Apuntes Completos ISDCM (Revisado)

1. Introducción a la Seguridad y Criptografía

1.1. Conceptos Básicos de Seguridad

- Amenazas típicas a la seguridad en comunicaciones: Interceptación, Manipulación, Suplantación (Impersonation) y Repudio (No Repudio - Repudiation).
- Servicios de seguridad: Confidencialidad, Integridad, Autenticación, No Repudio, Control de Acceso, Disponibilidad.

1.2. Criptografía

- Principios de Kerckhoffs: La seguridad de un sistema criptográfico debe residir en la clave, no en el algoritmo.
- Confusión: Cada bit del texto cifrado debe depender de varias partes de la clave. Un pequeño cambio en la clave provoca un gran cambio en el texto cifrado. (No confundir con: "un pequeño cambio en el texto en claro lleve a un gran cambio en la clave", eso es incorrecto).
- Difusión: Si se cambia un bit del texto en claro, aproximadamente la mitad de los bits del texto cifrado deberían cambiar, y viceversa.

1.3. Criptografía Simétrica (Clave Secreta)

- Utiliza la misma clave para cifrar y descifrar.
- Ejemplos: DES (Data Encryption Standard), AES (Advanced Encryption Standard).
  - DES es obsoleto respecto a AES.
  - AES y DES son mecanismos de cifrado por bloques.
- No se puede usar AES para firma digital directamente (se usan algoritmos asimétricos para ello).

1.4. Criptografía Asimétrica (Clave Pública)

- Utiliza un par de claves: una pública (para cifrar o verificar firma) y una privada (para descifrar o firmar).
- RSA:
  - Generación de claves:
    1. Elegir dos números primos grandes, p y q.
    2. Calcular $n = p * q$.
    3. Calcular $\phi(n)=(p-1)(q-1)$.
    4. Elegir un entero e (clave pública) tal que $1<e<\phi(n)$ y $mcd(e, \phi(n)) = 1$.
    5. Calcular d (clave privada) tal que $d \equiv e-1 (mod \phi(n))$.
  - Cifrado: $C=Me(mod n)$.

    - Descifrado: $M = C^d \pmod n$.
    - Firma: $S = (\text{hash}(M))^d \pmod n$ (generalmente se firma el hash del mensaje).
    - Verificación de firma: $\text{hash}(M) = S^e \pmod n$.
    - Computacionalmente, el cálculo de claves en RSA es más costoso que en ElGamal.
- ElGamal:
    - Basado en el problema del logaritmo discreto.
    - Generación de claves:
        1. Elegir un número primo grande p y un generador α del grupo multiplicativo Z_p^*.
        2. Elegir un entero aleatorio a (clave privada) tal que $1 \le a \le p-2$. (La generación es simplemente la elección de este número).
        3. Calcular $KP = \alpha^a \pmod p$ (clave pública). La clave pública KP depende de la clave privada a.
    - Cifrado: Para un mensaje M, el emisor elige un entero aleatorio k (efímero, $1 \le k \le p-2$).
        - $C1 = \alpha^k \pmod p$.
        - $C2 = M \cdot (KP)^k \pmod p$.
        - El par $(C1, C2)$ es el texto cifrado que se envía al receptor. C1 es necesario para el descifrado.
    - Descifrado: $M = C2 \cdot (C1^a)^{-1} \pmod p = C2 \cdot C1^{p-1-a} \pmod p$.
- Funciones Hash:
    - Producen un resumen de longitud fija (hash) a partir de una entrada de longitud variable.
    - Propiedades: Unidireccionalidad (difícil obtener M de H(M)), resistencia a colisiones (débil: difícil encontrar M' con H(M')=H(M) para un M dado; fuerte: difícil encontrar cualquier par M, M' con H(M)=H(M')).
    - Usos: Integridad de datos, firmas digitales (se firma el hash del mensaje).

2. Infraestructura de Clave Pública (PKI)

2.1. Certificados Digitales (X.509)

- Documento electrónico que vincula una clave pública con una identidad.
- Contenido principal: Versión, Número de serie, Algoritmo de firma del certificado, Emisor (Issuer), Periodo de validez (Not Before, Not After), Sujeto (Subject), Información de la clave pública del sujeto (algoritmo, clave pública), Identificadores únicos (opcional), Extensiones (opcional), Firma del emisor sobre todo lo anterior.
- Un certificado sólo incluye la firma de la CA emisora, incluso en PKIs jerárquicas

(no incluye la firma de la CA raíz ni otras CAs del árbol en el propio certificado del sujeto). No incluye la firma del propietario del certificado.

2.2. Autoridad de Certificación (CA)

- Entidad de confianza responsable de emitir y revocar certificados.
- Verifica la identidad de los solicitantes antes de emitir un certificado.

2.3. Modelos de Confianza PKI

- PKI Jerárquica:
  - Estructura de árbol con una CA Raíz (Root CA) en la cima.
  - La CA Raíz emite certificados para CAs subordinadas.
  - La confianza se establece a través de una cadena de certificación (ruta de certificación) hasta la CA Raíz (ancla de confianza).
- Cross-Certification (Certificación Cruzada):
  - Permite que CAs de diferentes dominios (diferentes PKIs) confíen entre sí.
  - Las CAs emiten certificados mutuos.
  - Permite extender el número de certificados aceptados emitiendo certificados para nuevas CAs o CAs de otras jerarquías.
- Bridge CA (Puente de Certificación):
  - Una CA que actúa como un hub para interconectar diferentes PKIs (jerárquicas o no).
  - Establece relaciones de certificación cruzada con las CAs raíz (o CAs principales) de las PKIs que conecta.
  - No necesariamente ahorra tener que añadir nuevas CAs; la Bridge CA es en sí misma una CA adicional. Funciona sobre cualquier tipo de infraestructura PKI.
- Modelo Plano (Plain): Una única CA que emite certificados para todos los usuarios. Su certificado es auto-firmado.
- Lista de Confianza (Trust List Model): El usuario mantiene una lista de CAs en las que confía.

2.4. Validación de Certificados

- Proceso para verificar la autenticidad y validez de un certificado.
- Implica:
  1. Verificar la firma de la CA emisora (usando la clave pública de la CA emisora, obtenida de su propio certificado).
  2. Verificar la validez temporal (que no esté caducado ni sea futuro).
  3. Verificar el estado de revocación.
  4. Construir y validar la ruta de certificación hasta un ancla de confianza (trust anchor). Si el certificado de la CA emisora no es un ancla de confianza, se repite el proceso para el certificado de la CA emisora, y así sucesivamente,

hasta llegar a un ancla de confianza o determinar que no se puede construir una ruta válida.
    - Si un usuario A (con certificado de CA_A) quiere validar un certificado de Usuario B (emitido por CA_B), y CA_A y CA_B pertenecen a jerarquías independientes sin cross-certification o bridge, UserA no podrá validar el certificado de UserB porque no encontrará una ruta de certificación hasta una CA en la que UserA confíe.
- Revocación de Certificados:
    - CRLs (Certificate Revocation Lists): Listas de certificados revocados publicadas periódicamente por la CA. Desventajas: tamaño, latencia.
    - OCSP (Online Certificate Status Protocol): Protocolo para verificar el estado de un certificado en tiempo real.
        - Un cliente OCSP envía una petición a un Respondedor OCSP.
        - El Respondedor OCSP devuelve una respuesta firmada con el estado del certificado (good, revoked, unknown).
        - OCSP no construye la ruta de certificación; solo verifica el estado de un certificado individual. El cliente OCSP es responsable de construir la ruta.
- SCVP (Simple Certificate Validation Protocol):
    - Un servidor SCVP puede construir y validar la ruta de certificación de un certificado dado, además de verificar su estado de revocación, descargando esta tarea del cliente.

2.5. Time Stamping Authority (TSA)

- Autoridad de Sellado de Tiempo.
- Proporciona una prueba de que ciertos datos existían en un momento determinado.
- Proceso:
    1. El solicitante envía un hash de los datos originales (H(M)) a la TSA (por privacidad, no se envía M).
    2. La TSA combina este hash con una marca de tiempo (timestamp, T): (H(M) + T).
    3. La TSA calcula un hash de esta combinación: H(H(M) + T).
    4. La TSA firma este segundo hash con su clave privada. El resultado es el sello de tiempo.

3. Seguridad en la Capa de Transporte (TLS, QUIC)

3.1. TLS (Transport Layer Security)

- Protocolo que proporciona seguridad a las comunicaciones sobre TCP (usado en HTTPS). Sucesor de SSL.

- Objetivos: Confidencialidad (cifrado), Integridad (MAC), Autenticación (certificados).
- Versiones:
  - TLS 1.2 (RFC 5246): Ampliamente utilizado.
  - TLS 1.3 (RFC 8446): Versión más reciente, más rápida y segura. Elimina algoritmos obsoletos, handshake más corto (a menudo 1-RTT), soporta 0-RTT (Zero Round Trip Time resumption).
- CipherSuite: Define los algoritmos criptográficos a usar: algoritmo de intercambio de claves, algoritmo de autenticación, algoritmo de cifrado simétrico, algoritmo MAC. La negociación de la cipher suite es parte del handshake.
  - En TLS 1.3, hay 5 cipher suites especificadas. Una de ellas (`TLS_CHACHA20_POLY1305_SHA256`) utiliza un cifrador de flujo (ChaCha20), las otras usan AES (cifrador por bloques).
- Certificados en TLS 1.3: Son obligatorios para el servidor. El certificado del cliente es opcional (solicitado por el servidor si se requiere autenticación mutua).
- Handshake TLS 1.2 (simplificado):
  1. `ClientHello`: El cliente envía algoritmos soportados, nonce.
  2. `ServerHello`: El servidor elige CipherSuite, envía su certificado, nonce. Puede solicitar certificado del cliente (`CertificateRequest`).
  3. `Certificate` (Servidor): Certificado del servidor.
  4. `ServerKeyExchange` (opcional): Si la clave pública del certificado no es suficiente para el intercambio de claves (e.g., DHE).
  5. `ServerHelloDone`: Servidor termina su parte inicial.
  6. `Certificate` (Cliente, si solicitado): Certificado del cliente.
  7. `ClientKeyExchange`: Cliente envía información para generar la clave simétrica (e.g., pre-master secret cifrado con clave pública del servidor, o parámetros DH del cliente).
  8. `CertificateVerify` (Cliente, si envió certificado): Firma para verificar posesión de clave privada.
  9. `ChangeCipherSpec` (Cliente): Indica que comenzará a usar los parámetros negociados.
  10. `Finished` (Cliente): Mensaje cifrado para verificar el handshake.
  11. `ChangeCipherSpec` (Servidor): Indica que comenzará a usar los parámetros negociados.
  12. `Finished` (Servidor): Mensaje cifrado para verificar el handshake.
  13. Intercambio de datos de aplicación cifrados.
- Handshake TLS 1.3 (simplificado):
  - Más eficiente, a menudo 1-RTT.
  - `ClientHello`: Cliente propone CipherSuites, versiones, parámetros para

intercambio de claves (e.g., claves Diffie-Hellman), puede incluir datos para 0-RTT.
    - `ServerHello`: Servidor elige parámetros, envía su parte de la clave DH, certificado, `Finished`.
    - Cliente verifica, envía `Finished`.
    - El intercambio de claves suele ser (EC)DHE (Diffie-Hellman Efímero). RSA para intercambio de claves está obsoleto.
    - Diffie-Hellman es un protocolo de acuerdo de claves. La clave simétrica no se cifra con DH, sino que se deriva del secreto compartido obtenido mediante DH.
- 0-RTT (Zero Round Trip Time Resumption): En TLS 1.3, permite al cliente enviar datos en el primer mensaje si ha habido una sesión previa (usando Pre-Shared Keys - PSK).

3.2. QUIC (Quick UDP Internet Connections)

- Nuevo protocolo de transporte sobre UDP.
- Características:
    - Multiplexación de flujos sin bloqueo HOL (Head-of-Line blocking).
    - Control de flujo por flujo.
    - Conexiones de baja latencia (0-RTT o 1-RTT).
    - Cifrado integrado (seguridad equivalente a TLS 1.3). No necesita añadir TLS encima.
    - Migración de conexión (mantiene la conexión si cambia la IP/puerto del cliente).
- HTTP/3 utiliza QUIC como capa de transporte y mantiene las características de HTTP/2.

4. Seguridad en Correo Electrónico (S/MIME)

4.1. S/MIME (Secure / Multipurpose Internet Mail Extensions)

- Estándar para añadir seguridad a los mensajes MIME (correo electrónico).
- Servicios:
    - Confidencialidad: Cifrado del contenido del mensaje (EnvelopedData).
    - Integridad: Asegura que el mensaje no ha sido modificado (SignedData).
    - Autenticación del Origen: Verifica la identidad del remitente (SignedData).
    - No Repudio del Origen: El remitente no puede negar haber enviado el mensaje (SignedData).
- Utiliza criptografía de clave pública (certificados X.509) y criptografía simétrica.
- Basado en PKCS#7 / CMS (Cryptographic Message Syntax).
- Tipos de contenido S/MIME:

- EnvelopedData: Contiene datos cifrados y la información para descifrarlos (claves simétricas cifradas con la clave pública del destinatario). No incluye firma.
- SignedData: Contiene datos firmados (hash del contenido firmado con la clave privada del remitente) y el certificado del firmante. Se recomienda firmar sobre un digest.
- SignedAndEnvelopedData: Combina ambos.
- AuthenticatedData: Proporciona autenticación e integridad mediante MACs.

5. Seguridad XML (XML Encryption, XML Signature)

5.1. XML Encryption

- Recomendación del W3C para cifrar datos (XML o arbitrarios) y representar el resultado en XML.
- Elemento <EncryptedData>:
  - Reemplaza el elemento o contenido cifrado en el documento original, o es el elemento raíz de un nuevo documento si se cifra el documento completo.
  - Atributos: Type (indica qué se cifró, e.g., http://www.w3.org/2001/04/xmlenc#Element o http://www.w3.org/2001/04/xmlenc#Content), MimeType.
  - Hijos:
    - <EncryptionMethod> (opcional): Algoritmo usado para el cifrado (e.g., AES-256-CBC).
    - <ds:KeyInfo> (opcional, de XML Signature): Información para obtener la clave de descifrado. Puede contener <EncryptedKey> (clave simétrica cifrada con clave pública del receptor) o <RetrievalMethod>.
    - <CipherData>: Contiene los datos cifrados.
      - <CipherValue>: Datos cifrados directamente (codificados en Base64).
      - <CipherReference URI="...">: URI que apunta a los datos cifrados (que pueden estar en un documento externo o en otra parte del mismo documento).
- Proceso de cifrado:
  1. Se genera una clave simétrica (clave de sesión).
  2. Se cifran los datos con la clave simétrica.
  3. Se cifra la clave simétrica con la clave pública del destinatario (o se usa una clave precompartida).
  4. Se construye el elemento <EncryptedData>.
- Se puede cifrar un elemento XML completo, el contenido de un elemento XML, o datos arbitrarios. El contenido cifrado (resultado de la encriptación) no siempre se incluye directamente en el <EncryptedData>; puede ser referenciado.

5.2. XML Signature

- Recomendación del W3C para firmar datos (XML o arbitrarios) y representar la firma en XML.
- Proporciona integridad, autenticación del firmante y no repudio.
- Elemento `<Signature>`:
  - Elemento raíz de la firma.
  - Atributo `Id` (opcional).
  - Hijos:
    - `<SignedInfo>`: Información que realmente se firma. Su canonicalización es lo que se firma.
      - `<CanonicalizationMethod>`: Algoritmo para la canonicalización del `<SignedInfo>` antes de firmar (e.g., `http://www.w3.org/2001/10/xml-exc-c14n#`). La canonicalización asegura una representación en bytes consistente.
      - `<SignatureMethod>`: Algoritmo usado para la firma (e.g., `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`).
      - Uno o más elementos `<Reference>`:
        - Atributo `URI`: Identifica el recurso a firmar (puede ser "" para el documento contenedor, "#id" para un elemento local, o una URL externa).
        - `<Transforms>` (opcional): Secuencia de transformaciones aplicadas al recurso antes de calcular el digest (e.g., Canonicalización, XPath, Enveloped Signature Transform `http://www.w3.org/2000/09/xmldsig#enveloped-signature`).
        - `<DigestMethod>`: Algoritmo de hash usado (e.g., `http://www.w3.org/2001/04/xmlenc#sha256`).
        - `<DigestValue>`: Valor del hash (Base64) del recurso (transformado).
    - `<SignatureValue>`: Valor de la firma (Base64) de la canonicalización del `<SignedInfo>`.
    - `<KeyInfo>` (opcional): Información sobre la clave de verificación (e.g., certificado X.509, clave pública, nombre de la clave).
    - `<Object>` (opcional): Permite incluir los datos firmados dentro de la propia firma (firma envolvente).
- Tipos de Firma:
  - Enveloped (Envainada): La firma (`<Signature>`) está dentro del XML que firma (e.g., `<Signature>` es hijo del elemento raíz del documento firmado). Se necesita la transformación "Enveloped Signature Transform".
  - Enveloping (Envolvente): El XML firmado está dentro de la firma (en un elemento `<Object>`).

    - Detached (Separada): La firma y los datos firmados están separados (la firma referencia los datos mediante URI).
- En todos los casos (enveloped, enveloping, detached), el elemento `<Signature>` se incluye en algún documento XML (ya sea el mismo que se firma o uno separado).

6. Control de Acceso y Privacidad (XACML)

6.1. Conceptos de Privacidad

- PII (Personally Identifiable Information): Información que puede usarse para identificar a un individuo (e.g., nombre, DNI, datos médicos).
- PET (Privacy Enhancing Technologies): Tecnologías para proteger la privacidad eliminando o reduciendo PII o previniendo su recolección no deseada.
- GDPR (General Data Protection Regulation):
    - Privacy by Design: Integrar la protección de datos en el diseño de sistemas y procesos.
    - Data Minimization: Recoger y procesar solo los datos personales estrictamente necesarios para un fin específico. No se refiere al formato de almacenamiento (binario vs ASCII), sino a la cantidad de información.

6.2. XACML (eXtensible Access Control Markup Language)

- Lenguaje basado en XML (estándar de OASIS) para definir políticas de control de acceso y un modelo de procesamiento para evaluar peticiones de acceso.
- Arquitectura XACML:
    - PEP (Policy Enforcement Point): Punto donde se fuerza la decisión de acceso. Intercepta la petición, la envía al PDP, y aplica la decisión.
    - PDP (Policy Decision Point): Evalúa las políticas aplicables contra la petición y retorna una decisión (Permit, Deny, Indeterminate, NotApplicable). Necesita información del PAP (políticas) y del PIP (atributos).
    - PIP (Policy Information Point): Proporciona atributos adicionales sobre el sujeto, recurso, acción o entorno que el PDP pueda necesitar.
    - PAP (Policy Administration Point): Donde se crean y gestionan las políticas.
    - PRP (Policy Retrieval Point): Donde el PDP obtiene las políticas.
- Componentes de una Política XACML:
    - `<Request>`: Contiene los atributos de la petición de acceso:
        - `<Attributes Category="...">`: Define atributos para el sujeto (`urn:oasis:names:tc:xacml:1.0:subject-category:access-subject`), recurso (`urn:oasis:names:tc:xacml:3.0:attribute-category:resource`), acción (`urn:oasis:names:tc:xacml:3.0:attribute-category:action`), entorno.
        - `<Attribute AttributeId="..." DataType="...">`: Define un atributo específico con su valor.

    - <Response>: Contiene la decisión del PDP.
        - <Result>: Decisión (Permit, Deny, Indeterminate, NotApplicable), Status, Obligations, Advice.
    - <PolicySet>: Contenedor de otras <Policy> o <PolicySet>.
        - Atributos: PolicySetId, Version, PolicyCombiningAlgId.
        - <Target>: Define a qué peticiones aplica este PolicySet.
    - <Policy>: Contenedor de <Rule>.
        - Atributos: PolicyId, Version, RuleCombiningAlgId.
        - <Target>: Define a qué peticiones aplica esta Policy.
    - <Rule>: Unidad básica de una política.
        - Atributos: RuleId, Effect ("Permit" o "Deny"). El Effect indica la consecuencia si la regla se cumple (Target y Condition son verdaderas). Si Effect="Permit" y la regla se cumple, se permite la acción. Si Effect="Deny" y la regla se cumple, se deniega.
        - <Target>: Define a qué peticiones aplica esta Rule.
        - <Condition> (opcional): Expresión booleana que debe evaluarse a true para que la regla aplique su efecto. Usa funciones y atributos.
    - <Target>:
        - Contiene <AnyOf> elementos, que a su vez contienen <AllOf> elementos, que contienen <Match> elementos.
        - <Match MatchId="...">: Compara un valor de atributo de la petición con un valor dado en la política.
            - MatchId: Función de comparación (e.g., urn:oasis:names:tc:xacml:1.0:function:string-equal, urn:oasis:names:tc:xacml:1.0:function:anyURI-equal).
            - <AttributeValue DataType="...">: Valor literal a comparar (definido en la política).
            - <AttributeDesignator Category="..." AttributeId="..." DataType="..." MustBePresent="false/true">: Especifica el atributo de la <Request> a usar en la comparación.
- Algoritmos de Combinación (Combining Algorithms):
    - Para PolicySets (PolicyCombiningAlgId) y Policies (RuleCombiningAlgId).
    - Ejemplos: urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides, permit-overrides, first-applicable.
- Obligations y Advice:
    - <ObligationExpressions> / <AdviceExpressions>: Acciones que el PEP debe realizar si la política/regla resulta en Permit/Deny.
- Uso en eHealth/Genomics:
    - Definir políticas granulares para acceso a información médica sensible.

- Roles (e.g., "researcher", "doctor"), acciones (e.g., "getData", "viewRecord", "Modify", "Print"), recursos (e.g., "file.mgg#chromosome3", "urn:hospital:lab_result").

6.3. SAML en XACML

- Las aserciones SAML pueden ser usadas por el PIP para proveer atributos al PDP durante la evaluación de una política XACML. El PEP puede recibir la petición de acceso con aserciones SAML.

7. Protocolos Específicos de Seguridad Web

7.1. SAML (Security Assertion Markup Language)

- Estándar XML para intercambiar datos de autenticación y autorización (aserciones) entre dominios de seguridad.
- Actores Principales:
  - Principal (User): El usuario que intenta acceder a un recurso. Se identifica ante el IdP.
  - IdP (Identity Provider): Entidad que autentica al Principal y emite Aserciones SAML.
  - SP (Service Provider): Entidad que provee el servicio y confía en el IdP para la autenticación/autorización.
- Aserciones SAML: Declaraciones hechas por un IdP, expresadas en XML. Incluyen una firma para integridad y autenticación.
  - Authentication Assertion: Declara que el Principal fue autenticado en un momento específico por un método particular.
  - Attribute Assertion: Declara atributos asociados al Principal.
  - Authorization Decision Assertion: Declara si una petición de acceso a un recurso específico es permitida o denegada.
- Protocolos SAML: Definen cómo se solicitan y obtienen las aserciones (e.g., `AuthnRequest` Protocol).
- Bindings SAML: Especifican cómo se encapsulan los mensajes SAML en protocolos de transporte estándar. La comunicación entre IdP y SP puede ser a través del navegador del usuario.
  - HTTP Redirect Binding: Mensajes SAML (deflated, base64-encoded, URL-encoded) en parámetros GET de URL (típicamente para `AuthnRequest` del SP al IdP).
  - HTTP POST Binding: Mensajes SAML (base64-encoded) en un formulario HTML auto-enviado (típicamente para `Response` del IdP al SP).
  - SOAP Binding: Mensajes SAML en el cuerpo de mensajes SOAP.
- Perfiles SAML: Casos de uso concretos.

    - Web Browser SSO Profile (Single Sign-On):
        - SP-Initiated: Usuario accede a SP -> SP redirige a IdP con `AuthnRequest` (e.g., vía HTTP Redirect) -> Usuario se autentica en IdP -> IdP envía `Response` (con aserción) a SP (via navegador, e.g., vía HTTP POST) -> SP valida aserción y da acceso.
        - IdP-Initiated: Usuario se autentica en IdP -> IdP envía `Response` (con aserción) a SP -> SP valida aserción y da acceso.

7.2. OAuth 2.0

- Framework de autorización abierto. Permite a aplicaciones de terceros acceder a recursos HTTP en nombre del propietario del recurso, sin compartir las credenciales (contraseña) del propietario del recurso con la aplicación cliente.
- Roles:
    - Resource Owner (RO): El usuario que autoriza el acceso a sus recursos.
    - Client: La aplicación que solicita acceso a los recursos del RO.
    - Authorization Server (AS): Emite tokens de acceso al Client después de autenticar al RO y obtener su autorización. Puede ser la misma entidad que el Resource Server.
    - Resource Server (RS): Servidor que aloja los recursos protegidos del RO. Acepta y valida tokens de acceso.
- Tipos de Tokens:
    - Access Token: Credencial usada por el Client para acceder a los recursos protegidos. Vida corta.
    - Refresh Token (opcional): Credencial usada para obtener nuevos Access Tokens cuando el actual expira. Vida más larga.
- Grant Types (Flujos de Autorización):
    - Authorization Code Grant: Para aplicaciones web con backend.
        1. Client redirige RO al AS con una `Authorization Request` (incluye `redirect_uri`, `scope`).
        2. RO se autentica y autoriza al Client en el AS.
        3. AS redirige RO de vuelta al Client (al `redirect_uri`) con un `authorization_code` (esto es un tipo de `Authorization Grant`).
        4. Client intercambia `authorization_code` por un `access_token` (y opcionalmente `refresh_token`) directamente con el AS (en el Token Endpoint).
    - Implicit Grant (obsoleto para nuevas implementaciones): Para aplicaciones de una sola página (SPA) o móviles (sin backend). Access token se devuelve directamente en la redirección (menos seguro).
    - Resource Owner Password Credentials Grant: RO proporciona sus credenciales directamente al Client. Usar con precaución, solo para clientes

de confianza.
    - Client Credentials Grant: Para acceso a recursos propios del Client (comunicación máquina a máquina).
- Scopes: Limitan el alcance del acceso que el Client solicita (e.g., "read_profile", "post_tweet"). Se incluyen en la "Authorization Request".

7.3. OpenID Connect (OIDC)

- Capa de identidad construida sobre OAuth 2.0. Permite a los Clients verificar la identidad del End-User basándose en la autenticación realizada por un Authorization Server (que también actúa como OpenID Provider - OP).
- Añade el concepto de ID Token.
- ID Token: Un JWT (JSON Web Token) que contiene claims (información) sobre el End-User y su autenticación (e.g., `iss`, `sub`, `aud`, `exp`, `iat`). Es firmado por el AS/OP.
- UserInfo Endpoint: Un endpoint protegido por OAuth donde el Client puede obtener claims adicionales sobre el End-User usando el Access Token.
- Flujos similares a OAuth 2.0 (e.g., Authorization Code Flow, Implicit Flow, Hybrid Flow).

7.4. JSON Web Token (JWT)

- Estándar abierto (RFC 7519) para crear tokens de acceso que afirman un número de claims. Compacto y URL-safe, codificado en Base64Url. Es una secuencia de caracteres ASCII.
- Comúnmente usado como Access Token en OAuth 2.0 y como ID Token en OpenID Connect.
- Estructura (JWS - JSON Web Signature): `header.payload.signature`
    - Header (JOSE Header): Objeto JSON (Base64Url encoded) que describe el token y las operaciones criptográficas.
        - `alg`: Algoritmo de firma (e.g., `HS256` - HMAC SHA-256, `RS256` - RSA SHA-256). `none` es posible pero no recomendado.
        - `typ`: Tipo de token, usualmente "JWT".
    - Payload: Objeto JSON (Base64Url encoded) que contiene los claims.
        - Registered Claims: `iss` (issuer), `sub` (subject), `aud` (audience), `exp` (expiration time), `nbf` (not before), `iat` (issued at), `jti` (JWT ID).
        - Public Claims: Definidos por quienes usan JWTs, deben evitar colisiones (e.g., registrándolos en IANA o usando URIs).
        - Private Claims: Claims personalizados entre partes que acuerdan su uso.
    - Signature (Opcional):
        - Para `HS256`: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`.

        - Para RS256: Firma RSA del base64UrlEncode(header) + "." + base64UrlEncode(payload) usando la clave privada del emisor.
- La firma (si está presente) asegura la integridad y, en caso de algoritmos asimétricos, la autenticidad del emisor. Header y Payload son obligatorios.

7.5. JSON Web Encryption (JWE)

- Estándar (RFC 7516) para representar contenido cifrado usando JSON y Base64Url. Estandarizado por IETF (no W3C).
- Estructura: protected_header.encrypted_key.iv.ciphertext.authentication_tag (concatenados por ".")
    - Protected Header (JOSE Header): Objeto JSON (Base64Url encoded) con parámetros sobre el cifrado y la clave. Es obligatorio.
    - Encrypted Key: Clave de cifrado de contenido (CEK) cifrada con la clave pública del receptor o derivada de un secreto compartido. Es obligatorio.
    - Initialization Vector (IV): Vector de inicialización para el cifrado simétrico. Obligatorio.
    - Ciphertext: Contenido cifrado. Obligatorio.
    - Authentication Tag: Para verificar la integridad y autenticidad del texto cifrado (en cifrados AEAD como AES GCM). Obligatorio.
- Algoritmos de Cifrado de Clave (Key Encryption): RSAES-OAEP, AES Key Wrap (A128KW, A256KW), ECDH-ES.
- Algoritmos de Cifrado de Contenido (Content Encryption): AES GCM (A128GCM, A256GCM), AES CBC HMAC SHA2 (A128CBC-HS256, A256CBC-HS512).

8. Gestión de Derechos Digitales (DRM)

8.1. Propiedad Intelectual Multimedia

- Derechos Morales: Paternidad, integridad, divulgación, retirada. (Irrenunciables e inalienables, no se pueden vender ni negociar su exclusividad).
- Derechos Patrimoniales (Económicos): Reproducción, distribución, comunicación pública, transformación. (Exclusivos, remunerados, transferibles, se pueden vender).
- Derechos Conexos: Artistas intérpretes o ejecutantes, productores de fonogramas, entidades de radiodifusión.

8.2. MPEG-21 REL (Rights Expression Language)

- Lenguaje basado en XML (Parte 5 de MPEG-21) para expresar derechos y condiciones para el uso de contenido digital.
- Concepto central: Licencia (<license>).
- Componentes clave de una licencia:

- <grant>: Otorga derechos. Identifica un recurso, qué operaciones están permitidas sobre él (y bajo qué condiciones) y para quién es el "grant".
  - <principal>: A quién se le otorgan los derechos (usuario, grupo).
  - <right>: El derecho otorgado (e.g., <play>, <copy>, <print>).
  - <resource>: El contenido digital al que aplica el derecho.
  - <condition>: Condiciones que deben cumplirse para ejercer el derecho (e.g., <validityInterval> para tiempo, <count> para número de usos, <territory> para ubicación).
- <issuer>: Entidad que emite la licencia.

8.3. MPEG-21 CEL (Contract Expression Language)

- Lenguaje basado en XML (Parte 20 de MPEG-21) para la expresión de contratos digitales relacionados con contenido multimedia.

8.4. DRM y Navegadores Web

- EME (Encrypted Media Extensions):
  - Estándar W3C que permite a las aplicaciones web reproducir contenido multimedia cifrado.
  - No define un sistema DRM, sino una API común para que el navegador interactúe con Módulos de Descifrado de Contenido (CDM).
  - CDM (Content Decryption Module): Software (a menudo propietario y específico del navegador/OS) que gestiona las claves y descifra el contenido.
  - Key System: Identifica un sistema DRM específico (e.g., Widevine, PlayReady, FairPlay).
  - Proceso típico:
    1. La aplicación web detecta contenido cifrado (e.g., a través de MSE).
    2. Solicita acceso a un Key System disponible.
    3. Crea una MediaKeySession (sesión de claves).
    4. La aplicación obtiene datos de inicialización (initData) del medio.
    5. Pasa initData a la MediaKeySession para generar una petición de licencia (license request).
    6. El CDM genera la petición de licencia.
    7. La aplicación envía esta petición a un servidor de licencias del proveedor de contenido.
    8. El servidor de licencias devuelve una licencia.
    9. La aplicación pasa la licencia al CDM.
    10. El CDM procesa la licencia, obtiene las claves de descifrado y las usa para descifrar el contenido multimedia.
- MSE (Media Source Extensions):

- Estándar W3C que extiende `HTMLMediaElement` para permitir a JavaScript generar flujos de medios para reproducción.
- Permite a las aplicaciones gestionar el búfer de datos de audio/video/texto.
- Define un objeto `MediaSource` con uno o más objetos `SourceBuffer`.
- Las aplicaciones pueden añadir segmentos de datos a los `SourceBuffer`.
- Usado junto con EME para reproducir contenido protegido: MSE gestiona los segmentos de medios (cifrados o no) y EME/CDM se encarga del descifrado.