))\}$ of plaintext and the corresponding ciphertext, and his goal is to find the keys $k1$ and $k2$. Then, he can employ the following algorithm to obtain the keys:

1. Compute two sets C and M, where C contains ciphertexts of m encrypted with all possible keys, and M contains plaintexts of c decrypted with all possible keys.
2. Find the common element in C and M.
3. From the common element, we can obtain the two keys (by tracing which keys were used to obtain it from m and c respectively).



For k-bit keys, this requires approximately 2^{k+1} operations using approximately 2^{k+1} units of storage space. We can also have a tradeoff between time and space by exhaustively selecting the last s bits of k_1 and k_2, and carrying out meet-in-the-middle attack for each selection.

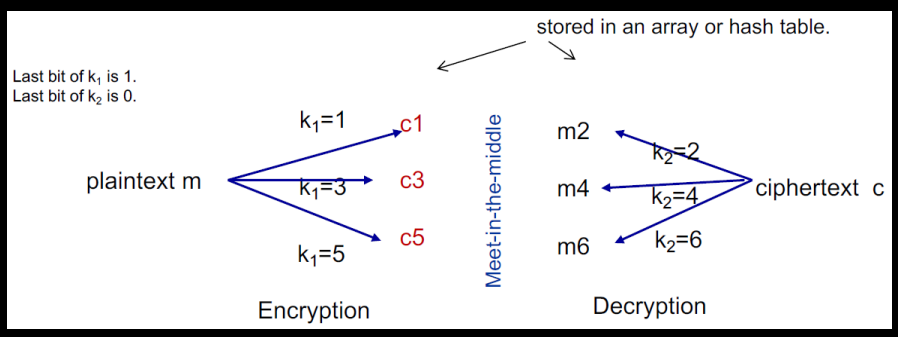


Figure 2: Meet-in-the-middle attack with exhaustive search on the last bit.

This reduces the storage requirement to 2^{k-s+1}, but the number of cryptographic operations increases to 2^{k+s+1}.

The solution to meet-in-the-middle attacks is 3DES (i.e., encrypt with DES 3 times) but using 2 keys (e.g., $c = \mathrm{E}_{k1}(\mathrm{E}_{k2}(\mathrm{E}_{k1}(m)))$).

2.4.2 Padding oracle attack

Padding oracle attack is an attack on the mode-of-operation, rather than on the cipher. A **padding oracle** is an oracle which outputs:

- YES, if the plaintext is in the correct padding format.
- NO, otherwise.

A **padding format** is a standard used to encode information regarding *the number of padded bits* in the plaintext. **PKCS#7** is one such standard, where:

- If $n > 0$ bytes are to be padded in the last block, the last n bytes will be padded with n's.
- If the last block is full, an extra block of all 0s is added.

Suppose the attacker has (IV$\|c$) (i.e., one block of IV and one block of c), access to a padding oracle, and he knows that the last block of the plaintext is padded with k bytes. To obtain the last non-padding byte of the plaintext, the attacker can perform the following steps:

1. Modify the last k bytes in the IV by XOR'ing with k values $[t_1, t_2, ..., t_k]$ such that the decrypted plaintext x would be of the form:

$$[x_1, x_2, ..., x_j, k+1, k+1, ..., k+1]$$

2. Exhaustively search for a t_0 s.t. the padding oracle returns true. Specifically, the following algorithm is used:

```
for t_0 = 0x00 to 0xFF do
    v ← IV ⊕ [0, ..., 0, t_0, t_1, ..., t_k]
    send the two-block query (v||c) to padding oracle
    if oracle returns YES then
        return (k + 1) ⊕ t_0
```

For an in-depth walkthrough of the padding oracle attack, see here.

2.5 *Cryptography pitfalls*

Some pitfalls in cryptography include:

1. **Predictable/Missing initialization vector (IV)**: predictable IVs are vulnerable to the BEAST attack.
2. **Reusing one-time-pad**.
3. **Predictable secret key generation**.
4. **Designing own cipher**.
5. **Reliance on obscurity**: hiding the design of the system in order to achieve security.
    - Typically, system designs will be exposed with time despite obscurity.
    - Systems should adhere to the **Kerckhoff's principle**: a system should be secure even if everything about the system, except the secret key, is public knowledge.

3 AUTHENTICATION

Lecture 3
26th January 2023

Authentication is the process of assuring that the communicating entity, or origin of a piece of information, is the one that it claims to be. Note that *authenticity implies integrity*.

There are two different settings where authentication is considered:

- **Entity authentication**: verifying the authenticity of entities involved in a connection (for connection-oriented communication).
    - Mechanisms: password, challenge & response, cryptographic protocol.
- **Data-origin authentication**: verifying the origin of a piece of information (for connection-less communication).
    - Mechanisms: crypto primitives such as MAC/digital signatures.

3.1 *Passwords*

Passwords are similar to secret keys in symmetric encryption. However, they are *generated by humans* and *can be remembered by humans*.

A password system typically consists of the following processes:

1. **Bootstrapping**: user and server establish a common password.
    - This can be done via either:
        (a) Server/User chooses a password, and sends it to the user/server through another communication channel.

(b) Default password.

2. **Authentication**: server authenticates an entity.
   - This can be done via either:
     (a) **Protocols**: i.e., an interaction between the user and server.
       – A protocol may be a **weak authentication** system, i.e., vulnerable to **replay attacks**.
       – It may also be a **strong authentication** system, where information sniffed during user-server interactions cannot be used to impersonate the user.
     (b) Without any interaction (e.g., sending a message along with the authentication details).
   - The server verifies the authenticity of the user by checking the username and password against the *password file*.

replay attack: information sniffed from the communicated channel can be replayed to impersonate the user.

password file: a file recording users and their corresponding passwords.

3. **Password reset**: we need to authenticate the entity before allowing password change, either via:
   (a) Provision of the old password, or
   (b) Another authentication method (e.g., security questions, recovery emails).
     - **Security questions** are a mechanism for **fallback authentication**, and is a **self-service password reset**. Its properties include:
       – Better usability: users can still login even if their passwords are lost.
       – Lower cost: eliminates the need for having a helpdesk.
       – Weaker security: attackers have another means to obtain access to the system.
     - Password reset using a **recovery email** is more secure and more widely used nowadays.

3.2 *Attacks*

3.2.1 Attack bootstrapping

The attacker may:

- Intercept the password during bootstrapping (e.g., stealing postal mail to retrieve passwords).
- Use "default" passwords (typically for IoT devices).
  – The current practice is to ship IoT devices with the default password, and require the user to change the password after the initial login.

3.2.2 Search for the password

An attacker can know whether a guess is correct by feeding it to the login session. The guess can be generated via the following ways:

- **Exhaustive search**: testing all combinations to guess the passwords.
- **Dictionary attack**: testing the passwords stored in a dictionary (e.g., containing words from the English dictionary, commonly-used passwords, etc.).
- Exhaustively test combinations of words in the dictionary & all possible capitalizations and substitutions of each character.
- **Social information**: gathering social information about the user (e.g., mobile number), and inferring the password thereafter.

This works because passwords are human-generated, thus common patterns/words will be present across passwords.

There are two main types of dictionary attacks:

- **Online dictionary** attack: the attacker must interact with the authentication system during the searching process.
- **Offline dictionary** attack: there are two main phases, namely
  1. The attacker obtains some information D about the password, via some interactions.
  2. The attacker carries out the searches using D, without interacting with the system (e.g., by testing against the hashes of words in the dictionary).

3.2.3 Steal the password

Stealing can occur in various manners:

1. **Sniffing**: e.g.,
   - **Shoulder sniffing**: a.k.a. look-over-the-shoulder attack.
   - **Sniffing the communication**: some legacy communication systems (e.g., FTP, HTTP) simply send the password over the public network without any encryption.
   - **Sniffing wireless keyboards** which employ insecure encryption methods.
   - **Side-channel attacks**: using information in the physical world, e.g., sounds made by the keyboard.
   - **Key-loggers**: a software (e.g., computer viruses) or hardware which captures keystrokes and sends the information back to the attacker.
2. **Phishing**: victims are tricked to voluntarily send their password to the attacker.

    - Typically, it tricks the user to visit a website, which is a spoofed login web.
    - The attacker would cast thousands of emails, hoping that at least one recipient would fall prey to the attack.
    - Phishing can also be carried out over phone calls (i.e., *vishing*).
3. **Spear phishing**: phishing which is target to a specific group of users (e.g., NUS staff).
4. **Cache**: when using a shared workstation, information keyed in could be cached and accessed by the next user.
5. **Insider attack**: malicious (or compromised) system administrators may attempt to steal the password file.

3.3 *Preventive measures*

Some preventive measures include:

- **Workshops** and **reminders**.
- **Embedded phishing exercises**: authorized entities send out "phishing" emails to employees.
- **Blacklisting**: organizations actively monitor for phishing sites and blacklist them; users avoid blacklisted sites using browsers or firewalls.
- Using **strong passwords**: i.e., with high *entropy*.
    - Suppose a set P contains N unique passwords. If a password is chosen randomly and uniformly from P, the entropy of the password is $\log_2 \mathrm{N}$ bits.
    - If the password is not chosen uniformly, the entropy is

$$-\sum_{i=1}^{\mathrm{N}} p_i \log_2 p_i$$

      where p_i is the probability that the i^{th} word in P is chosen.
- **Enhancing password system**:
    - To make online attacks more difficult, systems can intentionally *add delay into login session* upon failed attempts, or *lock the account* after several failed attempts.
    - To make offline attacks more difficult, a *key derivation function (KDF)* can be applied to the password.
    - Systems can also check for weak passwords when users register/change their password.
    - Systems may also require regular password changes.
- **Hashing and salting** the password file.

3.4 *ATMs*

To authenticate an ATM user, a card and a PIN is typically required.

- The card contains a magnetic strip which stores the user account id (it simplifies the input of the account id into the system).
  - Data is encoded into the magnetic strip using well-known standards. Given access to a card, anyone can copy the card by reading the information from the card, and writing it to a spoofed card.
- The PIN plays the role of a password.

An **ATM skimmer** steals a victim's account id and PIN. It consists of:

1. A **card-reader** attached on top of the existing ATM reader. This allows the attacker to spoof the ATM card.
2. A **camera** overlooking a keypad, or a **spoofed keypad** on top of the existing keypad. This allows the attacker to obtain the PIN.
3. Some means to record and transit the information back to the attacker.

Some preventive measures which could be taken include:

- **Anti-skimmer device**: a device that prevents external card readers from being attached onto the ATM.
- **Shielding** the keypad.

3.5 *Biometric authentication*

Biometric authentication uses the unique physical characteristics of a person for authentication.

1. During **enrolment**, a **template** of a user's biometric data is captured and stored. Similar to bootstrapping in password systems.
2. During **verification**, the biometric data of the user is captured and compared with the template using a *matching algorithm*, which decides to accept or reject the user.

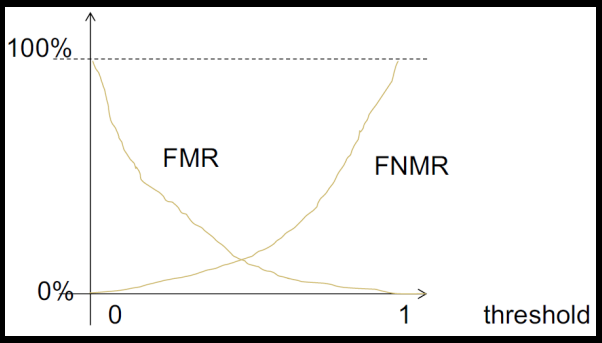
Some differences between passwords and biometric data include:

Password	Biometric data
Can be changed/revoked	Cannot be changed
Requires remembering	Does not require remembering
Zero non-matching rate	Probability of error
Users can share the password with another person	Cannot be shared

Specifically, there is inevitable noise in capturing biometric data leading to errors in matching during verification. Typically, the matching algorithm makes a decision based on some adjustable threshold (ranging from 0 to 1), which affects the **false match rate (FMR)** and **false non-match rate (FNMR)**.

$$\text{FMR} = \frac{\text{\# successful false matches}}{\text{\# attempted false matches}}$$

$$\text{FNMR} = \frac{\text{\# rejected genuine matches}}{\text{\# attempted genuine matches}}$$



Other types of error rates include:

- **Equal error rate (EER)**: rate when FNMR = FMR.
- **False-to-enroll rate (FER)**: rate of inability in capturing users' biometric data for enrolment (e.g., due to injury).
- **Failure-to-capture rate (FTC)**: rate of failure in capturing users' biometric data during authentication (e.g., due to dirty fingers).

Attacks on biometric authentication include **spoofing**: biometric data could be easily spoofed (e.g., fake fingerprints).

To prevent these attacks, biometric systems may include **liveness detection** to verify that the entity scanned by the scanner is indeed "live", instead of a spoofed material.

3.6 *n-factor authentication (nFA)*

n-factor authentication (typically 2FA) requires users to have at least two different authentication factors. Factors include:

1. Something you know: e.g., password, PIN.
2. Something you have: e.g., security token, mobile phone, ATM card.
    - An **one time password (OTP) token** is a hardware that generates a password that can only be used once. Two types of tokens include:

(a) **Time-based**: a password is generated based on a shared secret between the token and the server, and the current time interval.
(b) **Sequence-based**: an event (e.g., pressing of a button) triggers a change of the password.

OTP is typically only valid for a short period of time or specific to a transaction, thus even if the session is compromised, confidentiality of the factor is still preserved.

3. Who you are: e.g., biometric data.
4. Where you are: e.g., geographical location, IP address.
5. What you do: actions and gestures, e.g., picture password.

2-step verification:

- In 2-step verification, there are typically two communication channels.
  - The main channel is the platform used by the user.
  - The **out-of-band** channel is a separate channel (e.g., SMS, email) used for additional authentication.

Note: using an email account as an additional factor is not truly 2FA, since both authentication factors are of the type "something you know".

4 PUBLIC-KEY CRYPTOGRAPHY (PKC)

Lecture 4
2nd February 2023

A **public/asymmetric key scheme** uses different keys for encryption and decryption. The goal of PKC is confidentiality, but it is also useful for authentication. The requirements of PKC include:

- **Correctness**: with the *private key*, a user can decrypt and obtain the plaintext.
- **Security**: given the *public key* and ciphertext (without *private key*), it is difficult to determine the plaintext.

This implies that it must be difficult to get the private key from the public key.

Note that in PKC, the **encryption oracle** is always available to anyone, since anyone can encrypt using the public key.

The main advantages of PKC include:

1. Entities do not need to know each other before broadcasting public keys.
2. Only a single *secure broadcast channel* is required to distribute keys; in SKC, multiple *individual secure channels* need to be set up between each pair of nodes to communicate the shared keys.
3. Less keys are required in PKC (i.e., n public keys and n private keys) than in SKC (i.e., $\binom{n}{2} = \frac{n(n-1)}{2}$).

Popular PKC schemes include RSA, ElGamal, and Paillier.

RSA is a factorization-based encryption scheme, whereas ElGamal and Paillier are discrete-log based.

4.1 *RSA*

"Classroom" RSA is the basic form of RSA which is rarely used in practice. Practical RSA typically involves additional modifications:

- Padding: to destroy its homomorphic property.
- Choosing strong primes.
- Having a fast and secure way to generate primes.
- Having some form of IV such that encryption of the same plaintext at a different time would give different ciphertexts.
- Secure implementation to guard against side-channel attacks.

"Classroom" RSA key pair generation employs the following algorithm:

1. Randomly choose 2 large primes p, q, and compute $n = pq$.
2. Randomly choose an *encryption exponent* e s.t. $\gcd(e, \phi(n)) = 1$.
3. Find the *decryption exponent* d, where $de \equiv 1 \mod \phi(n)$.
4. Publish (n, e) as the public key and keep (n, d) as the private key.

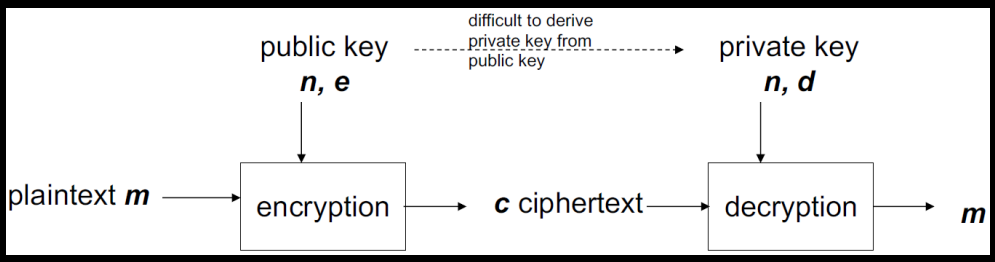
In practice, the value of e is typically fixed to the prime 65537.

$\phi(n) = (p-1)(q-1)$ is **Euler's totient function**, which counts the number of positive integers $< n$ that are coprime to n.

Subsequently, encryption and decryption can be done using the key pair:

- **Encryption**: given plaintext m, the ciphertext $c \equiv m^e \mod n$.
- **Decryption**: given ciphertext c, the plaintext $m \equiv c^d \mod n$.

Note that in "classroom" RSA, encryption and decryption still works when we swap d and e.



Correctness of RSA:

- For any positive $m < n$, and any pair of public/private keys k, we have

$$\begin{aligned} D_k(E_k(m)) &= (m^e)^d \mod n \\ &= m^{de \bmod \phi(n)} \mod n && \text{property of modulo} \quad (1) \\ &= m^1 \mod n && \text{from the definition of } d \\ &= m && \because m < n \end{aligned}$$

where line 1 above is due to the following property of modulo:

$$\forall m, n, r, \quad m^r \mod n = m^{r \bmod \phi(n)} \mod n$$

where n is the product of two primes (e.g., p, q) and $\phi(n) = (p-1)(q-1)$.

Algorithmic issues:

- **Encryption/Decryption**: there is an efficient algorithm for computing exponentiation, thus enabling efficient encryption (i.e., computing $m^e \bmod n$) and decryption (i.e., computing $c^d \bmod n$).
- **Choosing primes** (step 1): to find a random prime, we randomly pick a number and test whether it is prime. By the prime number theorem, the probability of a randomly chosen k-bit number being prime is $\approx 1/\ln 2^k$.
- **Value of** d (step 3): d can be efficiently computed from e and n using the extended Euclidean algorithm.

Security of RSA:

- The problem of getting the RSA private key from the public key is as difficult as the problem of factorizing n.
  - However, it is not known whether the problem of getting the plaintext from the ciphertext is as difficult as factorization (i.e., the **RSA problem**).
- State-of-the-art machines can perform factorization efficiently (e.g., quantum computers are able to perform *discrete log* in polynomial time), thus **post-quantum cryptography** will be needed in future, e.g.:
  - **Lattice-based cryptography**: based on the (hard) problem of finding the shortest lattice point given the basis of a lattice.
  - **Multivariate polynomial**: based on the problem of finding the modulo of a multivariate polynomial.
- Note that RSA is not necessarily better than AES.
  - RSA is a form of PKC, whereas AES is a form of SKC (the two are not directly comparable).
  - RSA is broken under quantum computing, but it is not clear whether AES is broken under quantum computing.
  - RSA encryption and decryption is significantly slower than AES.

post-quantum cryptography: PKC that are secure against quantum computers.

4.2 *Data authenticity*

Lecture 5
9th February 2023

A **(cryptographic) hash** is a function that takes an arbitrary large message as input, and outputs a fixed size *digest* (e.g., 160 bits).

For cryptographic hashes, an adversary may have 3 different goals:

1. **2nd pre-image attack**: given F, find an F′ s.t. $h(\mathrm{F}) = h(\mathrm{F}')$ and $\mathrm{F} \neq \mathrm{F}'$.
2. **One-way** (a.k.a. **pre-image attack**): given d, find an F′ s.t. $h(\mathrm{F}') = d$.
3. **Collision attack**: find F, F′ s.t. $h(\mathrm{F}) = h(\mathrm{F}')$ and $\mathrm{F} \neq \mathrm{F}'$.

Note that collision attack is easier than 2nd pre-image attack and one-way.

If an attacker is able to perform 2nd pre-image attack or one-way, he will also be able to perform collision attacks.

- **Security requirement (collision-resistance)**: it is difficult for an attacker to find two different messages m_1, m_2 such that $h(m_1) = h(m_2)$.
  - Hash algorithms that are not collision-resistant (and thus not secure) include:
    * Taking selected bits from the data.
    * CRC checksum.
  - Secure hash algorithms include:
    * SHA (SHA-1 and SHA-0 are not secure under **birthday attack**).
    * MD (MD5 and below are not secure).

  More formally, a hash function h is collision-resistant if there is no method which can significantly outperform the **birthday attack**.

  - The birthday attack is similar to exhaustive search on encryption schemes, based on the birthday problem.
  - Suppose there are M messages, and each message is tagged with a value randomly chosen from $\{1, 2, ..., T\}$, where $T = 2^{|d|}$ rep. the number of possible values for a digest of length $|d|$.

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

A hash that is collision-resistant is also **one-way** secure.

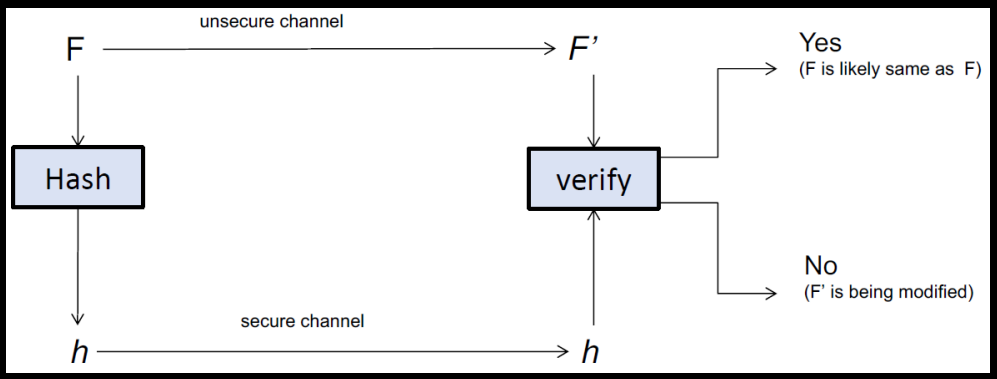
$\Pr\{\text{same}\} \approx 1 - \exp\frac{-M^2}{2T}$

Let S rep. set of k distinct elements, where each element is an n-bit binary string. Select a set T of m n-bit strings independently and randomly. Then, $\Pr\{S \cap T \neq \varnothing\} \geq 1 - 2.7^{-km2^{-n}}$.

4.2.1 Unkeyed hash

One application of unkeyed hash is verification of the authenticity of a file downloaded from an online source.

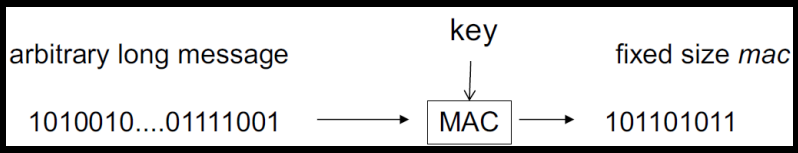
Here, we assume that the file will be downloaded through an unsecure channel (e.g., another website hosting the file), whereas the digest of the authentic file can be retrieved from a secure channel (e.g., displayed on a trusted website).



The user who downloaded the file can then compute the digest $h(F')$ for the downloaded file, and verify that it matches the digest obtained from the trusted source (i.e., $h(F) = h(F')$).

4.2.2 Symmetric key hash (mac)

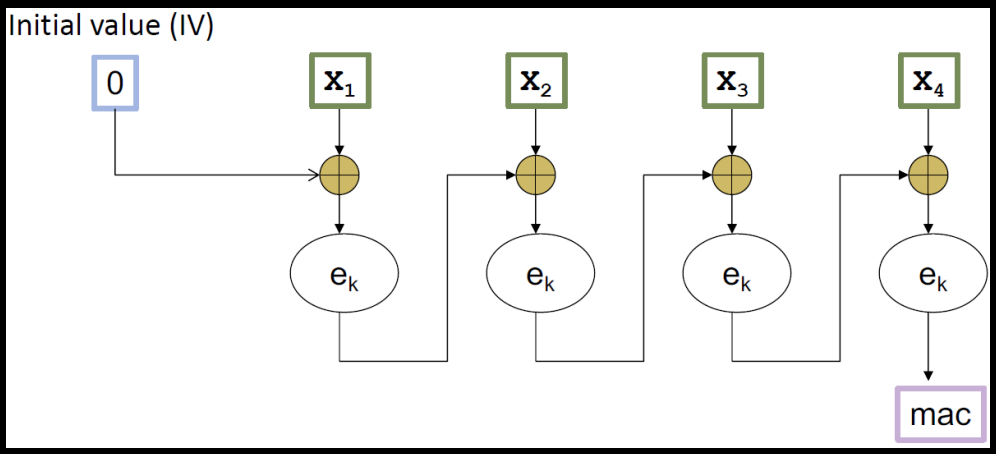
A **keyed hash** is a function that takes an arbitrary large message and a secret key as input, and outputs a fixed size **message authentication code (mac)**.



- **Security requirement (forgery)**: after seeing multiple valid pairs of messages and their corresponding mac, it is difficult for the attacker to forge the mac of a previously unseen message.

MAC can be constructed in multiple ways, including:

- **CBC-mac**: based on AES operated under CBC mode.

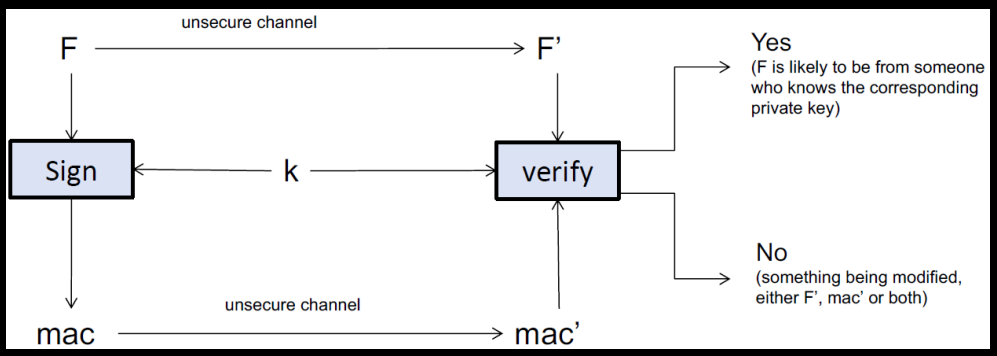


  In CBC-mac, the IV is always set to 0.
- **HMAC**: converts an existing secure hash (e.g., SHA) to a mac.

$$\text{HMAC}_k(x) = \text{SHA}\left((k \oplus opad) \,||\, \text{SHA}\left((k \oplus ipad) \,||\, x\right)\right)$$

  where:
  - $opad$ = 3636...36 (i.e., outer pad), and
  - $ipad$ = 5c5c...5c (i.e., inner pad).

One application of mac is also file authentication, where there isn't a secure channel to deliver the digest. In such scenarios, we can protect the digest with the help of mac.



Typically, the mac is appended to F, and stored as a single file or transmitted through the communication channel together. Subsequently, an entity can verify the authenticity of F using the secret key.

However, mac still has several limitations:

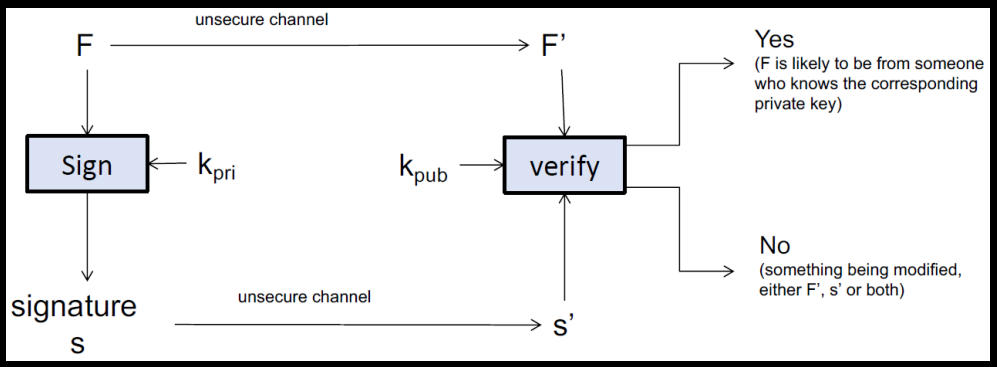
- The secret key has to be shared with the user through a secure channel.
- The scheme does not guarantee **non-repudiation**.

non-repudiation: assurance that someone cannot deny previous commitments or actions.

4.2.3 Asymmetric key hash (signature)

An asymmetric key hash takes an arbitrary large message and a private key as input, and outputs a fixed size **signature** which can be verified by the corresponding public key.

- **Security requirement**: without knowing the private key, it is difficult to forge a signature.

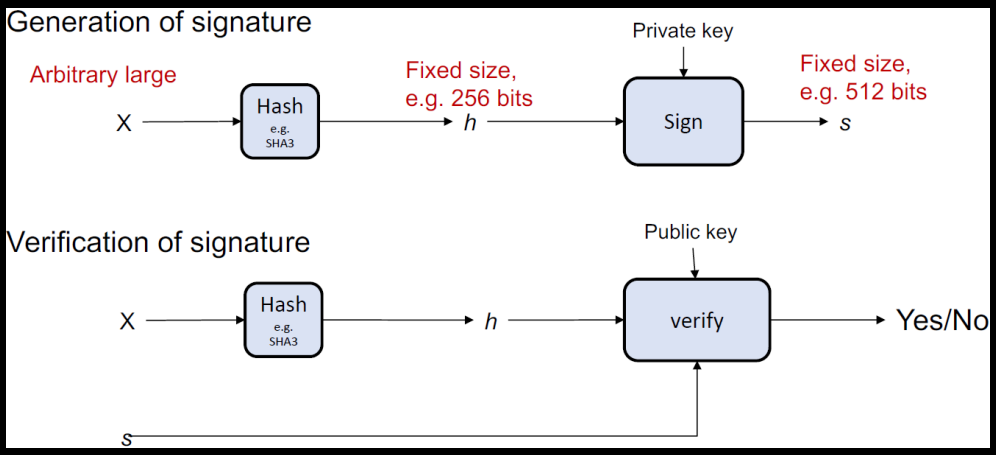


Similar to mac, the computed signature is typically appended to F and stored as a single file. Subsequently, an entity can verify the authenticity of F using the associated public key.

Signatures guarantee **non-repudiation** because only the person who knows the private key is able to sign the message.

Signature schemes typically consist of two components:

1. Unkeyed hash: for efficiency, since signing a large file is very slow.
2. Sign/Verify algorithm: typically, an asymmetric key scheme is used.



Some popular signature schemes include:

- **RSASSA**: uses RSA for the sign/verify component.
  - For signature generation, we have $s = \mathrm{E}_{\mathrm{RSA}}(k_{\mathrm{priv}}, h(\mathrm{X}))$.
  - For verification of (X, s), s is authentic if $h(\mathrm{X}) = \mathrm{D}_{\mathrm{RSA}}(k_{\mathrm{pub}}, s)$.
- **Digital signature algorithm (DSA)**: uses discrete log for security.

5 PUBLIC KEY INFRASTRUCTURE

Lecture 6
16th February 2023

Public keys need to be securely distributed, to prevent **spoofing attacks**. There are 3 mechanisms for distributing public keys:

spoofing attack: situation in which a person or program successfully identifies as another by falsifying data, to gain an illegitimate advantage.

1. **Public announcement**: owner broadcasts his public key on some existing platform (e.g., social media, email, physical name card).
   (−) Not standardized.
   (−) No systematic way to search/verify the public key.
2. Publishing on a **publicly available directory**.
   (+) Users can search for a public key associated to a "name" by querying the public directory.
   (−) No systematic way to verify the information (since anyone can post their public keys in the server).
   (−) Not everyone may trust the server.

For emails, the server could verify by sending a confirmation email to the address; but other "names" cannot be verified easily.

3. **Public key infrastructure (PKI)**: standardized system that distributes public keys, and is deployable at a large scale. It is based on two key features:
   - **Certificate**: digital document *minimally* containing the following:
     (a) **Name**: e.g., email, domain name, etc.
     (b) **Public key** of the owner.
     (c) **Time window** that this certificate is valid.
     (d) **Signature** of the CA.

     Other information which may be included are:
     - **Usage of certification**:
       * **Type** of the name: e.g., whether it is an email address or domain name.
       * Whether the name can take the role of a CA.
     - **Digest** (fingerprint): for verification without using the CA's public key.
     - **Metadata**: e.g., type of algorithm (ECC, RSA), key length, etc.
   - Chain-of-trust of **Certificate Authority (CA)**: trusted authority that manages a directory of public keys.
     - An entity can request adding its public key to the directory.
     - Anyone can send queries to search the directory.
     - The CA also has its own public-private key pair.
       * Most OS and browsers have a few **root CAs** (i.e., CAs which have their public keys pre-installed).
       * Other CAs' public keys can be added through the **chain-of-trust**, or from other sources.

     CAs are responsible for issuing certificates, and verifying that the information contained in a certificate is correct (e.g., an applicant truly owns the domain name which he requested a certificate for). A fee is required to get a certificate signed by a CA.

   Some standardization bodies for PKI include:
   - **ITU-T X.509**: specifies formats for certificates, certificate revocation lists, and a certification path validation algorithm.
   - **Public-Key Infrastructure (X.509) Working Group (PKIX)**: creates Internet standards on issues related to PKI, based on X.509 certificates.

public PKI: refers to the PKI adopted on the Internet for domain names, emails, etc.

private PKI: systems for other applications (e.g., only for use by a certain company), has its own set of "CA".

chain-of-trust: user trusts the root CA → information signed by the CA is also trusted (e.g., a name can take the role of a CA, other user's public key, etc.)

5.1 *Self-signed certificates*

Self-signed certificates are signed by their owners, to be verified using the public key listed in the certificate.

- They are convenient for the manual installation of public keys. A **root certificate** is a self-signed certificate by a CA.

5.2 *Certificate revocation*

Non-expired certificates may be revoked due to various reasons, e.g.:

- Owner's private key was compromised.
- Entity left an organization.
- Business entity closed.
- Issuing CA was compromised (least likely).

There are two different approaches to certificate revocation:

1. **Certificate Revocation List (CRL)**: CAs periodically sign and publish a revocation list.
2. **Online Certificate Status Protocol (OCSP)**: verifier can send the OCSP respondent a query to check whether a certificate is still valid.

5.3 *Attacks on PKI*

Some attacks on PKI include:

- **Implementation bugs**:
  - e.g., some browsers do not display substrings in the "name" field after null characters (i.e., \0) in the address bar, but include them when verifying the certificate.
    $\implies$ viewers may think that they are connecting to "www.nus.edu.sg" when they are actually connecting to "www.nus.edu.sg\0.hacker.com".
- **Abuse by CA**: a rogue/malicious CA can forge any certificate, and act as the man-in-the-middle of any SSL/TLS connection.
- **Social engineering**: exploits confusing naming conventions so that verifiers are tricked into using a wrong name. E.g.,
  - **Typosquatting**: using a domain name that is similar with the name of a legitimate website (e.g., "www.nus.edv.sg").
  - **Sub-domain**: creating a subdomain which is similar to the domain name of a legitimate website (e.g., "www.nus.edu.sg.111.com").

5.4 *Authentication*

Lecture 7
2nd March 2023

In **authentication**, an entity wants to convince others that they are authentic. This can be done by showing that it knows some "secrets".

- **Attack model** (**sniff-then-impersonate**): Eve can sniff the communication between the authentic Alice and Bob, and use the stolen information to impersonate Alice.
- **Method**:
  - **Weak authentication**: e.g., password. However, this is susceptible to **replay attacks**.
  - **Challenge-response**:
    * (**SKC variant**) Suppose Alice and Bob have a shared secret k, and have agreed on using mac. If Bob wants to authenticate an entity P who claims to be Alice,

      P refers to "prover".

      1. (Challenge) Bob picks a random message m and sends to P.
      2. (Response) P computes $t = \text{mac}_k(m)$ and sends to Bob.
      3. If the tag received is the mac of m, P is Alice.
         · Even if Eve has sniffed the communication and obtained multiple pairs of m, she will still fail to get the secret key k and cannot forge the mac for unseen messages.
         · Since the challenge m (a.k.a. **cryptographic nonce**) is randomly chosen, Eve cannot replay the response; this ensures **freshness** of the authentication process.

    * (**PKC variant**) Suppose Bob wants to authenticate an entity P who claims to be Alice.
      1. (Challenge) Bob picks a random message m and sends to P.
      2. (Response) P signs m with his private key and attaches a certificate.
      3. Bob verifies the certificate is valid and belongs to Alice.
      4. Bob extracts the public key from the certificate, and verifies that the signature $\text{sign}(m)$ is correct $\rightarrow$ P is Alice.
         · Similarly, Eve cannot derive Alice's private key and forge the response, and the nonce m ensures freshness.

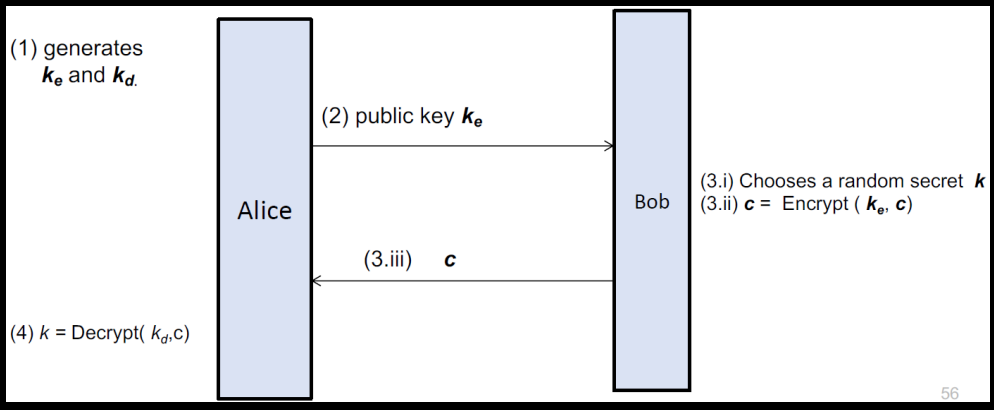
    This is a **unilateral authentication** protocol, since it only authenticates Alice. Protocols which verify both parties are called **mutual authentication**.

5.5 *Key-exchange*

In **key-exchange**, two entities want to establish a common key, which can be used to protect subsequent communication between them.

- **Attack model**: sniff, then steal session key.
- **Method**:
  - **PKC-based**:
    1. Alice generates a public-private key pair.

2. Alice sends the public key k_e to Bob.
3. Bob carries out the following:
   (a) Randomly chooses a secret k.
   (b) Encrypts k using k_e.
   (c) Sends the ciphertext c to Alice.
4. Alice uses her private key k_d to decrypt and obtain k.



By the security of PKC, the attacker cannot get any information on the plaintext from the public key and ciphertext.

– **Diffie-Hellman**: assume both Alice and Bob have agreed on two *public* parameters, a *generator* g and a large prime p.
  1. Alice randomly chooses a and computes $x = g^a \bmod p$, Bob randomly chooses b and computes $y = g^b \bmod p$.
  2. Alice sends x to Bob, and Bob sends y to Alice.
  3. Alice obtains $k = y^a \bmod p$, Bob obtains $k = x^b \bmod p$.

  Correctness:

  * $k = y^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p = x^b \bmod p$.

  Security (**computational Diffie-Hellman (CDH) assumption**):

  * Given g, p, x, y, it is computationally infeasible to find $k = g^{ab} \bmod p$.

5.6 *Authenticated key-exchange*

A key-exchange protocol assumes that the adversary can only sniff, but is not malicious. To prevent malicious attacks, **authenticated key-exchange** is required.

1. A **session key** k is first shared between the communicating parties using key-exchange.
2. Subsequently, all communication will be protected using k.

The session key ensures forward secrecy in addition to protection against Mallory.

- **Attack model**: steal session key and/or impersonate.

- **Method**:
  - Sign all interactions in a key-exchange protocol.
  - **Station-to-station protocol**:
    1. Alice randomly chooses a and computes $x = g^a \bmod p$, Bob randomly chooses b and computes $y = g^b \bmod p$, and signs y to obtain signature s.
    2. Alice sends x to Bob, and Bob sends (y, s) to Alice.
    3. Alice verifies signature s and computes $k = y^a \bmod p$, Bob computes $k = x^b \bmod p$.

    * (**SKC variant**) Both entities share a symmetric key; an entity is authentic if it can prove to the other that it knows the key.
    * (**Password**) **Password-authenticated key agreement (PAKE)** protocols are secure protocols which are secure against offline dictionary attacks even if the entropy of the password is low.
- **Security requirement**:
  - Authenticity: Alice is assured that she is communicating with an entity who knows $\mathrm{B}_{\mathrm{private}}$, and Bob is assured that he is communicating with an entity who knows $\mathrm{A}_{\mathrm{private}}$.
  - Confidentiality: attacker is unable to get the session key.

5.7 *Channel security*

Suppose there is a **public channel** which facilitates communication but is not secure. How do we secure the communication under the presence of a malicious attacker?

SSL/TLS:

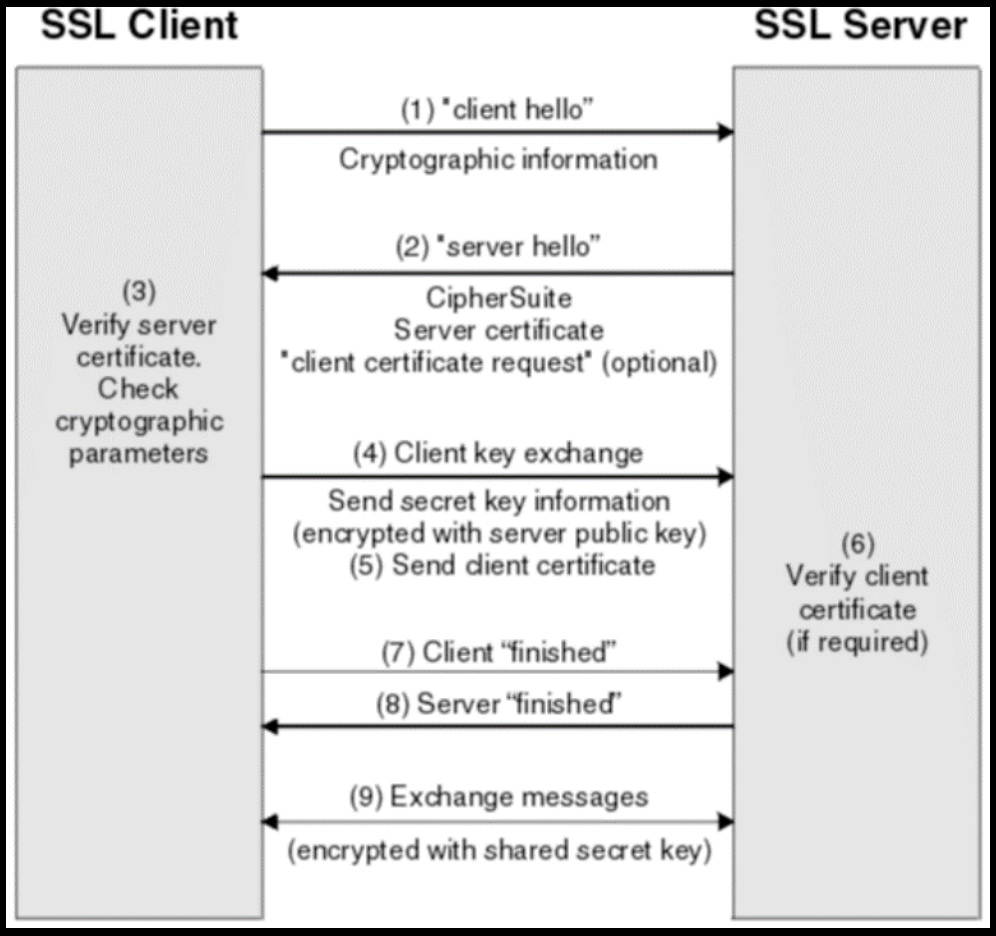
1. Using **long-term/master keys** (i.e., Bob's public-private key pair), carry out authenticated key-exchange (a.k.a. **handshake** in TLS).

   $\Longrightarrow$ Alice is convinced that she is interacting with Bob.

   $\Longrightarrow$ Both Alice and Bob have a shared **session key** not accessible by Mallory.
2. Subsequent communication protected by the session key.
   - E.g., in TLS, the actual data to be sent for the i^{th} message m_i is

     $$\mathrm{E}_k(i \| m_i) \,\|\, \mathrm{mac}_t\left(\mathrm{E}_k(i \| m_i)\right)$$

     where i rep. the sequence number.



Notes:

- SSL is the predecessor of TLS.
- HTTPS is built on top of TLS.

6 NETWORK SECURITY

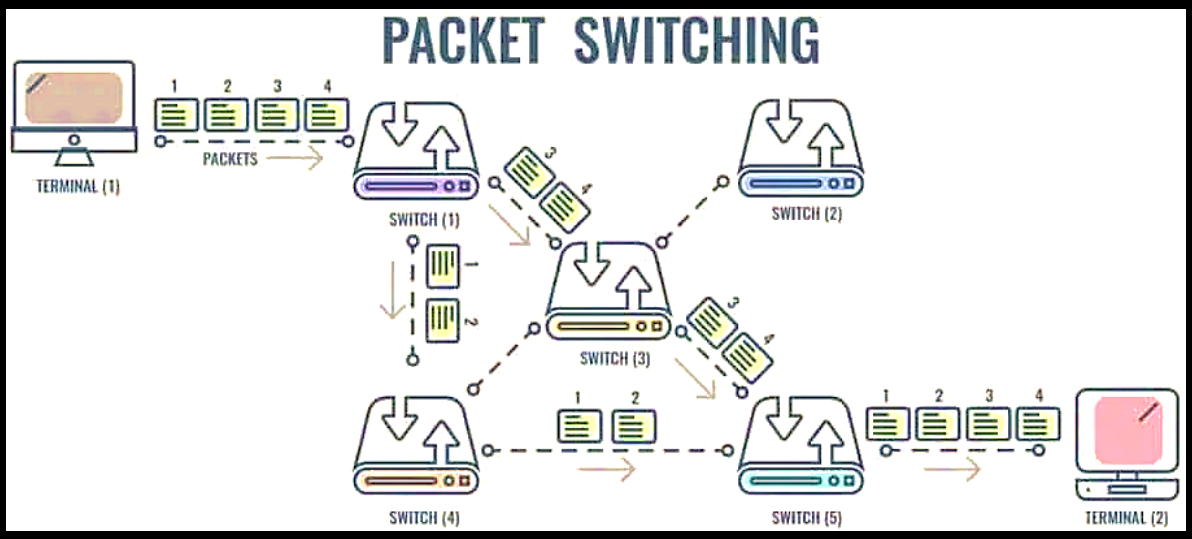
Lecture 8
16th March 2023

6.1 *Computer networks*

Computer networks establish communicating connections between entities. To share networking resources and enhance robustness, **packet switching** is employed:

1. Messages are broken into *packets/frames*.
2. Messages route via multiple switches and routers.
    - When accessing a website, the data goes through multiple hops via intermediate nodes.
    - To facilitate routing, intermediate nodes read routing/header data, and may also modify them (e.g., add hop count, translate address).
    - `traceroute` can be used to see the hops.

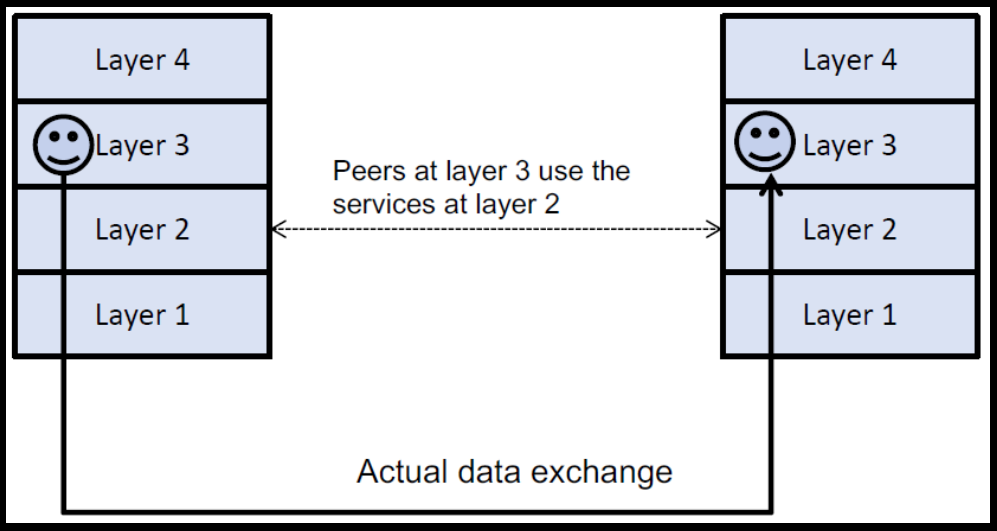
Intermediate nodes could be owned by different third parties, e.g., Internet service provider (ISP), company's firewall, etc.



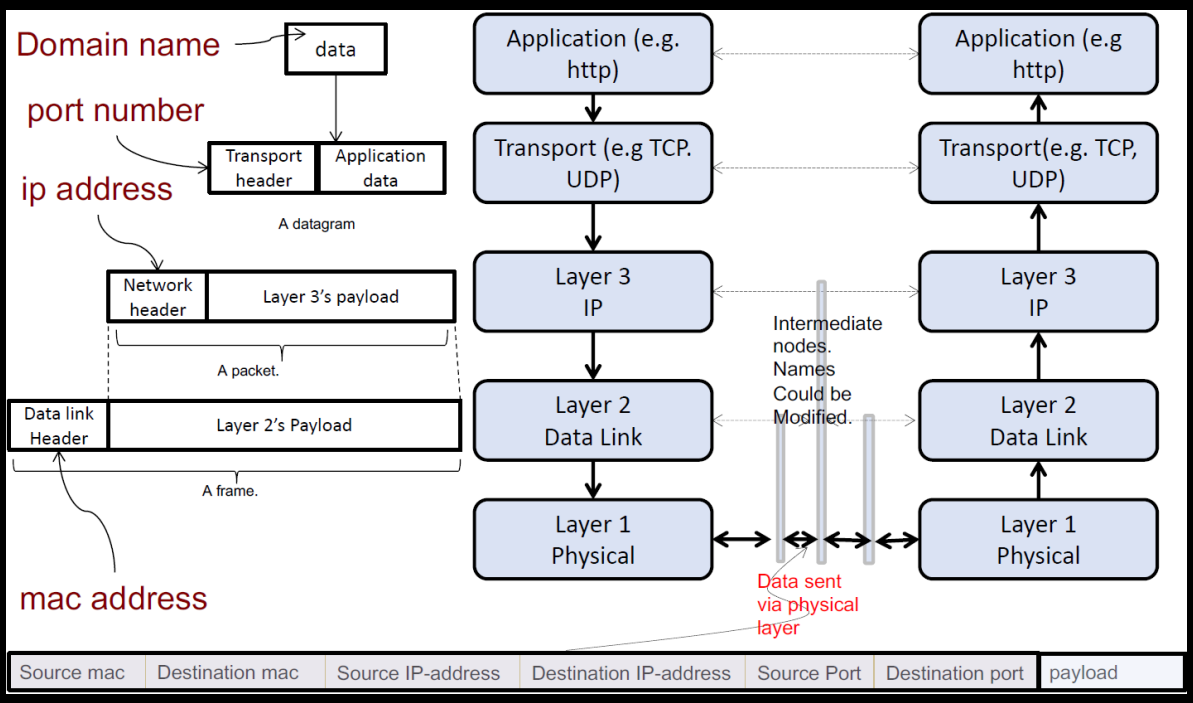
To handle the complex structure of networks, network protocols are abstracted as layers.

5. **Application** layer: in charge of domain names, e.g., https.
4. **Transport** layer: in charge of ports, e.g., TCP, UDP.
3. **Network** layer: in charge of IP addresses.
2. **Data link** layer: in charge of mac addresses.
1. **Physical** layer.

- A node has different names in different layers (e.g., google.com, port 80, 74.125.24.102, 10:12:A3:44:55:61).
- Layer N – 1 provides a virtual channel for entities in layer N.
- The *peer entities* in layer N communicate using the virtual channel in layer N – 1.



1. When a layer N channel is invoked to send a message m, the protocol in layer N transforms m into a **payload**, and generates a **header** (i.e., metadata) for the payload.
   - Each header contains at least two pieces of information:
     (a) **Source address/name** at layer N.
     (b) **Destination address/name** at layer N.
2. At the receiver end, layer N reconstructs the message m from the series of data units received.
   - At transport layer, each data unit = **datagram**.
   - At network layer, each data unit = **packet**.
   - At link layer, each data unit = **frame**.

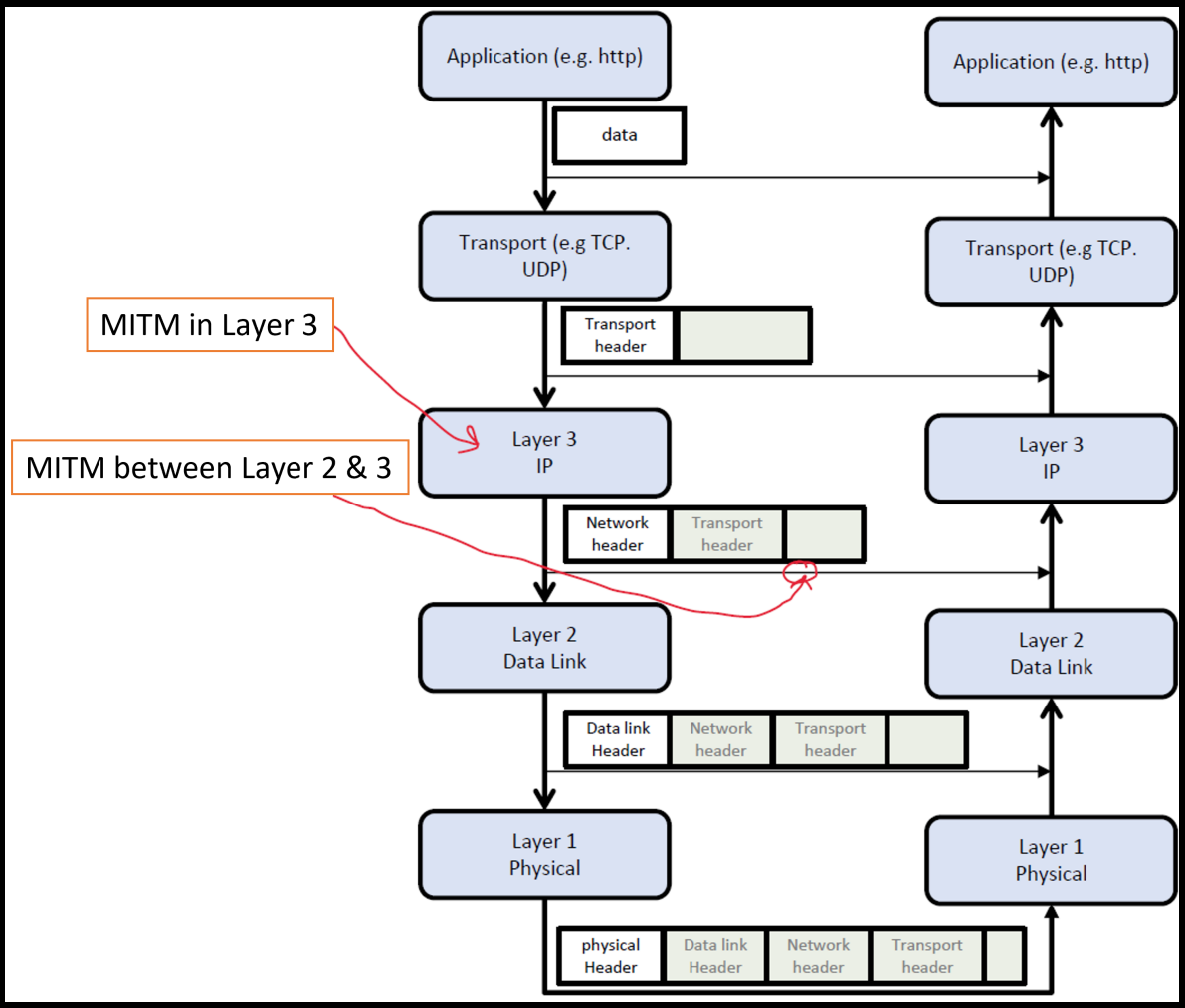


- **Networking** focuses on how to route messages via intermediate nodes.
- **Network security** focuses on the effects of having attackers among the intermediate nodes.

6.1.1 Challenges in network security

- **Intermediate nodes and layers**: there are many intermediate nodes, each handling routing-related information at a different layer → **man-in-the-middle** attacks are likely.
  - "MITM sits in layer x" implies that the attacker:
    * Can see the input to layer x.

  - Can decide what is the output of layer x.
  - Knows all the internal information stored in layer x.
- "MITM sits between layer $x-1$ and x implies that the attacker:
  - Can see the input and modify the output of layer x.
  - Does not have access to the internal data in layer x.



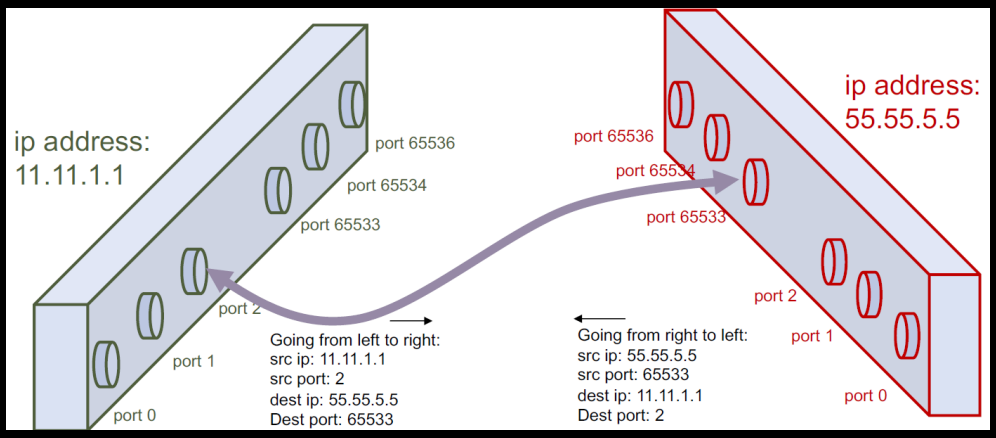
- **Legacy & security tradeoff**: the initial design of many networking protocols did not consider malicious attacks.
- **Management**: need for isolating and controlling data flows (e.g., using firewalls).

6.1.2 TCP & UDP

The transport layer and network layer are often treated as a single layer, with the address of a communicating identity being an **ip address** and a **port**.

- Each node in a network has $2^{16} - 1 = 65535$ ports.

- A communication channel between two nodes is established by connecting two ports.
- There are applications waiting to process data coming via certain ports.
  - If a node/process is listening to a port, it is a **open/listening port**.
  - Otherwise, it is a **closed port**.
    - Data sent to closed ports are dropped.



UDP/IP is **unreliable** (i.e., possibility of data loss or arriving out of order).

- To invoke UDP/IP to send messages, we can use `DatagramSend(src_port, dest_ip, dest_port)`.
  1. An IP datagram is first constructed, followed by an IP packet.
  2. The packet is then passed to the link layer.
  - The size of the message is limited to approx 65k bytes.
  - This invocation does not return a result indicating whether the destination has received the packet.

TCP/IP is **reliable**.

- To invoke TCP/IP, we can use:
  - `open_connect(src_port, dest_ip, dest_port)`: carry out some form of handshake protocol to make sure the recipient is listening.
  - `send()`: constructs IP packets and passes them to the link layer.
    - The protocol employs a mechanism for re-sending, re-ordering, and acknowledgement to ensure that the destination receives the message.
- Reliable but **not secure**: intermediate nodes along the communication route can still modify data in the header and payload (e.g., spoofing/reordering packets).

6.2 *Network attacks*

6.2.1 Name resolution

Resolution protocols help provide the corresponding name of a node at a lower layer (which is required by senders in higher layers to send over data). Some protocols include:

- **Domain name system (DNS)**: resolves domain names to IP addresses.
  1. Client sends a query to DNS server (using UDP).
  2. DNS sends answer back (using UDP).

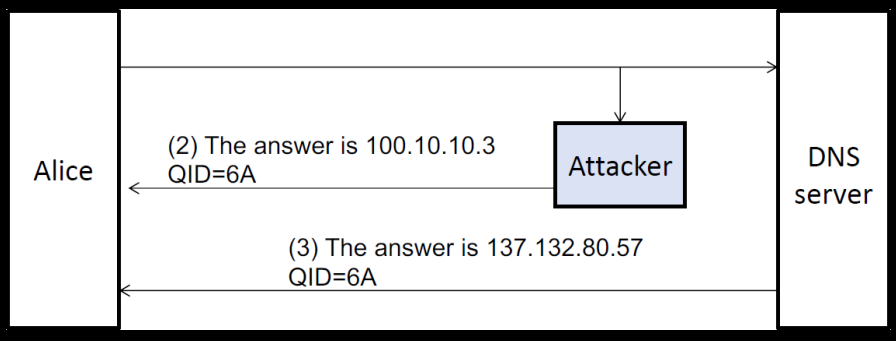
  – The client who initiates the query to the DNS server is a **resolver**.
  – The query contains a 16-bit **query ID**, and the response must contain the same QID.
  – If the address is found, we say that the domain name is **resolved**.
  – `nslookup` can be used to resolve domain names.

  With unprotected wifi, attackers who sit below the physical layer:

  – Can sniff data from the communication channel.
  – Can inject spoofed data into the communication channel.
  – Cannot remove/modify data sent by users.

  Attackers can perform **DNS spoofing**, which involves the following:

  1. User queries DNS server for the IP address of a valid website.
  2. Attacker sniffs and quickly spoofs a reply with the same QID.
  3. DNS server also sends a reply, but much later than the attacker.
  4. User takes the first reply as answer, and connects to the malicious IP address.



  Hence, a DNS server can be a **single point of failure** of the network. **Denial of service (DoS)** attacks on a web service can target the DNS server instead of the web server directly, since the web service will not be reachable when the DNS server is downed.
- **Address resolution protocol (ARP)**: resolves IP addresses to mac addresses.

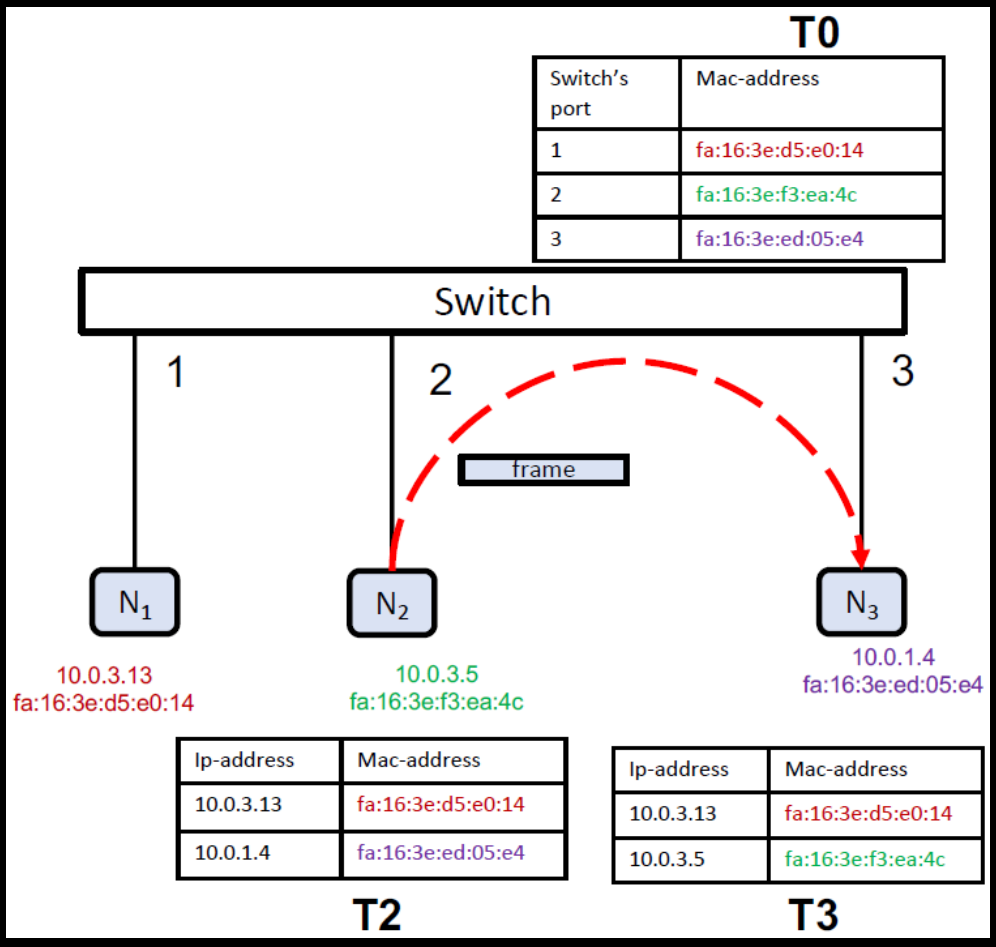
6.2.2 ARP poisoning

Lecture 9
23rd March 2023

Wifi **routers** often also serve as **switches**, and they contain gateways which perform routing.

- Switches connect two ports to each other, based on mac addresses.
- Switches keep a table which associate ports to mac addresses.

Resolution of IP address to mac address is done by nodes (e.g., desktop, phone); each node keeps an ARP table that associates IP addresses to mac addresses.

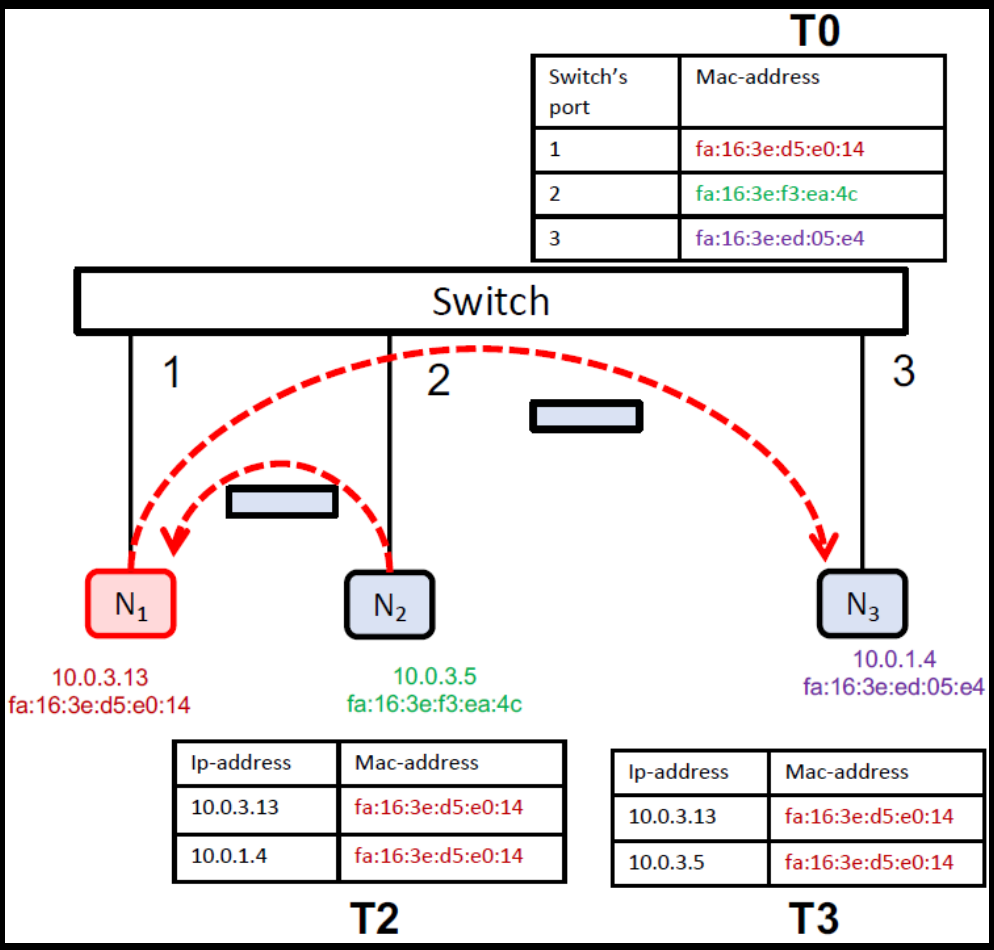


- Under normal circumstances, when a node N_i wants to send a packet to IP address a_{ip},
  1. N_i looks up the table T_i, and resolve a_{ip} to the mac adddress a_{mac}.
  2. N_i sends the frame to the switch, specifying the destination a_{mac}.
  3. Switch looks up its table, and redirects the frame to the corresponding port.

- If T_i does not have information about some IP address,
  1. N_i broadcasts a query or asks a specific node for the information.
  2. When N_i receives the answer, it updates T_i.
- It is also possible for a node to broadcast its own mac address, even though nobody asked for it.
  - E.g., when a laptop connects to a router for the first time.

ARP poisoning is an attack that modifies nodes' tables to gain MITM access. Suppose N_1 in the figure above is an attacker, then:

1. N_1 informs N_2 that the mac address of N_3 is its own address.
2. N_1 also informs N_3 that the mac address of N_2 is its own address.
3. After the tables are poisoned, all frames will be sent to N_1.
   - N_1 can relay the frames, or modify the frames before relaying.
   - N_1 becomes MITM in layer 2.



6.2.3 Denial of Service (DoS)

Denial of Service (DoS) is the prevention of authorized access to resources, or the delaying of time-critical operations.

- DoS is an attack on **availability**.
- Most DoS attacks simply flood the victims with overwhelming amounts of requests/data.
  - For DoS to be effective, large numbers of attackers are typically required (i.e., **Distributed Denial of Service (DDoS)**).

Availability: property of being accessible and usable upon demand by an authorized entity.

Reflection attacks are a type of DoS where attackers send requests to intermediate nodes, which in turn send overwhelming traffic to the victim → more difficult to trace.

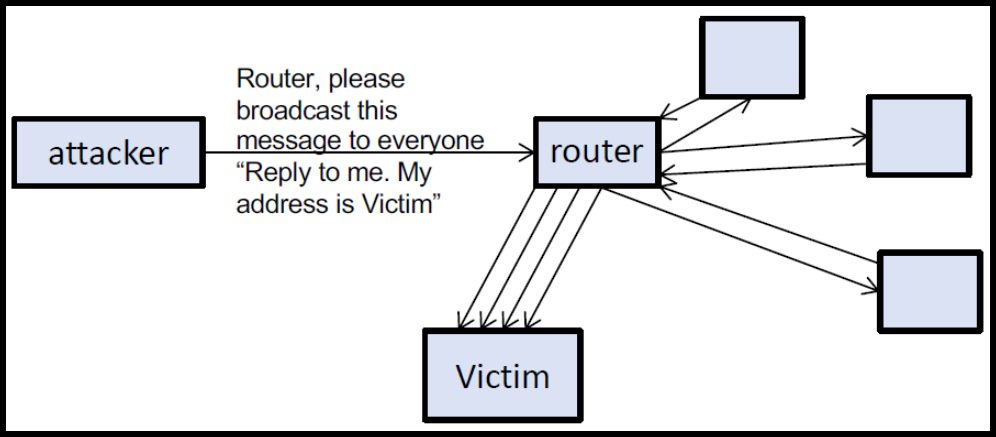
- **Amplification attacks** involve a single request triggering multiple responses from intermediate nodes.
- The **amplification factor** can be calculated as:

$$\frac{\text{size of traffic received by victim}}{\text{size of traffic sent by attacker}}$$

- Example of reflection attack using ICMP:
  1. Attacker sends the request `ICMP PING` to a router (i.e., instructing the router to broadcast this request), and the source IP address of this request is spoofed with the victim's IP address.
  2. The router broadcasts this request.
  3. Each entity who has received this request replies to it by sending an `echo reply` to the victim.
  4. The victim's network is overwhelmed with `echo reply`.

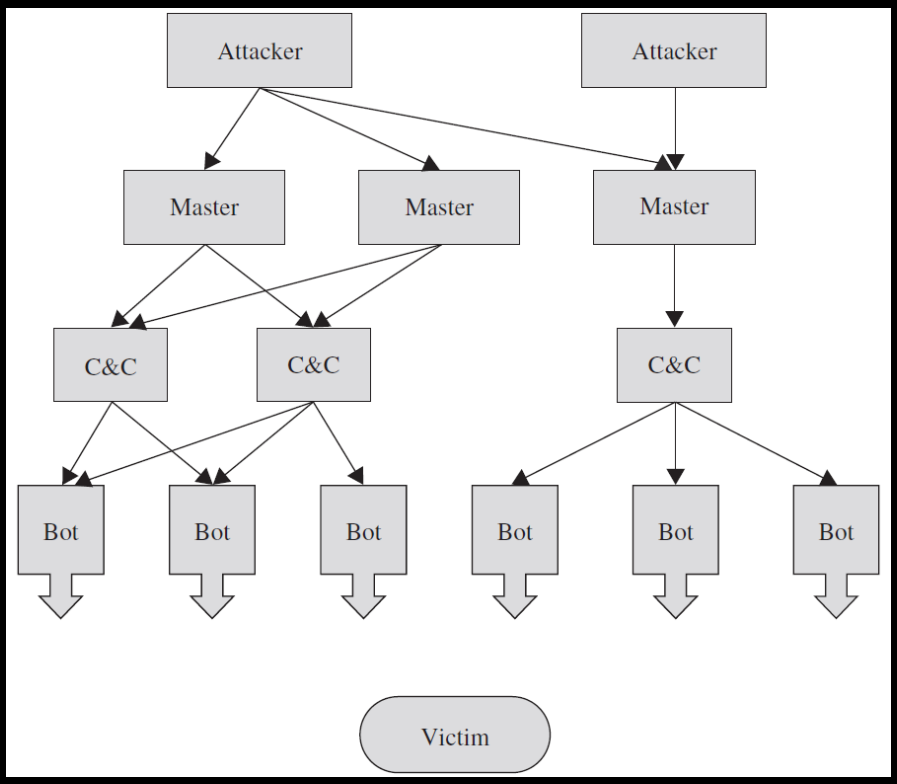
Note: this attack is no longer effective, since most routers are now configured not to broadcast by default.

Reflection attacks can also be carried out using DNS.



Botnets:

- A **bot**/zombie is a compromised machine.
- A **botnet**/zombie army is a large collection of connected bots, communicating via covert channels.
  - A botnet has a *command-and-control mechanism*; it can be controlled by an individual to carry out DDoS.
  - Often, communication is not directly sent to individual bots; instead, hierarchies are used to prevent revealing the location of the attacker.



  - Botnets can be used for DDoS flooding, vulnerability scanning, anonymizing HTTP proxy, email address harvesting, and cipher breaking.

6.3 *Useful tools*

- **Wireshark** listens to interactions between the OS and the network card driver.
  - It typically captures data in the **link layer**.
  - Modifications made by the network card may not be captured by Wireshark.
- **Nmap** can be used for **port scanning**, i.e., probing a host/server to identify open ports.

- Port scanners are useful for network adminstrators to scan for vulnerabilities.
- Port scanners can be used by attackers to detect open ports, since attackers cannot feed malicious data to closed ports.

6.4 *Securing the communication channel*

Security protocols can be used to protect different layers of the communication system.

- A security protocol that protects layer k will protect information in that layer and above.
- However, it is not feasible to only have protection at the lowest layer.
  $\because$ Intermediate nodes (e.g., routers) need to access some information in higher layers.
  $\implies$ Malicious intermediate nodes could be a MITM in higher layers.

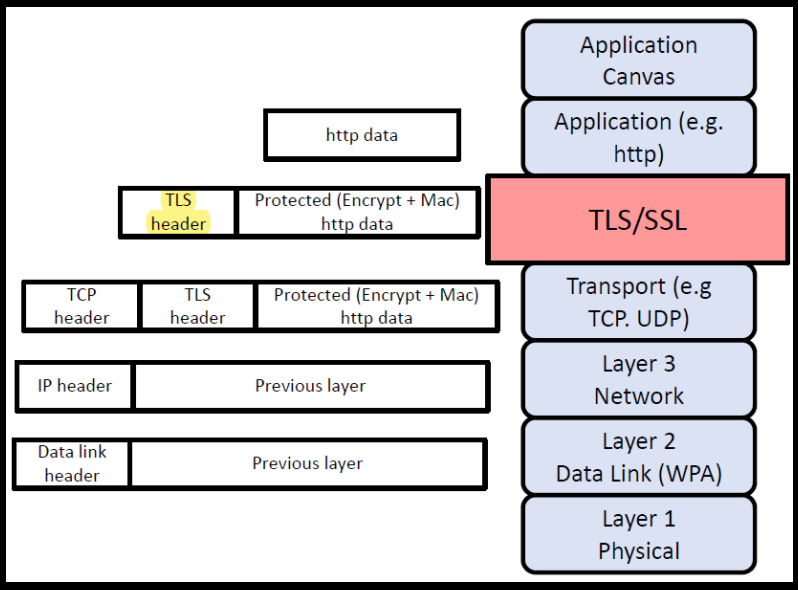
Typical implementation: SSL/TLS + WPA2.

Some popular protocols include:

- **SSL/TLS**:
  - Sits on top of the **transport layer**.
  - When an application (e.g., browser, email agent) wants to send data to the other end point,
    1. It passes the data and address (i.e., IP address + port) to TLS.
    2. TLS protects the data using encryption (confidentiality) and mac (authenticity).
    3. TLS instructs the transport layer to send the protected data.

SSL/TLS scans can be done with *SSL Labs* to check for known vulnerabilities of a server.

Note that TLS adds a header on top of the protected data.

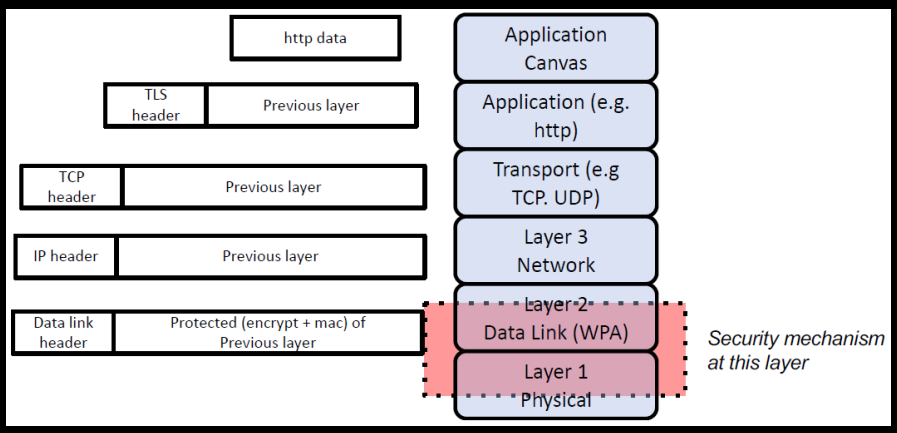


  - At the server,
    1. The transport layer passes the protected data to TLS.
    2. TLS decrypts the data, and verifies integrity using the mac.
    3. TLS passes the decrypted data to the application.

- **Wifi Protected Access II (WPA2)**: commonly employed in home Wifi access points.

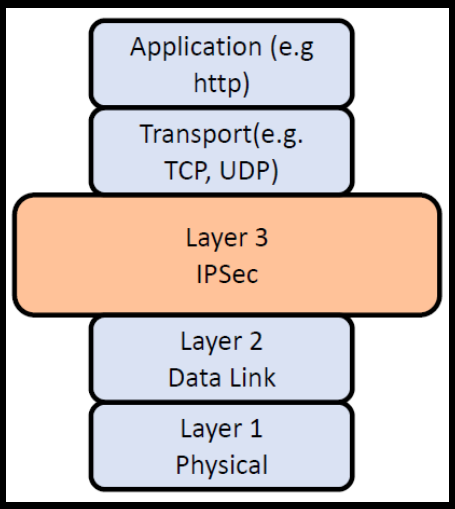
  - Provides protection to link and physical layers; not all information in the link layer is protected (specifically, the mac address is not encrypted).

After an attacker joins a network, he would be able to monitor the network to identify IP and mac addresses of nodes in the network.



- **IPSec**:

  - Provides integrity/authenticity protection of IP address, but not confidentiality.
  - MITM are unable to spoof the source IP address, but can learn the source and destination IP address of the sniffed packets.



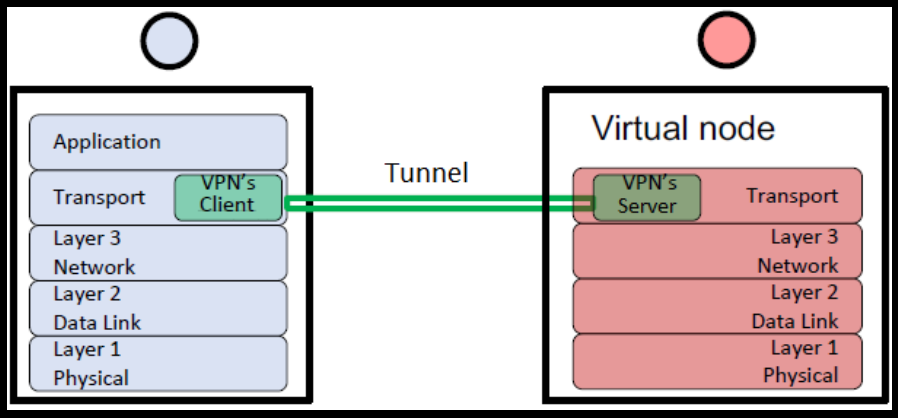
  - However, IPSec is expensive to implement and difficult to deploy, because the program has to modify the OS.

6.5 *VPN*

Using VPN, an entity X can "virtually" become a node in the network. Other entities who communicate with X would view X as a node in the network.

Conceptually, this is achieved as follows:

0. Organization Y has a VPN server, entity X has installed a VPN client.
    - The VPN client sits in some layer (commonly application, transport, or network; less commonly data link).
    - VPN that sit in the application layer are *application specific* (e.g., a proxy that carries out Google search, video streaming, etc.).
1. VPN client and server establishes a connection (called a **tunnel**).
    - The tunnel can be established in many ways, e.g., using SSL/TLS or IPSec.
2. When X communicates with another entity Z, the VPN client sends the data to the VPN server via the tunnel, instead of passing the data to the layer below.
3. The VPN server sets up a virtual node with the necessary details (e.g., IP address, mac address, etc.), and passes the received data to the transport layer of the virtual node.
4. From Z's point of view, Z is communicating with the virtual node having an IP address of organization Y.



6.6 *Access control tools*

Consider the computer network in an organization:

- Some nodes contain more sensitive information than others.
- Some nodes are more secure, whereas other nodes operate in a more hostile environment.

- Certain protocols do not have protection mechanisms (e.g., DNS, ARP, network printer, etc.); there is a need to prevent attackers from accessing such protocols.

Thus, there is a need to divide the computer network into segments and deny unnecessary access by applying the following principles:

- **Principle of least privilege**: in a particular abstraction layer of a computing environment, every module (e.g., a process, user, or program) must be able to access only the information and resources that are necessary for its legitimate purpose.
- **Compartmentalization**: information should be confined within compartments.

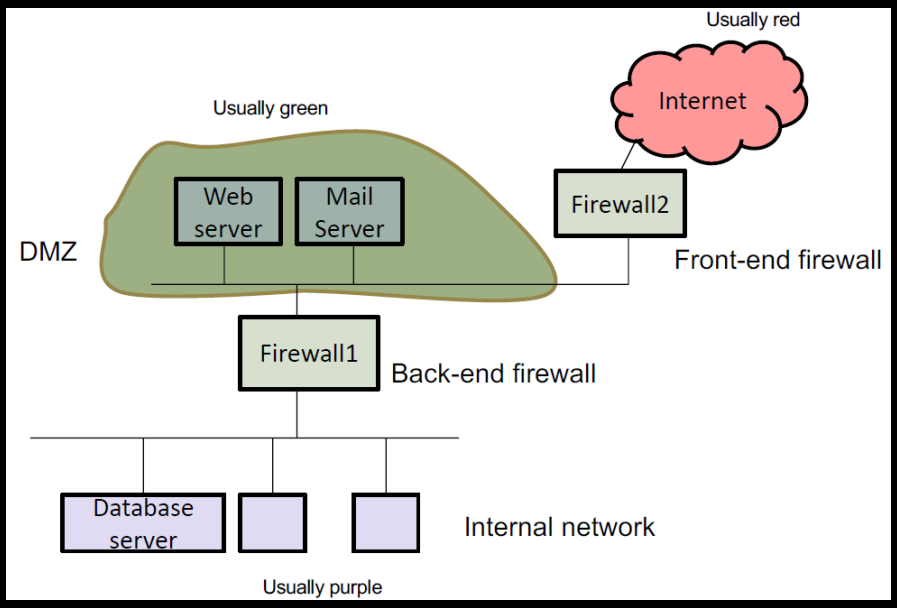
Some tools to control access to networks include **firewalls** and **intrusion detection systems (IDS)**.

6.6.1 Firewalls

A **firewall** controls what traffic is allowed to enter the network (i.e., **ingress** filtering) or leave the network (i.e., **egress** filtering).

Demilitarized zone (DMZ): refers to a subnetwork that exposes the organization's external service to (untrusted) internet.

- Firewalls are devices or programs that control the flow of network traffic between networks or hosts that employ differing security postures.



  - E.g., to access the contents of the database server, a user will have to first make a query to the web server, which will in turn query the database server, retrieve the results, and display them to the user.

- Firewalls enforce a set of rules provided by the network administrator.
  - How rules are specified differ on different devices and software.

Rule	Type	Direction	Source Address	Destination Address	Designation Port	Action
1	TCP	in	*	192.168.1.*	25	Permit
2	TCP	in	*	192.168.1.*	69	Permit
3	TCP	out	192.168.1.*	*	80	Permit
4	TCP	in	*	192.168.1.18	80	Permit
5	TCP	in	*	192.168.1.*	*	Deny
6	UDP	in	*	192.168.1.*	*	Deny

Matching condition *action*

Figure 3: Example of how firewall rules can be specified. In the examples, the rules are processed sequentially from top to bottom, and the first matching rule determines the action.

  - Some examples of firewall rules include:
    - Dropping packets with source IP address not within the organization's network.
    - **Whitelist**: drop all packets except for those specified in the whitelist.
    - **Blacklist**: accept all packets except for those specified in the blacklist.
- Firewall control can be achieved by:
  - **Packet filtering**: inspection of all packets.
    - Typically, only the TCP/IP packet's header information is inspected.
    - If the payload is inspected, it is called **deep packet inspection (DPI)**.
    - This may occur in the router, gateway/bridge, host, etc.
    - Possible actions taken after inspection include:
      - Allowing the packet to pass.
      - Dropping the packet.
      - Rejecting the packet (i.e., drop and inform the sender).
      - Log information.
      - Notifying the system admin.
      - Modifying the packet.
  - **Stateful inspection**: keeping a state of previously received packets (e.g., counting the number of connections a particular IP address has made in the past hour).
  - **Proxy**: modification of packets.

6.6.2 Intrusion detection systems (IDS)

An **intrusion detection system (IDS)** consists of a set of sensors which gather data, that are subsequently analyzed for intrusion. There are 3 types of IDS:

1. **Attack signature detection**: for attacks with specific, well-defined signatures (e.g., using certain port numbers, or certain source IP addresses).
2. **Anomaly detection**: detection of abnormal patterns (e.g., sudden surge of packets with a certain port number).
3. **Behavior-based**: anomaly detection that focuses on human behaviour (e.g., detection of users who deviate from certain profiles).

6.7 *Management*

Management is needed to monitor and adjust network characteristics. Some terminologies include:

- **Security Operations Center (SOC)**: centralized unit in an organization that monitors the IT systems and deals with security issues.
- **Security Information and Event Management (SIEM)**: software used by SOCs to monitor network and install sensors in some nodes, e.g., collects login logs and passes to the SIEM server to detect abnormal behaviours.
  - Popular SIEM systems include Splunk, and the ELK stack: Elasticsearch, Logstash, and Kibana.

The ELK stack provides centralized logging in order to identify problems with servers or applications.

Lecture 10
1st April 2023

7 ACCESS CONTROL

Definitions:

- **Principal/Subject**: an entity which wants to access an object with some operation. Specifically,
  - **Principals**: human users.
  - **Subjects**: entities in the system that operate on behalf of the principals.
- **Operations**: accesses to objects, can be classified into the following:
  - **Observe**: reads on a file.
  - **Alter**: writes, modification, or deletion of a file.
  - **Action**: execution of a program.
- **Objects**: entities that are being accessed.

  - Every object has an **owner**.
  - There are two options regarding decision of access rights to an object:
    * **Discretionary access control (DAC)**: rights are decided by the owner of the object.
    * **Mandatory access control (MAC)**: rights are decided by a system-wide policy.

Access control systems specify and enforce selective restrictions of access to a place or other resource. **Access control** provides a security perimeter which facilitates segregation of accesses.

- This segregation confines and localizes damage caused by attacks.
- Malicious activities outside of the boundary would not affect resources within the perimeter.
- Similarly, malicious activities within the boundary stays within.
- Design of security parameters/boundaries is guided by:
  - **Principle of least privilege**: a user or entity should only have access to the specific data, resources and applications needed to complete a required task.
  - **Compartmentalization**: to separate something into parts and not allow those parts to mix together.
  - **Defence in depth**: strategy that leverages multiple security measures to protect an organization's assets.
  - **Segregation of duties**: having more than one person required to complete a task.
    * The goal is to eliminate single point of failure.
    * With this, a single rogue system admin will be unable to corrupt all data.

7.1 *Access control matrix*

An **access control matrix** is used to specify the access rights of principals to objects. However, such a table will be very large and difficult to manage, thus it is seldom explicitly stored.

Instead, the access control matrix is often represented in two ways:

- **Access control list (ACL)**: stores the access rights to an object as a list.

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

For ACL, it is difficult to obtain the list of objects a particular subject has access to.

- **Capability list**: a subject is given a list of capabilities, where each capability is the access rights to an object.

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})

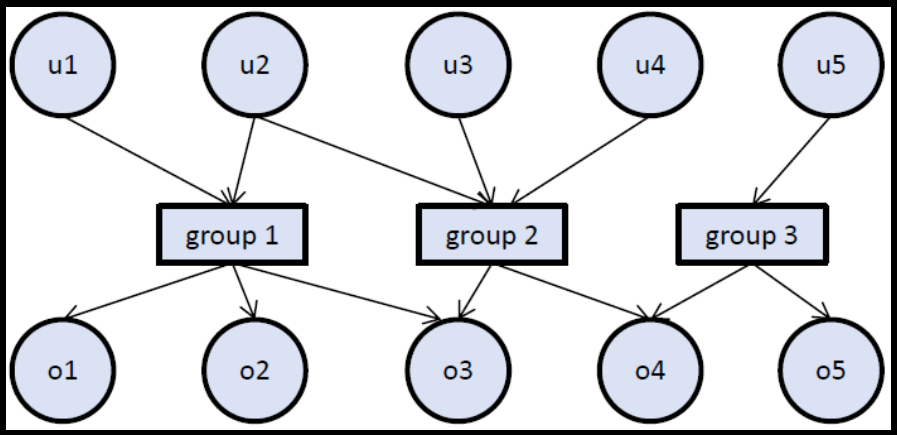
For capability list, it is difficult to get the list of subjects who have access to a particular object.

7.2 *Intermediate control*

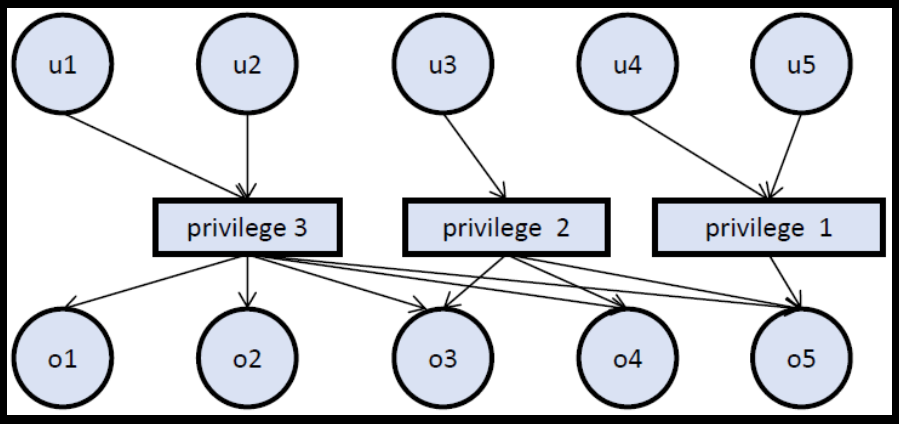
It is impractical for an owner to specify each entry in the access control matrix. One method of simplifying the representation is **intermediate control**, which groups the subjects/objects and defines access rights for each group.

- The grouping can be determined by the **role** of each subject.
  - A **role** associates with a collection of procedures.
  - In order to carry out these procedures, access rights to certain objects are required.
  - To design the access rights of a role, the *principle of least privilege* should be followed.

i.e., access rights that are not required to complete the role should not be assigned.

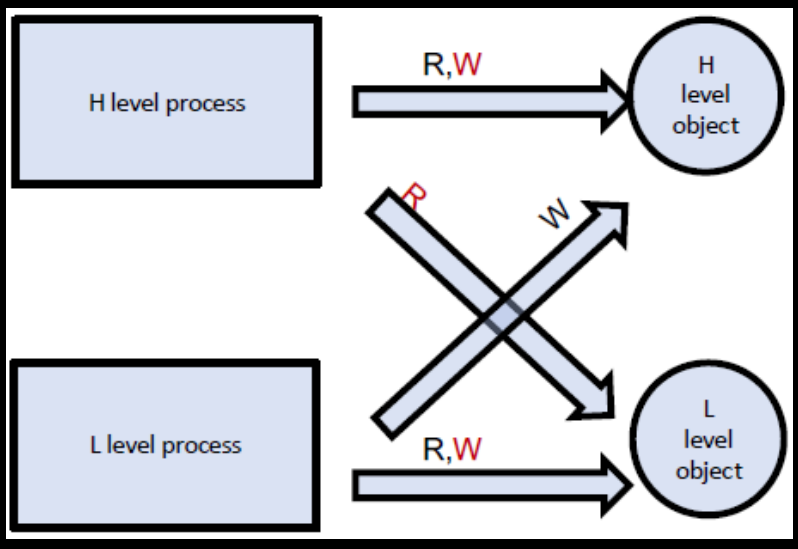


- **Privileges** can be viewed as another form of intermediate control.
  - Each privilege is represented by a number.
    - Objects with smaller numbers are more important.
    - Subjects with smaller numbers have higher privileges.
  - Subjects cannot access (i.e., both read/write) an object with a smaller number.
  - Privileges are also known as **protection rings**.



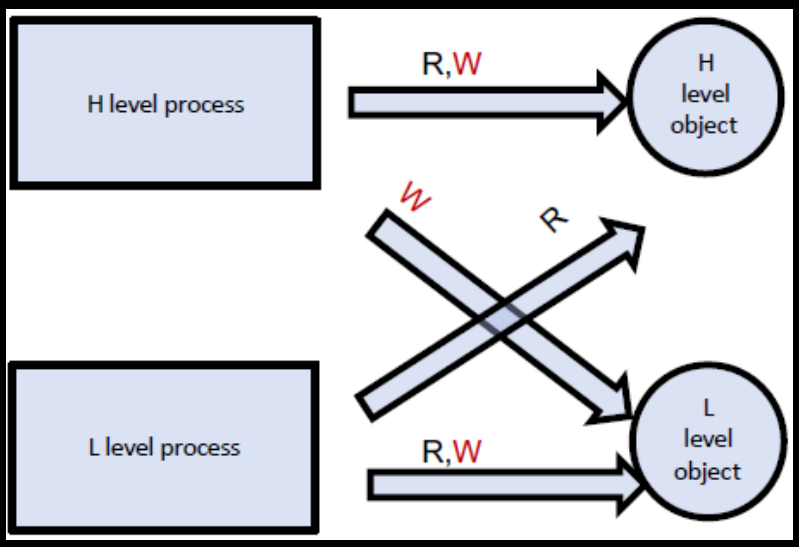
Two non-intuitive examples of intermediate control:

- **Bell-LaPadula**: restrictions include
  - **No read up**: subjects cannot read from objects at higher levels.
  - **No write down**: subjects cannot write to objects at lower levels.



This achieves *confidentiality* for higher-level objects, since information with higher privileges will not be disclosed easily.

- **Biba**: restrictions include
  - **No write up**: subjects cannot write to objects at higher levels.
  - **No read down**: subjects cannot read from objects at lower levels.



This achieves *integrity* for higher-level processes, since they will not be reading information written by processes with fewer privileges.

7.3 *Unix file system*

Note that Unix file permissions employ ACL.

In Unix systems, subjects are divided into groups. All objects (e.g., files, directories, memory devices, etc.) are treated as files. For each file, the owner specifies the rights for:

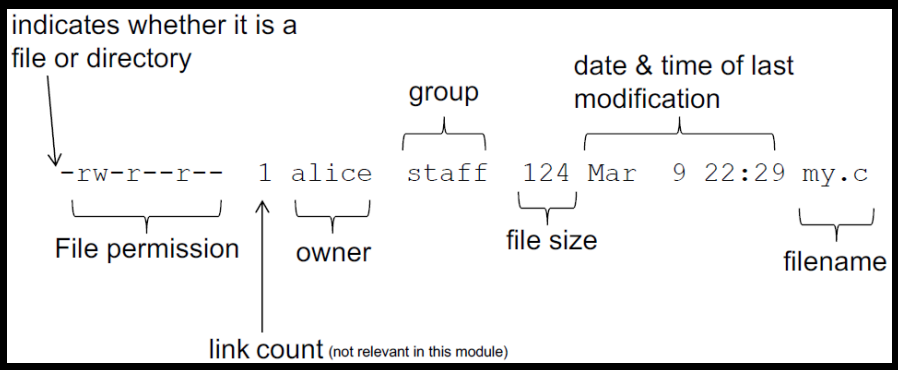
1. Owner,
2. Group (which the owner is in), and
3. Other (everyone).

For each group, flags for file permissions include:

- `r`: read.
- `w`: write.
- `x`: execute or `s`: allows user to execute with permission of owner.

A '-' indicates that access is not granted for that particular permission.

The figure below illustrates file system permissions in Unix.



Users:

- **Principals** have user identities (UIDs) and group identities (GIDs).
  - Information of the user accounts are stored in the "password" file, in the following format:

    `<username>:*:<UID>:<GID>:<GECOS>:<home dir>:<default prog>`

    - The file is made world-readable because information in the file may be required by non-root programs.
    - The `*` in the file stored users' hashed passwords in earlier versions of Unix, which is now replaced to prevent offline dictionary attacks.
      - The actual hashed passwords are stored somewhere else and not world-readable.
      - The `*` is kept for backward compatibility.
  - The **superuser** is a special user with UID 0, and usually with username `root`.
    - All security checks are turned off for `root`.
- **Subjects** are processes, and each process has a process ID (PID).
  - Every process belongs to some user.
  - When a process creates another process, it inherits the owner.

When a user wants to access a file, the following are checked in order:

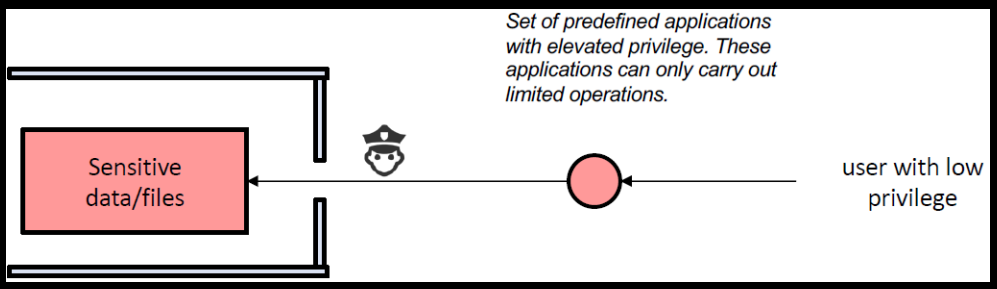
1. If the user is the owner, the permission bits for *owner* decide the access rights.
2. If the user is not the owner, but the user's group (GID) owns the file, the permission bits for the *group* decide the access rights.
3. If the user is not the owner, nor a member of the group that owns the file, then the permission bits for *other* decide the access rights.

Only the owner of the file, or the superuser, can change the permission bits.

7.4 *Controlled invocation*

In **controlled invocation**,

- The system provides a predefined set of applications that have access to a file F.
- These applications are granted **elevated privilege** so that they can freely access the file, and any user can invoke the applications.



  - These applications can be viewed as predefined "bridges" for the user to access sensitive data.
  - These applications should only perform the intended operations, so that the user stays within the planned boundaries when using the applications.
  - If the bridge is not implemented correctly, attackers can trick the bridge to perform illegal operations not expected by the programmer (i.e., **privilege escalation** attacks).

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.

In Unix,

- Processes can be created by executing a file.
- Each executable file has a *set user ID (SUID)* flag.
- Each process is associated with a **real UID** and an **effective UID**.
  - The **real UID** is inherited from the user who invokes the process.
  - If SUID disabled → **effective UID** = real UID.
    If SUID enabled → **effective UID** = file owner's UID.

Use case of `s` (setUID):

- Consider a scenario where the file `personal.txt` contains personal information of the users.
  - This is sensitive information, thus the system administrator sets it to non-readable except by `root`: `-rw-------`.
  - However, users should be allowed to self-view and self-edit some fields of their own profile.

  - Since the file's permissions is set to `-` for all users, a process created by any user cannot read/write it.
- The solution is to create an executable file `edit.exe` owned by `root`, i.e., with permissions `-r-sr-xr-x`.
  - The program is made world-executable so that *any* user can execute it.
  - The permission is also set to `s`, so when it is executed, its effective UID will be `root`.
  - Now, if Alice executes the file, the process's real UID is `alice`, but its effective UID is `root`.
  - Following the sequence in which rules are checked, the process can read/write the file `personal.txt`.

8 SOFTWARE SECURITY

Lecture 11
6th April 2023

8.1 *Control flow*

Computer architectures:

- **von Neumann architecture**: code and data are both stored in the same memory; there is no clear distinction of code and data.
- **Harvard architecture**: code and data are stored separately in separate hardware components.

Modern computers are based on the von Neumann architecture.

- The **program counter** is a *register* (i.e., a small and fast memory within the processor) that stores the address of the next instruction.
  1. After an instruction is executed, the processor fetches another instruction from the memory, whose address is specified in the program counter.
  2. After the new instruction is fetched, the program counter automatically increases by 1 before executing the fetched instruction.
  3. Thus, instructions are executed sequentially in memory.
- During execution, the program counter could be changed by the fetched instruction, e.g.:
  - **Direct branch**: replaced by a constant value specified in the instruction.
    - For direct branching, the value is typically hardcoded in the program.
  - **Indirect branch**: replaced by a value fetched from memory.

    - For indirect branching, the value is not hardcoded but determined by some data obtained during the runtime of the program.

8.1.1 Control flow integrity

Since code is stored as data, if an attacker can modify some memory (i.e., able to compromise **memory integrity**), they could potentially compromise the **control flow integrity** by either:

- Directly modifying code in the memory.
- Modifying the address in the indirect branch.

Typically, it is not so easy for an attacker to compromise memory. There are usually some restrictions (e.g., can only write to some particular location, can only write a sequence of consecutive bytes, can write but not read, etc.).

It is also generally easier to modify the data/heap/stack segments than the code segment.

Possible attack mechanisms:

1. Overwrite **existing execution code** with malicious code.
2. Overwrite a piece of control-flow information, i.e., replace a **memory location** storing a code address used by:
   (a) A direct jump (e.g., via `printf` *vulnerability*).
   (b) An indirect jump (e.g., via *buffer overflow*).

8.1.2 Call stack

Functions/Procedures/Subroutines break code into smaller pieces, thus facilitating modular design and code reuse.

- A function can be called from different parts of the program, and even recursively.
- The **call stack** stores important information of function calls (e.g., parameters to be passed, local variables, and control flow information such as return addresses).
  - **Stack frame**: each element of the call stack, including:
    - Parameters.
    - Return address.
    - Local variables.
    - Frame pointer: pointer to the previous stack frame in the stack.
  - **Stack pointer**: variable that stores the location of the first element.

  The stack is a LIFO data structure.

Using attacks such as **buffer overflow**, the attacker could modify the stack (i.e., **stack smashing**). Stack smashing has two effects:

- If return address is modified, the control flow is compromised.
- If local variables are modified, the computed result could be wrong.

Lecture 12
13th April 2023

8.2 *Unsafe functions*

Several features in C are considered unsafe, e.g.:

1. `printf()`: if formatting symbols (e.g., "%d") are present in the first parameter, the function will attempt to fetch values from the supposed location of the the subsequent parameters and display them, even if the function only took in 1 parameter.
   - Through this vulnerability, an attacker might be able to:
     - Obtain more information (loss of confidentiality).
     - Cause the program to crash (loss of execution integrity).
     - Modify memory content using `"%n"` (loss of execution integrity).

     If the program invoking `printf` has elevated privilege, the attacker can obtain information that was previously inaccessible.
   - Preventive measure: avoid using the following forms of `printf` where `t` is a variable:
     - `printf(t)`
     - `printf(t, a1, a2)`

     Instead, use raw strings (e.g., `printf("hello %s", "abc")`).
2. **Data representation**: some strings in C adopt the **null termination** convention, where the first occurrence of the null character (i.e., `\0`) indicates the end of the string.
   - Systems which use both null and non-null definitions to verify certificates may get confused, e.g., a browser implementation which:
     (a) Verifies a certificate based on non-null termination.
     (b) Displays the domain name based on null termination.

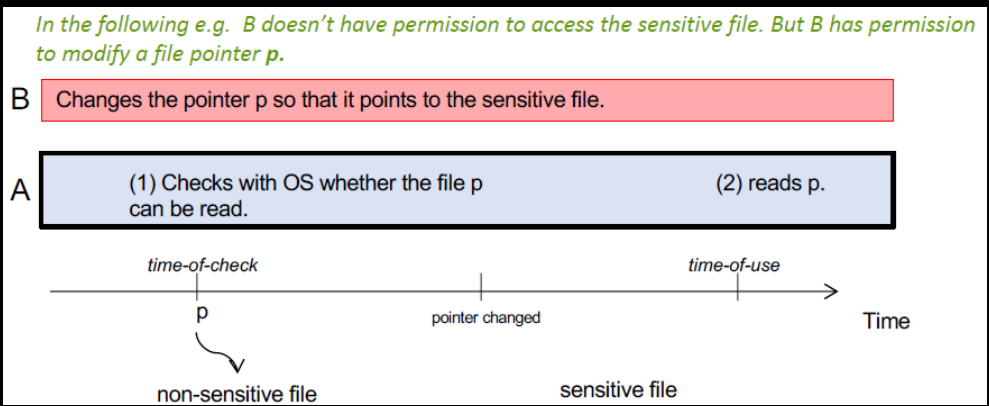
     This can enable attackers to direct victims to spoofed websites, e.g., luminus.nus.edu.sg\0.attacker.com, since only the section of the string before '\0' will be visible to the victim in his/her browser.
3. **IP address representation**: IP addresses can be represented as:
   (a) Strings: e.g., "123.123.123.123".
   (b) 4 integers, each with a size of 32 bits.
   (c) A single 32-bit integer.
   - Suppose the an IP address has the format "a.b.c.d". If a user stores IP addresses as `ip = (a<<24) + (b<<16) + (c<<8) + d`, an attacker could exploit the system by passing in invalid values (e.g., `a = 257`).

- Preventive measure: always convert IP addresses to a standard/canonical representation immediately.

4. **Buffer overflow**: C and C++ do not employ bound checking during runtime. This achieves efficiency, but is prone to bugs, e.g.:
   - `strcpy(s1, s2)`: copies the entire string `s2` to `s1`, even iif the length of `s2` is larger than the size of the buffer `s1`.
     - Preventive measure: use `strncpy(s1, s2, n)` instead, which takes in an additional parameter n which states that at most n characters will be copied.
   - **Stack smashing**: special case of buffer overflow targetting the stack.
     - If the return address (stored on the stack) is modified, the execution control flow can be changed (e.g., to run a piece of code injected by the attacker).
     - Preventive mechanisms: canaries and memory randomization.
5. **Integer overflow**: in most programming languages, integers "wrap around" (e.g., for an unsigned 8-bit integer, $254 + 2 = 0$).
6. **Code/Script injection**: scripting languages (e.g., SQL) are particularly vulnerable to injection attacks. E.g., SQL injection: `1' OR 1=1 --`.

Scripts: programs that automate the execution of tasks that could alternatively be executed line-by-line by a human operator.

7. **Undocumented access points**: programmers may insert undocumented access points to inspect states for debugging, and these access points may mistakenly remain in the final production system, providing a backdoor for the attackers.
8. **Race conditions**: **time-of-check-time-of-use (TOCTOU)** is a type of race condition illustrated by the following:
   (a) Process A checks the permission to access some data, followed by accessing the data.
   (b) Process B (e.g., attacker) swaps the data.



8.2.1 Preventive measures

1. **Input validation**: if the input provided by the user is not of the expected format, reject it.
   - **Whitelist**: list of items that are known to be benign and allowed to pass.
   - **Blacklist**: list of items that are known to be bad and to be rejected.

   However, it is difficult to design a filter that is (1) complete (i.e., does not miss out any malicious string), and (2) accepts all legitimate inputs.
2. **Parameterized queries**: mechanisms introduced in some SQL servers to protect against SQL injection.
   - They enable the SQL parser to know which parameters are the "data" and which parameters are the "script".
3. **Safe functions**: avoid functions that are "unsafe", e.g., `strcpy()`, `printf()`, and `access()`.
4. **Bound checking**: some programming languages (e.g. Java) perform bound checking during runtime, and terminate prematurely when the user attempts to access an index which is out of bounds.
   - This reduces program efficiency, but prevents buffer overflow.
5. **Type checking**: some programming languages perform type checking to ensure that the arguments of an operation during execution are always correct.
   - This checking could be done during runtime (i.e., *dynamic* type checking) or when the program is compiled (i.e., *static* type checking).
6. **Canaries**: secrets inserted at carefully selected memory locations during runtime. Checks are carried out during runtime to make sure that the values are not being modified. If so, halts.
   - This helps to detect stack overflow (e.g., when an attacker attempts to change the return address of a function on the stack).
7. **Memory randomization**: *address space layout randomization (ASLR)* helps to decrease the success of attacks, by randomizing the location where data and codes are stored.
8. **Code inspection**: this can be done either via manual or automated checking (e.g., taint analysis, sanitizers, etc.)
9. **Testing**:
   - White-box: tester has access to source code.
   - Black-box: tester does not have access to source code.
   - Grey-box: combination of the above.
   - Fuzzing: sending malformed inputs to discover vulnerabilities.

10. **Principle of least privilege**: in a particular abstraction layer of a computing environment, every module (e.g., a process, user, or program) must be able to access only the information and resources that are necessary for its legitimate purpose.

* * *