1 Security Policies

Threat model: Threat modeling is *a family of activities for improving security by identifying threats, and then defining* countermeasures to prevent, or mitigate the effects.

Describe the capabilities of the adversaries.

- What they can do and what they cannot do with the system.
- What system vulnerabilities they might exploit.
- What is trusted and what is not trusted.

Security policies.

- Describe desirable behavior of the system.
    - What should not happen and what should happen.
- Security policies can be expressed in terms of:
    - **Confidentiality** properties - Restrictions on who may read some entity.
        - Privacy
        - Anonymity
    - **Integrity** properties - Restrictions on how much trusted an entity should be.
    - **Availability** properties - Requirements that "good things" should happen.

enforcement mechanism is a system or process that ensures security policies and rules are followed, and enforces **CIA**.

- It ensures that the system behavior satisfies the security policy against the threat model.
- Policy and enforcement separation.
    - The same policy can be enforced by many different mechanisms.

Enforcement mechanisms are vital for:

- Protecting Sensitive Data: Ensuring only authorized users can access sensitive information.
- Maintaining System Integrity: Preventing unauthorized modifications to systems and data.
- Ensuring Availability: Protecting systems from attacks that could disrupt services and operations.
- Compliance: Meeting regulatory requirements and avoiding legal consequences associated with data breaches.

Assurance - Show that the enforcement actually does what it is supposed to do.

Property: refers to attributes or characteristics that are essential for enforcing security policies.

A useful way to map properties into defenses: **Confidentiality, Integrity, Availability (CIA):**

- **confidentiality** is limiting data access.
- **integrity** is ensuring your data is accurate.
- **availability** is making sure it is accessible to those who need it.

2. Enforcement

Security by design

- Description of system functionality
- Description of threat model
- Specification of security policies – what we want to achieve.
- Enforcement mechanism – How we will achieve it.
- Assurance – Same^

An **enforcement mechanism** ensures that the system behavior satisfies the security policy against the threat model.

- **Authentication** - whether accesses are granted (E.g. Passwords).
- **Access Control** – Type of authentication (E.g. Identity, group membership, role)
  - Access control **Matrix** - Check whether <S,O,Op> is in the access control matrix
- **Encryption**

Defense in Depth - Use a collection of complementary mechanisms to enforce a policy (E.g. Multi-factor authentication).

Secrecy of design - Open design (E.g. Crypto protocols) vs. Security by obscurity (E.g. Proprietary code)

Type checking - Analyze the program and check whether it follows certain rules.

Isolation – Involves separating critical system components to prevent unauthorized access and interference.

Redundancy – Refers to the duplication of critical components or functions to increase reliability and fault tolerance.

- Replication – Copies of data to ensure availability in case of failure (E.g. disk failure).
- Error correcting codes – Techniques to detect and correct errors in data, ensuring data integrity even if some errors occur.

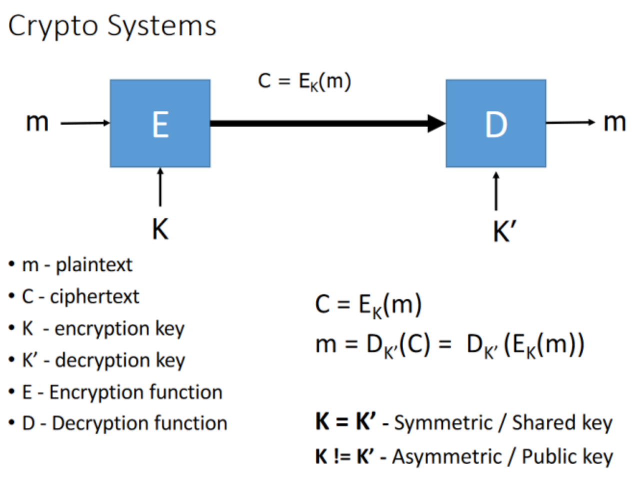
Audit – Involves recording and examining activity within a system to ensure compliance with security policies and to detect and investigate suspicious behavior.

- Ensuring that all actions can be traced (**Accountability**) back to a user (**Authentication**).

Economy of Mechanism – A simpler mechanism gives better assurance.

3. Crypto (Basic symmetric ciphers)

Cryptography – The practice and study of techniques for secure communication in the presence of third parties called adversaries.



- m - plaintext
- C - ciphertext
- K - encryption key
- K' - decryption key
- E - Encryption function
- D - Decryption function

$C = E_K(m)$

$m = D_{K'}(C) = D_{K'}(E_K(m))$

K = K' - Symmetric / Shared key

K != K' - Asymmetric / Public key

Attacks by Cryptanalyst

- Ciphertext-only attack (**COA**).
  - Determine plaintext and/or key.
  - Has access to the ciphertext of one or more messages and attempts to determine the plaintext and/or the key.
- Known-plaintext attack (**KPA**).
  - Has C - P pairs.
  - Determine key.
  - Has access to one or more pairs of plaintext and corresponding ciphertext encrypted with the same key, and aims to determine the key.
- Chosen-plaintext attack (**CPA**).
  - Can perform C = EK(P).
  - Determine key.
  - Hackers choose specific messages (plaintexts) to be encrypted using the target's encryption algorithm. Doing so produces corresponding ciphertexts. Hackers analyze the ciphertexts to reveal all or part of the secret encryption key.
- Chosen-ciphertext attack (**CCA**).
  - Can perform P = DK(C).
  - Determine key.
  - Same as above, but reversed.

Substitution ciphers

- **Simple substitution** – Each letter substituted by another letter or shifted**.**
  - **Caesar's Cipher**
  - Vulnerable to statistical attacks
- **Block substitution** – Data is processed in fixed-size blocks.
  - **Vigenère**
  - Vulnerable to statistical attacks
- **Polygraphic substitution –** Encrypt plaintext by replacing groups of letters (pairs) in a matrix system.

- **The Hill Cipher**
- Vulnerable to statistical attacks + long keys

Letter frequencies attack on substitution ciphers – most common letters makes it easy to see a pattern.

Caesar's Cipher – Shifted alphabet.

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**X**	**Y**	**Z**
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	W

Alphabet is shifted 3 rows to the left.

$$E_n(x) = (x + n\,)\; mod\; 26$$

$$D_n(x) = (x - n\,)\; mod\; 26$$

Represented using modular arithmetic.

Vigenère Cipher -> copy the key on the plaintext, you know have a "keytext". The intersection of th row and coloumn of the plaintext and "keytext" in the table, is the ciphertext.

- **Key extension** – Same logic as Ceasar's Cipher, but repeat the key on the plaintext and substitute each letter according to the key's letter – number value.

Key (K)	C	A	T	C	A	T
	2	0	19	2	0	19

P=	C	R	Y	P	T	O
	2	17	24	15	19	14

C=	E	R	R	R	T	H
	4	17	17	17	19	7

- **Table extension** – Same result but visualized with a table: Row C, Column C, --> Result E

The Hill Cipher – is a polygraphic substitution cipher using n x n matrix table.

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix} = \begin{pmatrix} 67 \\ 222 \\ 319 \end{pmatrix} \equiv \begin{pmatrix} 15 \\ 14 \\ 7 \end{pmatrix} \pmod{26}$$

First matrix -> key (block-length^2)

Second matrix -> plaintext (ACT)

Transposition Cipher – method of encryption which scrambles the positions of characters (transposition) without changing the characters themselves. They differ from substitution ciphers, which do not change the position of units of plaintext but instead change the units themselves.

One Time Pad – Use a **random** key that is equal length to plaintext and combine the key and the message using modular addition (letters have a corresponding number).

- The ciphertext is unbreakable if the key is; truly random, equal length to message, never used again, secret.

- **XOR** is perfect since you can apply the same **One Time Pad** on a message and ciphertext and get the same answers.
- **Two Time Pads** – Is insecure since you use the same pad twice and can therefore uncover the messages via eavesdropping. C = Ciphertext, M = message, P = Pad.
  $C1=M1\oplus P$
  $C2=M2\oplus P$
  $C1\oplus C2=(M1\oplus P)\oplus(M2\oplus P)=M1\oplus M2$

Stream Ciphers – Same logic as **One Time Pad** but the key is pseudo random and generated from a secret key that is (most likely) shorter than the message.

4. Advanced Encryption Standard (AES)

- Block cipher, 128 bits (16 bytes) blocks.
- Keys are 128, 192, and 256 bits.
- AES-256 considered best choice for symmetric cryptosystems.

AES: 10 rounds and 4 basic steps

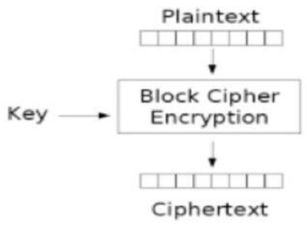
- Start *P* = plaintext, *K* = key:
  - $X_0 = P \oplus K$
    - Convert plaintext and key to hexadecimal and XOR.
- For each round 1 … 10 do:
  1. **SubByte**: S-Box substitution (lookup table, very quick)
     - Use S-Box on X_0
  2. **ShiftRows**: Transposition
     - Row0: shift left 0.)
     - Row 1: shift left 1.) All will wrap around.
     - Row 2: shift left 2.)
     - Row 3: shift left 3.)
  3. **MixColumns**: Hill Cipher
     - Mix columns similarly to hill cipher.
  4. **AddRoundKey**: XOR PRND pad
     - XOR the result-block with a derived subkey from the main key.

Lookup Table (Galois Field) in AES stores 256 entries (GF(2^8)).

Attacks on AES – highly secure, timing attacks -> table is now stored at CPU and is immune to this.

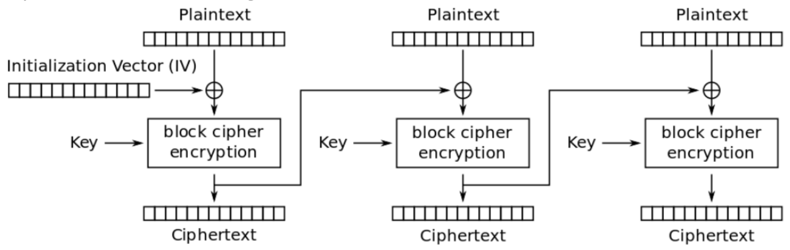
Block Cipher modes of Operation: Different ways to operate block cipher on fixed size blocks.

- Electronic Code Book (ECB) mode.



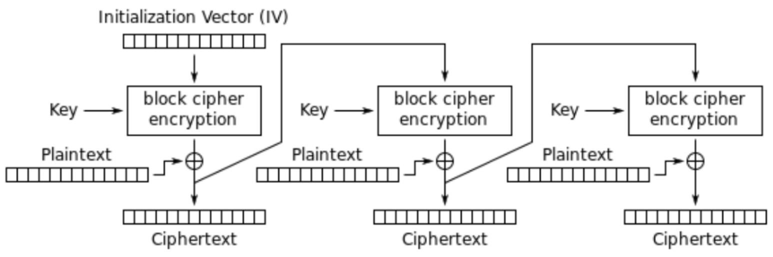
  i. Possible to see patterns in images.

- Cipher-block Chaining (CBC) mode.



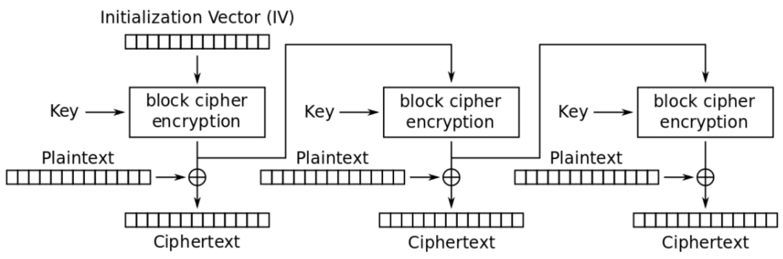
  i. Is resilient to loosing blocks since a single block only depends on the previous block.
  ii. Parallel decryption
  iii. Decrypt -> swap plaintext and ciphertext, swap arrows with attachments (other arrows etc.), swap encryption with decryption.

- Cipher Feedback (CFB) mode.



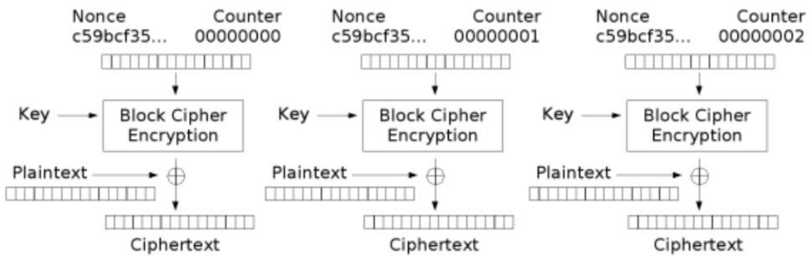
   i. Uses the encryption method to decrypt (swap plaintext and ciphertext).

- Output Cipher Feedback (OFB) mode.



   i. Uses the encryption method to decrypt (swap plaintext and ciphertext).

- Counter (CTR) mode.



   i. Uses the encryption method to decrypt (swap plaintext and ciphertext).

Initialization Vector – A block of bits used to randomize encryption.

- Usually does not need to be secret.
- Never reused.

Padding – Many crypto schemes operate on fixed blocks of data.

- **PKCS#7** – pad the end of the message with same values, k - (l mod k) octets.
- **ISO/IEC 7816-4:2005** – first byte of bad is 0x80, the rest is 0x00.

ECB, CBC, OFB, CFB, CTR protects **confidentiality**, But no **authenticity.**

5. Public-Key Crypto

Public key -> K^+

Privat key -> K^-

Congruence –> $a \equiv b \ (mod\ n)$ – same reminder when dividing with n.

Fermat’s Little Theorem:

$$x^{p-1} \equiv 1 \ (mod\ p)$$

when p is prime, gcd(x,p) = 1, and $0 < x < p$

6. Authentication (of Machines)

Who is at the other end of the socket?

- **IP address** not sufficient (spoofing)
  - The attacker sends packets from a false IP address to disguise their identity.

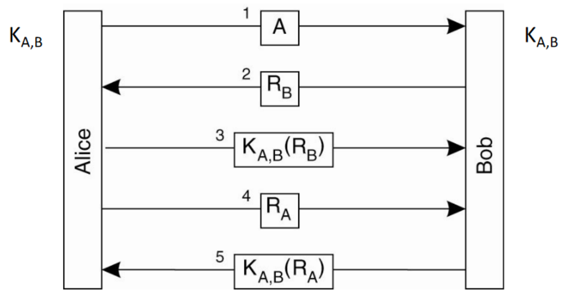
Dolev–Yao Adversary Model

- Assumes an active adversary that can;
  overhear, intercept, delay, replay, and synthesize any message.
- But cannot break crypto.

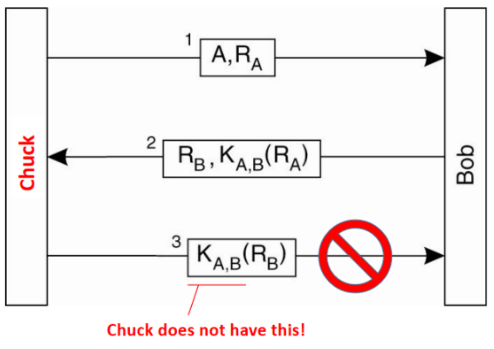
Shared Key Authentication (SKA) –> Convince each other that they know the secret.

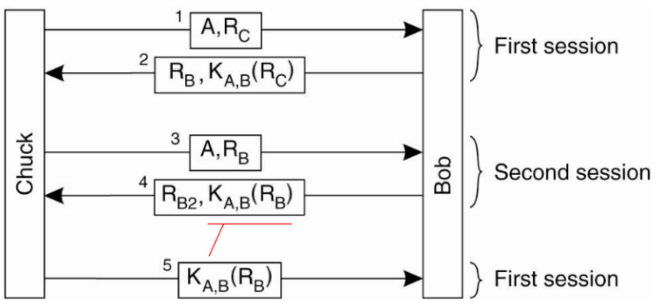
- A (Alice) and B (Bob) holds a pre-shared key $K_{A,B}$
- Value encrypted with key K
  - $K(value)$ or $\{value_1, value_2\}$
- Nonce R_1 and R_2
  - random strings, single purpose

Shared Key ($K_{A,B}$) Authentication



Reflection Attack on "optimized" version.



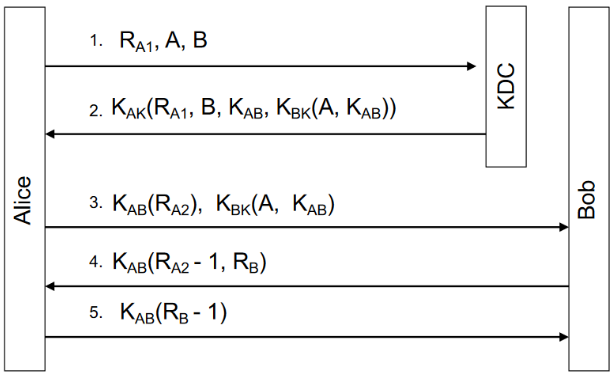


How can we safely share secrets.

1. Alt 1: Common group-key for all parities
   a. Requires a high-level of trust between all parties.
   b. Keys are not very secret if shared by many nodes.
2. Alt 2: Unique key for each pair of principles.
   a. N(N-1)/2 keys: one for each node pair.
   b. Huge key management issue.
3. Alt 3: **Key Distribution Service (KDC).**
   a. N - keys: one for each node, shared with KDC.
   b. Session keys generated peer communication.

Needham-Schroeder Protocol

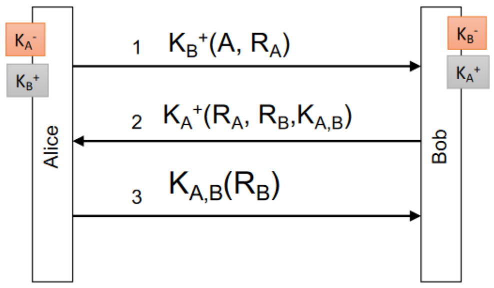
- Principals:
  - Alice (A) and Bob (B)
  - Key Distribution Center (KDC)
- Pre-Shared Secrets:
  - $K_{A, K}$ - Shared key between principal and KDC (one for each principal)
  - $K_{A, B}$ - Session keys generated by KDC for use by A and B



Alice
KDC
Bob
1. R_{A1}, A, B
2. $K_{AK}(R_{A1}, B, K_{AB}, K_{BK}(A, K_{AB}))$
3. $K_{AB}(R_{A2}), K_{BK}(A, K_{AB})$
4. $K_{AB}(R_{A2} - 1, R_B)$
5. $K_{AB}(R_B - 1)$

Public-Key Authentication

- Negotiating a shared key is slow. (speed order ~1000)
- K_A^+, K_B^+: Public keys. R_A, R_B: Nonce.

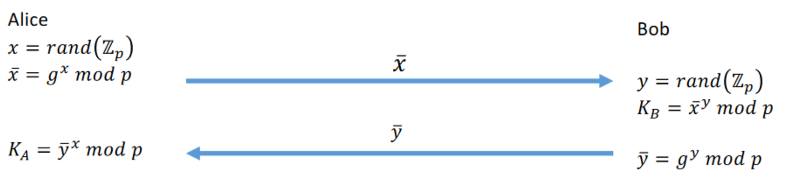


Forward Secrecy (FS) principle –> Diffie-Hellman

- Ensures that the compromise of long-term keys (like private keys used in a public-private key pair) does not compromise the session keys used in past communications.
- This is crucial for maintaining the **confidentiality** of past communications.
- Diffie-Hellman can be used in combination with other key exchange protocols (e.g., TLS, SSH)

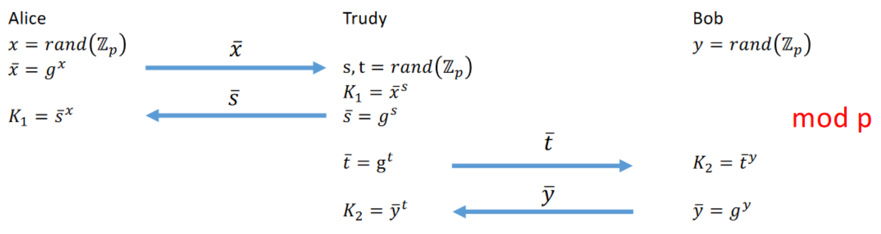
Diffie-Hellman (DH) key exchange protocol –> A method that allows two parties to securely share a secret key over a public channel.

1. Both parties agree on the public parameters *p* and *g*.
2. Private and Public Keys Generation.
    a. Private key: Choose a random integer *x*.
    b. Public key: $\bar{x} = g^x \bmod p$
3. Exchange public keys.
4. Compute shared private keys.
    a. $K_{A,B} = \bar{x}^x \bmod p$



Claim $K_A = K_B$ (a shared key)

- Public Parameters:
    - *p*: A large prime number.
    - *g*: A primitive root modulo *p* (also known as a generator).
- Secure since it is difficult to solve $\bar{x} = g^x \bmod p$ without knowing *x* (*x* is private).
    - Immune against eavesdropping.
- **Not Immune against Man in the Middle Attacks**.



Short answer: Trudy sends **TWO** of her private keys to A and B.

7. Discretionary Access Control (DAC)

Confidentiality and **integrity** are often enforced using a form of authorization known as **access control**. Maps **user ids** to rights on operations.

Assumes:

1. Authentication scheme.
2. Predefined set of operations (op).
3. A **reference monitor** checks execution.

Two **Access-Control** Approaches:

- Discretionary Access Control (DAC) –> the creator of an object can choose who has access.
  - Individual users can set rights that allow or deny access to objects.
  - Initial assignment by object owner.
- Mandatory Access Control (MAC).
  - Access restricted in system-wide policies.

Reference Monitor

- Every access; read, write, or execute, must go through the reference monitor.
- The primary function of the reference monitor is to enforce **access control policies**. It checks whether a subject (such as a user or a process) has the necessary permissions to perform a requested operation (such as reading or writing) on an object (such as a file or a database record).
- In many operating systems, the kernel acts as the reference monitor.
- The reference monitor is crucial for maintaining the **security** and **integrity** of a system.

Security commandments

- **Principle of Least Privileges (PoLP)** –> Each Principal should only have the least privileges it needs to accomplish its task.
  - impossible to implement if all objects and operations require the same privileges.
- Separation of privileges –> Divide system and access.

Protection matrices:

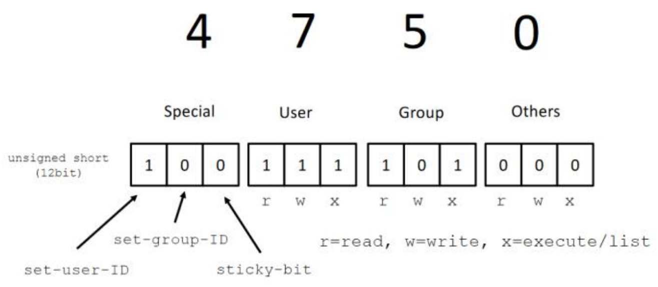
Specifies which users or system processes are granted access to resources, as well as what operations are allowed on given resources.

- **The Access-Control Matrix Model**
  - One row for each subject (user), One column for each object (file).
  - Cells has the rights for each subject to each object.
- **Access Control List**
  - Files hold subjects and operations.
  - *object (file) –> {Alice: read, write; Bob: read}*
- **Capabilities (or bearer tokens)**
  - Subjects holds files and operations.
  - *Subject (User) –> {f1: op_1, op_2; f2: op_2, op_3}*

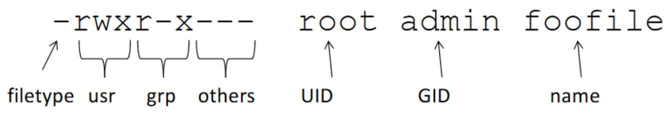
        - Subjects provide their capabilities to functions (JSON Web Tokens)
- **Linux Access Control List (Matrix Model)**
    - One row for each subject (uid/group uid), One column for each object (file).
    - Cells are rwx rights encoded as 4 x 3 rights bits (octets).
      read, write, execute.

Linux ACL

- Each file is described by an inode data structure.
    - Principals represented as uint.
    - i_mode encodes file permission: 4 x 3 bits (octets).



    - Can be written as:



Information Flow and DAC:

- A creates secret file and wants only B to read it.
- B wants to leak the file to C.
- Kernel prevents B from adding C to the ACL

- B reads the file and writes a copy.
- B allows C to read the copy.
- ACL/DAC cannot prevent this.

8. Mandatory Access Control (MAC) – used in companies

Who makes the authorization rules?

- If triplet <subject, object, action> is enough –> DAC
- If triplet <subject, object, action> is NOT enough –> MAC

In a **MAC** system, access policies are centrally controlled and cannot be altered by individual users.

- **Separation of Duty** –> requires multiple individuals to complete and authorize sensitive task.
  - Cooperation now becomes necessary to perpetrate fraud. Which is difficult.
  - Code developer and Code reviewer are different people.
    - *Auth (public, code, release, code developer, code reviewer, approval status) –> {Allowed, Not allowed}*

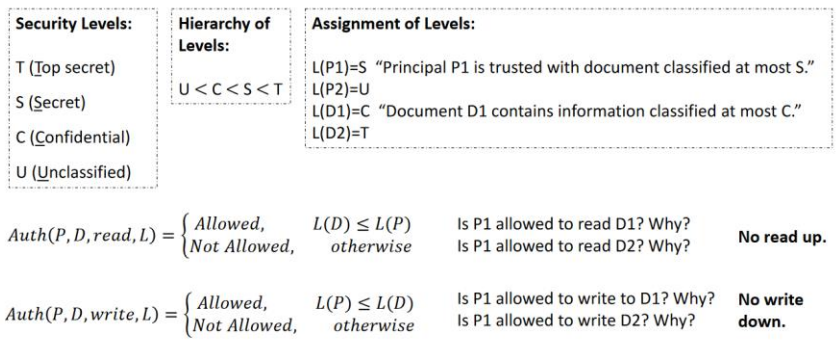
Ethical Wall (Chinese Wall) **Policies**

- Alice may read file A or file B. But once Alice reads A, she cannot read B, and vice versa.

Role-Based Access Control (RBAC)

- Subjects are assigned to roles; each role grants a set of privileges.
- Roles inherit roles from less superior roles.
- **PoLP** –> pick the lowest possible assigned role to accomplish a task.

Multi-level Security (MLS) –> Roles define different levels of security (access to documents etc.)



Security Levels:

T (Top secret)
S (Secret)
C (Confidential)
U (Unclassified)

Hierarchy of Levels:

U < C < S < T

Assignment of Levels:

L(P1)=S "Principal P1 is trusted with document classified at most S."
L(P2)=U
L(D1)=C "Document D1 contains information classified at most C."
L(D2)=T

$$Auth(P, D, read, L) = \begin{cases} Allowed, & L(D) \leq L(P) \\ Not\ Allowed, & otherwise \end{cases}$$

Is P1 allowed to read D1? Why?
Is P1 allowed to read D2? Why?
No read up.

$$Auth(P, D, write, L) = \begin{cases} Allowed, & L(P) \leq L(D) \\ Not\ Allowed, & otherwise \end{cases}$$

Is P1 allowed to write to D1? Why?
Is P1 allowed to write D2? Why?
No write down.

Protects **confidentiality**.

Enforcing MLS for confidentiality means that information flows only towards higher levels. This restriction is not always practical!

- Solution: **Information Flow Control**

SUMMARY

- policies with increased expressiveness.
- capture complicated requirements.
- might be difficult or impossible to enforce precisely, might be conservative and have many false negatives.

9. Credential-based Authorization

Authorization policies specify conditions under which access requests may be granted. Many different schemes:

- Access control lists.
- Capabilities.
- Multi-level security.

Guard operation for credential-based authorization is a security measure that enforces access control policies by verifying the credentials of users or systems attempting to access resources.

- **Credential Verification**: verifies the credentials (e.g., username and password, tokens, certificates)
- **Access Control Enforcement**: checks an **access control policy** or list to determine if the entity has the necessary permissions to access the requested resource.
- **Logging and Auditing**

Goal Formula acts as a rule that the guard operation uses to determine whether the presented credentials meet the necessary criteria for authorization. "IF-Statement"

Credentials convey beliefs about principals and/or the system state; guards employ logical inference to grant requests only when some specified goal formula is shown to hold.

Credential-based authorization allows for **delegation**, meaning that a user who has been granted specific access rights can further grant those rights to other users.

Credentials Authorization Logic –> Logical expressions used in delegation and authorization.

$$\text{SAYS-E: } \frac{P \textbf{ says } (P \textbf{ says } \mathcal{C})}{P \textbf{ says } \mathcal{C}} \qquad \text{SAYS-IMP-E: } \frac{P \textbf{ says } (\mathcal{C} \Rightarrow \mathcal{C}')}{(P \textbf{ says } \mathcal{C}) \Rightarrow (P \textbf{ says } \mathcal{C}')} \qquad \text{SAYS-AND-E: } \frac{P \textbf{ says } (\mathcal{C} \wedge \mathcal{C}')}{(P \textbf{ says } \mathcal{C}) \wedge (P \textbf{ says } \mathcal{C}')}$$

$$\text{DELEG-E: } \frac{P' \textbf{ speaksfor } P}{(P' \textbf{ says } \mathcal{C}) \Rightarrow (P \textbf{ says } \mathcal{C})} \qquad \text{HAND-OFF: } \frac{P \textbf{ says } (P' \textbf{ speaksfor } P)}{P' \textbf{ speaksfor } P}$$

Some **Inference rules**.

MÅ KANSKJE LES PÅ NYTT

10. Signatures, Hashes, and Certificates

Threat model: **Man in the Middle**

Digital signatures: RSA public and private key

- For **integrity**: sign with private key and verify with public key.
- For **confidentiality**: encrypt with public key and decrypt with private key.
- **Does not satisfy**: "B should accept only messages written by A"
  - **MitM** can combine two encrypted messages from A, and send it to B. A did technically not send it.

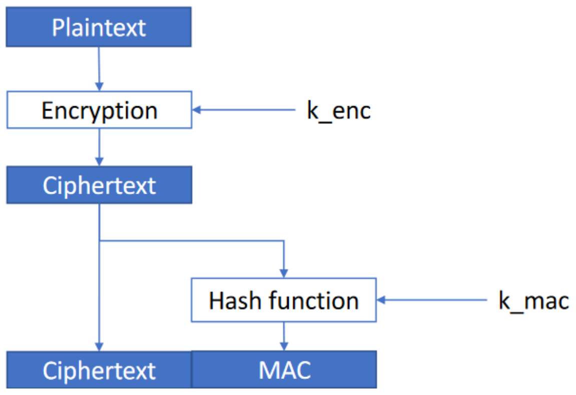
Cryptographic Hash Functions take an input (or 'message') and return a fixed-size string of bytes. (E.g. SHA-256 produces 256-bit hash values)

- **Deterministic:** Maps messages of arbitrary size to values of fixed size.
- **One-way:** Easy to compute but hard to invert.
- **Collision resistant:** Hard to find two messages that yield the same hash value.
- **Avalanche effect:** A small change in the input (even a single bit) should produce a significantly different hash value.

- **Use Case**: Digital signature.
  Sign messages with the hash value of the sent message.
- **Integrity** and **confidentiality** of data. But not confidentiality of messages since you cannot get the message from a hash value.

Message Authentication Codes (MAC) –> takes the message and the secret key (shared) as inputs and produces a fixed-size output known as the MAC or tag.

Authenticated Encryption –> Protects **confidentiality** and **integrity** of messages.



Digital Certificates –> Prove that it's your public key. Verified by a trusted third party known as a Certificate Authority (CA).

11. Web Security

Authentication –> The process of verifying the identity of a user (or machine) interacting with your system.

Authentication points:

- How the web server authenticates users.
- How database authenticate the web server.
- How users authenticate the web service.

Backend Authentication –> How the server authenticates to the database.

- Web server is trusted (nodejs example)
- Put environment variables in files (credentials)
- Key-Vault (e.g., Azure) –> Stores secrets and certificates.

Client Authentication –> Server must decide on rights of the caller.

Different levels of authentication

- Distinguish between groups of principals.
  - e.g., browser fingerprinting.
- Identity repeated request are from the same principal.
  - cookies, local username-password database.
- Identify principals across service domains.
  - Social login (Facebook, Google), federated identities (FEIDE).
- Map system identities to real-world identities.
  - e.g., BankID.

Basic HTTP Authentication (of users)

- GET request… Server responds with Unauthorized and ask for;
- Browser gets credentials.
- Browser computes token.
- Browser append token to request and redo

HTTP cookies are small pieces of data sent from a website and stored on the user's device by the web browser while the user is browsing. They are used to remember information about the user, such as login status, preferences, and other session information, to enhance the browsing experience.

- (name, **domain**, **path**) - > values
  - **Domain** and **path** attribute set by server.
- 3rd Party Cookies (domain != origin) –> Enables tracking of users between sites.

HTTP Sessions –> On login, server computes a (random) session id and store locally (browser). Client includes session ID cookie in all subsequent requests.

- Session ID is a transient credential.

HTTP Cookie Security

- Persistent cookies: stored on disk at client.
    - Can track users between browser restarts.
    - Must be protected on the client's disk.
- Secure cookie: require encrypted connection (HTTPS) –> Very common.

If attacker can **steal cookies**, he can impersonate the user. **Defense**: Browsers enforce a Same-Origin Policy.

Cross-Origin Resource Sharing (CORS) –> is a security feature implemented by web browsers that allows web applications to request resources from a different domain (origin) than the one that served the web page. This mechanism is essential for enabling secure cross-origin requests and data sharing between different websites.

MÅ KANSKJE LES PÅ NYTT

12. OpenID and OAuth2

Machine-to-Machine (M2M) Calls –> automated communication between devices or systems without human intervention. API server wants to access 3rd party server on user’s behalf, based on **valid credentials** only.

OAuth 2.0 Authorization Framework –> allows third-party applications to obtain limited access to user resources without exposing the user's credentials.

- Industry standard.
- Enables the use of external authentication providers.
- access tokens carried in HTTP header.

4 roles in OAuth2

1. **Resource Owner:** The user who owns the protected resources and can grant access to these resources.
    - Often a person (end-user) and her browse
2. **Client**: The application requesting access to the resource owner's resources. It could be a web application, mobile app, or any other service.
3. **Authorization Server**: The server that authenticates the resource owner and issues **access tokens** to the client. It is responsible for handling authorization grants.
    - Issues **ID tokens** as proof of completed authentication.
4. **Resource Server**: The server that hosts the protected resources and accepts access tokens to allow access to resources.

Tokens grants access (transient credentials).

JSON Web Token –> Auth server include claims as a JSON object in token.

Access Tokens:

- Credentials used to access protected resources.
- The string is usually opaque to the client.
- Represents scopes and durations of access, granted by the resource owner.
- Enforced by the resource server and authorization server.

Refresh tokens:

- Used to obtain a new access token (when current expires).
- Issued to the client by the authorization server.

ID Tokens:

- Stores user profile information for use by the client.

Resource Owner Password Credential Flow **(ROPC)**

Application Types

- **Public client applications** –> Cannot securely store secrets, such as client credentials. because public clients run on platforms that are accessible to end-users.
    - **Single Page Applications (SPAs)** –> web applications that load a single HTML page and dynamically update content as the user interacts with the app.
        - web apps in which tokens are acquired by a JavaScript or TypeScript.
    - **Mobile Applications.**
    - **Desktop Applications:** Applications installed and run on personal computers.
- **Confidential client application** –> can securely store and manage secrets, such as client credentials. These applications typically run on servers or in environments where the security of the application's credentials can be ensured.
    - Applications that run on a web server.
    - Backend Services.
    - Trusted Applications such as internal tools and management systems.

OAuth2 Authorization Flows

- Public Client Application:
    - Authorization Code Flow with PKCE.
    - Implicit Flow.
- Confidential Client Applications
    - Authorization Code Flow
    - Client Credentials Flow

Authorization Code Grant

- Redirection-based flow.
- Client must be capable of:
    - interacting with the web browser.
    - receiving incoming requests (via redirection) from the authorization server.
- used to obtain both access tokens and refresh tokens.
- optimized for confidential clients (that has client secrets).

Implicit Grant Flow

- redirection-based flow
    - optimized for public clients known to operate a particular redirection URI.
- client receives the access token as the result of the authorization request.
    - Unlike code flow.
- Does not include client authentication.
- access token is encoded into the redirection URI,
    - may be exposed to the resource owner and other applications residing on the same device.
- No longer a suitable authentication method.
    - E.g., due to removal of 3rd party cookie

13. HTTPS

How database authenticate the web server:

1. **Establishing a Secure Connection**: Choose a cipher and has function and the database sends the encrypted certificate to the web server, also containing the public key.
2. **Verification of the Certificate**: The web server verifies if the certificate is signed by a Certificate Authority (CA), then generates a random number that is used for the encryption in this session.
3. **Generation of Shared Secret Keys**: Use the random number to create shared keys.
4. **Verify Web Server Certificates**: The database verifies the web servers certificate to ensure mutual authentication.
5. **Access Control and Privilege Delegation**: restrict what the authenticated web server can do.

How the web server authenticates users:

- Username and password.
- Certificates.
- OAuth.
- Session cookies and Token-Based authentication.
- Biometric authentication.

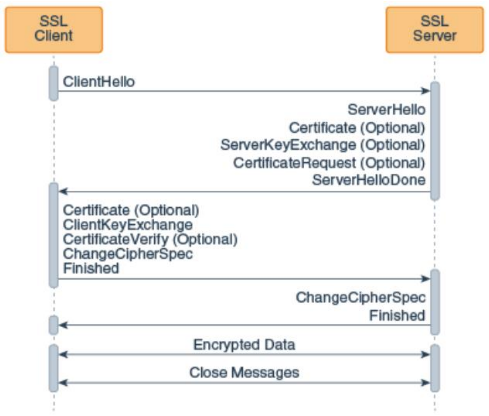
How users authenticate the web service –> Kinda the same steps as above, only the web service sends the certificate to the user.

- HTTPS: it uses SSL/TLS (Secure Sockets Layer/Transport Layer Security) to encrypt.
  - Web service provides SSL/TLS certificates.
- Domain Name System (DNS): translates domain names to IP addresses.
- Public Key Infrastructure (PKI): Digital signature that contains the public key of the web service.

HTTPS = HTTP + Transport Layer Encryption

TLS (Transport Layer Security) is the successor of SSL (Secure Sockets Layer).

- TLS handshake protocol –> after initiating a connection.
  1. **Client hello**: includes TLS version, cipher suits etc.
  2. **Server hello**: selecting the TLS version and cipher suite etc.
  3. Client verifies the serves **X.509 Certificate**.
  4. **Key exchange**: client uses the server's public key to send a **pre-master secret**, which is then used by both client and server to create a session symmetric key.
  5. Send **Finished** messages encrypted with the symmetric key to indicate that the handshake is complete.
- TLS Record protocol –> Manages how data is packaged, transmitted, and unpacked securely. Its primary purpose is to provide **confidentiality**, **integrity**, and **authenticity** to the data being transmitted.
  It is used after the Handshake, and divides the data to be transmitted into small chunks called "**records**". A **Message Authentication Code (MAC)** is added to each record.



Cipher suite: defines how to encrypt the transmitted data, ensure its integrity, and authenticate each other.

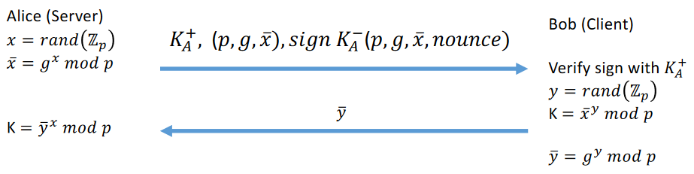
Generating **Symmetric Key** for Data Encryption in TLS:

1. PreMasterSecret generated by client.
2. PreMasterSecret from DH key exchange (provides **PFS**)

PreMasterSecret –> Its just the session key derived from using **PFS.**

Perfect Forward Secrecy (PFS) –> Protect past sessions against future compromises.

- Uses Ephemeral + Authenticated Diffie-Hellman.
  - **Authenticated DH** (Diffie-Hellman)
    - Server Sends public components prime *p*, generator *g*.



This is different because since server sends *p* and *g*.

  - **Ephemeral Diffie-Hellman (EDH)** –> not reusing *x* between sessions.

Session IDs –> enables session reuse.

Session Tickets –> After the full handshake, server asks for a session ticket containing session information and is stored by the client.

TLS and Security Monitoring

- Application data is always encrypted.
- Parameters visible to MiM.
  - IP Address.
  - Preferred ciphers.
  - Server name from certificate

X.509 Certificates

- Issuer information
- Subject information
  - Common Name (CN)
  - Organization (O)
  - Organizational Unit (OU)
  - Country (C)
- Subject public key
- Extensions
- Signature

Certificates are validated against DNS names –> Encoded in **CN** or **Sub Alt** Name fields.

- Standard SSL
  - Valid for a single domain (e.g., www.uit.no)
- Wildcard SSL
  - Valid for multiple hosts (e.g., *.uit.no)
- Multi-Domain SSL
  - Valid for multiple domains (e.g., www.uit.no and www.corporesano.no)

Domain Validation –> Prove that you own and control the DNS domain in your certificate.

- Send a Certificate Signing Request (CSR) to the CA
- Download the issued certificate and install it on the user's Web Server.

14. Passwords

Rainbow table attacks –> Table containing pairs of plaintext passwords and their corresponding hash values.

- Defenses:
  - Salt –> random value added to the plaintext password before hashing. Each user gets a unique salt, which is stored alongside the hash.
  - Pepper –> Similar to Salt, but value is not stored, therefore you have to iterate through every potential pepper value.
    - Costly.

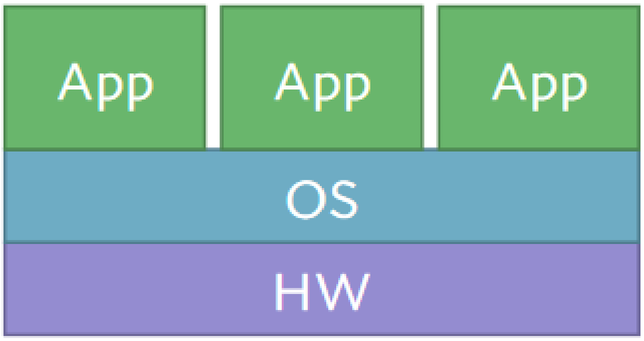
14. Isolation

A type of enforcement that protects the asset against, an attacker that exists within the environment.

Isolation approaches:

- Guard the interface.
  - **Firewall** –> Can restrict both incoming and outgoing network traffic.
  - **Same-origin policy**.
- Protect the interface with a secret.
  - **Tunneling** –> Encryption –> SSH.
- **Translation mapping**
- **Time-multiplexing**

User-space/OS –> User-mode instructions and System-mode instructions.



System calls might undermine process isolation and create *covert channels*. –> Transfer information between processes that are not supposed to be allowed to communicate.

Sandboxing –> Execution environments that impose application-specific restrictions on access to system resources.

Virtual machines –> There may be *covert channels*.

- **Intel SGX** –> protects data while it is actively used in memory.
  - *Enclaves*: running code and memory isolated from the rest of system.
  - Minimum Thread Control Block (TCB): only processor is trusted, nothing else (e.g., the OS is not trusted).
  - All writes to memory are encrypted.
  - Processor prevents access to cached enclave data outside of enclave.
- ○ **Side-channels** –> Attackers might control the OS.
  - Memory access patterns.
  - State of processor caches as enclave executes.
  - State of branch predictor.

15. Surveillance

Surveillance for Accountability –> Record attacks to hold the attacker accountable.

Audit –> Record/log all relevant actions performed by users.

An **intrusion detection system (IDS)** is a software or hardware system that is used to detect signs of malicious activity on a network or individual computers.

- It only detects the attack after it has happened. This can be dangerous if the attack is immediately damaging.

Network IDS (NIDS) –> Detect malicious behavior based on traffic patterns and content.

Host IDS (HIDS) –> Monitor audit files and system logs on a single machine.

How to detect an intrusion:

- **Signature-based detection**: Look for activity that matches the structure of a known attack.
- **Heuristic-based detection**: Develop a model of what normal activity looks like. Flag any activity that deviates from normal activity.

Questions

Authentication –> determination of the identity or role that someone has

- Server to client –> digital certificate, proves that the server is legitimate and it’s the owner of the public key.
- Client to server -> Credentials; password, etc. or tokens.

Authenticity –> The property that data originated from its purported source.

- requires a Public-Key Infrastructures (PKIs)