

Grado en Ingeniería Informática

Sistemas distribuidos

# Diseño de arquitecturas distribuidas

Departamento de Tecnología Informática y Computación

2023 – 2024



# contenido de teoría

## TEORÍA

Nº  
**CLASES**

- |  |     |
|--|-----|
| <b>1. Fundamentos de la computación distribuida</b>  | 1   |
| <b>2. Diseño de arquitecturas distribuidas</b>       | 2   |
| <b>3. Tecnologías web y middleware</b>               | 2   |
| <b>4. Seguridad</b>                                  | 4   |
| <b>5. Coordinación y control de tiempo en los SD</b> | 2,5 |
| <b>6. Archivo distribuido. Caso blockchain</b>       | 2,5 |

# 1. Introducción

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## ¿Qué es un sistema distribuido?

Es un entorno informático compuesto de **varios componentes** que se distribuyen entre numerosas computadoras (u otros dispositivos informáticos) en una misma **red de comunicaciones**.

Estos componentes son **independientes** y coordinan **esfuerzos** para completar un trabajo de manera más **eficiente** que si un solo dispositivo hubiese sido el responsable de la tarea.

La **computación distribuida** es el campo de la informática que estudia los sistemas distribuidos.

## 2. Fundamentos de la computación distribuida

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

### ¿Cómo funciona un sistema distribuido?

Las funciones más importantes de un sistema distribuido son:

- Heterogeneidad
- Conurrencia
- Escalabilidad
- Tolerancia a fallos
- Recursos compartidos
- Seguridad

## 2. Fundamentos de la computación distribuida

1. Introducción

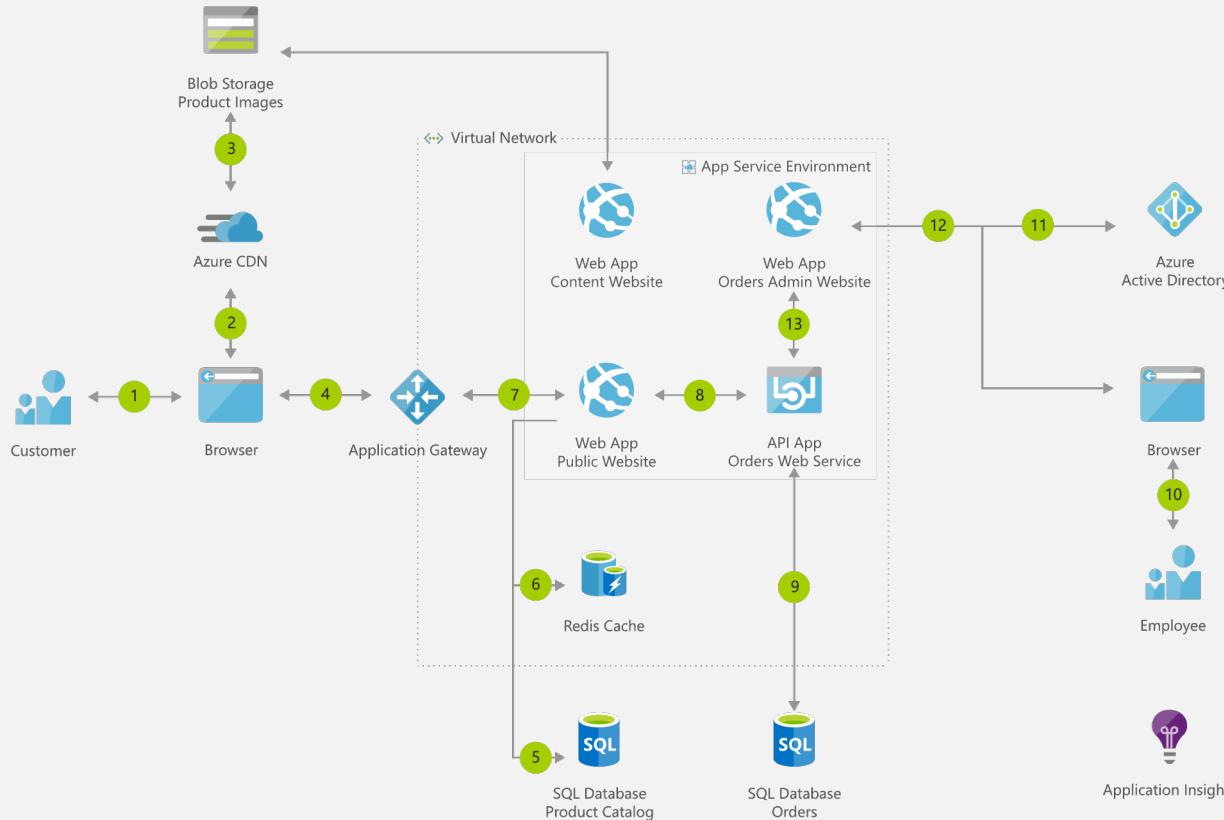
2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud



## 2. Fundamentos de la computación distribuida

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

### Ejemplos de sistema distribuidos

- Redes de computadores (1970)
- Redes de telecomunicaciones: VoIP
- Procesamiento paralelo: sistemas en la nube
- Bases de datos distribuidas: Cassandra o MongoDB
- Sistemas de tiempo real: control aéreo, logística o comercio electrónico

## 2. Fundamentos de la computación distribuida

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

### Beneficios de un sistema distribuido

El objetivo principal de un sistema distribuido es permitir la **escalabilidad**, el **rendimiento** y la alta disponibilidad de las aplicaciones.

Los **principales beneficios** incluyen:

- **Escalado horizontal ilimitado:** se pueden agregar máquinas cuando sea necesario.
- **Baja latencia:** al tener máquinas ubicadas geográficamente más cerca de los usuarios se reducen los tiempos de atención.
- **Tolerancia a fallos:** si un servidor o centro de datos falla el resto puede seguir dando servicio.

## 2. Fundamentos de la computación distribuida

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

### Inconvenientes de un sistema distribuido

La mayor desventaja de un sistema distribuido es la **complejidad**.

Más máquinas y más flujo de datos generan los siguientes problemas:

- **Integración y coherencia de los datos:** la capacidad de sincronizar el orden de los cambios en los datos cuando los nodos se están iniciando, deteniendo o fallando.
- **Errores de redes y comunicaciones:** mensajes de error que no se envían en el orden correcto pueden provocar fallos en la comunicación y la funcionalidad.
- **Gastos generales:** más componentes implica destinar más recursos a tareas de balanceo de carga, supervisión, monitorización y visualización.

# 3. Arquitecturas distribuidas

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Tipos de arquitecturas distribuidas

- Cliente - Servidor
- Peer-to-Peer (P2P)
- Middleware orientado a mensajes
- Cluster y grid
- Arquitectura Orientada a Servicios (SoA)

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Cliente - Servidor

Servidor como recurso compartido para múltiples clientes.

Ejemplos: impresora, base de datos o servidor web.

El cliente decide cuándo usar un recurso compartido, cómo usarlo y mostrarlo, e incluso cambiar los datos del recurso y enviarlo de nuevo al servidor.

Ejemplo: un repositorio de código.

El servidor es un proveedor de un **servicio en espera pasiva** y el cliente solicita el servicio.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Cliente - Servidor

- Paradigma de paso de mensajes: **petición - respuesta.**
- Permite **abstraer el acceso remoto a los recursos.**
- Gestión centralizada
- Mecanismos de sesión, concurrencia y escalabilidad
- Cuellos de botella

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

## Peer-to-Peer (P2P)

No existe una máquina centralizada o especializada que haga el trabajo pesado.

Toda la toma de decisiones y responsabilidades se dividen por igual entre las máquinas involucradas y todas ellas pueden asumir roles de cliente o servidor.

Ejemplos: Blockchain.

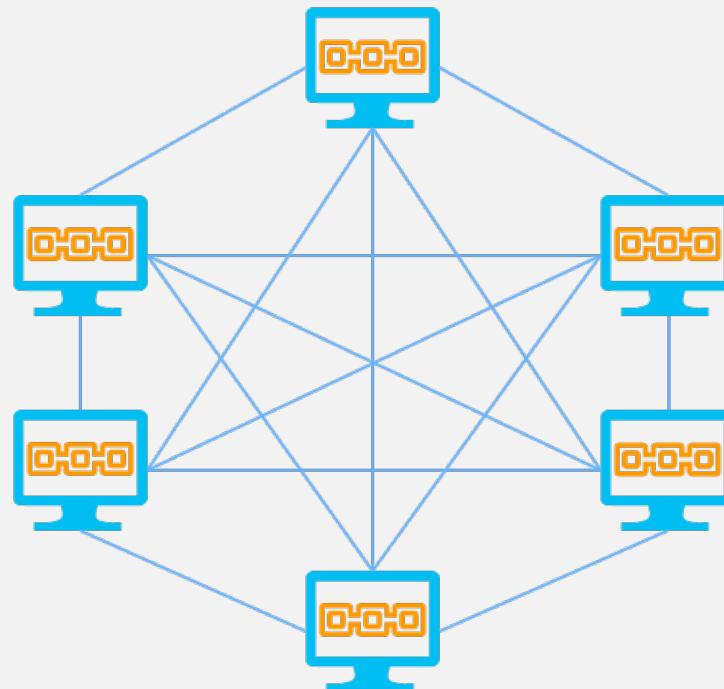
Existe una gestión distribuida del recurso. Implica menor control sobre el recurso.

# 3. Arquitecturas distribuidas

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Peer-to-Peer (P2P)



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Middleware orientado a mensajes

Permite a las aplicaciones distribuidas comunicarse mediante el **envío de mensajes**. El middleware se encarga de que **todos los mensajes lleguen a su destino**.

**La comunicación es asíncrona.**

Existe dos modelos de este tipo de middleware:

- Punto a punto
- Publicación / suscripción

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Middleware orientado a mensajes

### Punto a punto

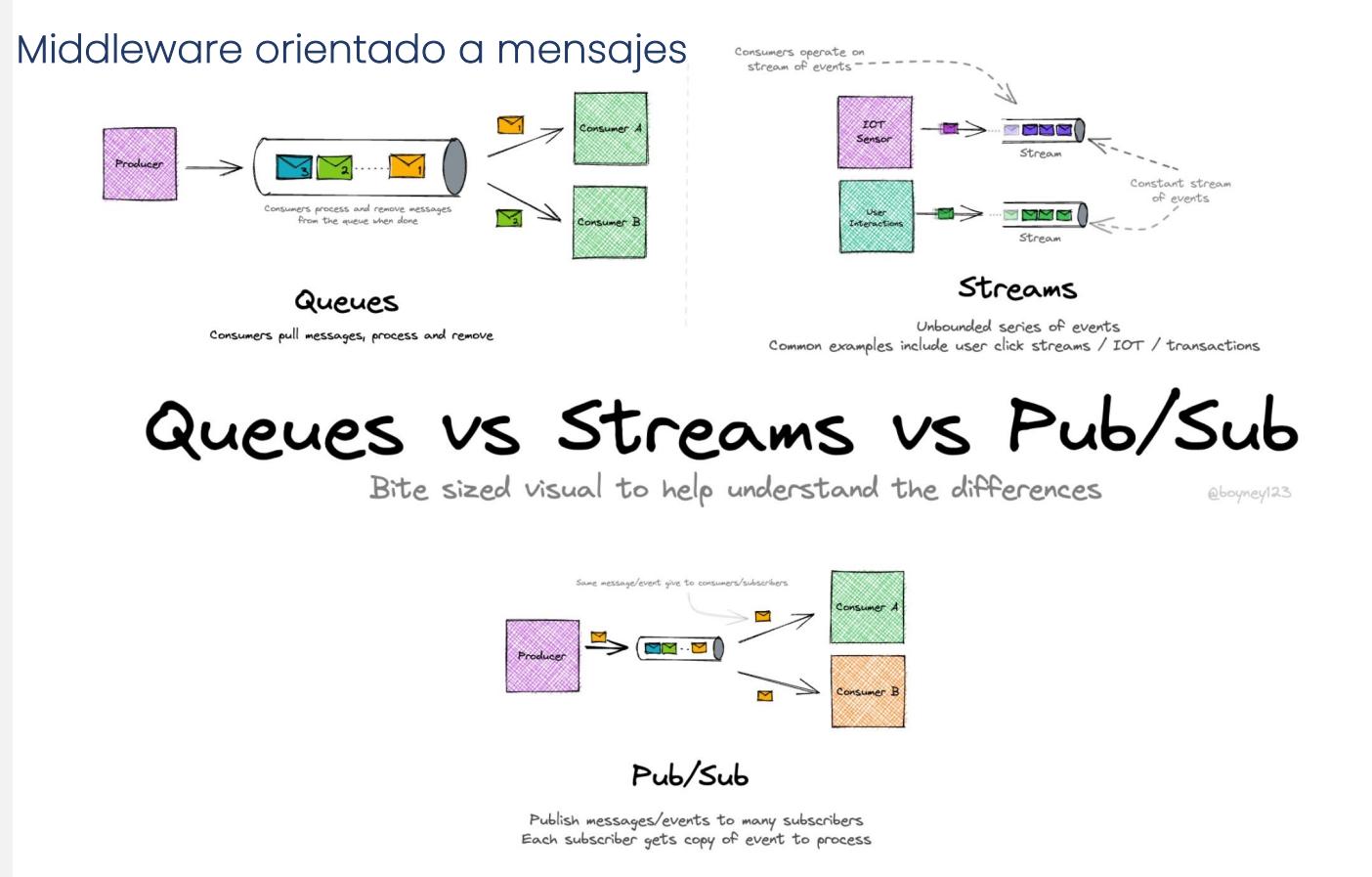
- Los mensajes van dirigidos a un único receptor.
- El mensaje queda almacenado en la cola hasta que el receptor pueda o quiera leerlo.

### Publicación / suscripción

- Emisores de información y suscriptores o consumidores de esa información.
- Los consumidores pueden suscribirse a un determinado tipo de mensajes.
- Los emisores envían mensajes al middleware y él se encarga de hacerlo llegar a los suscriptores.

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud

### 3. Arquitecturas distribuidas



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Cluster y grid

Infraestructura para ofrecer **mayor capacidad de procesamiento y almacenamiento**.

Un **cluster** es un sistema distribuido compuesto por una **colección de sistemas autónomos** interconectados mediante redes privadas y fuertemente enlazados que se emplea como **un recurso computacional unificado**.

Un **grid** es una arquitectura conceptualmente similar al cluster pero que además **permite compartir recursos no unidos geográficamente** para resolver problemas a gran escala.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Cluster y grid

La gran diferencia es que un **cluster** es homogéneo mientras que el **grid** es heterogéneo.

Las computadoras de un **grid** pueden ejecutar diferentes sistemas operativos y tener distinto hardware mientras que en un **cluster** tienen mismo hardware y sistema operativo.

A diferencia de la computación paralela un proyecto de computación con **grid** no tiene la dependencia de tiempo asociada. Las máquinas se utilizan cuando están inactivas para ejecutar el cómputo necesario y además se pueden ejecutar tareas ajenas al **grid**.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura orientada a servicios

Permite la **abstracción** de diferentes **procesos de negocio** denominados **servicios**.

Los servicios se prestan a otros servicios a través de un protocolo de comunicación.

### Componentes

- Proveedor de servicios
- Consumidor de servicios
- Servicio de registro

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

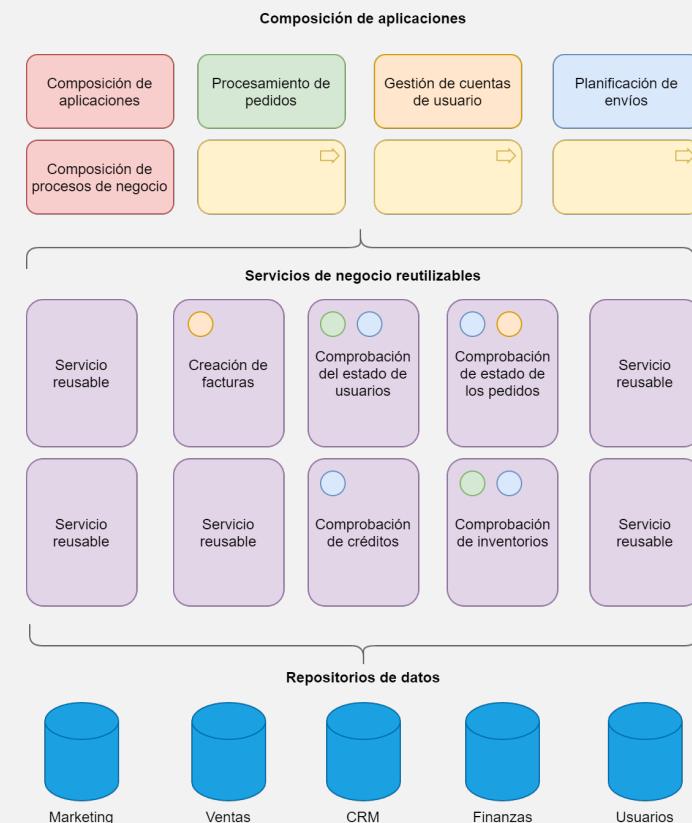
6. Cloud

### 3. Arquitecturas distribuidas

#### Arquitectura orientada a servicios

##### Principios de la arquitectura

- Localización, descubrimiento y publicación
- Interoperabilidad
- Composición
- Autonomía y autocontenidos
- Reusabilidad
- Desacoplamiento
- Contrato bien definido
- Sin estado



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

Una arquitectura de microservicios es un **enfoque flexible y escalable** para el diseño de software.

### ¿Qué es un microservicio?

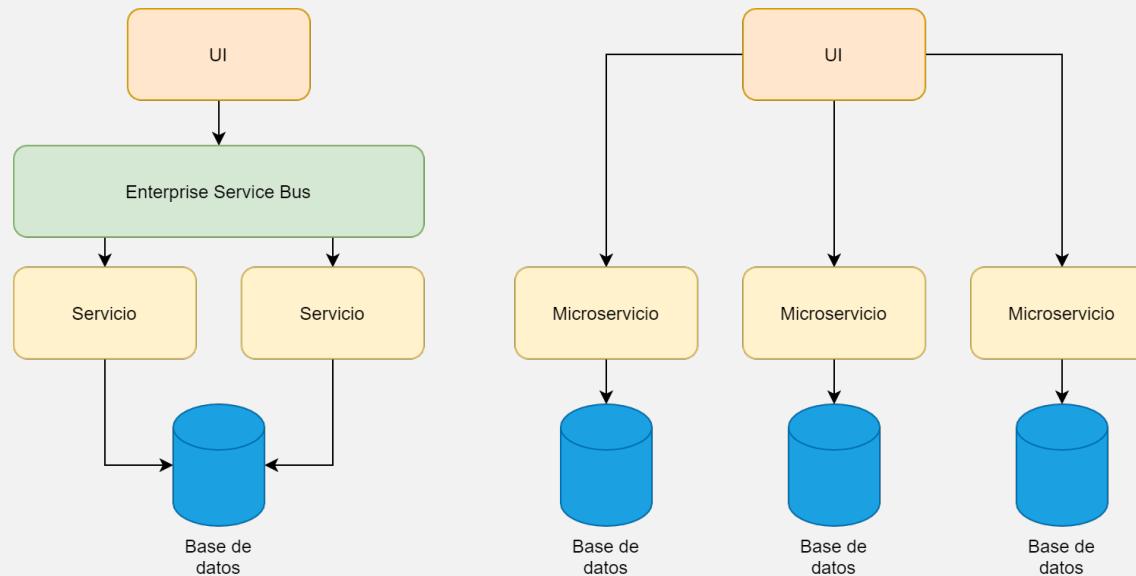
Es un servicio de **propósito único, altamente cohesionado y descentralizado**.

- El microservicio debe estar centrado en uno y sólo un único propósito. También debe estar centrado en el dominio y el objetivo.
- El microservicio debe ser autosuficiente en términos de requisitos de dominio e infraestructura de dominio. Debe tener todas las características que necesita para cumplir un único propósito.
- El microservicio debe de estar descentralizado de otros servicios e infraestructura desde un punto de vista lógico. Un cambio en el microservicio no debe involucrar cambios en ningún otro microservicio.

# 3. Arquitecturas distribuidas

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Arquitectura de microservicios



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

¿Cuál es la diferencia entre arquitectura orientada a servicios y microservicios?

La principal diferencia es el **alcance o scope**.

- Una arquitectura orientada a servicios tiene un **alcance empresarial**.
- Una arquitectura de microservicios tiene un **alcance de aplicación**.

Muchos de los enfoques básicos de cada arquitectura son **incompatibles** si se ignora la diferencia.

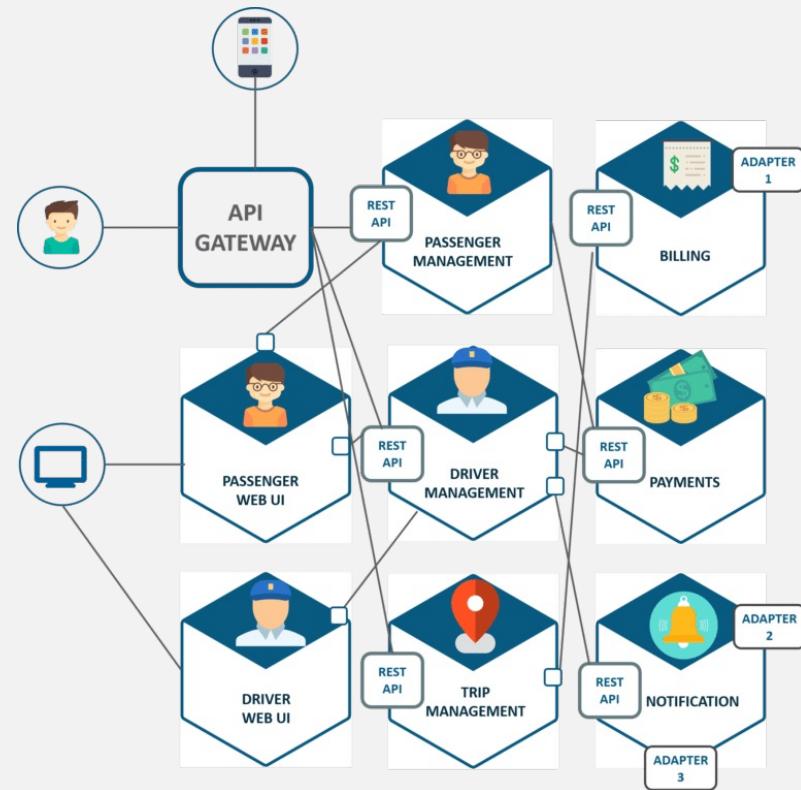
### 3. Arquitecturas distribuidas

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

Arquitectura de microservicios

Ejemplo

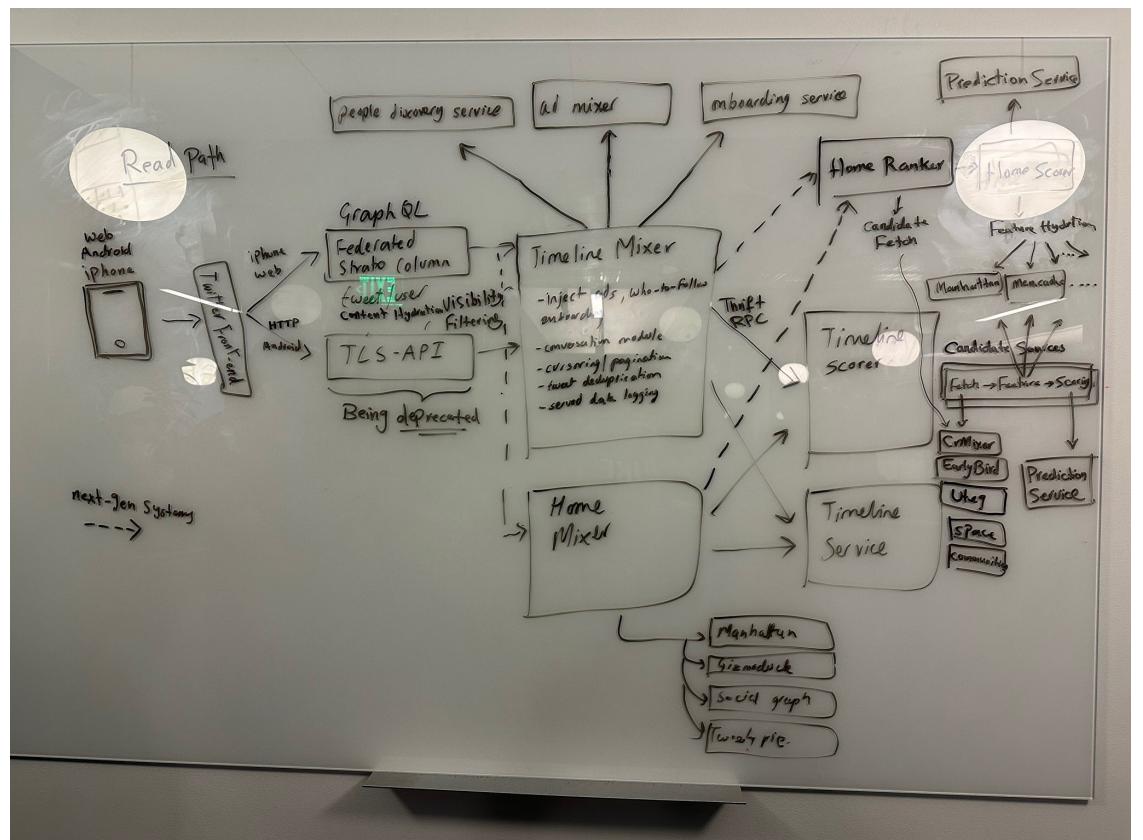
# Uber



1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud

### 3. Arquitecturas distribuidas

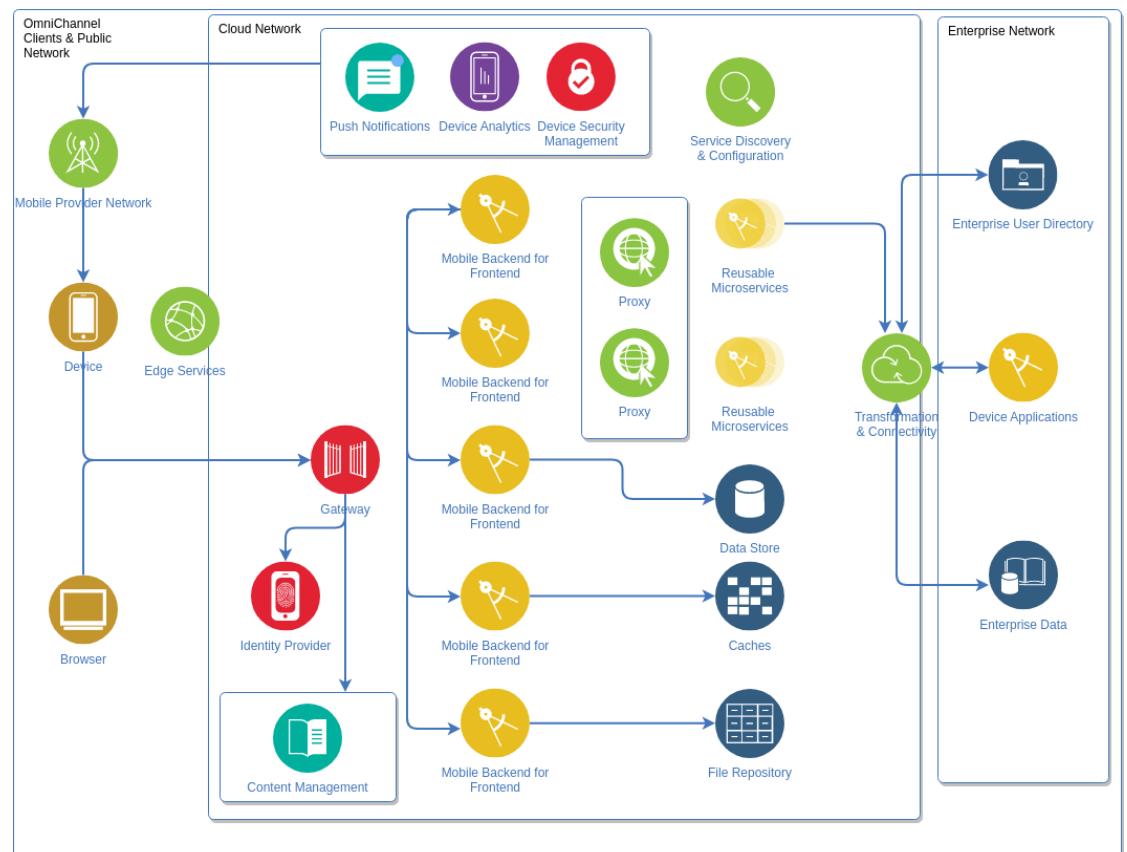
#### Arquitectura de microservicios



1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud

### 3. Arquitecturas distribuidas

#### Arquitectura de microservicios



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

Si se acepta la diferencia de alcance ambos enfoques pueden complementarse.

Casos de uso en los que afecta principalmente esta diferencia de alcance:

1. Reutilización de componentes
2. Llamadas síncronas
3. Duplicidad de los datos

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### 1. Reutilización de componentes

#### Arquitectura orientada a servicios

- El objetivo principal es la reutilización de las integraciones.
- La reutilización y el uso compartido de componentes aumentan la escalabilidad y la eficiencia.

#### Arquitectura de microservicios

- La reutilización de un componente genera dependencias y reducen la agilidad y la resiliencia.
- En algunos casos la mejor opción es reutilizar código para ayudar al desacoplamiento.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### 2. Llamadas síncronas

#### Arquitectura orientada a servicios

- Los servicios reutilizables están disponibles en toda la empresa utilizando protocolos síncronos.

#### Arquitectura de microservicios

- Las llamadas síncronas producen dependencias en tiempo real: pérdida de resiliencia.
- También pueden causar latencia afectando al rendimiento global.
- Se prefieren patrones de interacción basados en comunicación asíncrona.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### 3. Duplicidad de los datos

#### Arquitectura orientada a servicios

- Todas las aplicaciones obtienen y modifican datos de forma síncrona sobre su fuente principal.
- Reduce la necesidad de introducir patrones complejos de sincronización de datos.

#### Arquitectura de microservicios

- Cada microservicio tiene acceso local a todos los datos que necesita para garantizar su independencia de otros microservicios, incluso si esto agrega duplicidad de datos.
- Esta duplicidad agrega complejidad pero proporciona más agilidad y rendimiento.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Otras diferencias clave: comunicación

#### Arquitectura orientada a servicios

- Cada servicio debe compartir un mecanismo de comunicación común: bus de servicio empresarial.
- La administración y coordinación de los servicios se brindan a través del bus de servicio empresarial.
- El bus de servicio empresarial puede convertirse en un único punto de fallo para toda la empresa.
- Si un servicio se ralentiza todo el sistema puede verse afectado.

#### Arquitectura de microservicios

- Cada servicio se desarrolla de forma independiente con su propio protocolo de comunicación.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Otras diferencias clave: interoperabilidad

#### Arquitectura orientada a servicios

- Permite utilizar protocolos de mensajería heterogéneos como SOAP o AMQP.

#### Arquitectura de microservicios

- Utilizan protocolos de mensajería livianos para simplificar la comunicación como HTTP/REST o JMS.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Otras diferencias clave: gobernanza de datos

#### Arquitectura orientada a servicios

- Involucra recursos compartidos que permite la implementación de estándares comunes de gobernanza de datos en todos los servicios.

#### Arquitectura de microservicios

- No permite una gobernanza de datos coherente.
- Proporciona una mayor flexibilidad para cada servicio.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Otras diferencias clave: persistencia de datos

#### Arquitectura orientada a servicios

- Incluye una única capa de almacenamiento de datos compartida por todos los servicios de una aplicación determinada.

#### Arquitectura de microservicios

- Dedican un servidor o base de datos para el almacenamiento de los datos de cualquier servicio que necesite persistencia.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Otras diferencias clave: granularidad

#### Arquitectura orientada a servicios

- Puede abarcar desde servicios pequeños y especializados hasta servicios para toda la empresa.

#### Arquitectura de microservicios

- Se compone de servicios altamente especializados.
- Diseñados para hacer una única cosa y hacerla muy bien.

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

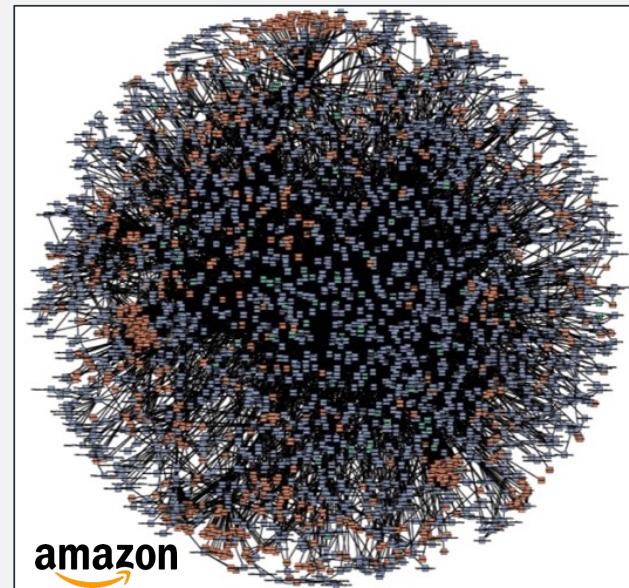
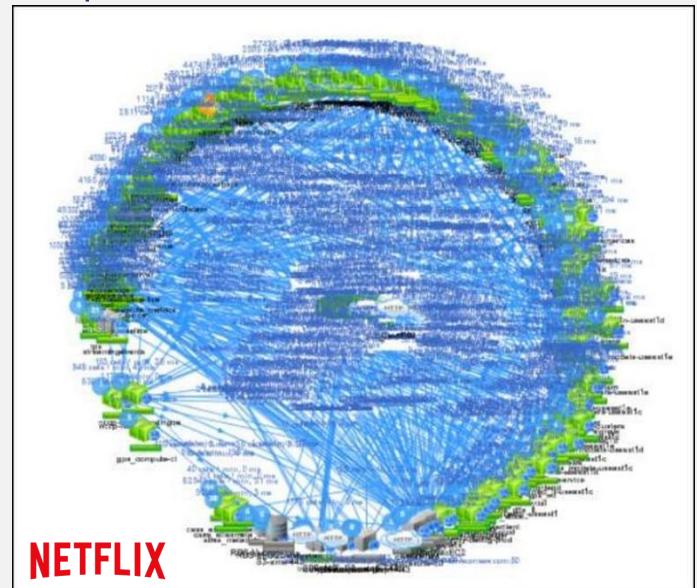
3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Cómo desarrollar arquitecturas de microservicios

#### División de equipos por **microservicio**

Un equipo se centra únicamente en el desarrollo de un microservicio concreto.

#### División de equipos por **área de conocimiento**

Un equipo especializado se centra en el desarrollo de un área para varios microservicios.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Beneficios de los microservicios

- Organización en torno a características o componentes de producto.
- Los cambios en una función o componente requieren un cambio solo en esa parte.
- La localización y arreglo de bugs es mucho más sencilla.
- La sinfonía entre servicios puede ofrecer soluciones más innovadoras.
- La gestión de proyectos se vuelve más sencilla.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Desventajas de los microservicios

- La cantidad de microservicios puede suponer una sobrecarga en la administración.
- La dotación de recursos en término de desarrolladores puede resultar costoso.
- Los equipos necesitan crecer a medida que lo hace un componente de la aplicación.
- Requiere de buena comunicación entre equipos para compartir conocimiento.
- La calidad del código puede ser diferente entre los microservicios.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

#### ¿Qué es un contenedor?

Un contenedor es un paquete de software estándar que agrupa el código de una aplicación con las **bibliotecas** y los archivos de **configuración** asociados, junto con las **dependencias** necesarias para que la aplicación se ejecute.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

¿Qué beneficios me proporciona la contenerización de software?

El problema de que una aplicación no se ejecute correctamente cuando se cambia de un entorno a otro es tan antiguo como el propio desarrollo de software.

Los contenedores solucionan este problema proporcionando una **estructura ligera e inmutable** para el **empaque y ejecución de aplicaciones**.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

¿Entonces es una máquina virtual?

¡NO! La filosofía de los contenedores es distinta a la de las máquinas virtuales.

Son tecnologías que persiguen un fin similar pero con enfoques totalmente diferentes.

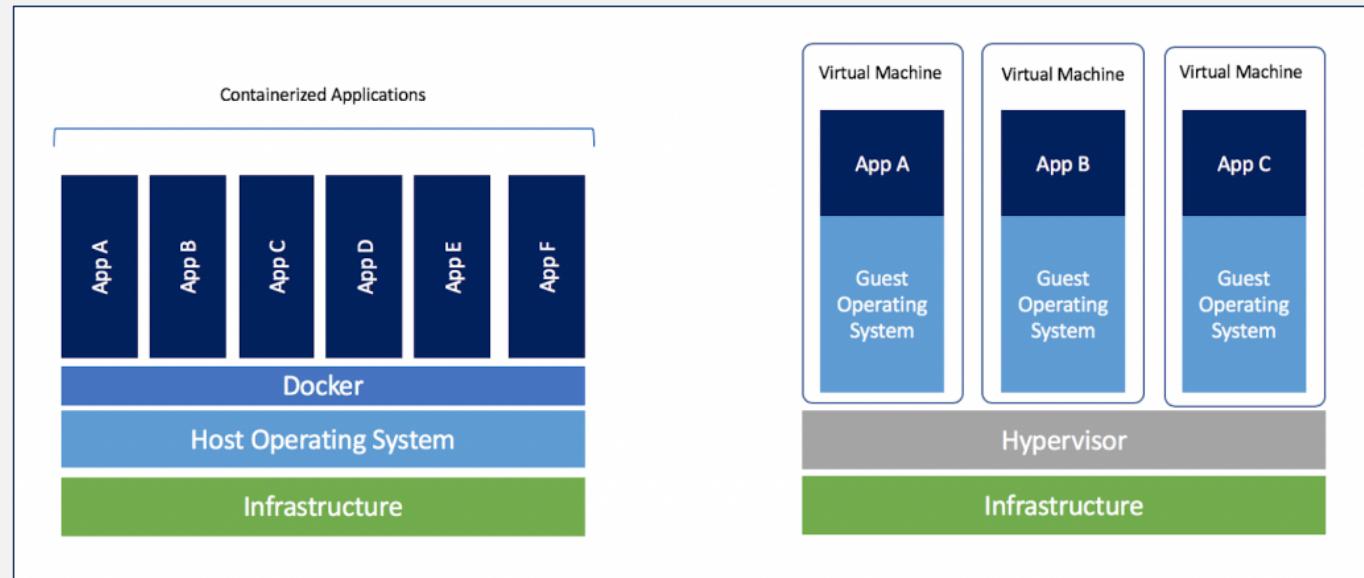
- Un contenedor funciona a partir de **imágenes que se pueden reutilizar**.
- Un contenedor **aísla una aplicación** y no un sistema operativo completo.
- Un contenedor es **mucho más ligero** que una máquina virtual.

# 3. Arquitecturas distribuidas

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Arquitectura de microservicios

### Contenedores



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

#### ¿Qué es Docker?

Docker es un proyecto de **código abierto** que **automatiza el despliegue** de aplicaciones dentro de **contenedores** de software, proporcionando una capa de abstracción y **automatización** de virtualización de aplicaciones en múltiples sistemas operativos.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

¿Cómo se genera un contenedor de Docker?

El Dockerfile son las **instrucciones** que Docker lee para construir una **imagen automáticamente**. Es un documento de texto que **contiene todos los comandos** que un usuario puede utilizar en la línea de comandos **para ensamblar una imagen**.

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

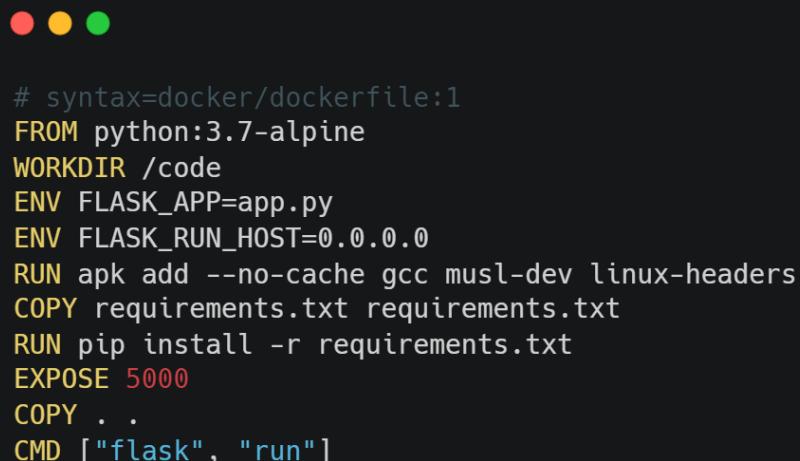
4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores



```
# syntax=docker/dockerfile:1
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

ENV ASPNETCORE_URLS=http://+:80

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["MyApp/MyApp.csproj", "MyApp/"]
RUN dotnet restore "MyApp/MyApp.csproj"
COPY . .
WORKDIR "/src/MyApp"
RUN dotnet build "MyApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MyApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

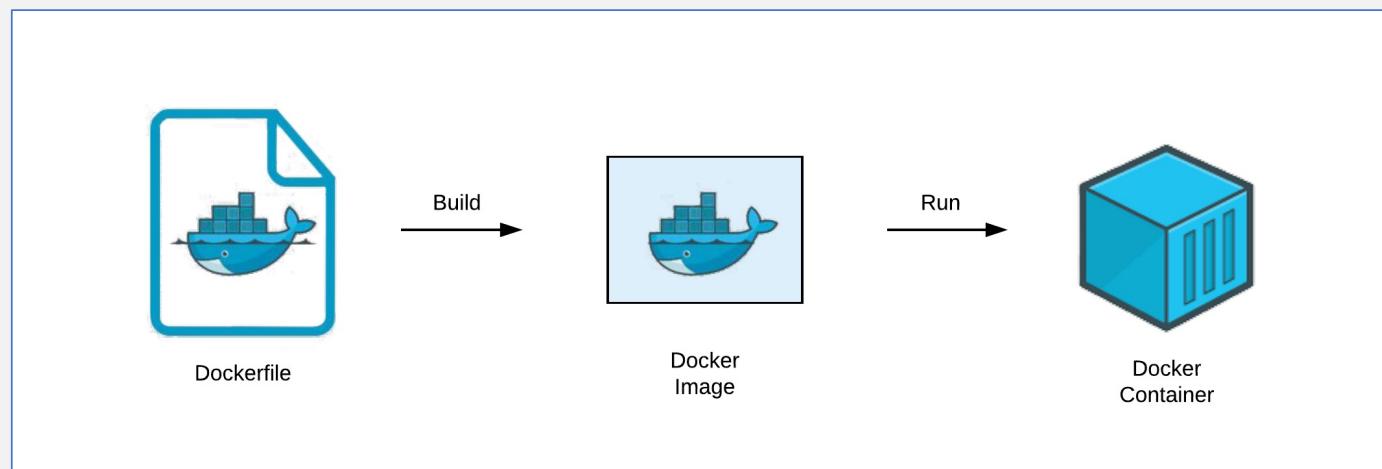
4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Contenedores



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

Comandos:

- docker build -t image\_name .
- docker images
- docker run -it name\_image command
- docker run -d -p host\_port:docker\_port
- docker ps
- docker container ls
- docker container start/stop id\_container
- docker rmi -f id\_image
- docker rm -f id\_container

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

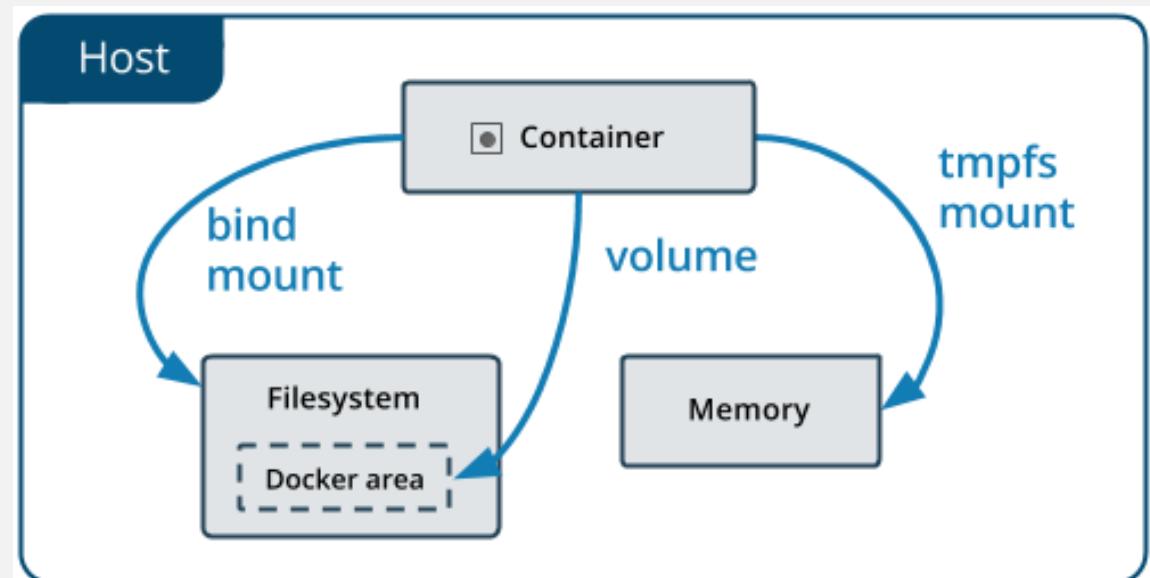
5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Volúmenes en contendores

- Volumes
- Bind Mounts
- tmpfs



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

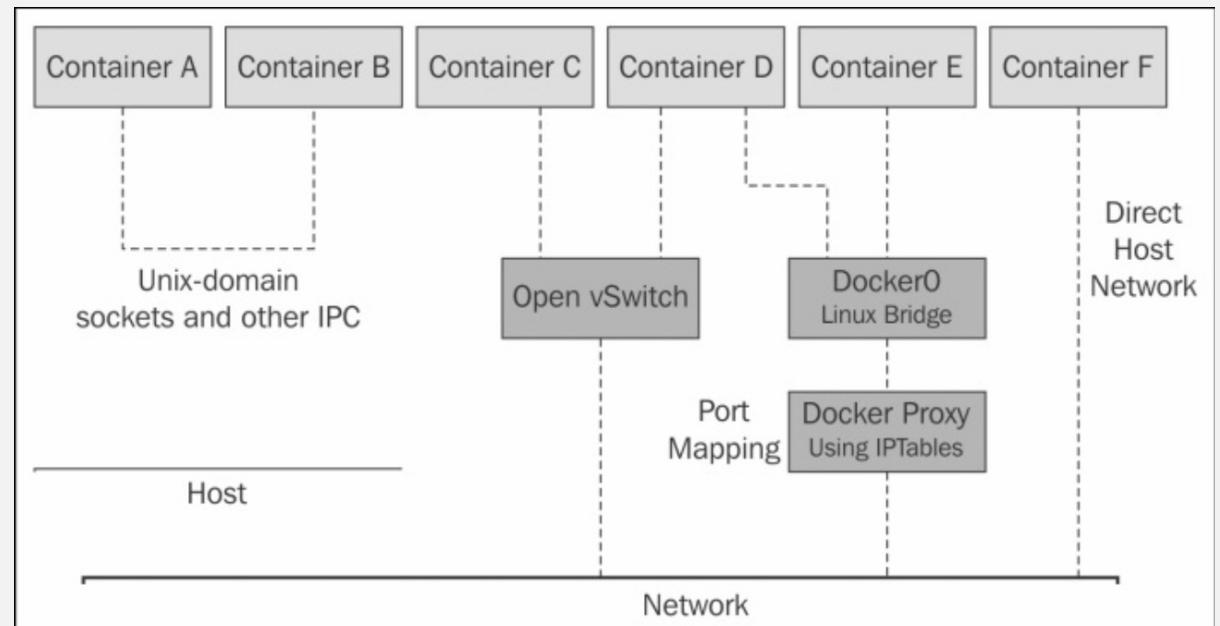
4. Comunicación

5. Servicios web

6. Cloud

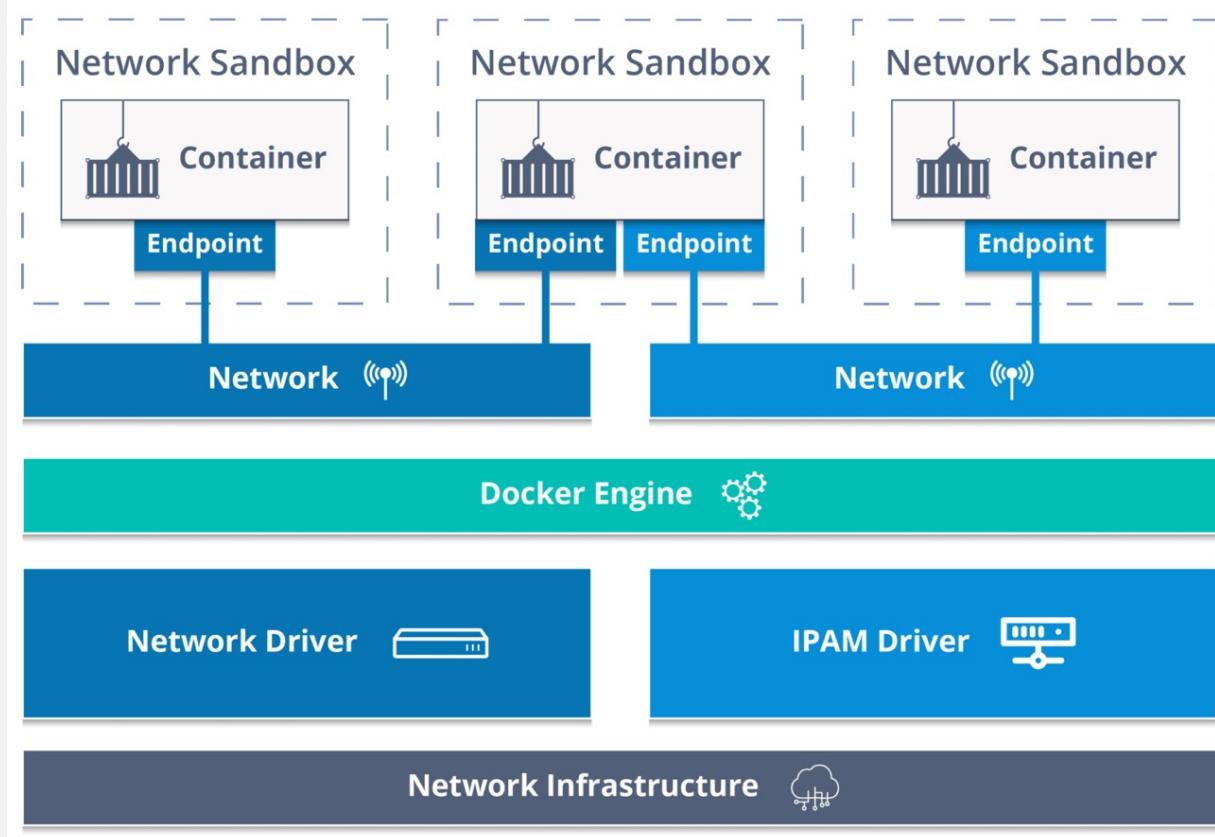
## Arquitectura de microservicios

### Redes en contenedores



### 3. Arquitecturas distribuidas

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

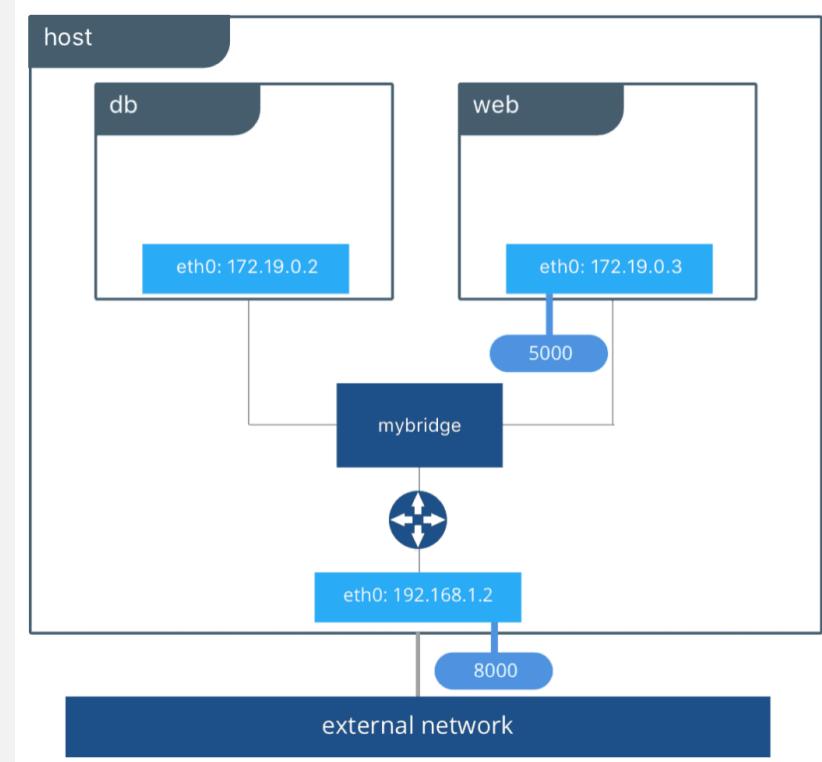
4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Docker Bridge



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

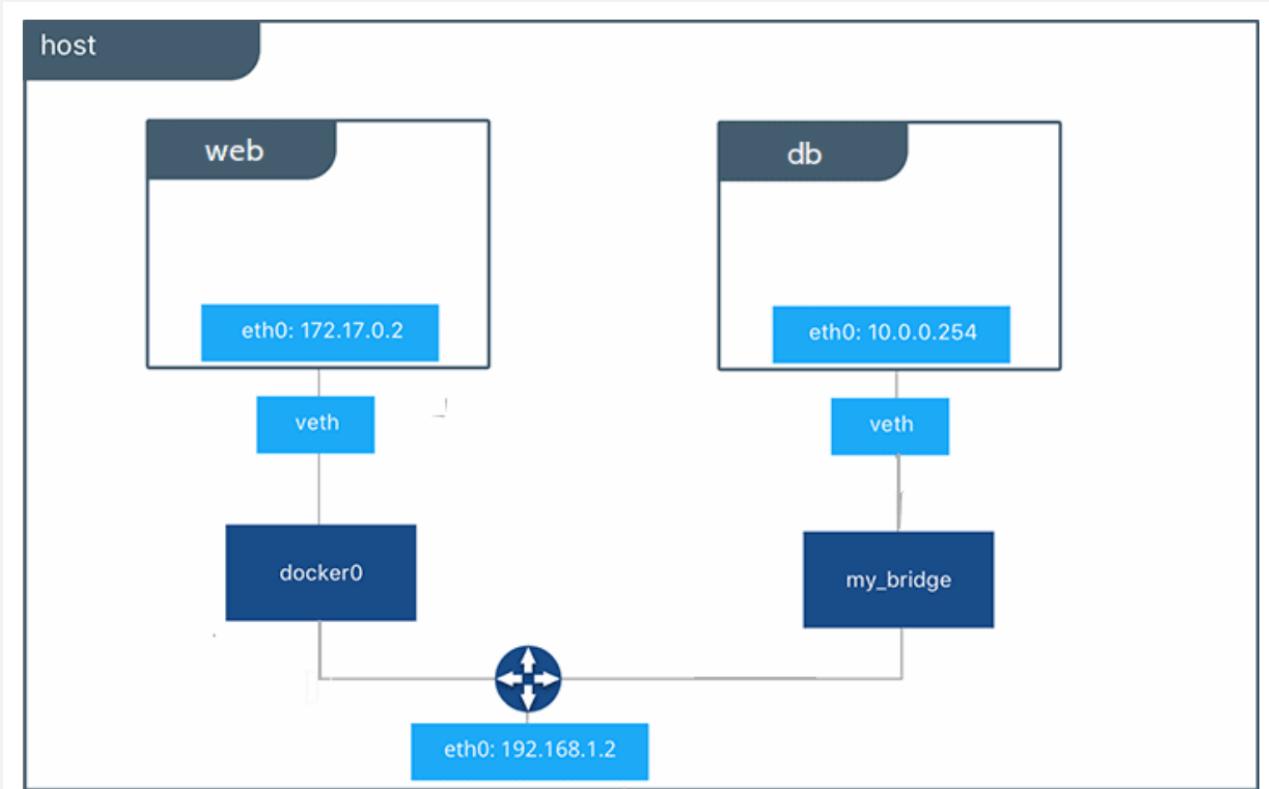
4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Docker Networking



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

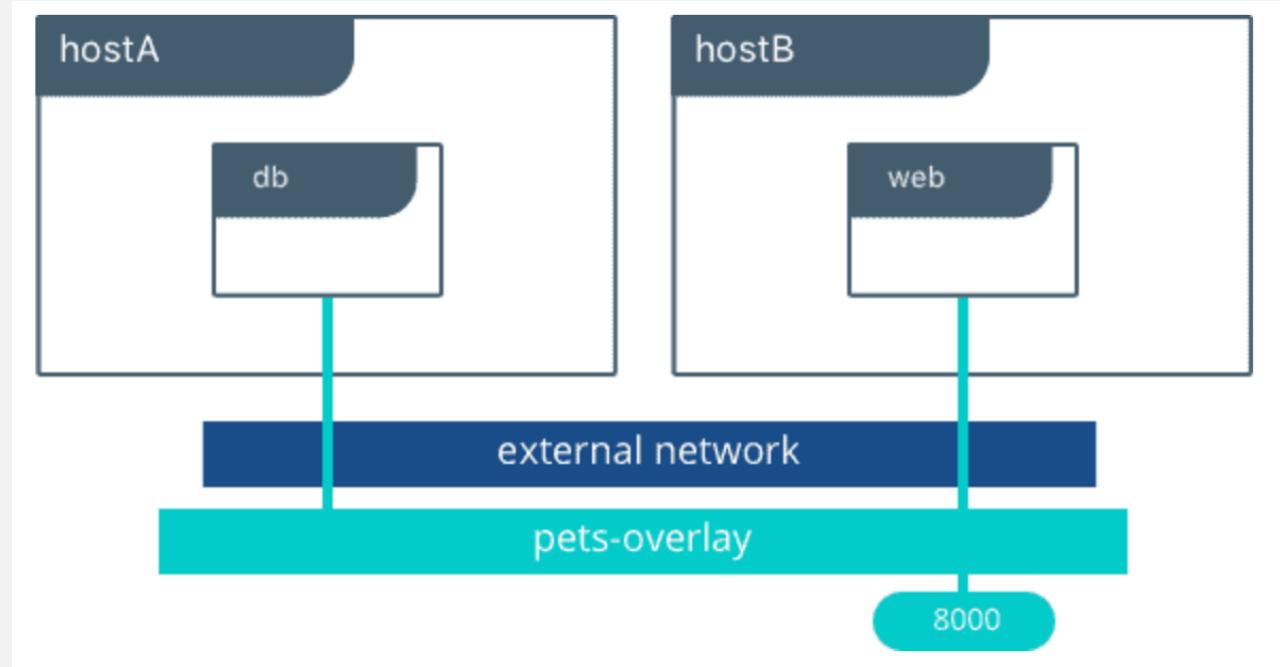
4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Overlay Networking



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Compose

```
services:  
  web:  
    build: .  
    ports:  
      - "8000:5000"  
    volumes:  
      - .:/code  
    environment:  
      FLASK_DEBUG: "true"  
  redis:  
    image: "redis:alpine"
```

<https://docs.docker.com/compose/gettingstarted/>

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Apache Kafka

- Plataforma de **intermediación de mensajes** que entrega datos de un punto a otro.
- Su singularidad radica en su capacidad para **distribuir, almacenar y procesar eventos**.
- Fácil de configurar.
- Maneja **grandes cantidades de datos** de diferentes fuentes con facilidad.
- Es **estable, seguro** y bien **respaldado** por la comunidad Apache.
- En continuo desarrollo.

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

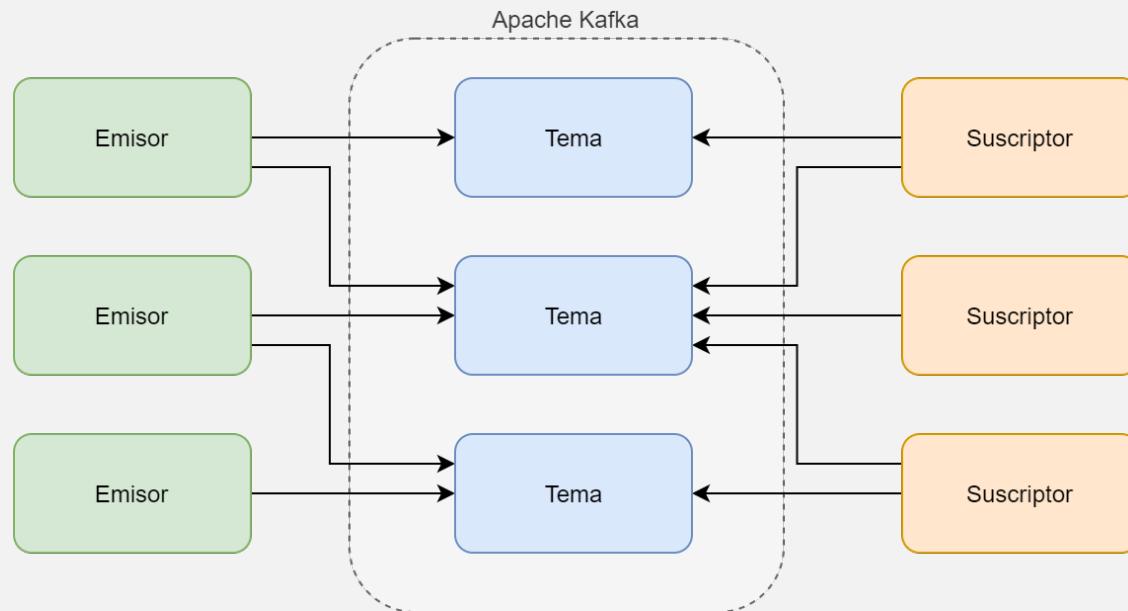


Apache Kafka es una plataforma de transmisión de eventos distribuida y de código abierto utilizada por miles de empresas para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica.

### 3. Arquitecturas distribuidas

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

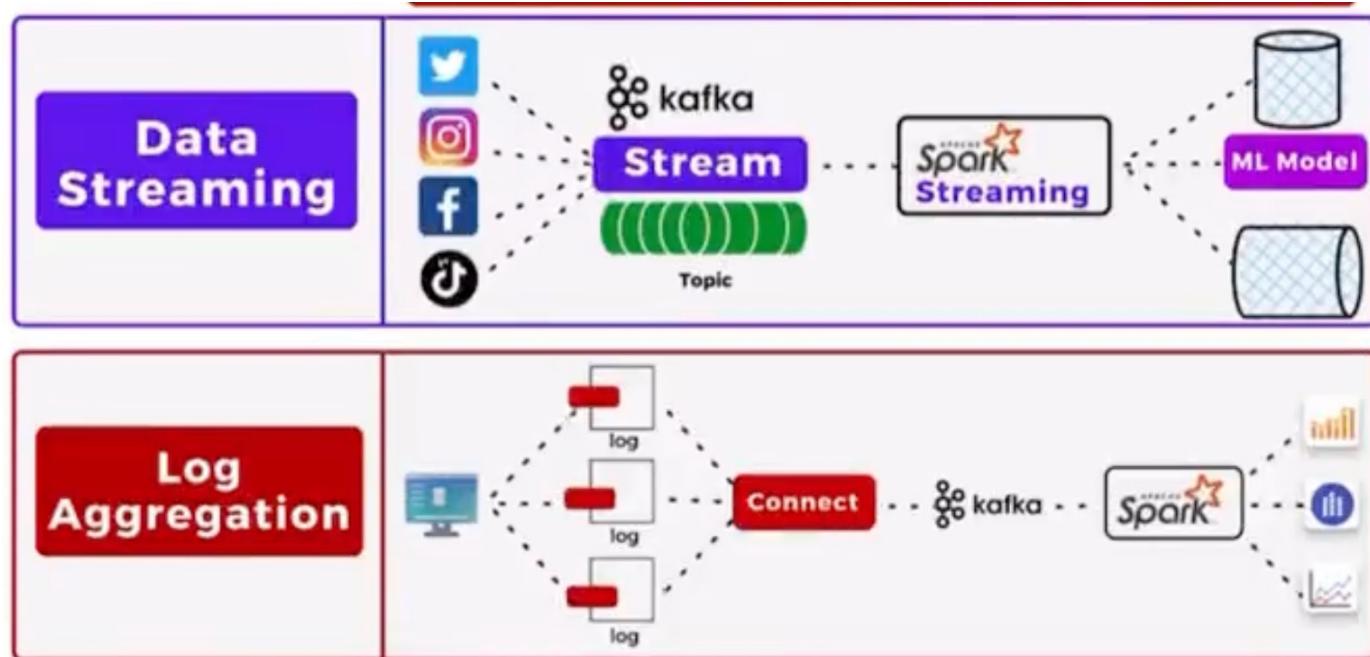
#### Arquitectura de microservicios



### 3. Arquitecturas distribuidas

#### Arquitectura de microservicios

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

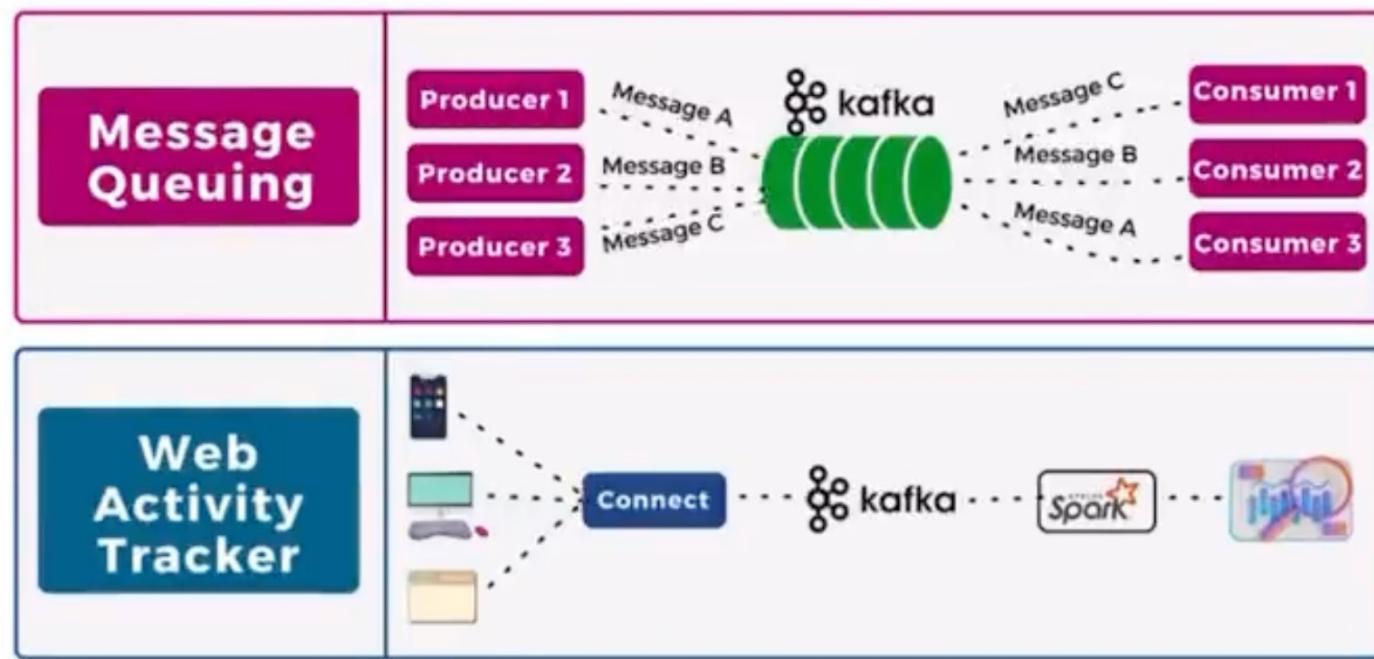
3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

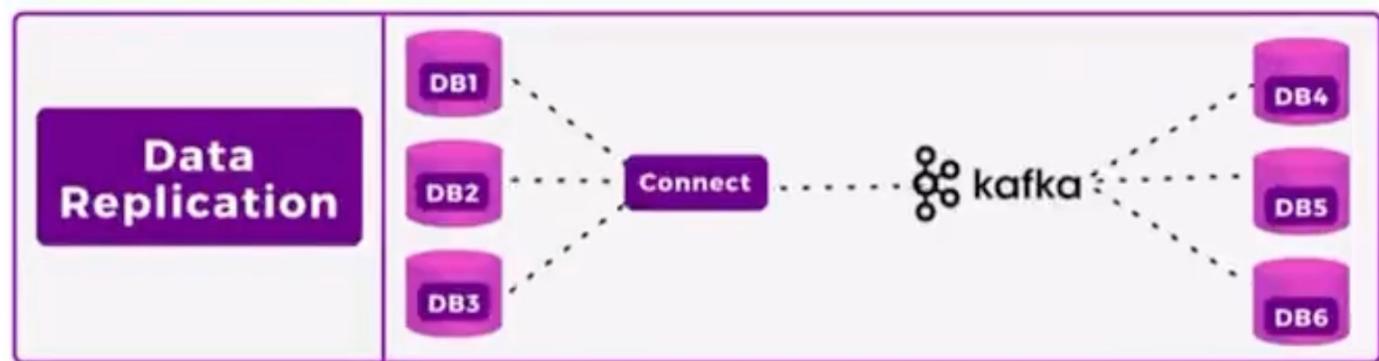
## Arquitectura de microservicios



# 3. Arquitecturas distribuidas

## Arquitectura de microservicios

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud



# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Apache Kafka: producer

```
● ● ●  
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],  
                         value_serializer=lambda x:dumps(x).encode('utf-8'))
```

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Apache Kafka: producer

```
● ● ●  
for e in range(1000):  
    data = {'number' : e}  
    producer.send('numtest', value=data)  
    sleep(5)
```

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Apache Kafka: consumer

```
● ● ●  
consumer = KafkaConsumer(  
    'numtest',  
    bootstrap_servers=['localhost:9092'],  
    auto_offset_reset='earliest',  
    enable_auto_commit=True,  
    group_id='my-group',  
    value_deserializer=lambda x: loads(x.decode('utf-8')))
```

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Arquitectura de microservicios

### Apache Kafka: consumer



```
for message in consumer:  
    message = message.value  
    collection.insert_one(message)  
    print('{} added to {}'.format(message, collection))
```

# 3. Arquitecturas distribuidas

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Patrones microservicios

- Sidecar
- Ambassador
- Adapters
- Replicated Load-Balanced Service

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

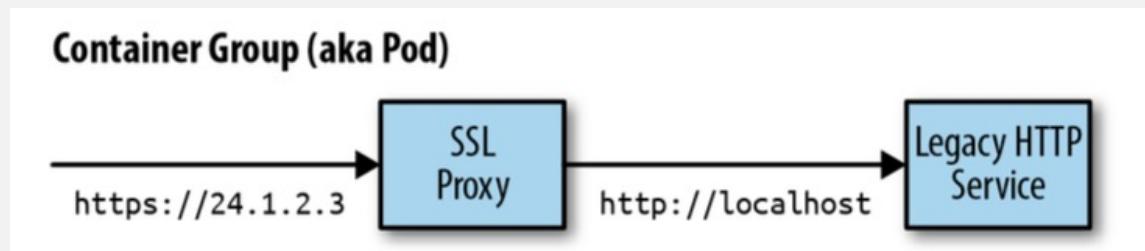
4. Comunicación

5. Servicios web

6. Cloud

## Sidecar

Mediante un contenedor legacy al cual no se le pueden añadir nuevas funcionalidades. Se añade un contenedor que lo complementa. Por ejemplo, se dispone de una aplicación legacy a la que hay que añadir una capa SSL para añadir seguridad.



### Ventajas

- añade funcionalidad sin necesidad de modificar el entorno

### Desventaja

- Añade complejidad a la arquitectura.

# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

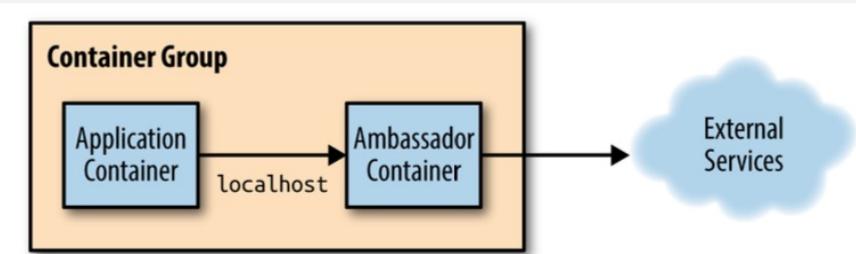
4. Comunicación

5. Servicios web

6. Cloud

## Abassador

Son empleados como capa de comunicación con otros servicios. Por ejemplo, sistemas de enrutamiento a servicios. "Sharding de bases de datos" o "Enrutamiento a diferentes bases de datos" o pruebas de test A/B



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

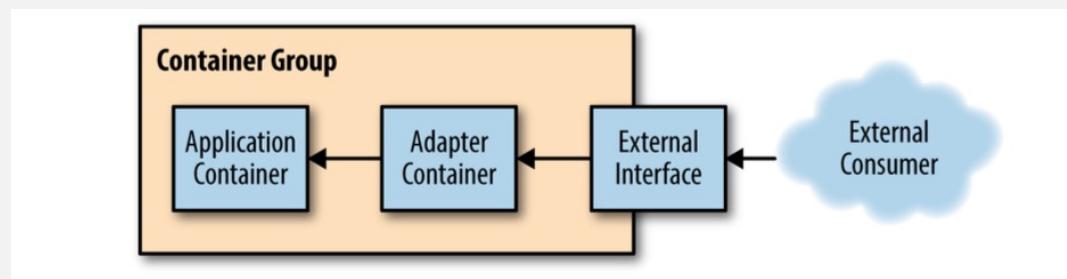
4. Comunicación

5. Servicios web

6. Cloud

## Adapters

Empleados para monitorizar y registrar los servicios a los que sirven en backend. Pueden realizar funciones de health-check o como proxy de monitorización de peticiones al backend.



# 3. Arquitecturas distribuidas

1. Introducción

2. Fundamentos

3. Arquitecturas

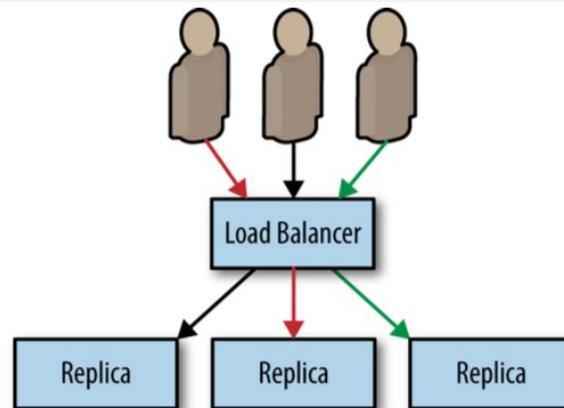
4. Comunicación

5. Servicios web

6. Cloud

## Replicated Load-Balanced Service

Creación de diferentes réplicas accesibles a través de una balanceador de carga. Se puede añadir funcionalidades de scale-down y scale-up. También se pueden añadir capas de caché. Se pueden aplicar varias capas.



# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Fundamentos de comunicación

- Mecanismos de comunicación entre procesos (IPC)
- Protocolos de comunicación

## Protocolos de comunicación

- Paradigma de mensajes: Sockets
- Llamada a procedimientos remotos: RPC
- Invocación de métodos remotos: RMI
- Intermediario de petición de objetos: ORB
- Servicios web: SOAP y RESTful

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Fundamentos de comunicación

### Mecanismos de comunicación entre procesos (IPC)

- Función básica de los sistemas operativos.
- Los procesos pueden comunicarse entre sí.
- Sistema de **paso de mensajes** que ofrece la red.
- La comunicación se establece siguiendo una serie de reglas: **protocolos de comunicación**.
- Los procesos pueden estar ejecutándose en una o más máquinas conectadas a una red.

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Fundamentos de comunicación

### Modelo OSI

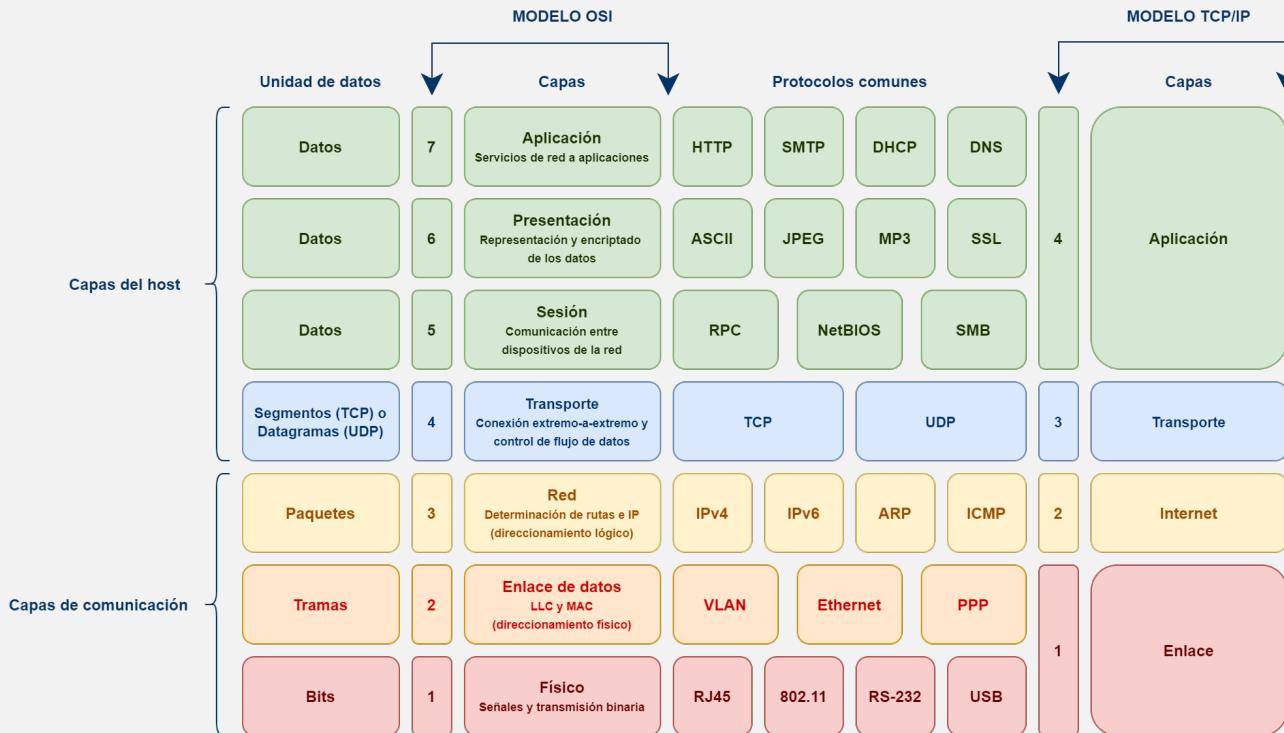
- El modelo OSI es un modelo **conceptual** utilizado principalmente para definir, discutir y comprender funciones de red individuales.
- Es **genérico e independiente del protocolo**.

### Modelo TCP/IP

- El modelo TCP/IP está **diseñado para resolver un conjunto específico de problemas** y no para funcionar como una descripción para todas las comunicaciones de red.
- Se basa en **protocolos estándar basados en Internet**.

# 4. Mecanismos de comunicación

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud



# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Fundamentos de comunicación

### Capa de transporte

#### Protocolo de control de transmisión (TCP)

- Uno de los protocolos fundamentales en Internet.
- Comunicación de forma segura.
- Garantiza que los datos son entregados en su destino sin errores y en el mismo orden en el que se transmiten.
- Da soporte a muchos aplicaciones y protocolos de aplicación.

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Fundamentos de comunicación

### Capa de transporte

#### Protocolo de datagrama de usuario (UDP)

- Protocolo ligero de transporte de datos que funciona sobre IP
- Proporciona un mecanismo para detectar datos corruptos en paquetes.
- No resuelve otros problemas como la pérdida o desorden de paquetes.

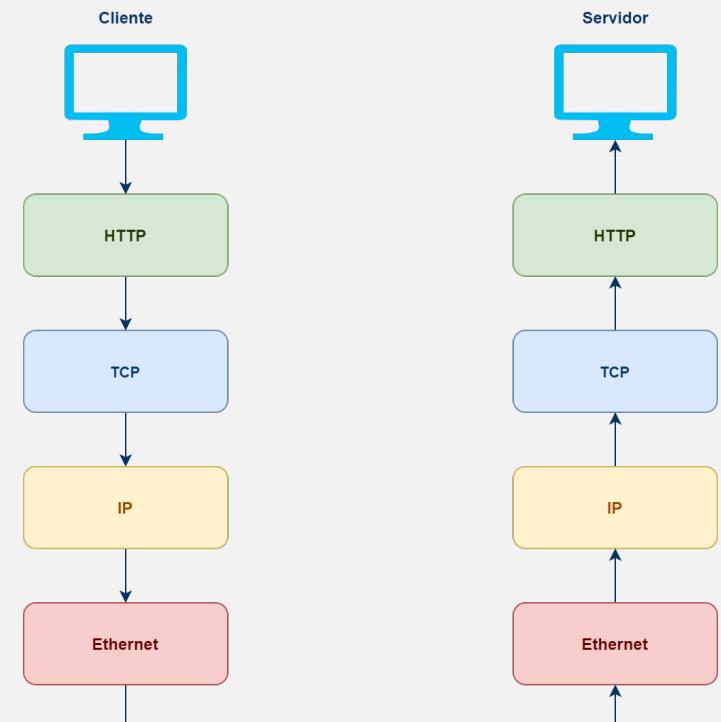
# 4. Mecanismos de comunicación

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Fundamentos de comunicación

### Protocolos de comunicación (capa aplicación)

- Protocolo de transferencia de hipertexto (HTTP)
- Protocolo de transferencia de archivos (FTP)
- Protocolo de oficina de correo (POP3)
- Protocolo de transferencia simple de correo (SMTP)
- Protocolo de configuración dinámica del host (DHCP)
- Protocolo simple de administración de red (SNMP)
- Protocolo extensible de mensajería y presencia (XMPP)



# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Paradigma de paso de mensajes

### Sockets

Un socket es un concepto abstracto por el que dos procesos pueden intercambiar un flujo de datos.

Un socket queda definido por:

- Un par de direcciones IP local y remota.
- Un protocolo de transporte (TCP o UDP).
- Un par de números de puerto local y remoto.

# 4. Mecanismos de comunicación

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Paso de mensajes

### Sockets stream

Los sockets stream ofrecen un **servicio orientado a la conexión**.

- Los datos se transfieren en un **flujo continuo**.
- Se basa en el **protocolo TCP**.
- Implica conexión entre dos sockets para transmitir información.
- Un socket atiende peticiones (servidor) y el otro solicita la conexión (cliente).
- El protocolo TCP incorpora de forma transparente la corrección de errores.
- No limita el tamaño máximo de información a transmitir.

# 4. Mecanismos de comunicación

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Paso de mensajes

### Sockets datagram

Los sockets stream ofrecen un servicio de transporte sin conexión.

- Se basa en el protocolo UDP.
- Los datos se envían y reciben en datagramas de tamaño limitado.
- Envío de información sin conexión previa.
- No garantiza la fiabilidad: duplicidad, pérdida o desorden de datagramas.
- El destino se encarga de la corrección de errores.
- Utilizado fundamentalmente en aplicaciones en tiempo real o streamings de audio y vídeo.

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

Paso de mensajes

Middleware de alto nivel

- RPC (Remote Procedure Call) o llamada a procedimientos remotos  
Uso de métodos remotos
- RMI (Remote Method Invocation) o invocación de métodos remotos  
Uso de objetos remotos
- ORB (Object Request Broker) o intermediario de solicitud de objetos  
Uso de objetos remotos
- Servicios web (SOAP y RESTful)  
Métodos como servicios

# 4. Mecanismos de comunicación

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Llamada a procedimientos remotos

### RPC

Herramienta para establecer estructuras colaborativas y operativas en redes y arquitecturas C/S.

- Proceso de **comunicación bidireccional orientada a solicitudes**.
- Envío de parámetros y retorno del valor de una función.
- Regulación de la comunicación entre procesos.
- El objetivo es **armonizar los niveles de procesamiento**.
- Concebido principalmente para la comunicación de varias máquinas.

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Llamada a procedimientos remotos

### RPC

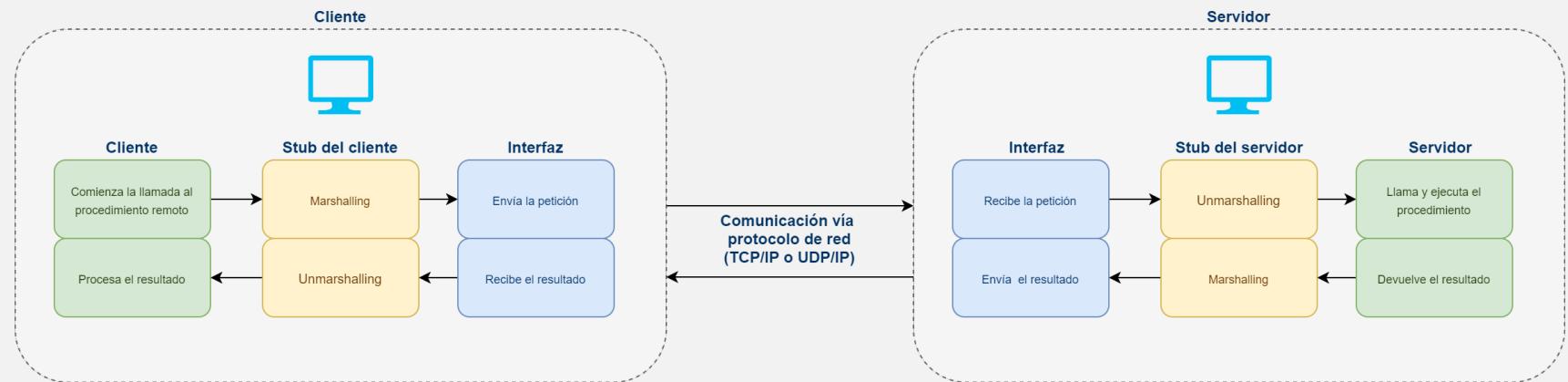
Siempre se ejecuta siguiendo un **patrón** determinado.

- En el emisor y en el receptor participan unas instancias especiales denominadas **stubs**.
- El **stub** del cliente actúa como **sustituto del procedimiento del servidor remoto** en el lado del cliente.
- El **stub** del servidor **sustituye los valores del cliente** que realiza la llamada en el lado del servidor.
- Los **stubs simulan** operar como una unidad local funcional.
- Los **stubs** actúan como interfaces del procedimiento.

# 4. Mecanismos de comunicación

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud

## Llamada a procedimientos remotos



# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Llamada a procedimientos remotos

### gRPC

Sistema de llamada de procedimiento remoto

- Desarrollado inicialmente por **Google**.
- Utiliza **HTTP/2** como transporte y **Protobuf** como lenguaje de descripción de la interfaz.
- Proporciona mecanismos y características de **autenticación**, transmisión bidireccional y control de flujos, enlaces bloqueantes, cancelaciones y tiempos de espera, entre otros.
- Genera enlaces **multiplataforma** entre cliente y servidor para muchos lenguajes: C#, C++, Dart, Go, Java, Kotlin, Node, Objective-C, PHP, Python y Ruby.
- Incluye conexión de servicios en arquitecturas **microservicios**: Netflix

# 4. Mecanismos de comunicación

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Llamada a procedimientos remotos

### Protobuf

Fichero con extensión .proto.

```
● ● ●  
message Person {  
    required string name = 1;  
    required int32 id = 2;  
    optional string email = 3;  
}
```

```
● ● ●  
service Greeter {  
    rpc SayHello (HelloRequest) returns (HelloReply) {}  
}  
  
message HelloRequest {  
    string name = 1;  
}  
  
message HelloReply {  
    string message = 1;  
}
```

# 4. Mecanismos de comunicación

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Llamada a procedimientos remotos

### gRPC

Generación de *stubs*.

```
public class GreeterImpl : Greeter.GreeterBase
{
    // Server side handler of the SayHello RPC
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        return Task.FromResult(new HelloReply { Message = "Hello " + request.Name });
    }
}
```

```
public static void Main(string[] args)
{
    Channel channel = new Channel("127.0.0.1:50051", ChannelCredentials.Insecure);

    GreeterClient client = new Greeter.GreeterClient(channel);
    String user = "you";

    HelloReply reply = client.SayHello(new HelloRequest { Name = user });
    Console.WriteLine("Greeting: " + reply.Message);

    channel.ShutdownAsync().Wait();
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Invocación de métodos remotos

### RMI

Mecanismo ofrecido por Java para invocar un método de manera remota.

- Forma parte del entorno estándar de Java.
- Proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas.
- Si se requiere comunicación entre otras tecnologías debe utilizarse CORBA o SOAP en lugar de RMI.
- Proporciona paso de objetos por referencia (no permitido por SOAP).
- Proporciona paso de tipos arbitrarios (funcionalidad no incluida en CORBA).
- A través de RMI Java puede **exportar un objeto** que estará **accesible a través de la red**.

# 4. Mecanismos de comunicación

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Invocación de métodos remotos

### RMI

Diferencias con RPC:

- **Enfoque:** RMI utiliza un paradigma orientado a objetos. RPC no transacciona con objetos sino con procedimientos o subrutinas.
- **Trabajo:** Con RPC llamas a un procedimiento remoto que se ejecuta como si fuese una llamada a un procedimiento local. RMI hace lo mismo pero pasa una referencia del objeto y del método.
- **Potencia:** RMI es más potente que RPC en términos de explotación de objetos, referencias (compartir un objeto con múltiples instancias de JVM), herencia, polimorfismo y excepciones. La tecnología está integrada en el lenguaje. RMI permite además invocación dinámica (las interfaces pueden cambiar en tiempo de ejecución).

# 4. Mecanismos de comunicación

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Invocación de métodos remotos

### RMI

Mecanismo ofrecido por Java para invocar un método de manera remota.

```
// HelloInterface.java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloInterface extends Remote {
    public void sayHello() throws RemoteException;
}
```

```
// HelloImpl.java
import java.rmi.RemoteException;
import javax.rmi.PortableRemoteObject;

public class HelloImpl extends PortableRemoteObject implements HelloInterface {
    public HelloImpl() throws RemoteException {
        // Invoca el enlace RMI y la inicialización remota del objeto.
        super();
    }

    public void sayHello() throws RemoteException {
        System.out.println("Hello World!");
    }
}
```

# 4. Mecanismos de comunicación

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Invocación de métodos remotos

```
// Server.java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {
    public Server() {}

    public static void main(String args[]) {
        try {
            Server server = new Server();
            HelloImpl stub = (HelloImpl) UnicastRemoteObject.exportObject(server, 0);

            // Vincula el stub del objeto remoto en el registro.
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("HelloImpl", stub);

            System.out.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

```
// Client.java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    private Client() {}

    public static void main(String[] args) {
        String host = (args.length < 1) ? null : args[0];

        try {
            Registry registry = LocateRegistry.getRegistry(host);
            HelloImpl stub = (HelloImpl) registry.lookup("HelloImpl");
            String response = stub.sayHello();

            System.out.println("Respuesta: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

# 4. Mecanismos de comunicación

---

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Intermediario de solicitud de objetos

### ORB

Middleware que permite a los objetos realizar llamadas a métodos situados en máquinas remotas.

- Componente fundamental de la arquitectura CORBA.
- CORBA es una alternativa a RMI.
- Es capaz de comunicar diferentes lenguajes.
- Comparte conceptos con RPC.
- Utiliza OMG IDL para definir interfaces de objetos y GIOP para estandarizar el formato intercambiable del mensaje.

## 4. Mecanismos de comunicación

1. Introducción

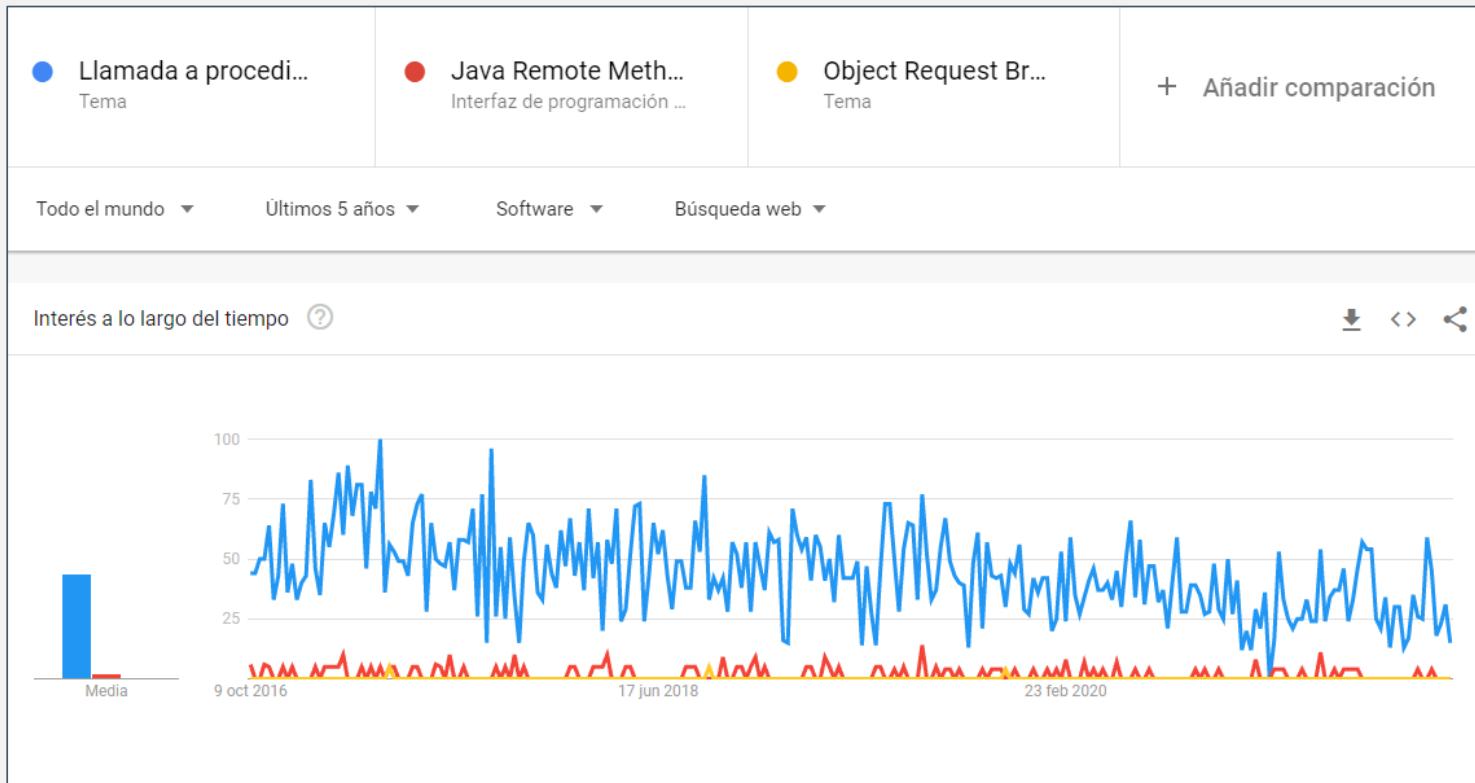
2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud



# 5. Servicios web

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios web

### SOAP

Protocolo de acceso a objetos simples

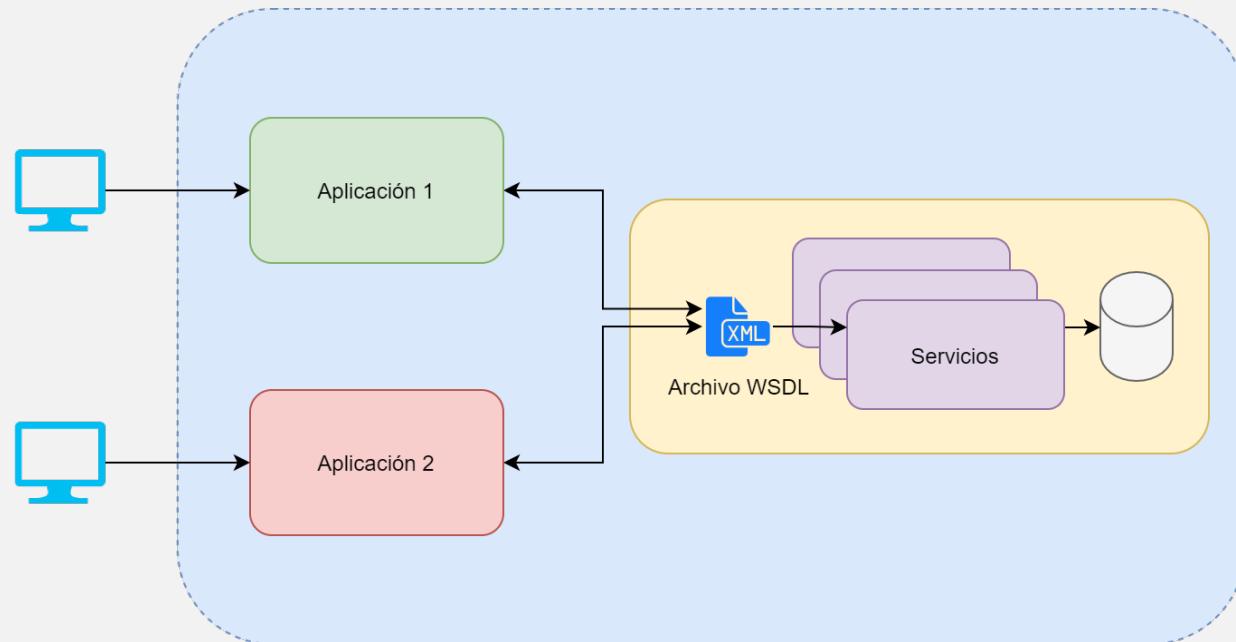
- Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse.
- Intercambio de datos en el **metalenguaje XML**.
- Creado por Microsoft e IBM entre otros.
- Recomendación y **estándares** de la World Wide Web Consortium (W3C).
- Acceso a un sistema de forma **ordenada y limitada**.
- **WSDL** para describir los servicios web.

# 5. Servicios web

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Servicios web

SOAP

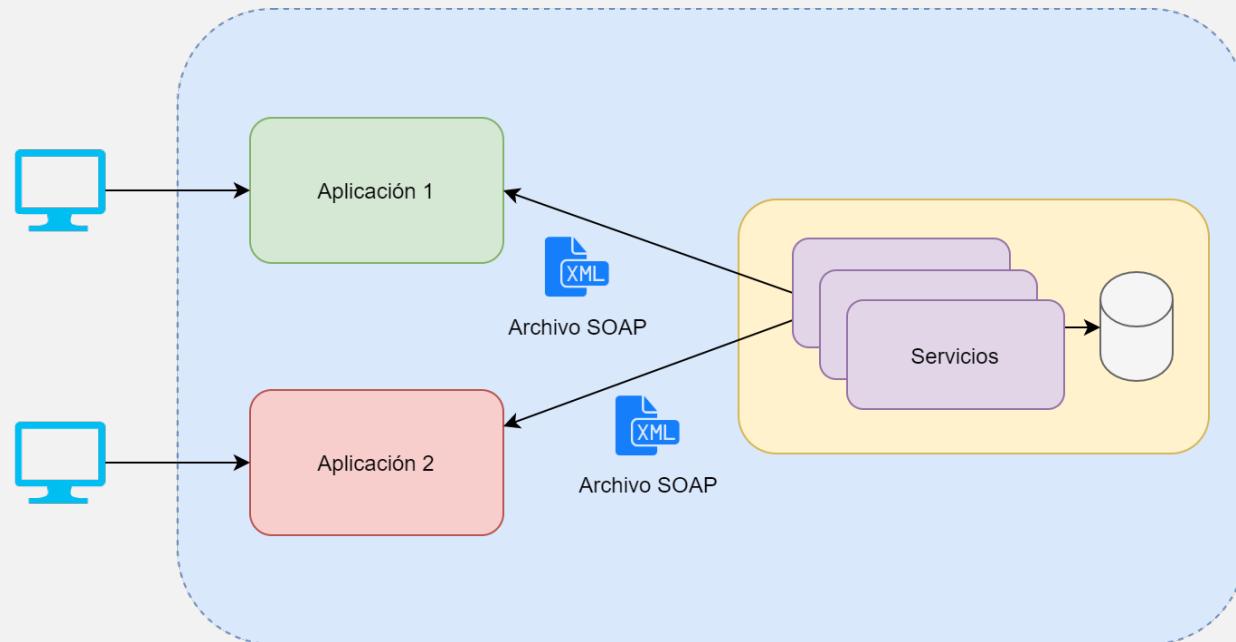


# 5. Servicios web

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Servicios web

### SOAP



# 5. Servicios web

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios web

### SOAP

Protocolo simple de acceso a objetos

```
<!-- WSDL -->
<message name="GetQuotationRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="GetQuotationResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossary">
  <operation name="GetQuotation">
    <input message="GetQuotationRequest"/>
    <output message="GetQuotationResponse"/>
  </operation>
</portType>
```

```
<!-- SOAP request -->
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset = utf-8
Content-Length: nnn

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-
  encoding">

  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotations">
    <m:GetQuotation>
      <m:QuotationsName>Microsoft</m:QuotationsName>
    </m:GetQuotation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# 5. Servicios web

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios web

### RESTful

Transferencia de estado representacional

- Un servicio web RESTful es aquel que está basado en la arquitectura REST.
- A diferencia de SOAP, REST más que un protocolo es un estilo de arquitectura que indica cómo realizar el intercambio y manejo de datos a través de servicios web.
- REST se diseñó para ser simple con el objetivo de lograr una rápida adopción del usuario.
- Está orientado al recurso: permite listar, crear, leer, actualizar y borrar recursos.
- Cada operación requiere dos cosas: el método URI (sustantivo) y HTTP (verbo).

# 5. Servicios web

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios web

### RESTful

Operación	Verbo (método HTTP)	Sustantivo (URI)	Parámetros	Resultado
Listar	GET	<code>/{{recursos}}</code>	No aplica	Lista del tipo de recurso
Crear	POST	<code>/{{recursos}}</code>	Dentro cuerpo del POST	Se crea un nuevo recurso
Leer	GET	<code>/{{recursos}}/{{recurso_id}}</code>	No aplica	El recurso con ese id
Actualizar	PATCH / PUT	<code>/{{recursos}}/{{recurso_id}}</code>	Cadena de consulta	Se actualiza / reemplaza el recurso con ese id
Borrar	DELETE	<code>/{{recursos}}/{{recurso_id}}</code>	No aplica	Se elimina el recurso con ese id

# 5. Servicios web

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios web

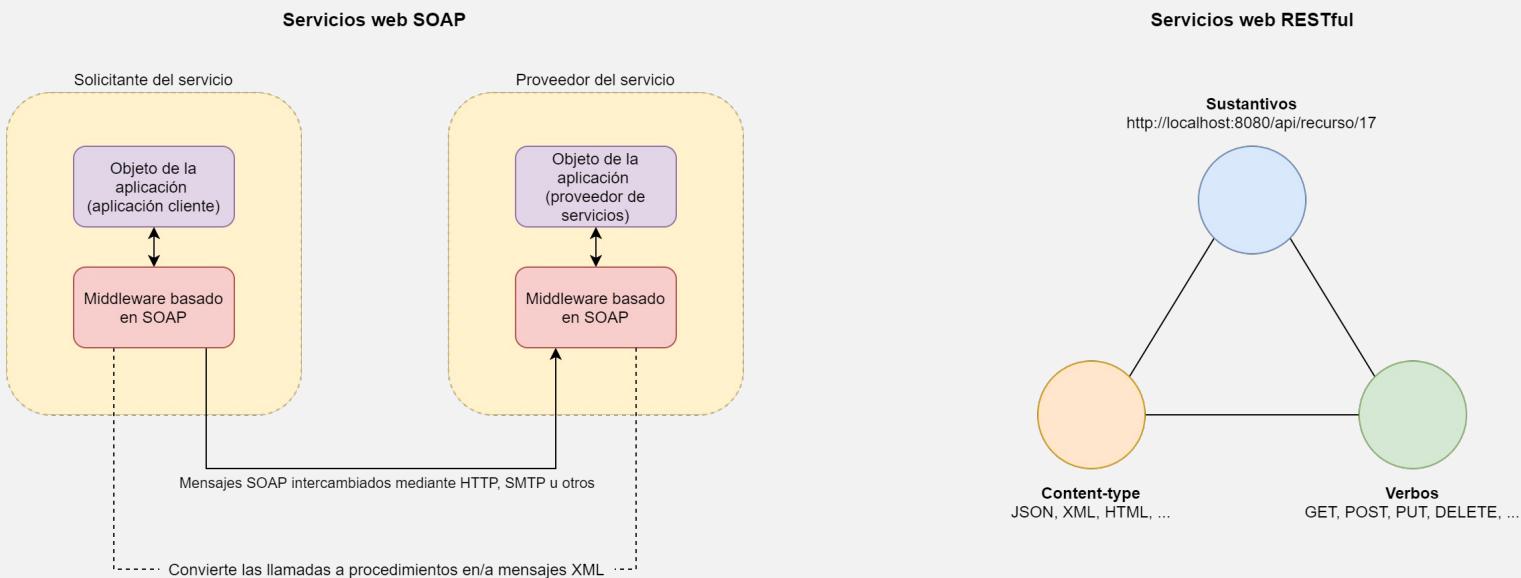
### SOAP vs RESTful

Servicios web SOAP	Servicios web RESTful
SOAP es un protocolo	REST es un estilo de arquitectura
SOAP no puede usar REST porque es un protocolo	REST puede usar SOAP porque puede usar cualquier protocolo
SOAP utiliza interfaces de servicios para exponer lógica de negocio	REST utiliza URLs para exponer lógica de negocio
SOAP define un estándar a seguir estrictamente	REST no define ningún estándar
SOAP requiere mayor ancho de banda y recursos	REST requiere menor ancho de banda y recursos
SOAP define su propia seguridad	Los servicios web RESTful heredan las medidas de seguridad de la capa de transporte
SOAP permite únicamente XML como formato de los datos	REST permite distintos formatos de datos: texto plano, HTML, XML, JSON, etc.

# 5. Servicios web

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Servicios web



# 5. Servicios web

1. Introducción

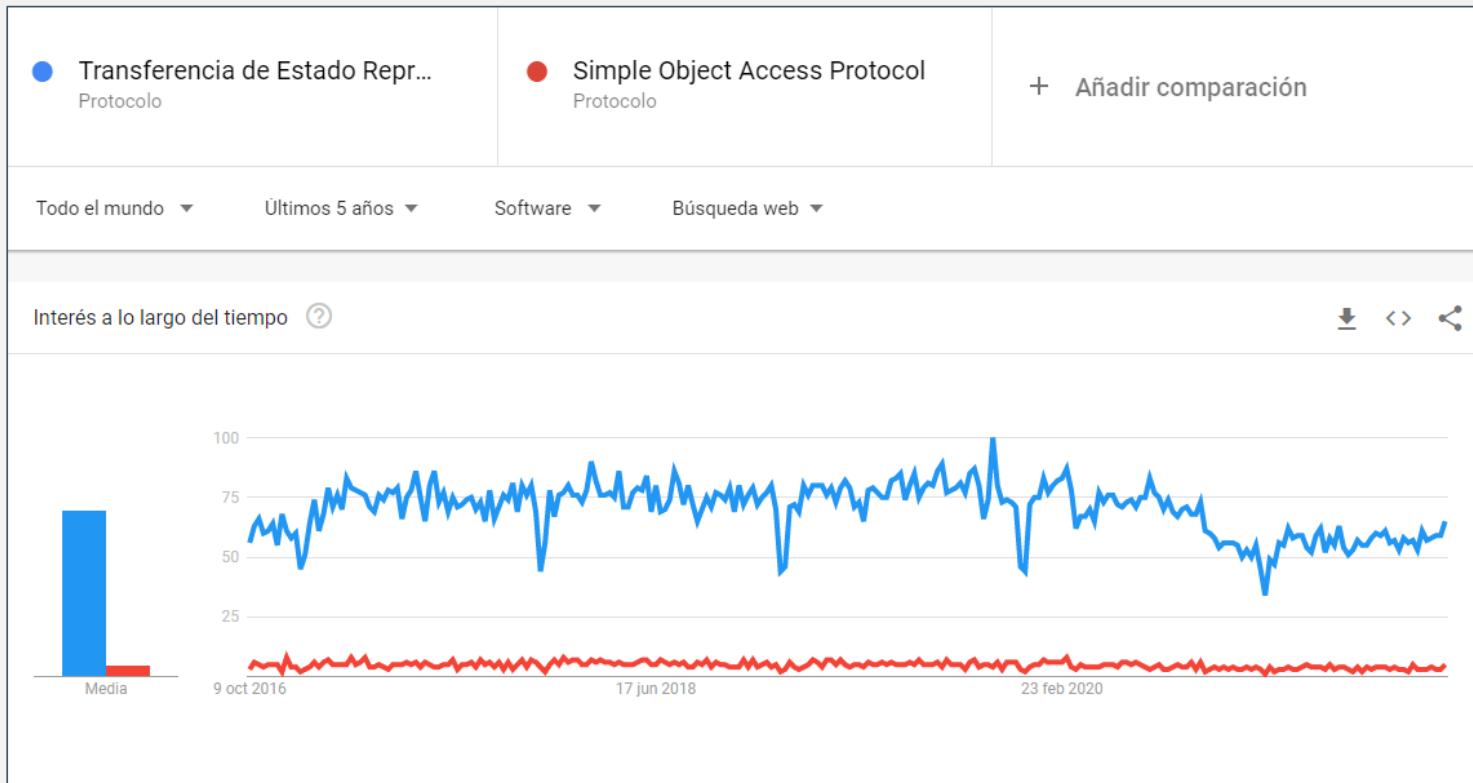
2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud



# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Cloud

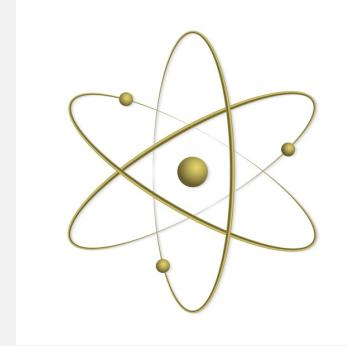
El Cloud un universo de servicios



Recursos  
programables



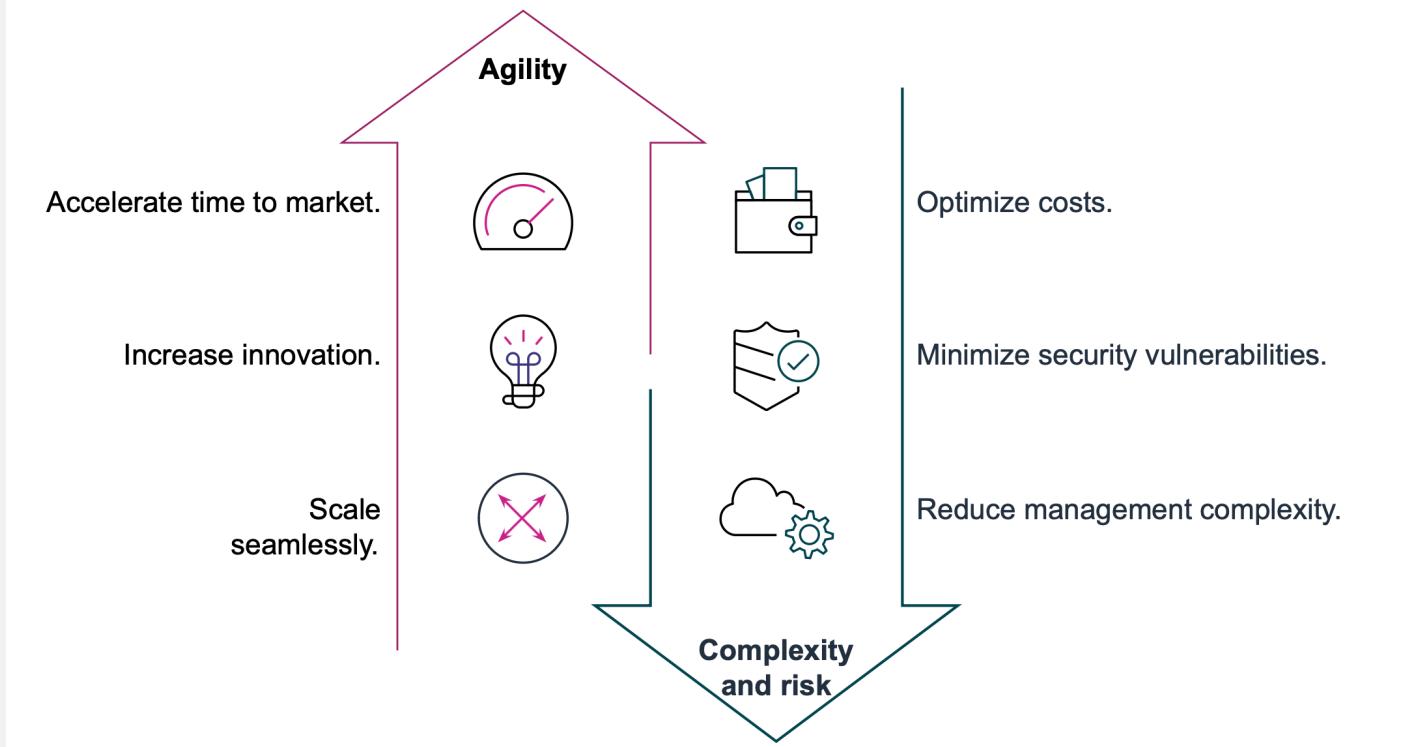
Pago por uso



Dinámico

# 6. Cloud

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud



# 6. Cloud

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

- Infrastructure As A Service
- Platform As A Service
- Software As A Service



# 6. Cloud

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Servicios



Analytics



Customer enablement



Developer tools



Customer engagement



Business applications



Application integration



Migration and transfer



End user computing



Machine learning



Serverless



Networking and content delivery



Database



Security identity and compliance



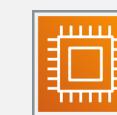
Management and governance



Storage



AWS cost management



Compute



Containers



Game tech



Satellite



Mobile



Robotics



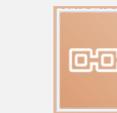
AR and VR



Internet of Things (IoT)



Media services



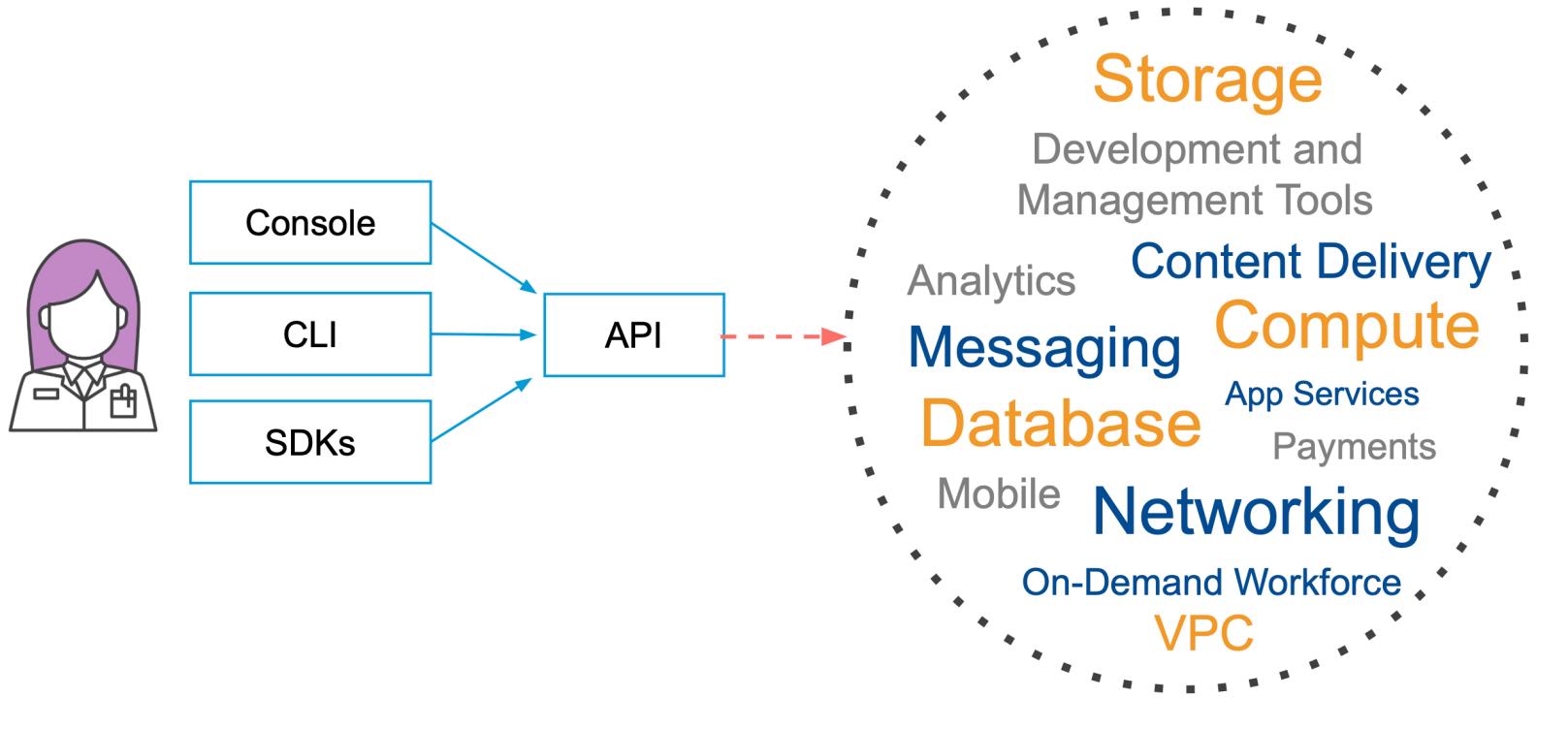
Blockchain



Quantum technologies

## 6. Cloud

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud



# 6. Cloud

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Características

- Los servicios son independientes.
- Acceso a través de consola, línea de comando o SDK.
- La seguridad es centralizada.
- Aumento progresivo de los servicios de manera continua.
- Los servicios son globales
- Proveedores:
  - Amazon Web Services, Microsoft Azure, Alibaba, Google Cloud Platform, Salesforce, IBM y Oracle.

# 6. Cloud

1. Introducción

2. Fundamentos

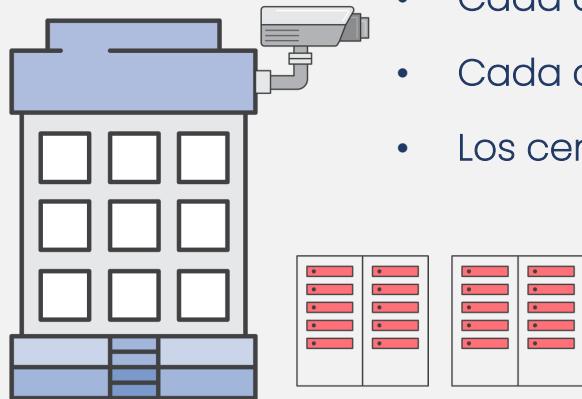
3. Arquitecturas

4. Comunicación

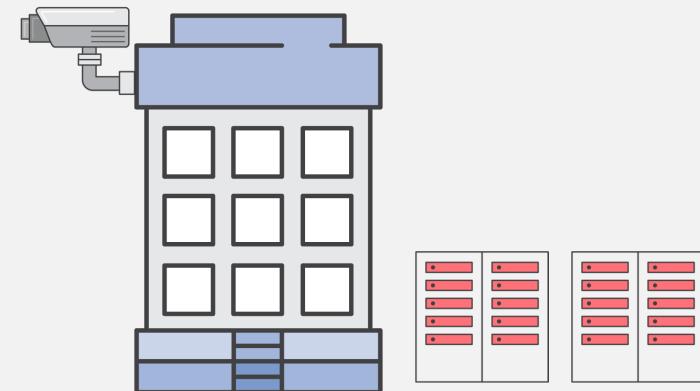
5. Servicios web

6. Cloud

## Centros de datos



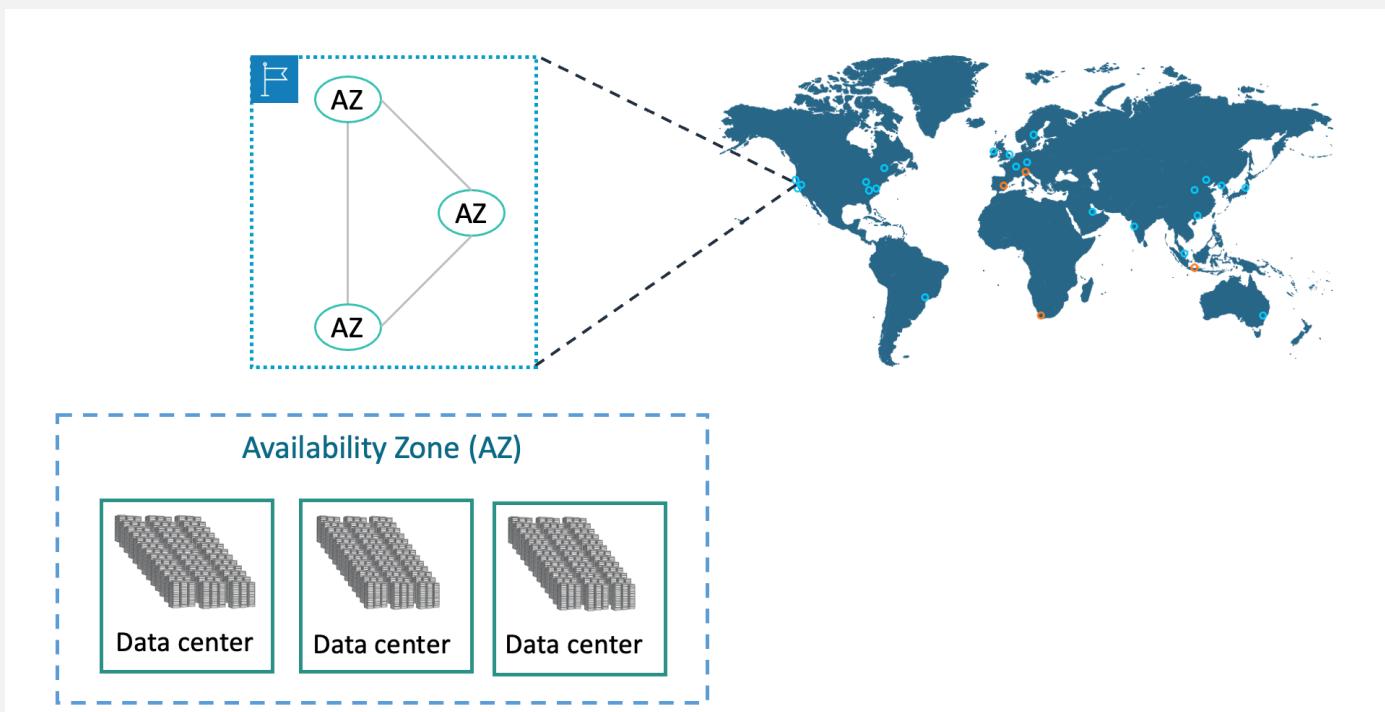
- Los proveedores cloud operan con centros de datos.
- Cada centro de datos alberga miles de servidores.
- Cada centro de datos tiene sus sistema de red.
- Los centros de datos se agrupan en zonas de disponibilidad.



# 6. Cloud

1. Introducción
2. Fundamentos
3. Arquitecturas
4. Comunicación
5. Servicios web
6. Cloud

## Zonas de disponibilidad



# 6. Cloud

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Zonas de disponibilidad

- Una región tiene al menos zonas de disponibilidad.
- Una zona de disponibilidad puede estar compuesta de uno o más CPD.
- Depende del servicio cloud su ámbito puede ser una zona o una región.
- Cada zona está aislada físicamente, energéticamente y a nivel de comunicaciones.

# 6. Cloud

1. Introducción

2. Fundamentos

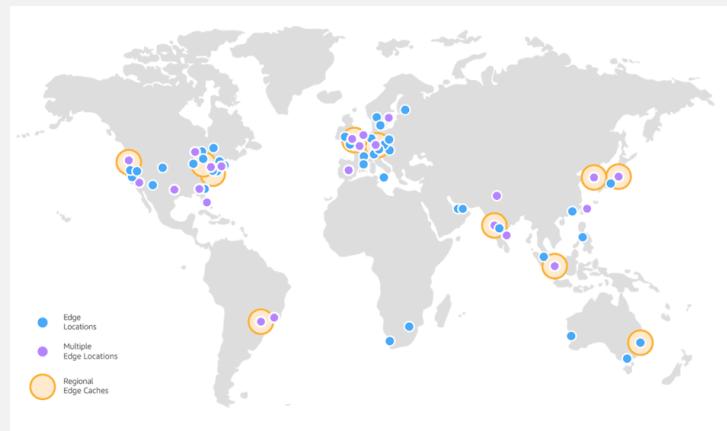
3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

## Regiones



# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud



Governance



Latency



Service availability



Cost

# 6. Cloud

---

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

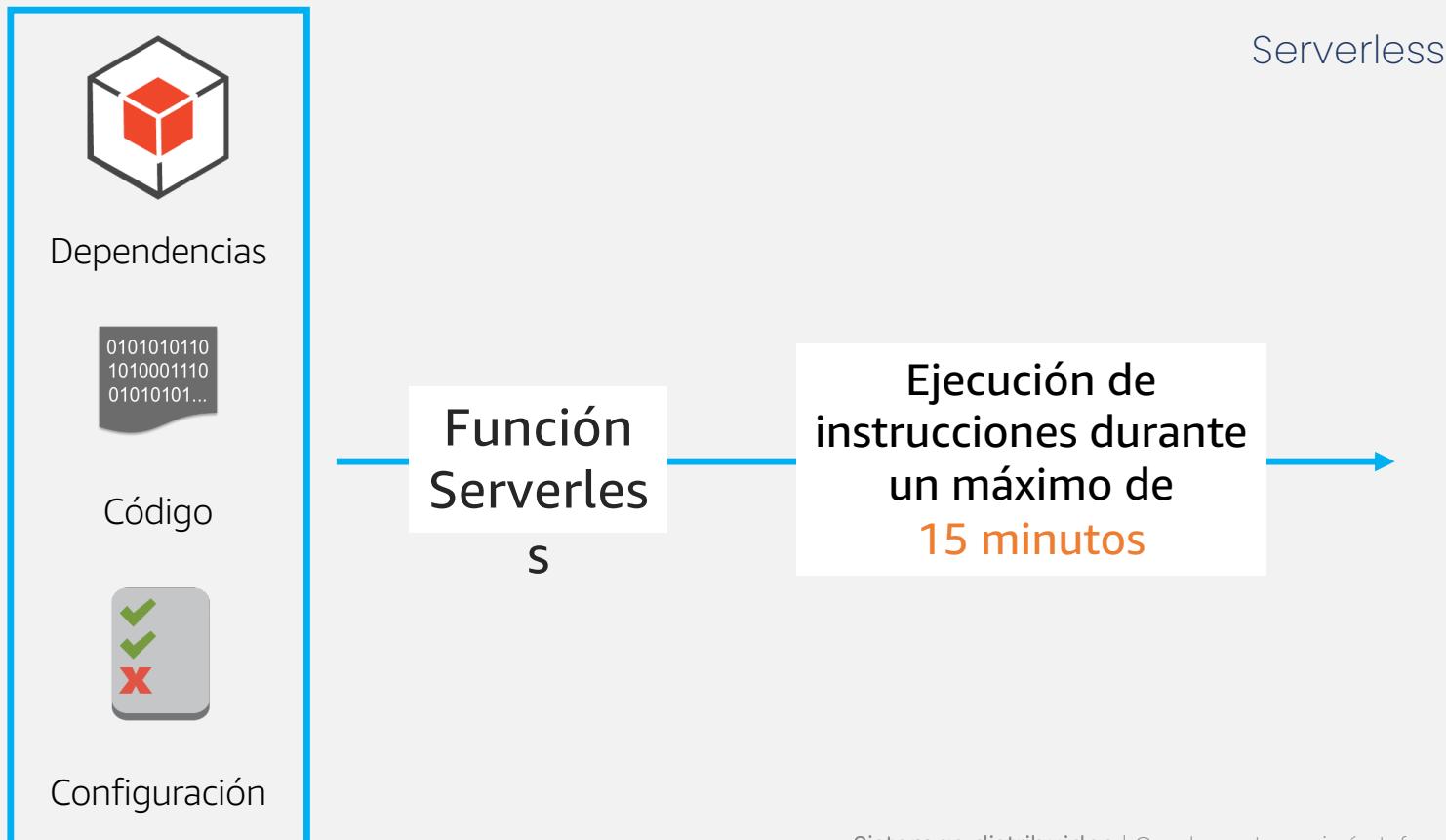
6. Cloud

## Serverless

- Desarrollar y ejecutar aplicaciones y servicios sin administrar servidores
- Su ejecución se basa en eventos
- Tecnología procedente del cloud
- Ejecuta código sin estado
- Admite infinidad de lenguajes de desarrollo

# 6. Cloud

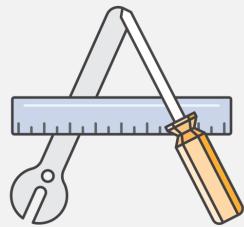
- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud



# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

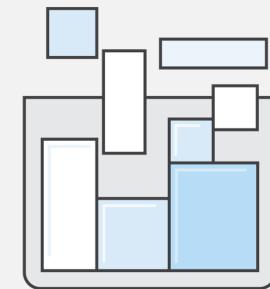
## Serverless



Se centra en su aplicación, no en la configuración.



Utiliza recursos informáticos solo bajo demanda.



Crear una arquitectura de microservicio

S.

# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Serverless



Secuencia de comandos del navegador web

AWS SDK para JavaScript

Serverless Function

```
lambda.invoke(pullParams,
  function(error, data) {
    if (error) {
      prompt(error);
    } else {
      pullResults =
        JSON.parse(data.Payload);
    }
});
```

```
{
  isWinner: false,
  leftWheelImage : {S :
    'cherry.png'},
  midWheelImage : {S :
    'puppy.png'},
  rightWheelImage : {S :
    'robot.png'}
}
```

# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud

## Serverless



Secuencia de comandos del navegador web

AWS SDK para JavaScript

Serverless Function

```
lambda.invoke(pullParams,  
function(error, data) {  
  if (error) {  
    prompt(error);  
  } else {  
    pullResults =  
    JSON.parse(data.Payload);  
  }  
});
```

```
{  
  isWinner: false,  
  leftWheelImage : {S :  
    'cherry.png'},  
  midWheelImage : {S :  
    'puppy.png'},  
  rightWheelImage : {S :  
    'robot.png'}  
}
```

# 6. Cloud

---

## Serverless

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

6. Cloud

Serverless gestiona lo siguiente:

- Servidores
- Necesidades de capacidad
- Implementación
- Escalado y tolerancia a errores
- Actualizaciones del sistema operativo o el lenguaje
- Métricas y registro

# 6. Cloud

---

## Serverless

1. Introducción

2. Fundamentos

3. Arquitecturas

4. Comunicación

5. Servicios web

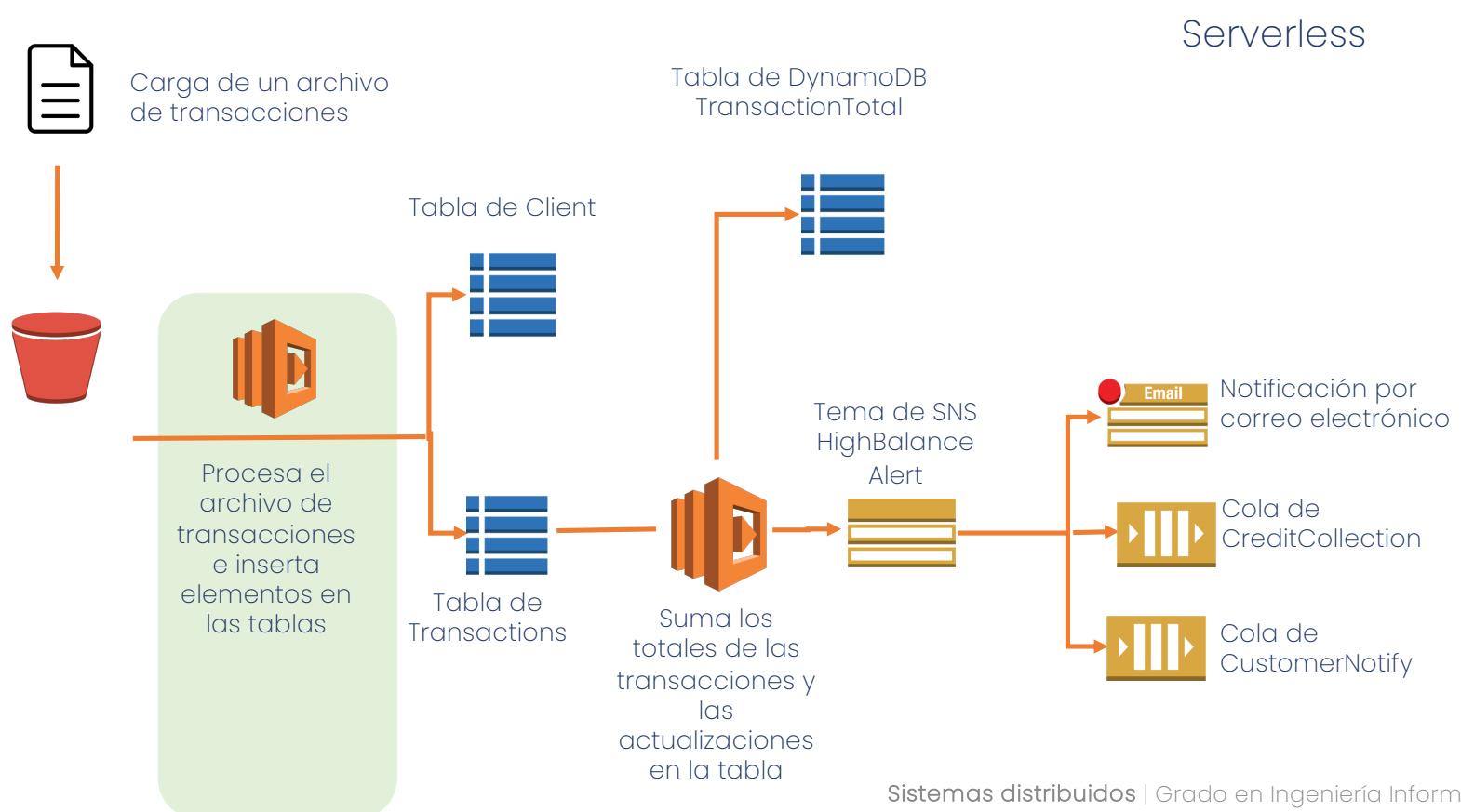
6. Cloud

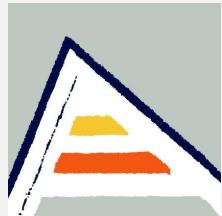
AWS Lambda le permite hacer lo siguiente:

- Aportar su propio código (incluso en bibliotecas nativas).
- Ejecutar código en paralelo
- Crear backends, controladores de eventos y sistemas de procesamiento de datos
- Nunca se abona por recursos inactivos

# 6. Cloud

- 1. Introducción
- 2. Fundamentos
- 3. Arquitecturas
- 4. Comunicación
- 5. Servicios web
- 6. Cloud





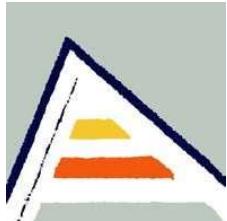
Grado en Ingeniería Informática

Sistemas distribuidos

# Diseño de arquitecturas distribuidas

Departamento de Tecnología Informática y Computación

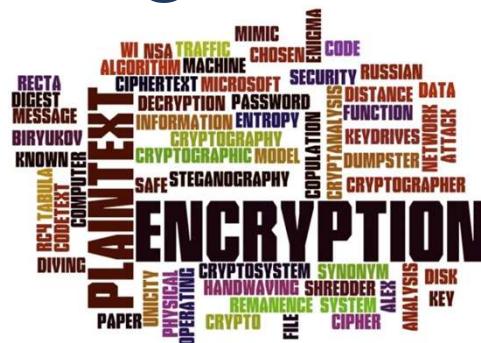
2021 - 2022



# Grado en Ingeniería Informática

# Sistemas distribuidos

# Seguridad



Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## **Coulouris: Distributed systems (2012) Chapter 11**

# recursos y principales

introducción

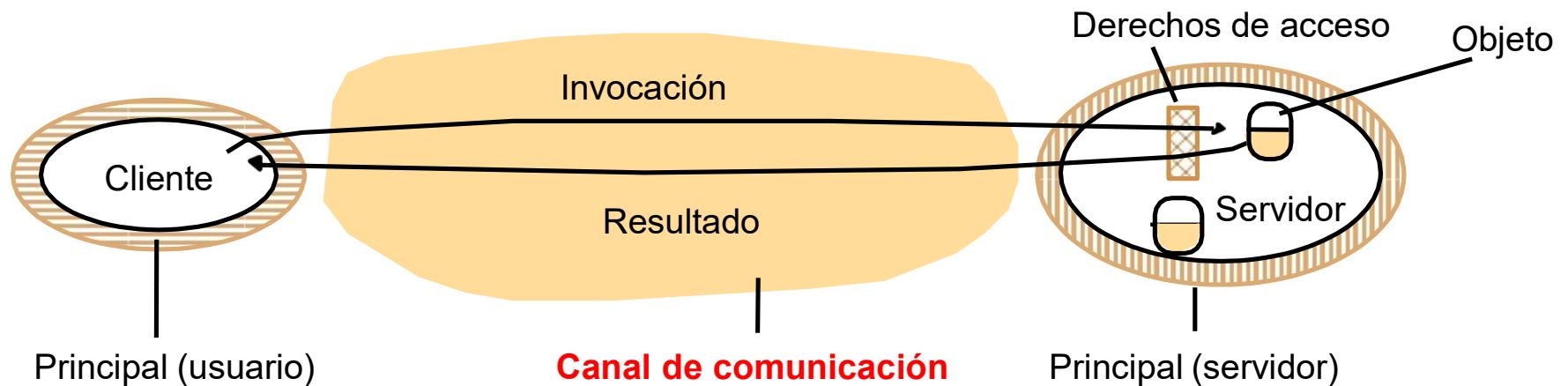
amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas



## ■ Recursos

- Buzón de correo, sistema de archivo, base de datos, parte de una web comercial...

## ■ Principal

- Usuario y/o proceso que tiene derechos para acceder y realizar acciones sobre los recursos
- Fundamental en la “Autenticación” y “Autorización”

# canales seguros

---

introducción

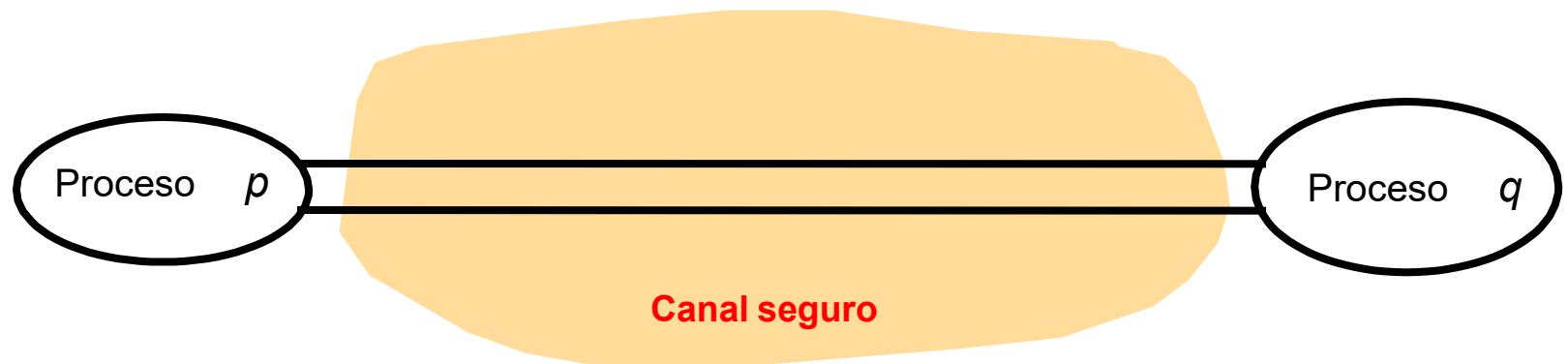
amenazas

criptografía

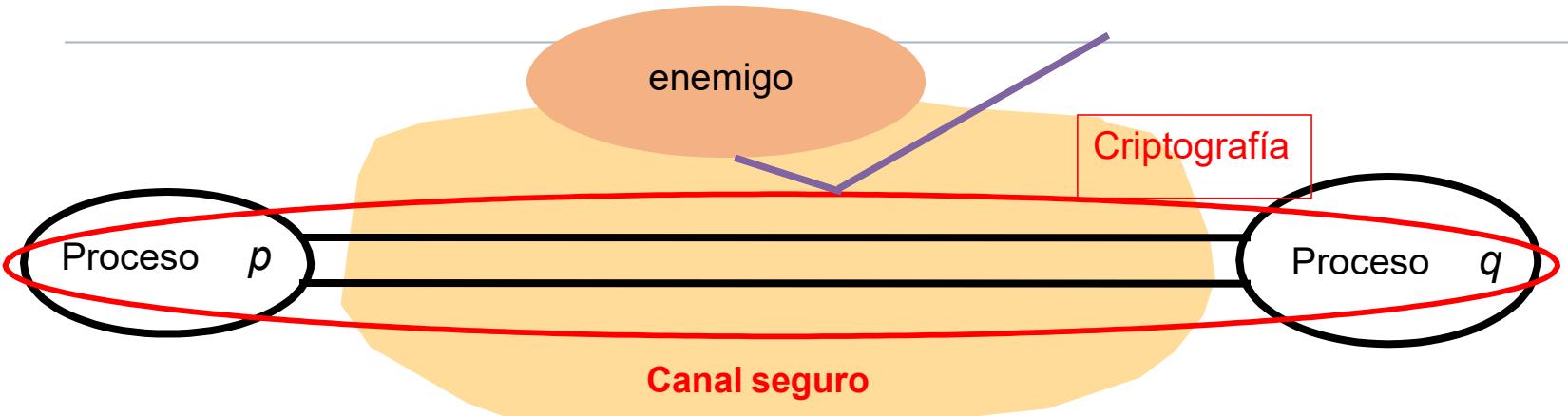
escenarios

autenticación y  
autorización

buenas  
prácticas



# canales seguros



- **Propiedades que debe cumplir**
  1. Cada proceso está seguro de la identidad del otro
  2. Los datos son privados y protegidos contra la manipulación
  3. Protección contra repeticiones y reordenación de datos
- **Se suele utilizar criptografía**
  - El secreto se preserva mediante ocultamiento criptográfico
  - Autenticación basada en la prueba de posesión de un secreto

# vulnerabilidades y amenazas

introducción

**amenazas**

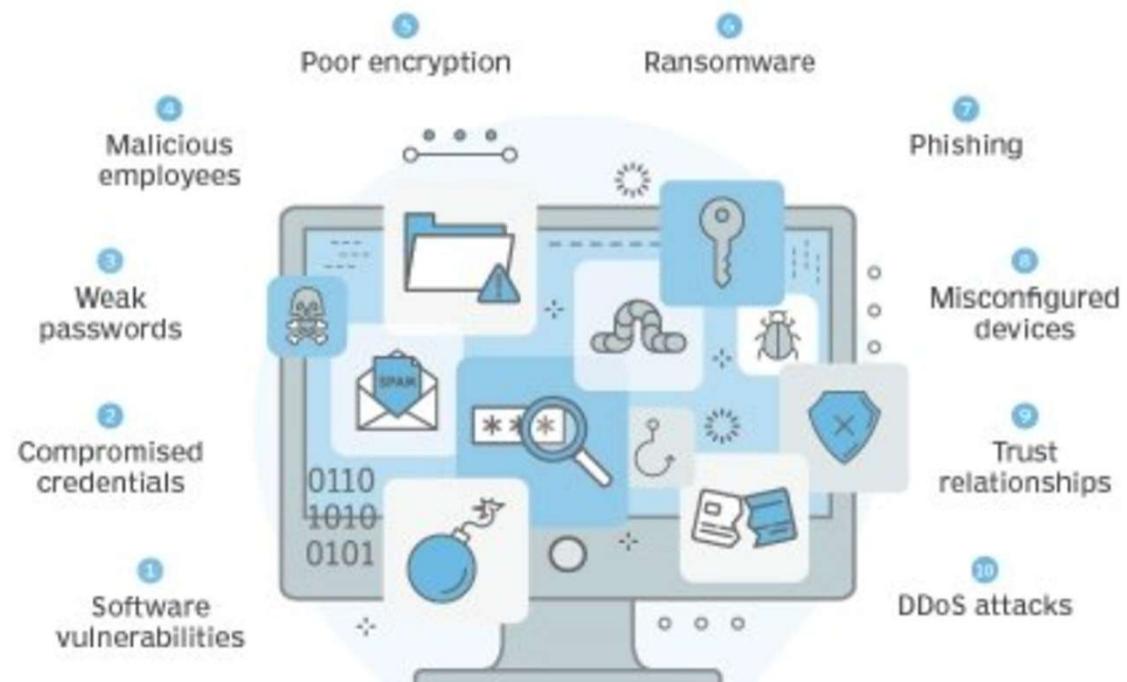
criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## 10 common attack vectors



# vulnerabilidades y amenazas



introducción

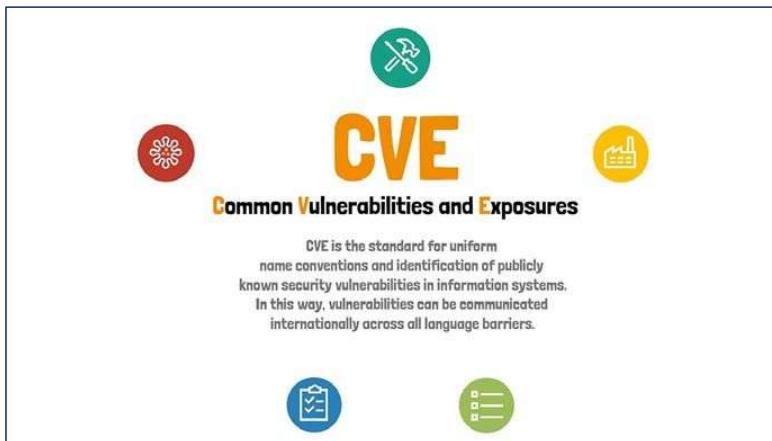
amenazas

criptografía

escenarios

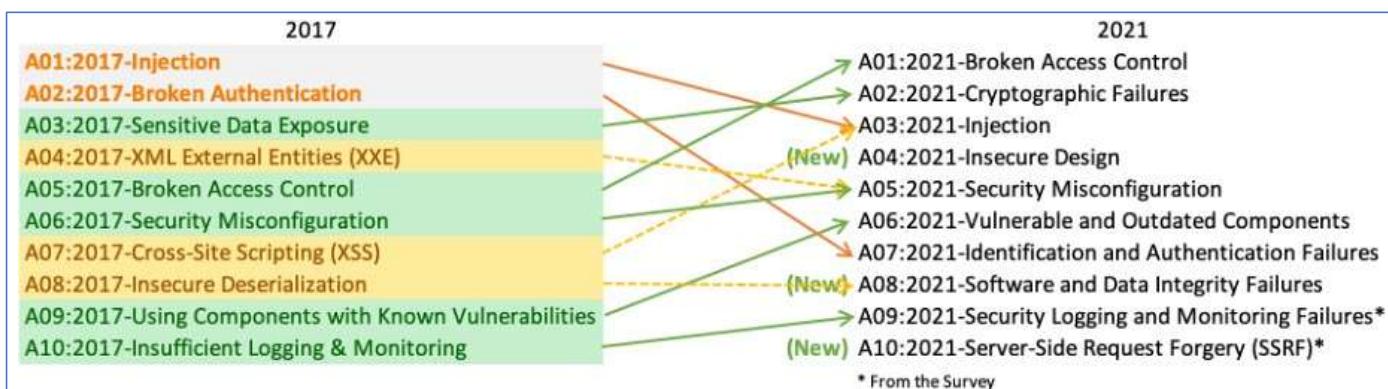
autenticación y  
autorización

buenas  
prácticas



Identifican, definen y catalogan vulnerabilidades descubiertas públicamente

<https://www.cve.org/>



Top 10 de los riesgos en aplicaciones Web

<https://owasp.org/>

# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Los sistemas distribuidos son vulnerables

### Servicios expuestos a Internet

- Visibles con herramientas como SHODAN, NMAP, Wireshark, ...
- Posibilidad de acceso remoto ¿ssh permitido? ¿puerto 22 abierto?
- **Errores de programación o en el diseño de la seguridad**
- Sistemas desactualizados (Windows XP en sistemas heredados)

The screenshot shows a web browser displaying search results from SHODAN. The URL in the address bar is `www.shodanhq.com/?q=Default+Password&feed=1`. The results list three IP addresses and their ports:

- 188.96.188.75:80**:  
HTTP/1.0 401 Unauthorized  
Date: Sun, 25 Oct 1970 09:54:08 GMT  
Server: Boa/0.93.15 (with Intersil External)  
Connection: close  
WWW-Authenticate: Basic realm="LOGIN Enter Password (default is medion, ignore username)"  
Content-Type: text/html
- 141.51.248.254:80**:  
HTTP/1.0 401  
Date: Sat, 21 Dec 1996 12:00:00 GMT  
WWW-Authenticate: Basic realm="Default password:1234"
- 203.223.200.183:8080**:  
HTTP/1.0 401 Unauthorized  
Server: GoAhead-Webs  
Date: Sat Jan 8 05:23:40 2000  
WWW-Authenticate: Basic realm="Default: admin/password"  
Pragma: no-cache  
Cache-Control: no-cache  
Content-Type: text/html

Annotations highlight specific parts of the results:

- A red box around the URL parameter `&feed=1` in the address bar is labeled "SHODAN results via RSS feed using &feed=1 URL parameter".
- A red box around the password "1234" in the second result is labeled "Finding lots of default passwords via SHODAN".

# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Errores de diseño y/o programación



**Un murciano recibe una factura telefónica de 19.500 euros tras enviar 100.000 SMS por un error de su compañía**

EFE NOTICIA 30.10.2021 - 17:38H

La empresa llegó a cortarle la línea móvil y a amenazarle con llevarlo al registro de morosos.

**You knitwits! Couple with KN19 TER number plate on their car get hit with £90 fine... thanks to woman in 'Knitter' shirt walking along bus lane**

- David Knight, 54, from Dorking, Surrey, received a fine for driving down a bus lane in Bath - but the builder immediately knew there had been a mistake
- He and his wife, Paula, 54, looked at photographic evidence which showed a woman walking in the bus lane with the word KNITTER on her t-shirt
- The council computer mixed up the wording on her t-shirt for a personalised registration plate on Mr Knight's Volkswagen van, KN19TER

By LIZ HULL FOR THE DAILY MAIL  
PUBLISHED: 22:49



Sistemas distribuidos | Grado en Ingeniería Informática

# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

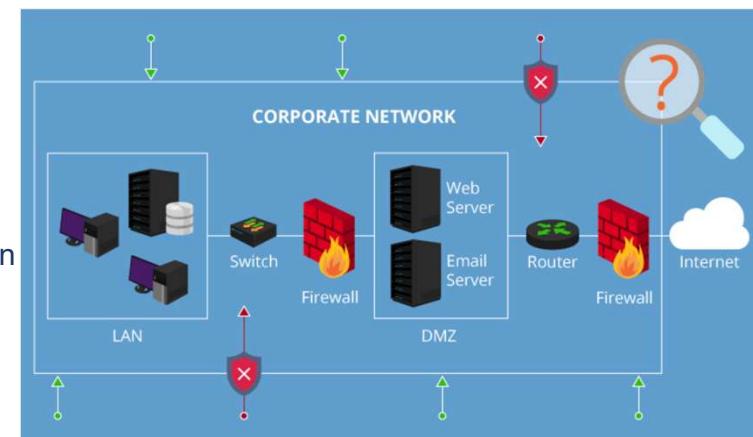
## Los sistemas distribuidos son vulnerables

### Servicios expuestos a Internet

- Visibles con herramientas como SHODAN
- Posibilidad de acceso remoto ¿ssh permitido? ¿puerto 22 abierto?
- Errores de programación o en el diseño de la seguridad
- Sistemas desactualizados

### Acceso físico a la infraestructura

- Concentradores y routers en lugares públicos accesibles sin autorización
- Salas de espera con regletas ethernet



# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Los sistemas distribuidos son vulnerables

### Servicios expuestos a Internet

- Visibles con herramientas como SHODAN
- Posibilidad de acceso remoto ¿ssh permitido? ¿puerto 22 abierto?
- Errores de programación o en el diseño de la seguridad
- Sistemas desactualizados

### Acceso físico a la infraestructura

- Concentradores y routers en lugares públicos accesibles sin autorización
- Salas de espera con regletas ethernet

### Proximidad a la infraestructura

- Wifi y honeypots
- Bluetooth y BLE
- RFID



# vulnerabilidades y amenazas

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Los sistemas distribuidos son vulnerables

### Servicios expuestos a Internet

- Visibles con herramientas como SHODAN
- Posibilidad de acceso remoto ¿ssh permitido? ¿puerto 22 abierto?
- Errores de programación o en el diseño de la seguridad
- Sistemas desactualizados

### Acceso físico a la infraestructura

- Concentradores y routers en lugares públicos accesibles sin autorización
- Salas de espera con regletas ethernet

### Proximidad a la infraestructura

- Wifi y honeypots
- Bluetooth y BLE
- RFID

### Desastres naturales

- Incendios
- Inundaciones

# vulnerabilidades y amenazas

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Incendio del principal *datacenter* en OVHCloud (marzo 2021)

[Enlace a la noticia: no revelarán el motivo del incendio hasta 2022](#)

(premonitorio en la 1<sup>a</sup> temporada de la serie MR. ROBOT)



# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

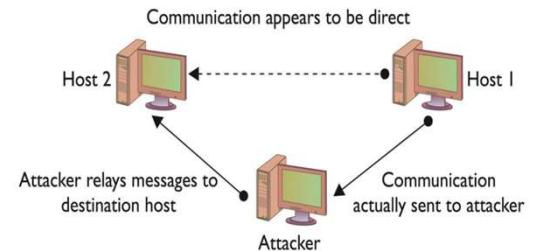
autenticación y autorización

buenas prácticas

## Tipos de ataque

### **Eavesdropping o escuchar a escondidas / Man-in-the-Middle (MitM)**

Consiste en obtener los mensajes que se envían y se reciben en un dispositivo objetivo

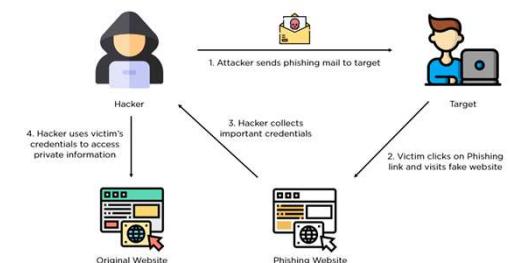


### Ingeniería social

Basado en técnicas más de psicología que de tecnología para engañar a la víctima normalmente suplantando la identidad de otra entidad o persona: nos cuesta decir no, siempre queremos ayudar, nos gusta que nos alaben, emociones como miedo, amor, codicia, ...

### **Phishing**

Tipo de ataque de ingeniería social: a través de mensajes de chat o correo electrónico



### Inyección de código

Inyección de cadenas que aprovechan vulnerabilidades en el diseño en implementaciones de protocolos y/o aplicaciones web. Normalmente usando métodos de acceso legítimos



### **Hijacking**

Secuestro de elementos del entorno de internet: urls, DNS, navegador, sesión, ...

# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Tipos de ataque

### Fuerza bruta

Aprovechando una criptografía débil o un método de autenticación vulnerable. Este tipo de ataques consiste en una búsqueda exhaustiva de claves por combinatoria. Suelen aprovechar diccionarios y conocimiento de esquemas de creación de contraseñas



### Manipulación de mensajes y reenvíos

Alterar el contenido de mensajes en tránsito o almacenar mensajes seguros y enviándolos con retraso

### Ingeniería inversa

Obtener información confidencial analizando el software (*firmware* o *software*) de un dispositivo o servicio con técnicas de ingeniería inversa



# vulnerabilidades y amenazas

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Tipos de ataque

### Ransomware

Cifrado de los datos de un sistema impidiendo el acceso a los mismos y solicitando una recompensa para liberar el sistema



### ¿Cómo funciona un ransomware?



\* También pueden ser otros links de Internet en los que se descarguen los archivos maliciosos al hacer clic.

\*\* Llamado Command and Control (C&C)

Fuente: Carbon Black Threat Report 2016

statista

# vulnerabilidades y amenazas

introducción  
amenazas  
criptografía  
escenarios  
autenticación y autorización  
buenas prácticas

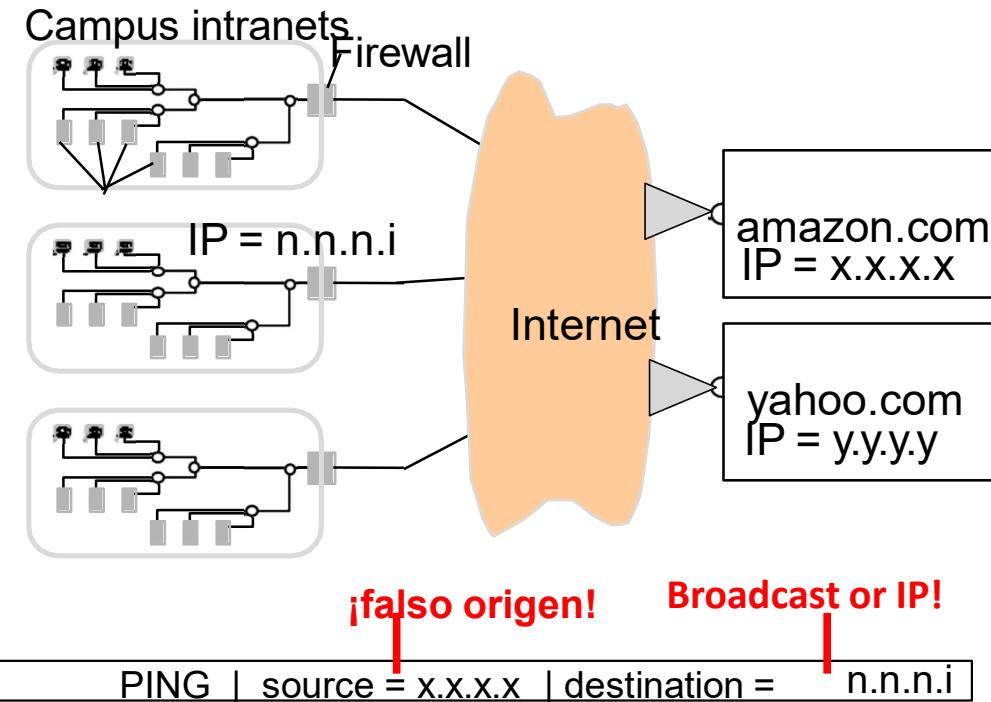
## Tipos de ataque

### DDoS (Denegación de servicio distribuido)

Se inundan los canales de comunicación de los servidores con peticiones aparentemente legítimas que impiden la conexión a los clientes

### [Año 2000 – uno de los primeros]

Ejemplo DDoS usando suplantación de IP  
*Desde un servidor malicioso se hace ping a muchas máquinas...*



# vulnerabilidades y amenazas

introducción  
amenazas  
criptografía  
escenarios  
autenticación y autorización  
buenas prácticas

## Tipos de ataque

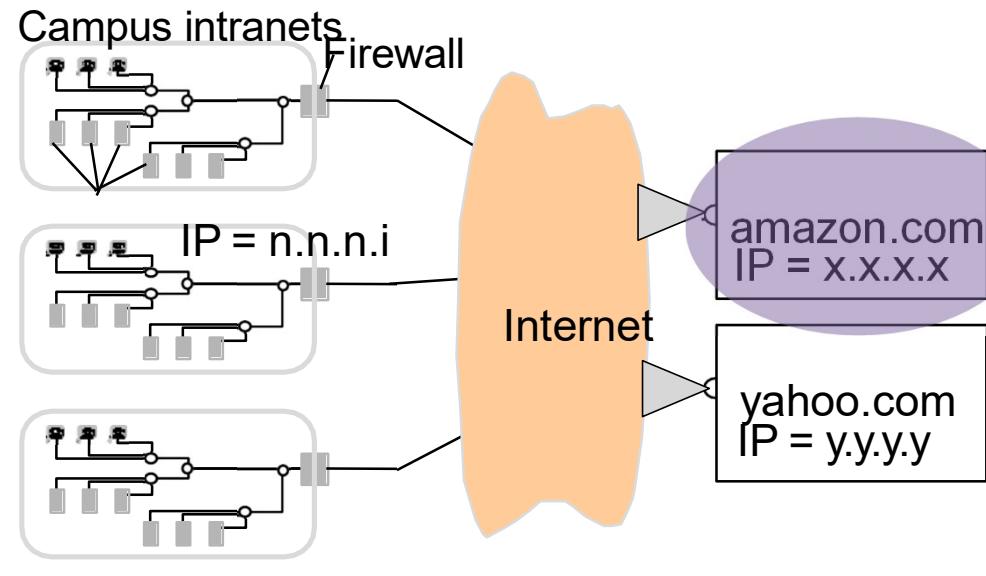
### DDoS (Denegación de servicio distribuido)

Se inundan los canales de comunicación de los servidores con peticiones aparentemente legítimas que impiden la conexión a los clientes

### [Año 2000 – uno de los primeros]

Ejemplo DDoS usando suplantación de IP  
*Desde un servidor malicioso se hace ping a muchas máquinas...*

*...como resultado todas ellas hacen pong:*



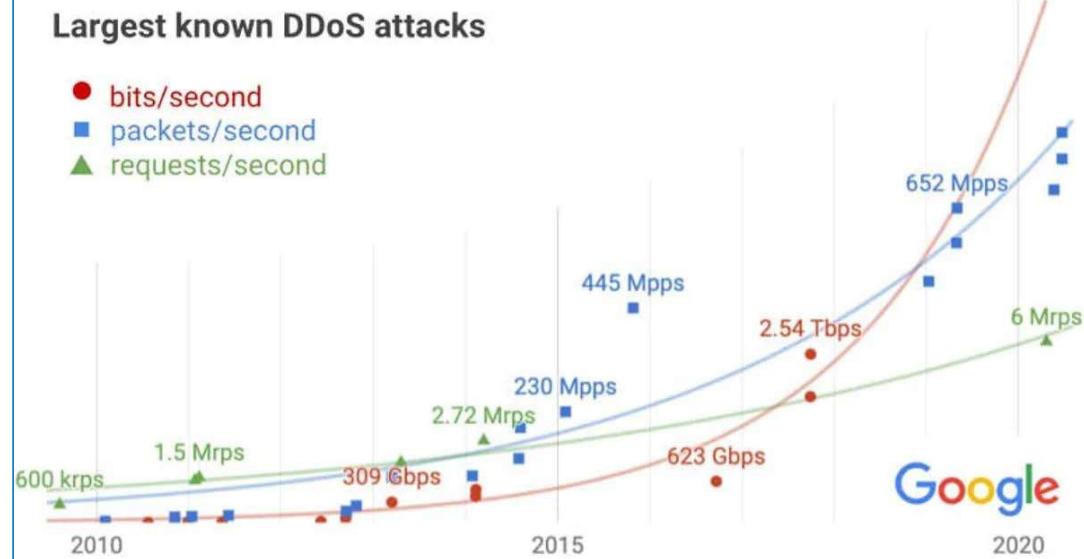
# vulnerabilidades y amenazas

introducción  
amenazas  
criptografía  
escenarios  
autenticación y autorización  
buenas prácticas

## Tipos de ataque

### *DDoS más relevantes:*

- Ataque a Google, sept. de 2017 (público en 2020)
- Ataque a AWS, febrero de 2020
- Ataque a GitHub, febrero de 2018
- Ataque a Dyn, 2016
- Ataque a GitHub, 2015
- Ataque a Spamhaus, 2013
- Ataque de Mafiaboy, 2000 (hacker de 15 años)
- Ataque a Estonia, 2007



# orígenes de la criptografía

introducción

amenazas

**criptografía**

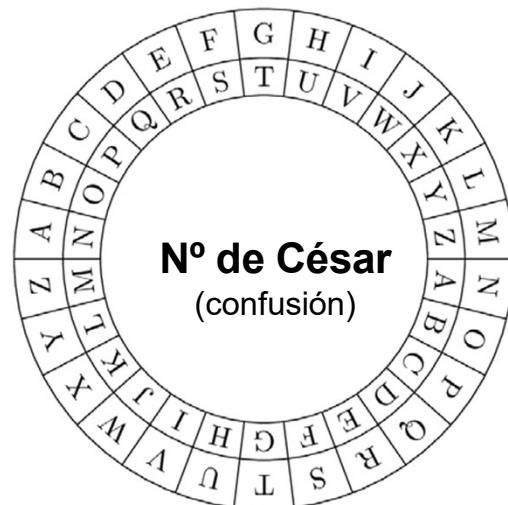
escenarios

autenticación y  
autorización

buenas  
prácticas

La **criptografía** es tan antigua como la historia de la Humanidad. Los primeros sistemas se basaban en los conceptos **CONFUSIÓN** y **DIFUSIÓN** pero usados por separado: altamente vulnerables

Confusión/Sustitución



Difusión/Permutaciones



**Scytale**  
(difusión)

# orígenes de la criptografía

---

introducción

amenazas

**criptografía**

escenarios

autenticación y  
autorización

buenas  
prácticas

Los sistemas criptográficos comenzaron a ser seguros cuando se combinaron ambas estrategias (cifrado simétrico):

## **Confusión**

La relación entre el texto cifrado y la clave sea lo más compleja posible. Una clave aleatoria del tamaño del texto en claro es un sistema robusto

## **Difusión**

La difusión pretende diluir el contenido del texto plano a lo largo del criptograma. La trasposición-permutación de los caracteres del texto en claro en el texto cifrado logran este objetivo

# algoritmos criptográficos

---

introducción

amenazas

**criptografía**

escenarios

autenticación y  
autorización

buenas  
prácticas

## Cifrado simétrico

El primero que se desarrolló. La misma clave para encriptar y para desencriptar

## Cifrado asimétrico

También llamado de clave pública y clave privada

## Cifrado híbrido

Combina tanto el cifrado simétrico como el asimétrico, aprovechando las mejores características de cada uno

## Funciones HASH o Resumen

A partir de una entrada M producen una cadena única de tamaño fijo  $H(M)$

# cifrado simétrico

introducción

amenazas

criptografía

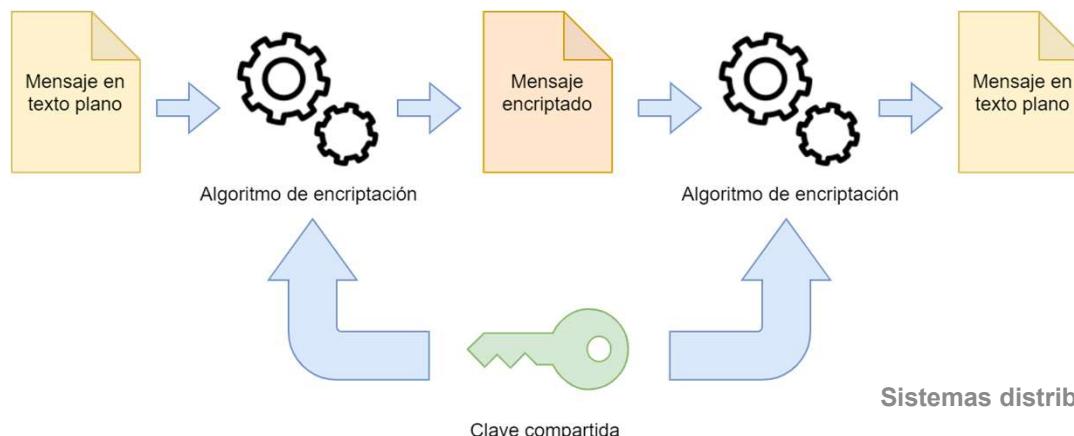
escenarios

autenticación y  
autorización

buenas  
prácticas

## Características

- Se utiliza una única clave privada para encriptar y desencriptar
- La clave debe ser conocida tanto por emisor como por receptor
- Los ataques que reciben suelen ser por fuerza bruta (o intercepción de la clave)
- Son muy rápidos y eficientes (en comparación con el cifrado asimétrico) y pueden **cifrar grandes cantidades** de información



# cifrado simétrico

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

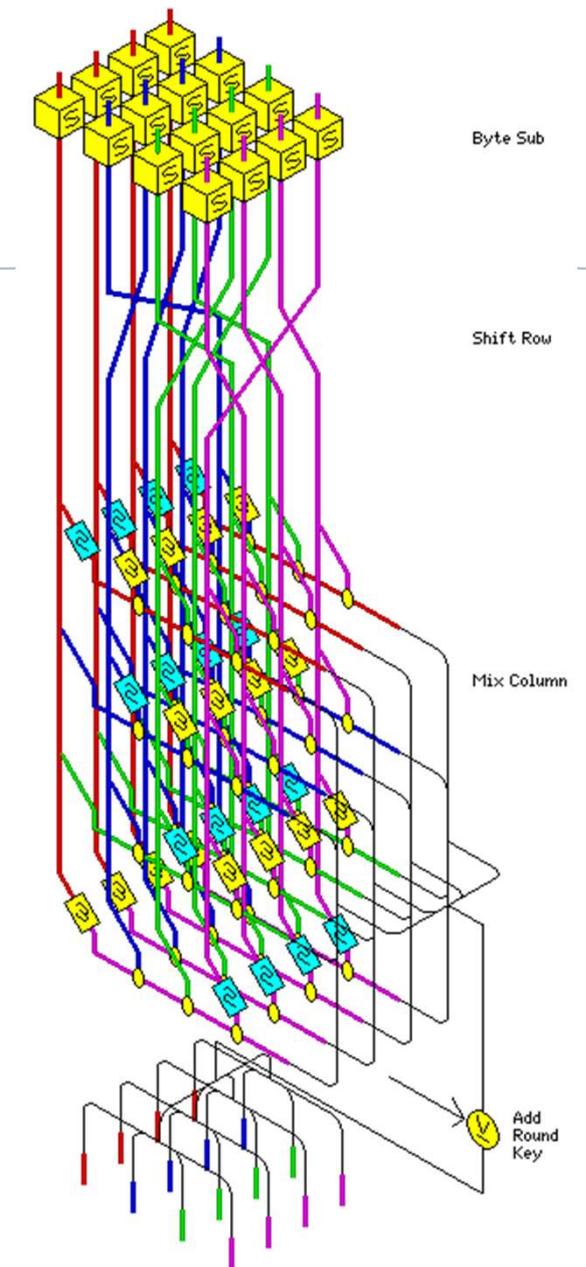
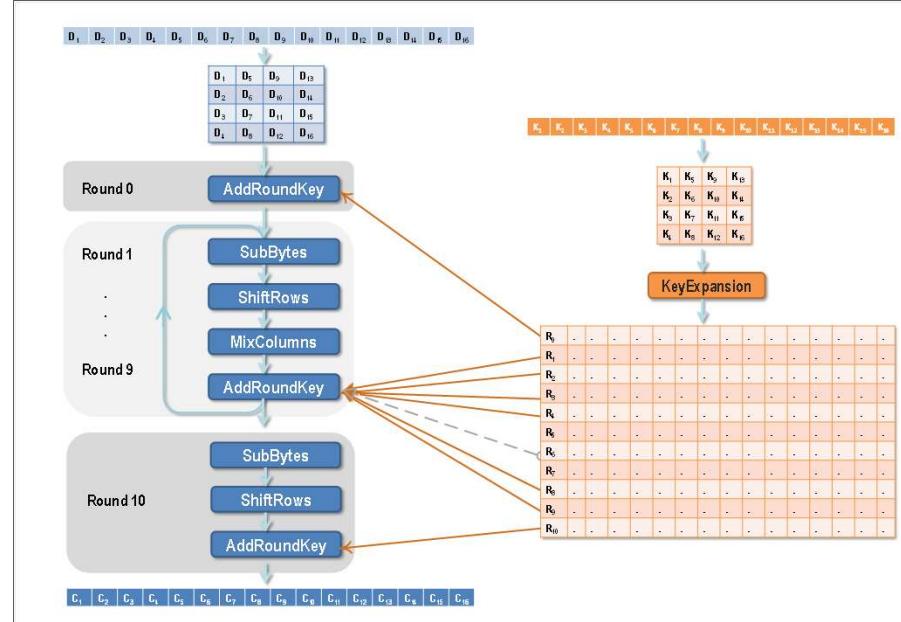
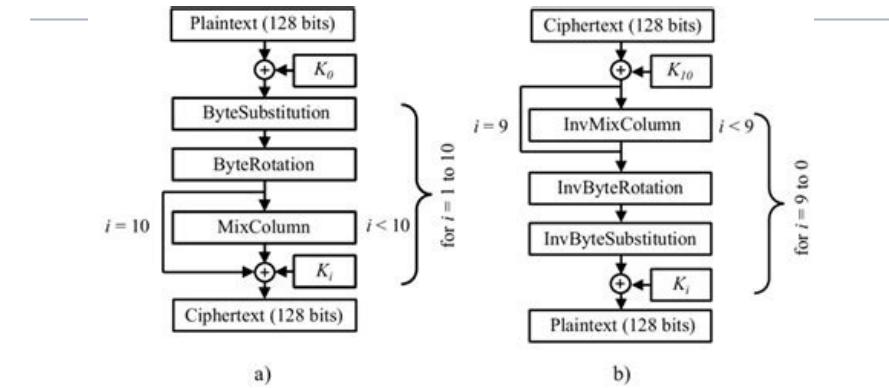
buenas  
prácticas

## Principales algoritmos

- **AES** (Advanced Encryption Standard): Sustituto avanzado de DES. Hasta la fecha no se ha encontrado ninguna vulnerabilidad desde 1997. Clave de 128, 160, 192, 224 o 256 bits. y sus variantes AES-CBC, AES-CFB, AES-OFB y **AES-GCM**. Permite uso de hardware específico (AES-NI). Cifrador de bloques de 128 bits. 100-150 Mbps.  
**Actualmente es el estándar utilizado**
- **Blowfish**: diseñado en 1993 por Bruce Schneier. Tiene un tamaño de bloque de 64 bits y una longitud de clave variable desde 32 bits hasta 448 bits
- **TwoFish**: Mejora del Blowfish, diseñado en 1998 por Bruce Schneier con cifrado de bloque de 128 bits y tamaños de clave de hasta 256 bits
- **ChaCha20. Cifrador de flujo** desarrollado en 2008 por Daniel J. Bernstein. Destaca por ser más rápido que AES cuando no se dispone de hardware de cifrado. Utiliza claves de 356 bits.
- **Triple DES**: Aplica DES tres veces con dos claves distintas (IBM, 1988). Clave de 168 bits (24 bytes). Bloques de 64 bits. Lento en comparación con los anteriores. Actualmente en desuso
- **TEA** (Tiny Encryption Algorithm): Desarrollado en la Universidad de Cambridge (1994). Clave de 128 bits (16 bytes). Bloques de 64 bits. **[NOTA]**: obsoleto, se mantiene como ejemplo explicativo de los cifrados simétricos]

introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas

# cifrado simétrico: AES - Rijndael



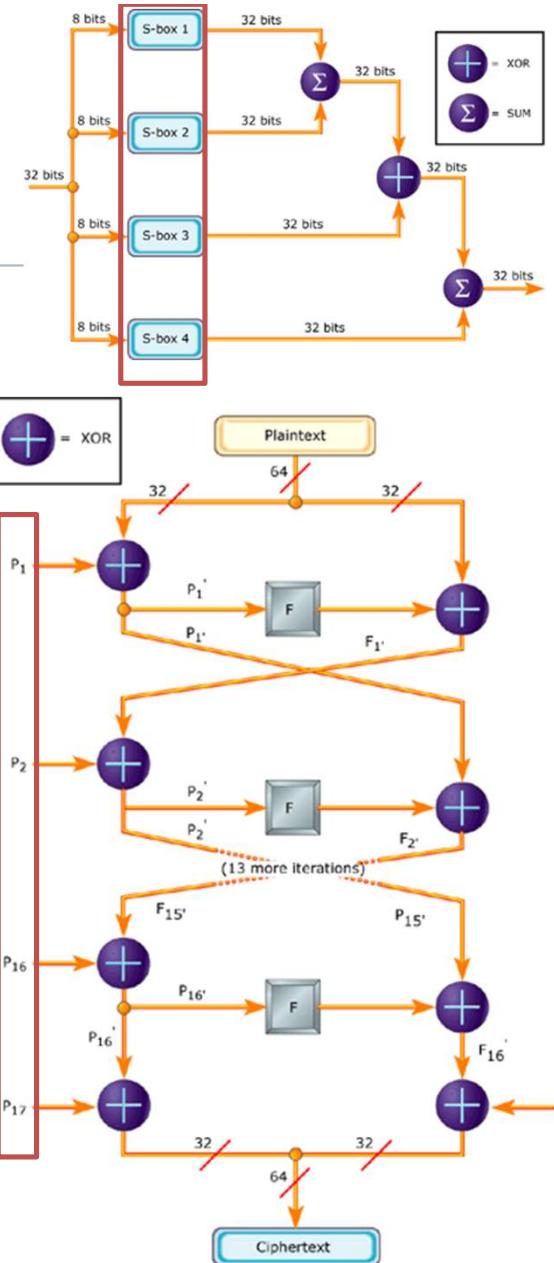
# cifrado simétrico: Blowfish

El mensaje de texto plano de 64 bits se divide:

1. Los 32 bits "izquierdos" hacen XOR con el primer elemento de una matriz P para crear un valor  $P'$ ,
2. Se pasa por una función de transformación llamada F,
3. Realiza XOR con los 32 bits "derechos" del mensaje para producir un nuevo valor  $F'$ .
4. Los datos el procesamiento Izq. y Drch. Se intercambian y se repite 15 veces.
5. Los valores  $P'$  y  $F'$  resultantes se combinan XOR con las dos últimas entradas de la matriz P y se recombinan para obtener el texto cifrado de 64 bits.

La función F divide una entrada de 32 bits en cuatro bytes y los utiliza como índices en una matriz S. Los resultados de la búsqueda se suman y XOR juntos para producir la salida

P y S son calculadas a partir de la clave del usuario.



# cifrado simétrico: ChaCha20

introducción

amenazas

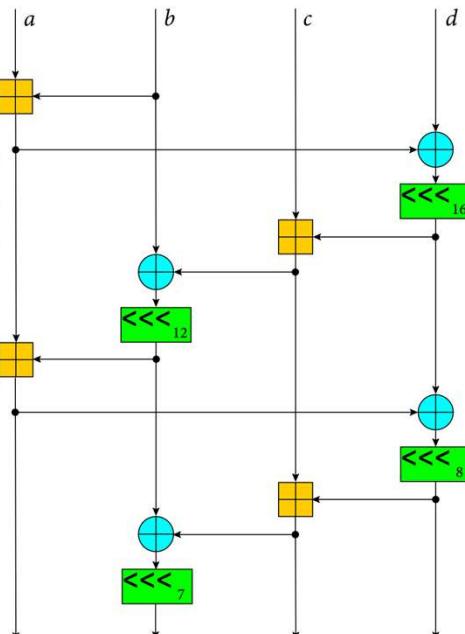
criptografía

escenarios

autenticación y autorización

buenas prácticas

```
// Odd round
QR(0, 4, 8, 12) // 1st column
QR(1, 5, 9, 13) // 2nd column
QR(2, 6, 10, 14) // 3rd column
QR(3, 7, 11, 15) // 4th column
// Even round
QR(0, 5, 10, 15)
QR(1, 6, 11, 12)
QR(2, 7, 8, 13)
QR(3, 4, 9, 14)
```



```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<= 8;
c += d; b ^= c; b <<<= 7;
```

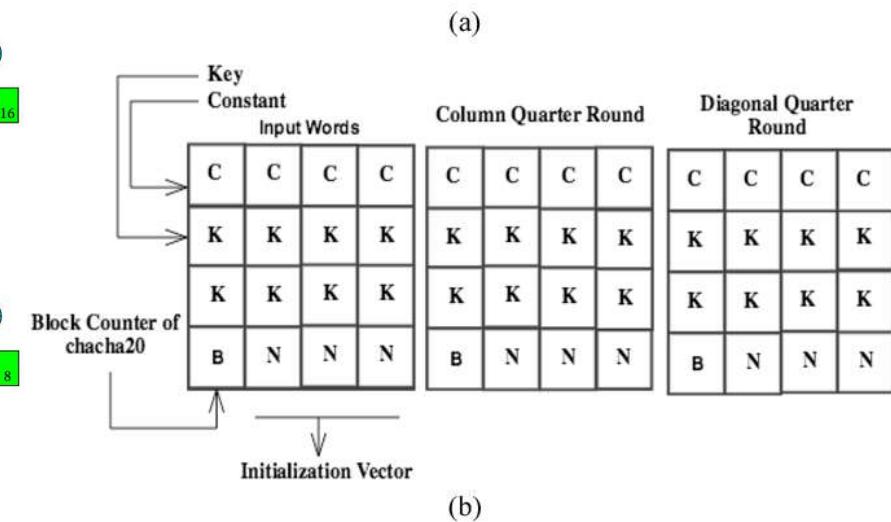
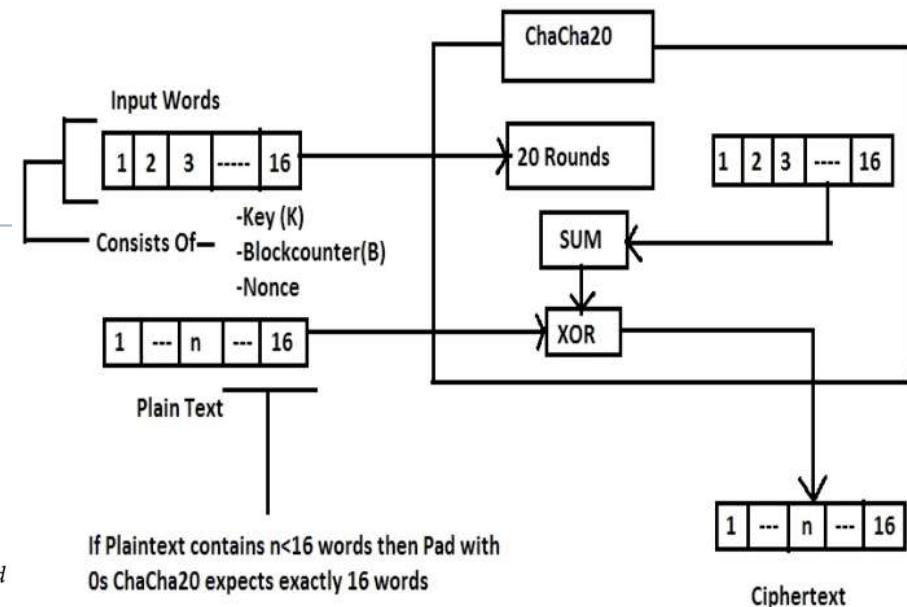


Figure 2 (a), (b): Mechanism of ChaCha20 Encryption Algorithm

# cifrado simétrico: 3DES - Data Encryption Standard

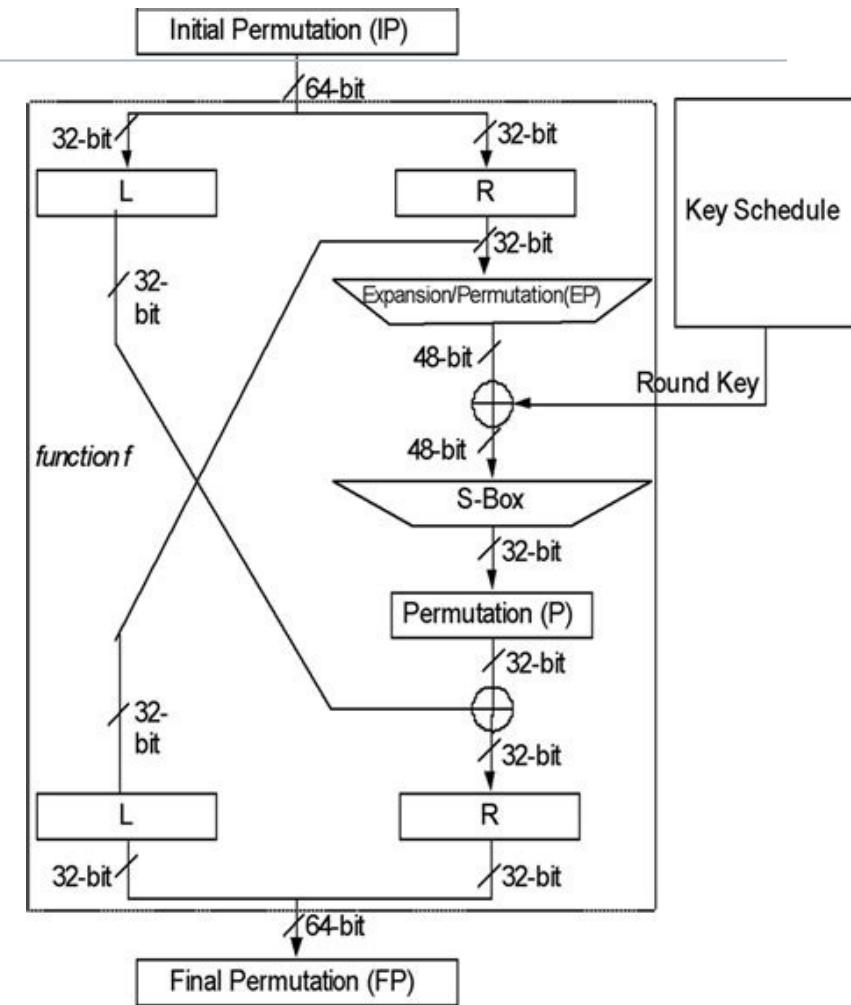
introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas

Hoy en día, DES se considera inseguro para muchas aplicaciones. Esto se debe principalmente a que el tamaño de clave de 56 bits es corto.

Las claves de DES se rompen en menos de 24 horas. Además, existen también resultados analíticos que demuestran debilidades teóricas en su cifrado, aunque son inviables en la práctica.

Se cree que el algoritmo es seguro en la práctica en su variante de [Triple DES \(3 claves; 3 pasadas\)](#), aunque existan ataques teóricos.

Este algoritmo no está exento de polémica ya desde su concepción por IBM y aceptación por la NSA. Las S-Boxes de DES fueron objeto de intensivo estudio durante años con la intención de localizar una [puerta trasera](#). El criterio del diseño de la S-Box fue que habían sido diseñadas para mostrarse resistentes a este ciertos ataques y que incluso una leve modificación de una S-Box podría haber debilitado significativamente DES.



# cifrado simétrico: TEA

introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas

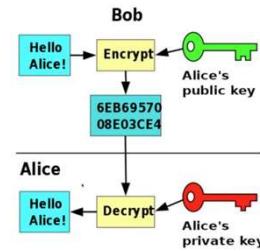
clave 4 x 32 bits

```
void encrypt(unsigned long k[], unsigned long text[]) {  
    unsigned long y = text[0], z = text[1];  
    unsigned long delta = 0x9e3779b9, sum = 0; int n;  
    for (n= 0; n < 32; n++) {  
        sum += delta;  
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);      5  
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y>>5) + k[3]);      6  
    }  
    text[0] = y; text[1] = z;  
}
```

texto  
plano y  
resultado  
2 x 32

XOR

desplazamiento



# cifrado asimétrico

introducción

amenazas

**criptografía**

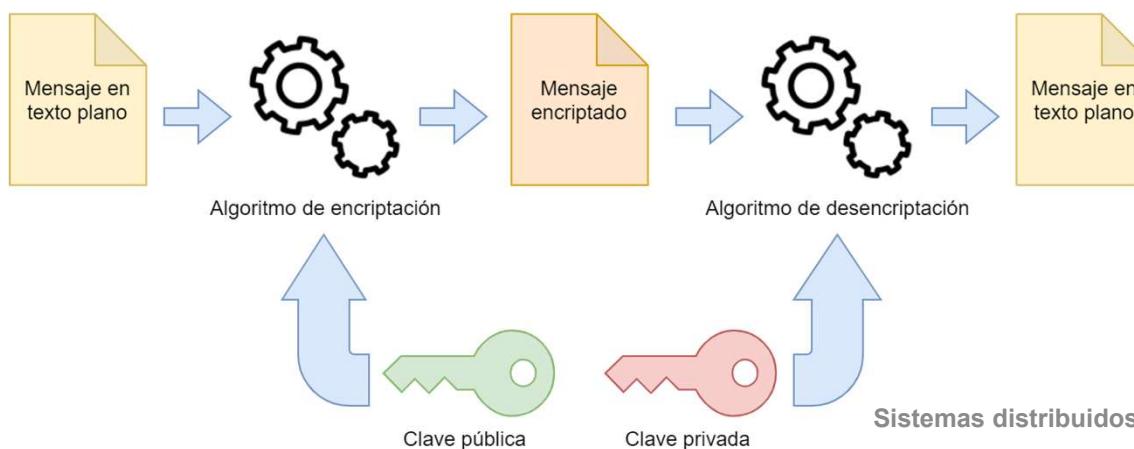
escenarios

autenticación y autorización

buenas prácticas

## Características

- Se genera un par de claves: **pública** (para cifrar) y **privada** (para descifrar)
- La clave pública se puede difundir entre quienes quieran enviar un mensaje cifrado al poseedor de la clave privada.  
La clave privada NUNCA debe ser revelada
- Es muy complejo computacionalmente y lento (comparados con el cifrado simétrico)
- La longitud de las claves son más largas que en el cifrado simétrico
- Idóneo para la distribución de claves y los sistemas de autenticación y **cifrar pequeñas cantidades** de información



# cifrado asimétrico

introducción

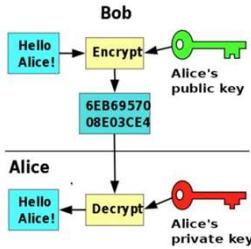
amenazas

criptografía

escenarios

autenticación y autorización

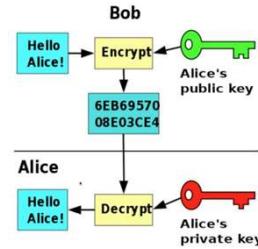
buenas prácticas



## Principales algoritmos

- **RSA** (Rivest, Shamir y Adleman, 1978): El primer algoritmo práctico y el que se utiliza con mayor frecuencia. Su seguridad radica en el problema de factorización de números primos y se basa en el producto de dos números primos enormes (del orden de  $2^{100}$ ). Clave entre 512 y 3072 bits.
- **ECC** (Elliptic Curve Cryptography, V. Miller y N. Koblitz, 1985) Se basa en las matemáticas de las curvas elípticas. Utilizado para intercambio de claves y firma digital. Es más rápida y utiliza claves más cortas (256 bits) que RSA. Una clave de 256 bits ofrece la misma seguridad que una clave de 3072 bits. Escala mejor que RSA.
- **DSA** (Digital Signature Algorithm, David W. Kravitz, 1991): Estándar del Gobierno Federal de los EEUU para firmas digitales. Sirve para firmar pero no para autenticar. La desventaja frente al RSA es que requiere mucho más tiempo de cómputo.

Estándares criptográficos (NSA Suite B): [http://www.nsa.gov/ia/programs/suiteb\\_cryptography/index.shtml](http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml)



# cifrado asimétrico: RSA

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

Para encontrar el par de claves  $K_{Pub}$  y  $K_{Priv}$ :

1. Elegir dos primos muy grandes,  $P$  y  $Q$  (p. ej. de 1024 bits cada uno) y calcular:

$$N = P \times Q$$

$$\Phi(N) = (P-1) \times (Q-1)$$

2. Para  $K_{Pub}$  elegir un número primo respecto a  $E$  ( $K_{Pub}$  no tiene factores comunes con  $\Phi(N)$ ).

3. Para encontrar  $K_{Priv}$  se resuelve la ecuación:  $K_{Priv} * K_{Pub} = 1 \text{ mod } E$

*KPriv es el elemento más pequeño divisible por KPub en la serie  $\Phi(N) + 1, 2\Phi(N) + 1, \dots$*

1. Two prime numbers  $P=7, Q=17$

$$2. n = P * Q = 17 * 7 = 119 \quad n = 119$$

$$3. \Phi(n) = (P-1) * (Q-1) = (17-1) * (7-1) = 16 * 6 = 96 \quad \Phi(n) = 96$$

4. Public key  $E = 5$ .

$$E = 5$$

$$5. \text{Calculate } d \quad d = ((\Phi(n) * i) + 1) / e \quad d = 77$$

$$d = ((96*1)+1) / 5 = 19.4$$

$$d = ((96*2)+1) / 5 = 38.6$$

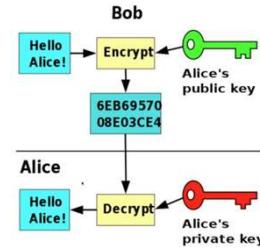
$$d = ((96*3)+1) / 5 = 57.8$$

$$d = ((96*4)+1) / 5 = 77 \quad (\text{Stop finding } d \text{ because getting integer value})$$

6. Public key =  $\{e, n\} = \{5, 119\}$ , private key =  $\{d, n\} = \{77, 119\}$ .

En RSA, por defecto se usa:  
 $\text{Power}(2,16) + 1 = 65537$   
17Bits

# cifrado asimétrico: RSA



Para encriptar según RSA, el texto plano  $M$  se divide en bloques de  $k$  bits donde  $2^k < N$   
 $k = 6$ , entonces  $2^6 = 64 (< N = 119)$  [en RSA original bloques de  $k = 2048$  bits]

La función de encriptación de un bloque de  $M$  es:  $E(K_{Pub}, N, M) = M^{K_{Pub}} \text{ mod } N$   
La función de desencriptación del bloque cifrado  $C$  es:  $D(K_{Priv}, N, C) = C^{K_{Priv}} \text{ mod } N$

6. Public key = {e, n} = {5, 119}, private key = {d, n} = {77, 119}.

7. Plain text PT = 6, CT =  $PT^E \text{ mod } n = 6^5 \text{ mod } 119 = 41$ . **Cipher Text = 41**

8. Cipher text CT = 41, PT =  $CT^d \text{ mod } n = 41^{77} \text{ mod } 119 = 6$ . **Plain Text = 6**

Rivest, Shamir and Adelman probaron que  $E'$  y  $D'$  son inversas mutuas:

$$E'(D'(x)) = D'(E'(x)) = x \quad 0 \leq P \leq N$$

**Lo que se encripta con la  $K_{Pub}$  solo se puede desencriptar con la  $K_{Priv}$**

**Lo que se encripta con la  $K_{Priv}$  solo se puede desencriptar con la  $K_{pub}$**  ←

**[RAZÓN POR LA QUE NO SE DEBE CIFRAR CON LA  $K_{PRIV}$  PARA OCULTAR INFORMACIÓN]**

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

# cifrado asimétrico: RSA

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Exercise - 1

**Question:** P and Q are two prime numbers. P=7, and Q=17. Take public key E=5. If plain text value is 6, then what will be cipher text value according to RSA algorithm? Again calculate plain text value from cipher text.

**Solution:**

1. Two prime numbers P=7, Q=17
2.  $n = P * Q = 17 * 7 = 119$  **n = 119**
3.  $\Phi(n) = (P-1) * (Q-1) = (17-1) * (7-1) = 16 * 6 = 96$   **$\Phi(n) = 96$**
4. Public key E = 5. **E = 5**
5. Calculate d = 77.  $d = ((\Phi(n) * i) + 1) / e$  **d = 77**  
 $d = ((96*1)+1) / 5 = 19.4$   
 $d = ((96*2)+1) / 5 = 38.6$   
 $d = ((96*3)+1) / 5 = 57.8$   
 $d = ((96*4)+1) / 5 = 77$  **(Stop finding d because getting integer value)**
6. Public key = {e, n} = {5, 119}, private key = {d, n} = {77, 119}.
7. Plain text PT = 6, CT =  $PT^E \text{ mod } n = 6^5 \text{ mod } 119 = 41$ . **Cipher Text = 41**
8. Cipher text CT = 41, PT =  $CT^d \text{ mod } n = 41^{77} \text{ mod } 119 = 6$ . **Plain Text = 6**

# cifrado asimétrico: RSA

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Exercise - 1

**Question:** P and Q are two prime numbers. P=7, and Q=17. Take public key E=5. If plain text value is 6, then what will be cipher text value according to RSA algorithm? Again calculate plain text value from cipher text.

**Solution:**

1. Two prime numbers P=7, Q=17

2.  $n = P * Q = 17 * 7 = 119$  **n = 119**

3.  $\Phi(n) = (P-1) * (Q-1) = (17-1) * (7-1) = 16 * 6 = 96$   **$\Phi(n) = 96$**

4. Public key E = 5. **E = 5**

5. Calculate d = 77.  $d = ((\Phi(n) * i) + 1) / e$  **d = 77**

$$d = ((96*1)+1) / 5 = 19.4$$

$$d = ((96*2)+1) / 5 = 38.6$$

$$d = ((96*3)+1) / 5 = 57.8$$

$d = ((96*4)+1) / 5 = 77$  (Stop finding d because getting integer value)

6. Public key = {e, n} = {5, 119}, private key = {d, n} = {77, 119}.

7. Plain text PT = 6, CT =  $PT^E \text{ mod } n = 6^5 \text{ mod } 119 = 41$ . **Cipher Text = 41**

8. Cipher text CT = 41, PT =  $CT^d \text{ mod } n = 41^{77} \text{ mod } 119 = 6$ . **Plain Text = 6**

# cifrado asimétrico: Elliptic Curve Cryptography

introducción

amenazas

criptografía

escenarios

autenticación  
autorización

buenas  
prácticas

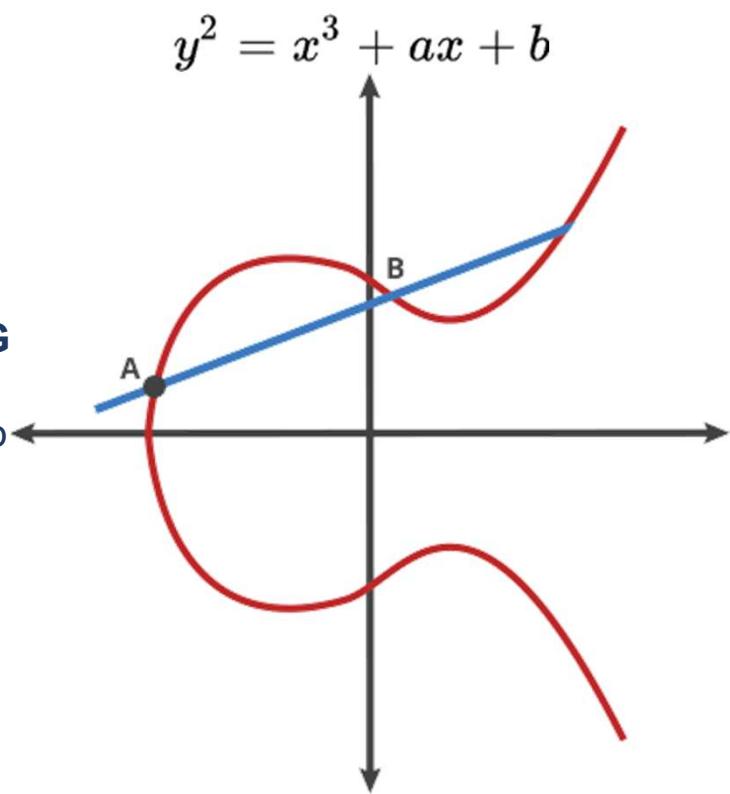


Un criptosistema de curva elíptica puede definirse eligiendo un número primo como máximo, una ecuación de la curva y **un punto público** en la curva. Una **clave privada** es un número privado, y es utilizado conjuntamente con la clave pública para el cálculo del cifrado.

El cálculo de la clave privada a partir de la clave pública en este tipo de criptosistema se denomina función de logaritmo discreto de curva elíptica, que no tiene solución conocida hasta la fecha.

Alice y Bob acuerdan primero utilizar la misma curva y algunos otros parámetros, y luego eligen un punto aleatorio **G** en la curva. Tanto **Alice como Bob** eligen números secretos ( $\alpha, \beta$ ). Alice multiplica el punto G por sí mismo  $\alpha$  veces, y Bob multiplica el punto G por sí mismo  $\beta$  veces.

Cada uno llega a nuevos puntos **A=αG**, y **B=βG** que intercambian los puntos entre sí. Partiendo de los nuevos puntos, Alice y Bob vuelven a multiplicar su nuevo punto por su propio número secreto. Bob y Alice multiplican su número secreto por el punto que reciben para generar el secreto **S**. Esto funciona porque, matemáticamente, **S=α(βG)=β(αG)**.



# cifrado híbrido

---

introducción

amenazas

**criptografía**

escenarios

autenticación y  
autorización

buenas  
prácticas

## Características

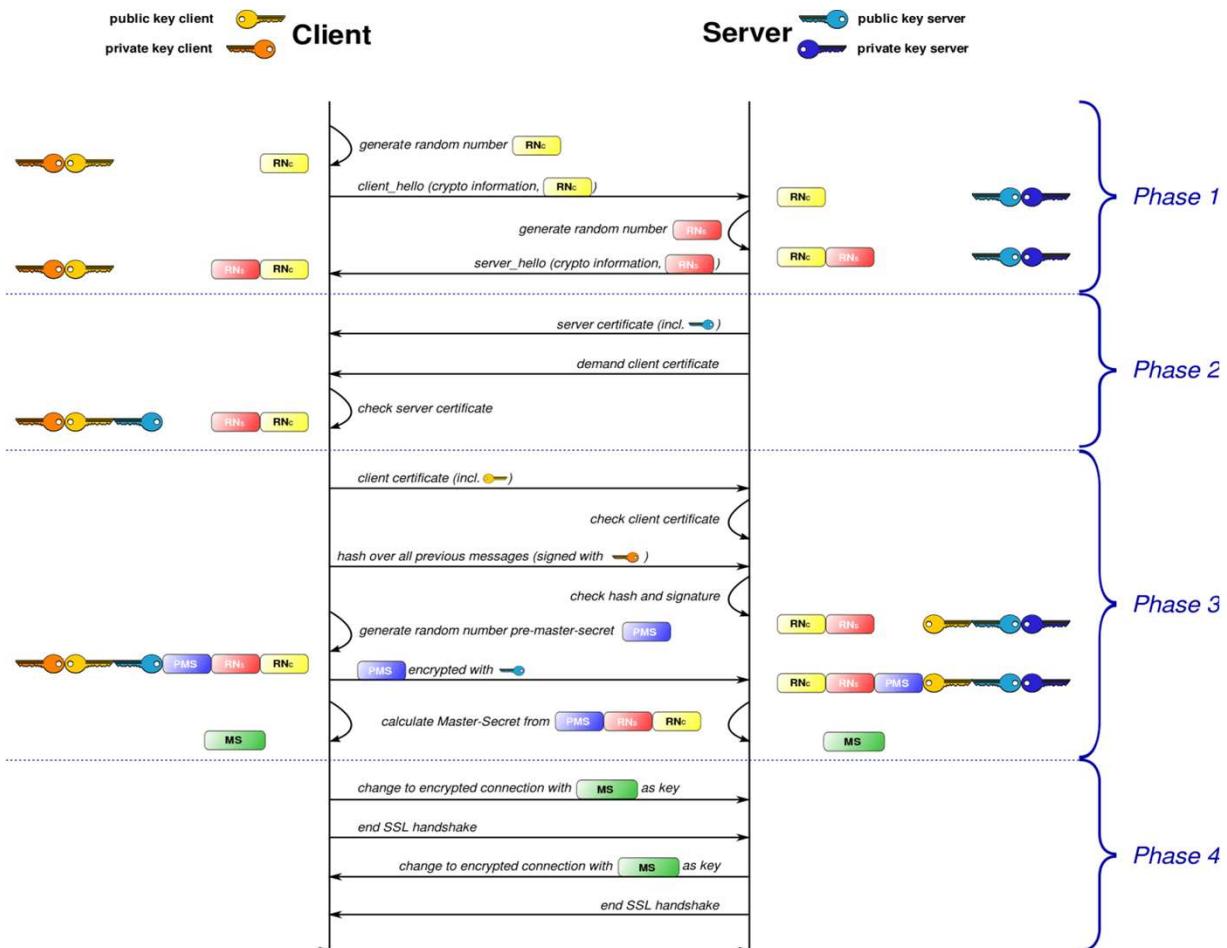
- **Combina el cifrado simétrico y asimétrico** para crear un sistema seguro que aprovecha las ventajas de ambos
- Resuelve el problema de la transmisión de la clave compartida del cifrado simétrico
- El emisor genera y cifra la clave compartida de sesión con la clave pública del receptor antes de transmitirla. El receptor obtiene la clave compartida descifrándola con su clave privada
- El resto de la sesión entre emisor y receptor se cifra con la clave compartida de ambos

## Protocolos que lo utilizan

- **SSL 3.0 (Secure Socket Layer - 1998)** y sus actualizaciones hasta actualización **TLS 1.3 (Transport Layer Security - 2018)**: Utilizan RSA y ECC como cifrado asimétrico para intercambio de claves. Utilizan AES-GCM para el resto de la comunicación mediante cifrado simétrico

# cifrado híbrido: SSL/TLS

introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas



## Fases:

Partiendo de que al menos el servidor posee su para clave pública/privada:

1. Se comprueba compatibilidad de las librerías de seguridad de cliente y servidor y se acuerdan algoritmos de cifrado
2. El servidor envía su “Clave Pública”.  
El servidor pide “Clave Pública” de cliente (opc.)  
El cliente comprueba validez “certificado/clave pública” del servidor
3. Cliente envía su “Clave Pública” (opcional, solo si tiene)  
Servidor comprueba “certificado/clave pública” del cliente (opcional)  
Cliente genera “Clave simétrica” MS, la cifra con “clave pública” de servidor y se la envía al servidor
4. Cliente y servidor pasan a modalidad “cifrado simétrico” con clave compartida MS  
La clave compartida MS se mantendrá solamente durante la sesión

# cifrado híbrido: SSL/TLS aspectos técnicos

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

- Instalación de las librerías OpenSSL (Ubuntu):  
**sudo apt-get update**  
**sudo apt-get install libssl-dev**
- Generación de los certificados (al menos para servidor):  
**openssl req -x509 -nodes 365 -newkey rsa:2048 -keyout certServ.pem -out certServ.pem**

Opciones:

- nodes**: Significa que no va a usar el algoritmo DES para cifrar la clave privada (¡ojo porque es un ejemplo! la clave privada aparece en el certificado sin cifrar)
- days 365**: El número de días de vigencia del certificado (y de las claves pública y privada), tras 1 año caducarán y ya no servirán
- rsa:2048**: Longitud de la clave generada y algoritmo de clave pública utilizado

Esta instrucción también solicita datos de dominio, sobre el propietario, email, “Common Name”, Country, etc. para que la CA (Certification Authority) pueda comprobar la autenticidad. En este caso se obvia, ya que crean certificados auto-firmados (self-signed)

# cifrado híbrido: SSL/TLS aspectos técnicos

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Interpretación del código de Servidor (extracto):

```
SSL_library_init();
portnum = Argc[1];
ctx = InitServerCTX();      /* inicializa SSL */
LoadCertificates(ctx, "mycert.pem", "mycert.pem"); /* carga de certificados */
server = OpenListener(atoi(portnum));    /* creación del socket servidor */
while (1)
{
    struct sockaddr_in addr;
    socklen_t len = sizeof(addr);
    SSL *ssl;
    int client = accept(server, (struct sockaddr*)&addr, &len); /* acepta conexiones entrantes */
    printf("Connection: %s:%d\n", inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
    ssl = SSL_new(ctx);          /* obtención de contexto de conexión seguro */
    SSL_set_fd(ssl, client);   /* conversión del socket en socket seguro SSL */
    Servlet(ssl);              /* se sirve la petición al servidor */
}
close(server);        /* cierre del socket servidor */
SSL_CTX_free(ctx);    /* liberación de contexto seguro */
```

# cifrado híbrido: SSL/TLS aspectos técnicos

## Interpretación del código de Cliente (extracto):

introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas

```
SSL_library_init();
ctx = InitCTX();
server = OpenConnection(hostname, atoi(portnum)); /* Conexión no-cifrada con servidor */
ssl = SSL_new(ctx);      /* crea un nueva conexión SSL */
SSL_set_fd(ssl, server); /* asocia el descriptor no-cifrada a la conexión cifrada*/
if ( SSL_connect(ssl) == FAIL ) /* conecta con el servidor de forma cifrada */
    ERR_print_errors_fp(stderr);
else
{
    const char *cpRequestMessage = "Usuario: %s  Contraseña: %s";
    ...
    sprintf(acClientRequest, cpRequestMessage, acUsername,acPassword); /* crea contestación */
    SSL_write(ssl,acClientRequest, strlen(acClientRequest)); /* encripta y envía el mensaje */
    bytes = SSL_read(ssl, buf, sizeof(buf)); /* obtiene la respuesta y la desencripta */
    buf[bytes] = 0;
    printf("\nRecibido: \"%s\"\n", buf);
    SSL_free(ssl);      /* libera la conexión cifrada */
}
close(server);      /* cierra el socket */
SSL_CTX_free(ctx); /* libera el contexto: algoritmos de cifrado, etc. */
```

# cifrado híbrido: SSL/TLS aspectos técnicos

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Compilación de los programas (en el repositorio UACloud):

```
gcc -Wall -o servidor servidor.c -L/usr/lib -lssl -lcrypto
```

```
gcc -Wall -o cliente cliente.c -L/usr/lib -lssl -lcrypto
```

## Ejemplo de ejecución de los programas:

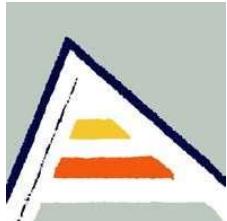
```
sudo ./servidor 8090           (se requiere ejecutarlo como root en un terminal o máquina)
```

```
./cliente 127.0.0.1 8090       (la dirección del servidor es localhost en otro terminal o máquina)
```

## Resultado:

Solamente **Usuario**: SD y **Contraseña**: 12345678 confirman el acceso por parte del servidor

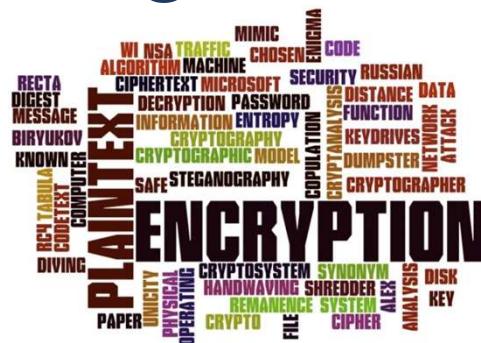
Si se analizan, con **WireShark**, los paquetes intercambiados entre Cliente y Servidor de las versiones seguras, se podrá verificar que las comunicaciones están cifradas, a diferencia de las versiones que usan los sockets no seguros



# Grado en Ingeniería Informática

# Sistemas distribuidos

# Seguridad



Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# funciones HASH

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

Son algoritmos matemáticos unidireccionales utilizados para asignar datos de cualquier tamaño a una cadena de bits de un tamaño fijo. También llamadas “de resumen seguro” o “digest”

## Propiedades

- **Velocidad:** Deben ser rápidos en calcular el valor de hash para una entrada dada
- **Determinismo:** Independientemente del tamaño de la información de entrada, la función de HASH siempre devuelve la misma salida para una entrada dada y del mismo tamaño
- **Resistencia a pre-imagen:** A partir del valor de hash de un mensaje es imposible obtener el mensaje original
- **Resistencia a la colisión:** Muy difícil encontrar a priori dos mensajes distintos que coincidan en el valor de su hash y más difícil encontrar una variante del mensaje original que coincida en el mismo hash (*ataque del cumpleaños*)
- **Efecto avalancha:** Un mínimo cambio en la entrada provoca un cambio masivo en la salida

# funciones HASH

introducción

amenazas

criptografía

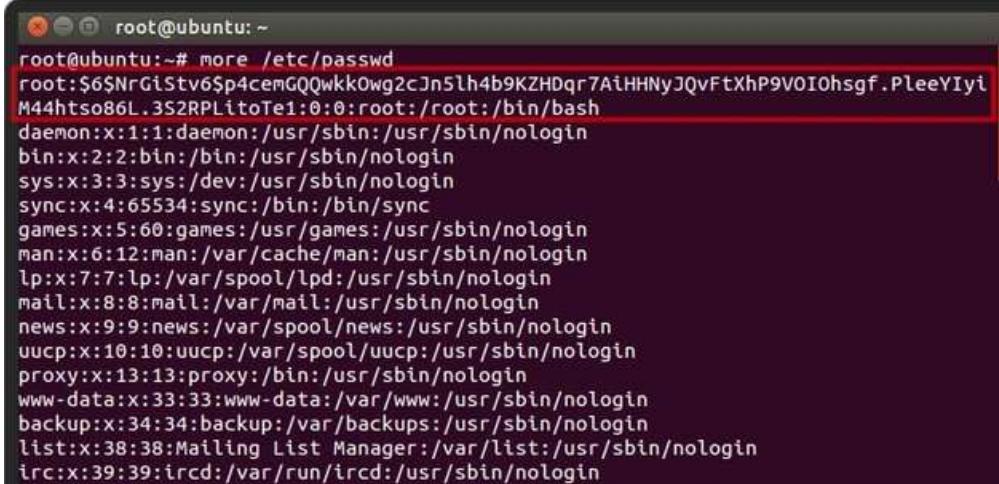
escenarios

autenticación y  
autorización

buenas  
prácticas

## Aplicaciones

- **Almacenamiento de contraseñas**



```
root@ubuntu:~# more /etc/passwd
root:$6$NrGiStv6$p4cemGQQwkk0wg2cJn5lh4b9KZHDqr7AiHHNyJQvFtXhP9VOIohsgf.PleeYIyi
M44htso86L.3S2RPLitoTe1:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
```

# funciones HASH

---

introducción

amenazas

**criptografía**

escenarios

autenticación y  
autorización

buenas  
prácticas

## Aplicaciones

- Almacenamiento de contraseñas
- Firma digital
- Protección de la propiedad intelectual

Documento electrónico generado el 07/06/2022 10:19:54. Página 1/2

Código de verificación (CSV): **JRL+09Y02OQ34MLWHWLC** Puede validar el documento en <https://seuelectronica.ua.es/>

# funciones HASH

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Aplicaciones

- Almacenamiento de contraseñas
- Firma digital
- Protección de la propiedad intelectual
- Peritajes informáticos
- Detección de Malware (p. ej. VirusTotal)



SHA256: 5e72ced0c5963fc711a5d26bedcc645802e025b6ffbc8d013d24b73d50ceb6  
File name: d772594ca48ed55d5d527caa7f1b08792f6e7e08  
Detection ratio: 23 / 54  
Analysis date: 2017-04-20 02:06:47 UTC (1 month ago)



The screenshot shows the WinMD5Free application window. At the top, it displays the file path: C:\Users\vlad\Desktop\vbs\Jessica\_Gostosa\_Caiu\_Na\_Net.vbs. Below this, there's a text input field labeled "File Name and Size:" containing the same file path. Next to it is a red-bordered box containing the text "Current file MD5 checksum value: f570beff40d696527fb63b91b473458a". Further down, there's a "Verify" button. The main part of the window is a table showing analysis results from various antivirus engines:

Antivirus	Result	Date
AegisLab	Troj_Downloader_Scriptic	20170419
AhnLab-V3	JS/Obfus	20170420
ALYac	VB:Trojan.Valnya.379	20170420
Avast	VBS:Agent-BRT [Tn]	20170420
Avira (no cloud)	VBS/Dldr.Agent.6394	20170420
Baidu	VBS:Trojan-Downloader.Agent.qr	20170420
CAT-QuickHeal	JS_Nemucod.APZ	20170420
Cyren	Trojan.LQEW-3	20170420
DrWeb	VBS_DownLoader.786	20170420
Emsisoft	Trojan-Downloader.Agent.(A)	20170420
ESET-NOD32	VBS/TrojanDownloader.Agent.OQA	20170419
Fortinet	WM/Moat.43484247ltr	20170420

# funciones HASH

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

## Aplicaciones

- Almacenamiento de contraseñas
- Firma digital
- Protección de la propiedad intelectual
- Peritajes informáticos
- Detección de Malware (p. ej. VirusTotal)
- Comprobación de la integridad de descargas en las redes de comunicaciones

```
a435f6f393dda581172490eda9f683c32e495158a780b5a1de422ee77d98e909 *ubuntu-22.04.3-desktop-amd64.iso
a4acfda10b18da50e2ec50ccaf860d7f20b389df8765611142305c0e911d16fd *ubuntu-22.04.3-live-server-amd64.iso
```

 Parent Directory	-	-	-
 SHA256SUMS	2023-08-10 18:33	202	
 SHA256SUMS.gpg	2023-08-10 18:33	833	
 ubuntu-22.04.3-desktop-amd64.iso	2023-08-08 01:19	4.7G	Desktop image for 64-bit PC (AMD64) computers (standard download)
 ubuntu-22.04.3-desktop-amd64.iso.torrent	2023-08-10 18:30	376K	Desktop image for 64-bit PC (AMD64) computers (BitTorrent download)
 ubuntu-22.04.3-desktop-amd64.iso.zsync	2023-08-10 18:30	11M	Desktop image for 64-bit PC (AMD64) computers (zsync metafile)
 ubuntu-22.04.3-desktop-amd64.list	2023-08-08 01:19	23K	Desktop image for 64-bit PC (AMD64) computers (file listing)
 ubuntu-22.04.3-desktop-amd64.manifest	2023-08-07 23:05	60K	Desktop image for 64-bit PC (AMD64) computers (contents of live filesystem)

# funciones HASH

introducción

amenazas

criptografía

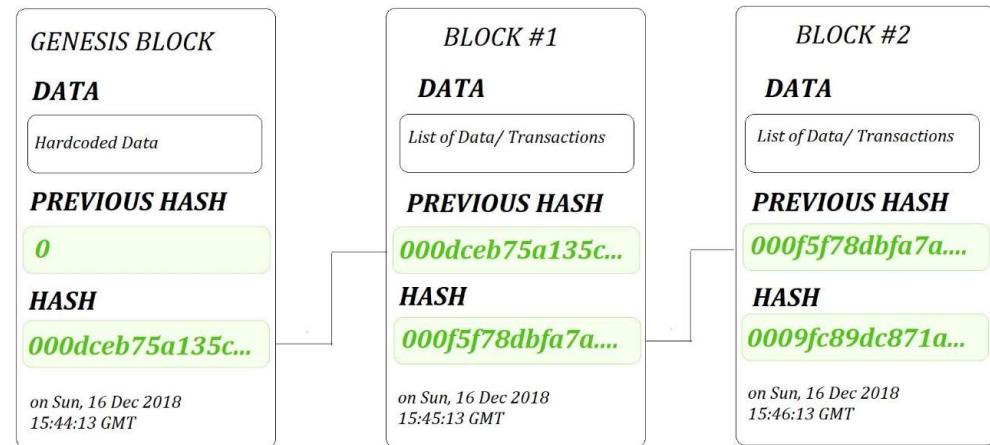
escenarios

autenticación y  
autorización

buenas  
prácticas

## Aplicaciones

- Almacenamiento de contraseñas
- Firma digital
- Protección de la propiedad intelectual
- Peritajes informáticos
- Detección de Malware (p. ej. VirusTotal)
- Comprobación de la integridad de descargas en las redes de comunicaciones
- Blockchain



# funciones HASH

introducción

amenazas

criptografía

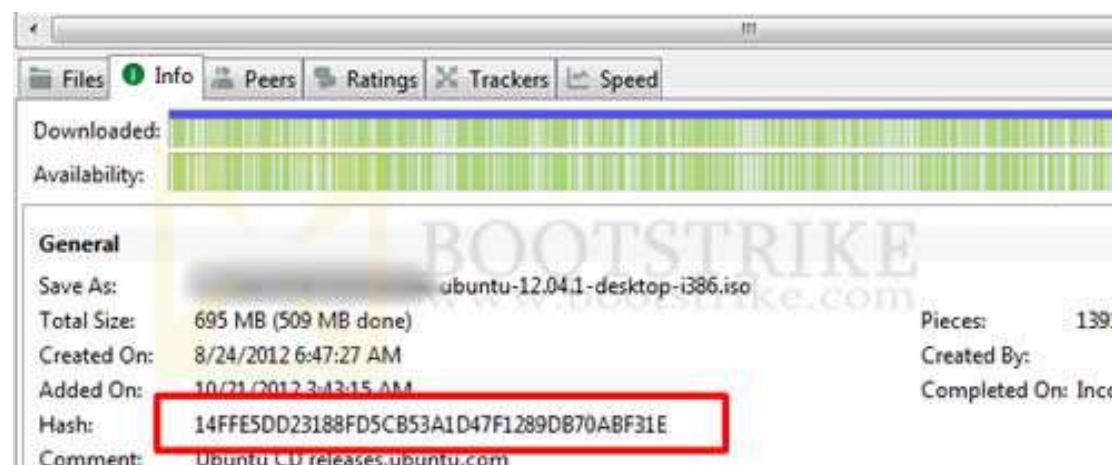
escenarios

autenticación y  
autorización

buenas  
prácticas

## Aplicaciones

- Almacenamiento de contraseñas
- Firma digital
- Protección de la propiedad intelectual
- Peritajes informáticos
- Detección de Malware (p. ej. VirusTotal)
- Comprobación de la integridad de descargas en las redes de comunicaciones
- Blockchain
- Transferencia de archivos mediante protocolos P2P (Torrent, Emule, ...)



# funciones HASH

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida

## Principales algoritmos

- **MD5:** Primera familia de algoritmos hash (1992). Salida de 128 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA1:** Salida de 160 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA256:** Familia SHA-2. Salida de 256 bits.  
Estándar actual => No se han encontrado **colisiones**
- **SHA512:** Familia SHA-2. Salida de 512.  
Futuro del estándar => No se han encontrado **colisiones**
- **SHA-3:** Nueva familia de hash diseñada en 2015.  
Más segura y más lenta que sus predecesores sin ayuda de hardware.  
Salida entre 224 bits y 512 bits.

# funciones HASH

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

**La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida**

## Principales algoritmos

- **MD5:** Primera familia de algoritmos hash (1992). Salida de 128 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA1:** Salida de 160 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA256:** Familia SHA-2. Salida de 256 bits.  
Estándar actual => No se han encontrado **colisiones**
- **SHA512:** Familia SHA-2. Salida de 512.  
Futuro del estándar => No se han encontrado **colisiones**
- **SHA-3:** Nueva familia de hash diseñada en 2015.  
Más segura y más lenta que sus predecesores sin ayuda de hardware.  
Salida entre 224 bits y 512 bits.

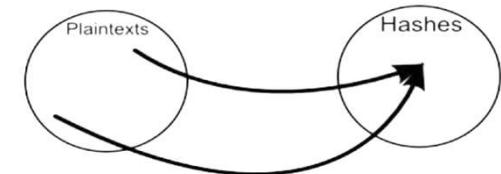


Hash Collision



Birthday Attack

# funciones HASH: ataque de cumpleaños



introducción

amenazas

criptografía

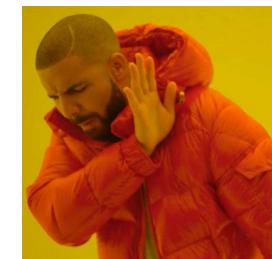
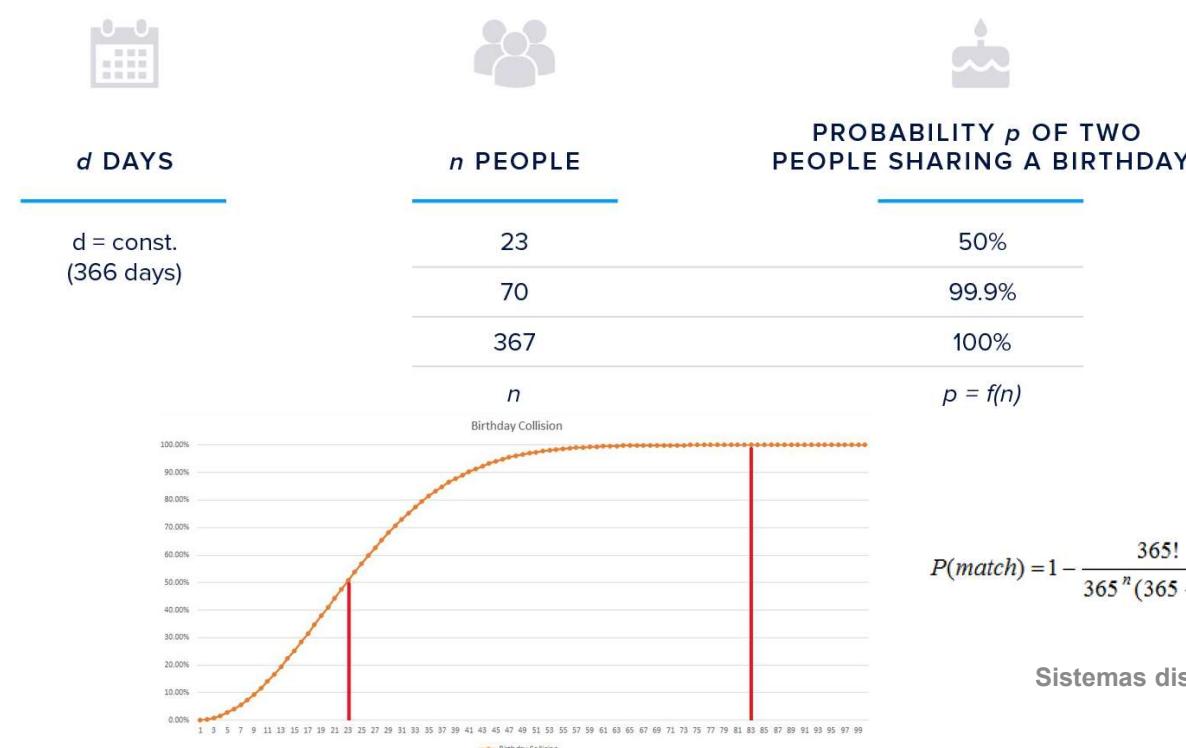
escenarios

autenticación y autorización

buenas prácticas

La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida

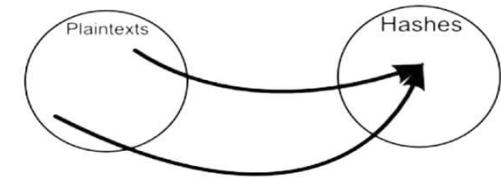
## BIRTHDAY PROBLEM



Hash Collision



Birthday Attack



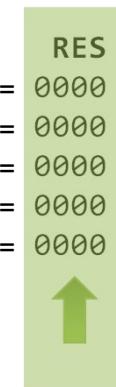
# funciones HASH: ataque de cumpleaños

introducción  
amenazas  
criptografía  
escenarios  
autenticación y autorización  
buenas prácticas



$\text{XOR}(A, B, C, D)$

A	B	C	D	RES
1011	0101	1110	0000	= 0000
1111	0001	1110	0000	= 0000
1111	0101	1010	0000	= 0000
1111	0101	1110	0100	= 0000
1111	0001	1010	0100	= 0000



Hash Collision



Birthday Attack

# funciones HASH

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida

## Principales algoritmos

- **MD5:** Primera familia de algoritmos hash (1992). Salida de 128 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA1:** Salida de 160 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.
- **SHA256:** Familia SHA-2. Salida de 256 bits.  
Estándar actual => No se han encontrado **colisiones**
- **SHA512:** Familia SHA-2. Salida de 512.  
Futuro del estándar => No se han encontrado **colisiones**
- **SHA-3:** Nueva familia de hash diseñada en 2015.  
Más segura y más lenta que sus predecesores sin ayuda de hardware.  
Salida entre 224 bits y 512 bits.



# funciones HASH: MD5

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida

## Principales algoritmos

- **MD5:** Primera familia de algoritmos hash (1992). Salida de 128 bits.  
Se han encontrado **colisiones** => Actualmente en desuso.

La entrada tiene que ser divisible entre 512 y la salida es 128 bits

$$a = b + ((a + \text{Proceso P } (b, c, d) + M[i] + T[k]) \ll\ll s)$$

Donde,

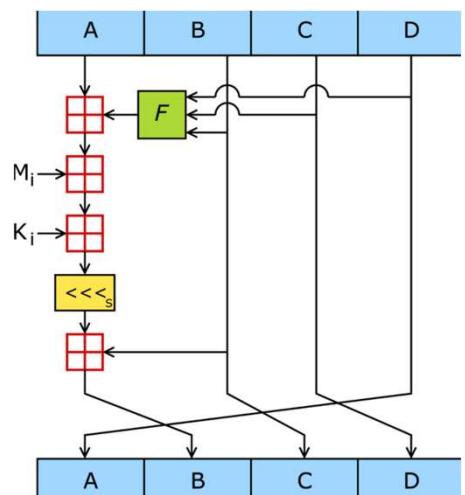
a, b, c, d = Encadenamiento de variables

Proceso F = Una operación no lineal

$M[i] = M[qx 16 + i]$ , que es la iésima palabra de 32 bits en el qésimo bloque de 512 bits de el mensaje

$t[k]$  = Una constante

$\ll\ll s$  = Desplazamiento circular a la izquierda en s bits



# funciones HASH: SHA1

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

La propiedad anticolisión del hash está directamente relacionada con el tamaño de salida

## Principales algoritmos

- **SHA1:** Salida de 160 bits.

Se han encontrado **colisiones** => Actualmente en desuso.

$$abcde = (e + \text{Proceso P} + s^5(a) + w[t] + K[t]), a, s^{30}, (b), c, d$$

Donde,

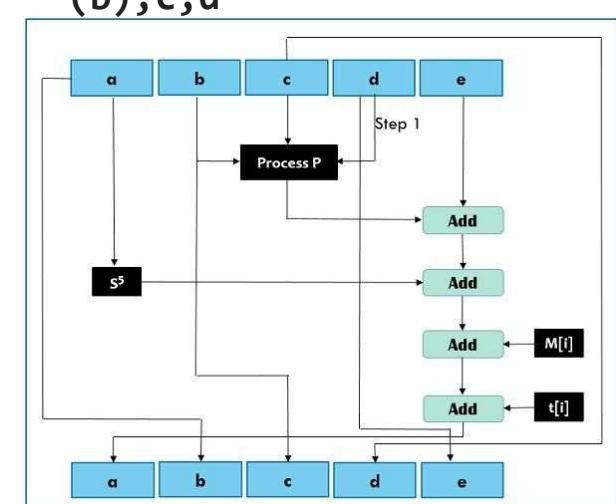
abcde = El registro construido con cinco variables a, b, c, d y e

Proceso P = La operación lógica,

$S^t$  = Desplazamiento circular a la izquierda del subbloque en t bits.

W[t] = Un subbloque derivado del actual subbloque

K[t] = Una de las cinco constantes aditivas



# funciones HASH: SHA-X

introducción  
amenazas  
**criptografía**  
escenarios  
autenticación y autorización  
buenas prácticas

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Bitwise operations	Collisions found	Example Performance (MiB/s) <sup>[8]</sup>
MD5 (as reference)		128	128	512	$2^{64} - 1$	32	64	and,or,xor,rot	Yes	335
SHA-0		160	160	512	$2^{64} - 1$	32	80	and,or,xor,rot	Yes	-
SHA-1		160	160	512	$2^{64} - 1$	32	80	and,or,xor,rot	Theoretical attack ( $2^{61}$ ) <sup>[9]</sup>	192
SHA-2	SHA-224 SHA-256	224 256	256	512	$2^{64} - 1$	32	64	and,or,xor,shr,rot	None	139
	SHA-384 SHA-512	384 512								
	SHA-512/224 SHA-512/256	512 224 256	512	1024	$2^{128} - 1$	64	80	and,or,xor,shr,rot	None	154
	SHA-3	224/256/384/512	1600 (5×5 array of 64-bit words)	1152/1088/832/576		64	24	and,xor,not,rot	None	

# Escenarios: 1. Cifrado con clave compartida

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Funcionamiento:

1. Alice y Bob comparten una clave secreta  $K_{AB}$
2. Alice usa  $K_{AB}$  y acuerda una función de encriptación  $\text{Encrypt}(K_{AB}, M)$  para codificar y enviar una serie de mensajes  $\{M_i\}_{KAB}$
3. Bob lee los mensajes encriptados usando la función  $\text{Decrypt}(K_{AB}, M)$ .

Alice y Bob pueden funcionar mientras estén seguros que  $K_{AB}$  no es conocida

## Problemas:

- **Distribución de claves:** ¿Cómo envía Alice la clave a Bob de forma segura?
- **Edad de la comunicación:** ¿Cómo detectar que el mensaje de Alice no es una copia capturada en un ataque MitM anterior?
- **Caducidad de la clave:** Si la clave se compartía con varios usuarios ¿Cómo eliminar a uno de los usuarios de la comunicación? ¿y si se filtra la contraseña?

# Escenarios: 1. Cifrado con clave compartida

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Funcionamiento:

1. Alice y Bob comparten una clave secreta  $K_{AB}$
2. Alice usa  $K_{AB}$  y acuerda una función de encriptación  $\text{Encrypt}(K_{AB}, M)$  para codificar y enviar un mensaje  $M$ .
3. Bob lee los datos y los descifra usando su propia clave secreta  $K_{BA}$  y la función  $\text{Decrypt}(K_{BA}, C)$  para obtener el mensaje original  $M$ .

## Autentificación:

Alice y Bob

En un principio **si** la **clave** de descifrado es preservada en **secreto** entre los pares presentes en la comunicación, el desencriptado o descifrado del mensaje es suficiente para identificar de forma inequívoca al emisor.

## Problemas:

- **Distribución de la clave:** ¿Cómo garantizar que la clave se distribuye de forma segura?
- **Edad de la clave:** ¿Qué hacer si la clave es capturada por un atacante?
- **Caducidad de la clave:** Si la clave se compartía con varios usuarios ¿Cómo eliminar a uno de los usuarios de la comunicación? ¿y si se filtra la contraseña?

# Escenarios: 2. autenticación y cifrado mediante servidor (clave compartida)

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

Bob es un servidor de ficheros (p.ej. servicio al que Alice quiere acceder);  
Sara es un servidor de autenticación. Sara conoce y comparte  $K_A$  con Alice y  $K_B$  con Bob

1. Alice envía un mensaje no encriptado a Sara identificándose y solicitando un **token** para acceder a Bob.
2. Sara responde a Alice con  $\{\{toKen\} K_B, K_{AB}\}_{KA}$ . que contiene una nueva clave  $K_{AB}$ .
3. Alice usa  $K_A$  para desencriptar la respuesta.
4. Alice envía a Bob el **token** recibido, su identidad y una respuesta R para acceder al fichero:  $\{token\} K_B, Alice, R$ .
5. **El token está formado por  $\{K_{AB}, Alice\} K_B$** . Bob usa  $K_B$  para desencriptarlo, chequea la identidad y usa  $K_{AB}$  para encriptar el resto de comunicación con Alice.

$$\{token\} K_B, Alice, R = \{\mathbf{K_{AB}, Alice}\} K_B , \mathbf{Alice}, R$$

# Escenarios: 2. autenticación y cifrado mediante servidor (clave compartida)

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

Bob es un servidor de ficheros (p.ej. servicio al que Alice quiere acceder);  
Sara es un servidor de autenticación. Sara conoce y comparte  $K_A$  con Alice y  $K_B$  con Bob

## Problemas:

- Distribución de claves (Punto único de fallo)
- Edad de la comunicación
- No controla la repetición y reinyección de mensajes
- No es válido para comercio electrónico, ¿por qué?

## Evolución:

- Protocolo Needham – Schroeder
- Kerberos

# Escenarios: 2. autenticación y cifrado mediante servidor (clave compartida)

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

Bob es un servidor de ficheros (p.ej. servicio al que Alice quiere acceder);  
Sara es un servidor de autenticación. Sara conoce y comparte  $K_A$  con Alice y  $K_B$  con Bob

## Problemas

- Distribución
- Edad de la clave
- No control de acceso
- No es válido para más de dos usuarios

No escala bien; útil en pequeñas empresas con un sistema de seguridad alto donde el IT conoce a los participantes y comparte las claves de forma segura e individualizada

## Evolución:

- Protocolo de autenticación
- Kerberos

# Escenarios: 3. autenticación y cifrado mediante clave pública

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

1. Bob solicita un par de claves pública/privada  $\langle K_{B\text{pub}}, K_{B\text{priv}} \rangle$  incluido en su certificado (emitido por una **autoridad de confianza**) y publica su  $K_{B\text{pub}}$
2. Alice obtiene la clave pública de Bob,  $K_{B\text{pub}}$  a partir de un certificado emitido por una **autoridad de confianza**
3. Alice crea una clave compartida  **$K_{AB}$  para esa sesión**, la encripta según  $K_{B\text{pub}}$  con un algoritmo de clave pública y envía el resultado a Bob
4. Bob usa  $K_{B\text{priv}}$  para desencriptar el mensaje y obtener  $K_{AB}$ .
5. El resto de comunicaciones de la sesión se cifran bidireccionalmente con  $K_{AB}$  (ambos tienen la clave compartida  $K_{AB}$ )

## Vulnerabilidad:

Alguien intercepte el solicitud de certificado de clave pública y le envíe el certificado del atacante (phishing) pudiendo desencriptarlo todo.

La **firma digital** (del certificado) lo impide

# Escenarios: 3. autenticación y cifrado mediante clave pública

introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

1. Bob solicita un par de claves pública/privada  $\langle K_{Bpub}, K_{Bpriv} \rangle$  incluido en su certificado (emitido por una **autoridad de confianza**) y publica su  $K_{Bpub}$
2. Alice obtiene el certificado emitido por la **autoridad de confianza**
3. Alice crea una conexión segura usando el algoritmo de cifrado que se indica en el certificado
4. Bob usa su clave privada para firmar el certificado
5. El resto de los participantes tienen la clave pública de Bob

SSL\_connection checks has failed?

```
SSL_library_init();
ctx = InitCTX();
server = OpenConnection(hostname, atoi(portnum)); /* Conexión no-cifrada con servidor */
ssl = SSL_new(ctx); /* crea un nueva conexión SSL */
SSL_set_fd(ssl, server); /* asocia el descriptor no-cifrado a la conexión cifrada*/
if ( SSL_connect(ssl) == FAIL ) /* conecta con el servidor de forma cifrada */
    ERR_print_errors_fp(stderr);
else
{
    const char *cpRequestMessage = "Usuario: %s  Contraseña: %s";
    /* ... */
}
```

## Vulnerabilidad

Alguien intercepte el solicitud de certificado de clave pública y le envíe el certificado del atacante (phishing) pudiendo desencriptarlo todo.

La **firma digital** (del certificado) lo impide

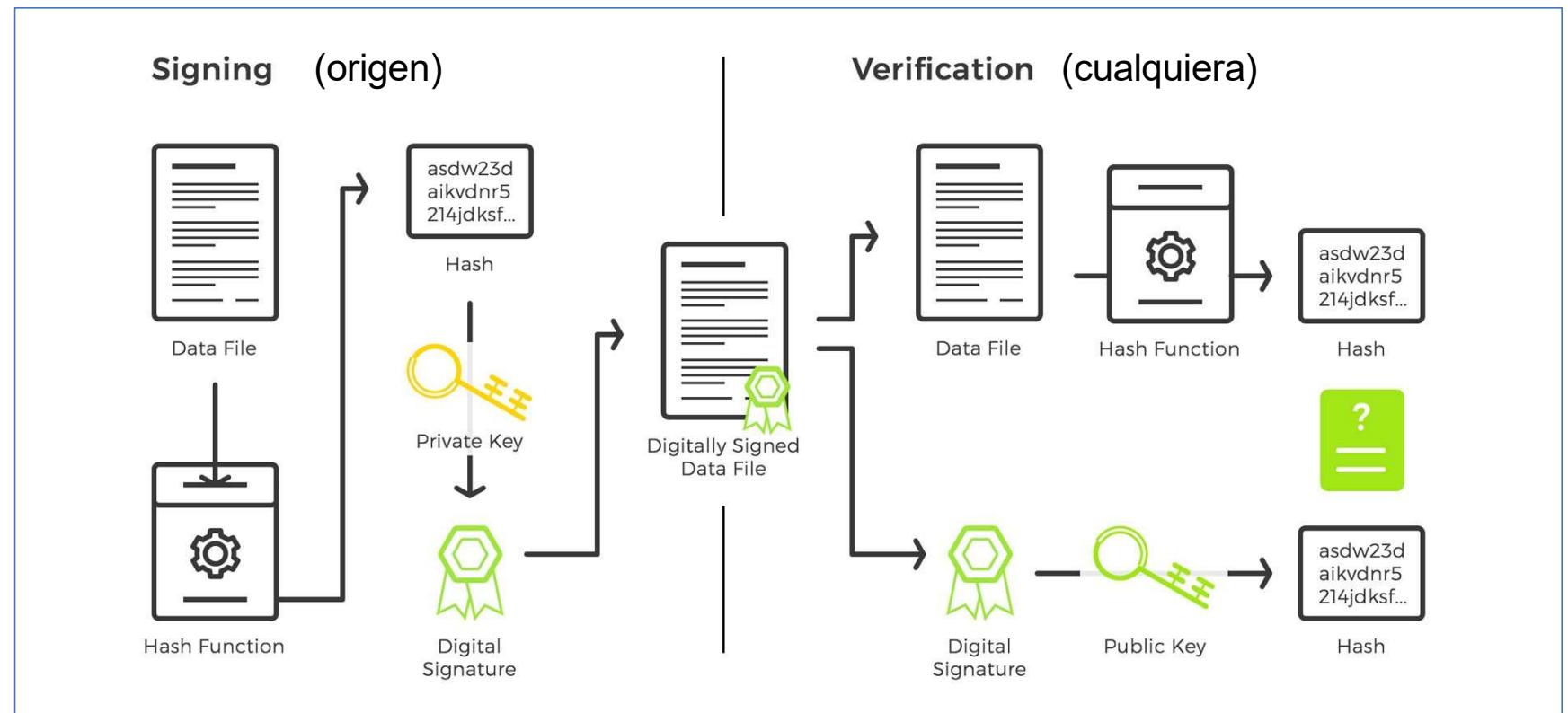
certificado emitido por una

criptado según  $K_{Bpub}$  con un

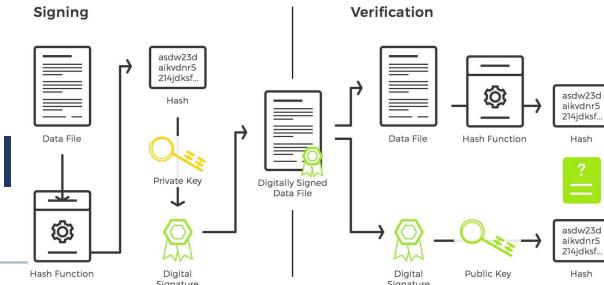
almente con  $K_{AB}$  (ambos

# Escenarios: 4. Hash y firma digital

introducción  
amenazas  
criptografía  
**escenarios**  
autenticación y autorización  
buenas prácticas



# Escenarios: 4. Hash y firma digital



introducción

amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas

Alice quiere publicar un documento **M** de forma que cualquiera pueda verificar su **integridad** y su **autoría** (firmar documento):

1. Alice calcula el HASH del documento:  $H(M)$
2. Alice encripta  $H(M)$  con su **clave privada** y lo concatena a **M** haciendo público el resultado:  $M + \{H(M)\}K_{Apriv}$
3. Bob obtiene el documento firmado, extrae **M** y computa  $H(M)$
4. Bob usa la **clave pública** de Alice para desencriptar  $\{H(M)\}K_{Apriv}$  y lo compara con el resumen calculado por él. **Si coincide, entonces el documento es válido. Si no coincide, el mensaje está manipulado o el firmante es un impostor**

## Características:

- Valida que el contenido del mensaje no ha sido modificado (gracias al HASH)
- Valida que el firmante es quien dice ser (gracias al uso de su clave pública)

# Escenarios: 4. Hash y firma digital

introducción

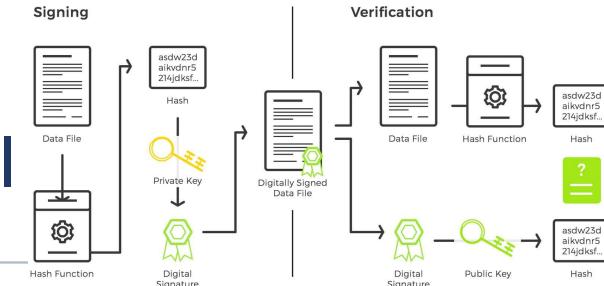
amenazas

criptografía

escenarios

autenticación y autorización

buenas prácticas



Alice quiere publicar un documento **M** de forma que cualquiera pueda verificar su **integridad** y su **autoría** (firmar documento):

1. Alice calcula el HASH del documento:  $H(M)$
2. Alice encierra el resultado en una firma digital
3. Bob obtiene la firma digital
4. Bob usa la clave pública de Alice para resumir el documento. Si el resultado coincide con el HASH que calculó Alice, el documento es válido.

Integridad del documento  
No olvido  
No repudio

I haciendo público el

)

K<sub>priv</sub> y lo compara con el  
mento es válido. Si no  
n impostor

## Características

- Valida que el documento no ha sido alterado (gracias al HASH)
- Valida que el firmante es quien dice ser (gracias al uso de su clave pública)

# certificado digital

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

Documento digital firmado por una autoridad de certificación (CA) de confianza que sirve como credencial de identificación de una entidad y para su autenticación

## Características:

- Posee un formato y estructura estándar con los datos de la entidad (según el tipo de certificado: personal, de página web, SSL/TLS, etc)
- Incluye siempre una fecha de inicio y de expiración
- Viene firmado por la CA para garantizar la integridad del mismo
- Requiere de un acuerdo sobre la construcción de las cadenas de certificados (cadena de confianza)

Forma cómoda para la transmisión de la clave pública

Lo encontramos en el DNIe, identificación de empresas, en páginas web, SSL/TLS, etc.

# certificado digital

introducción

amenazas

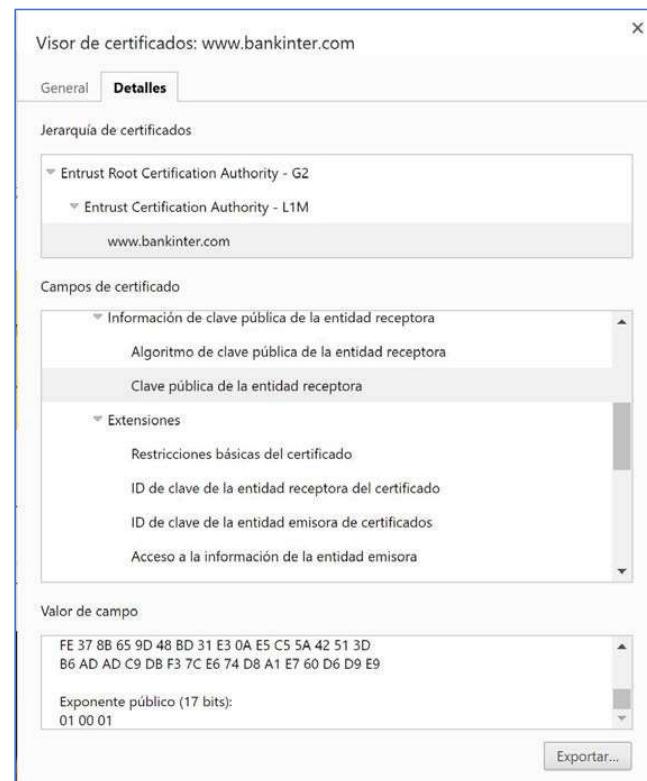
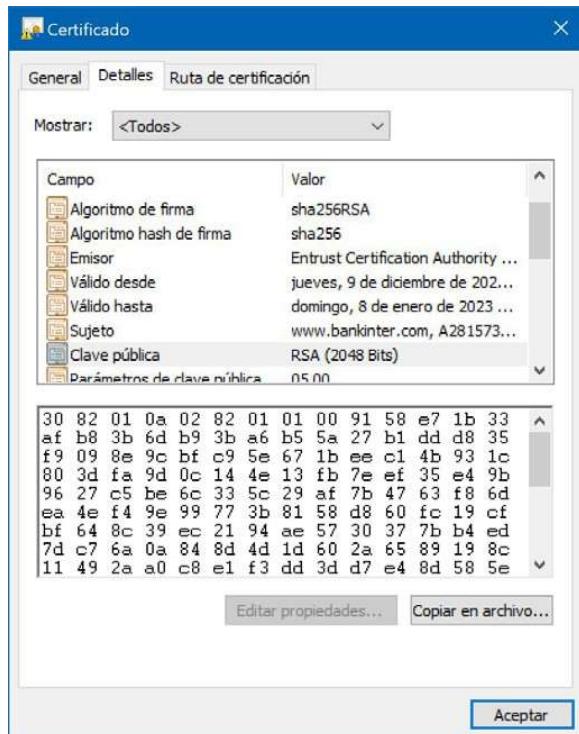
criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## Ejemplo: web de Bankinter (formato .crt)



# certificado digital

introducción

amenazas

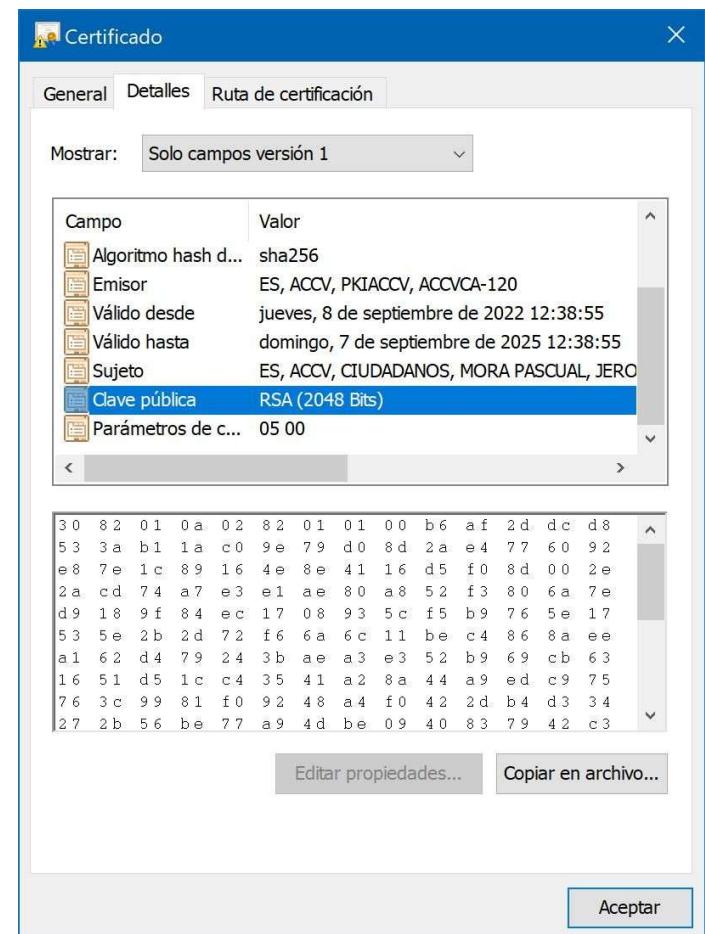
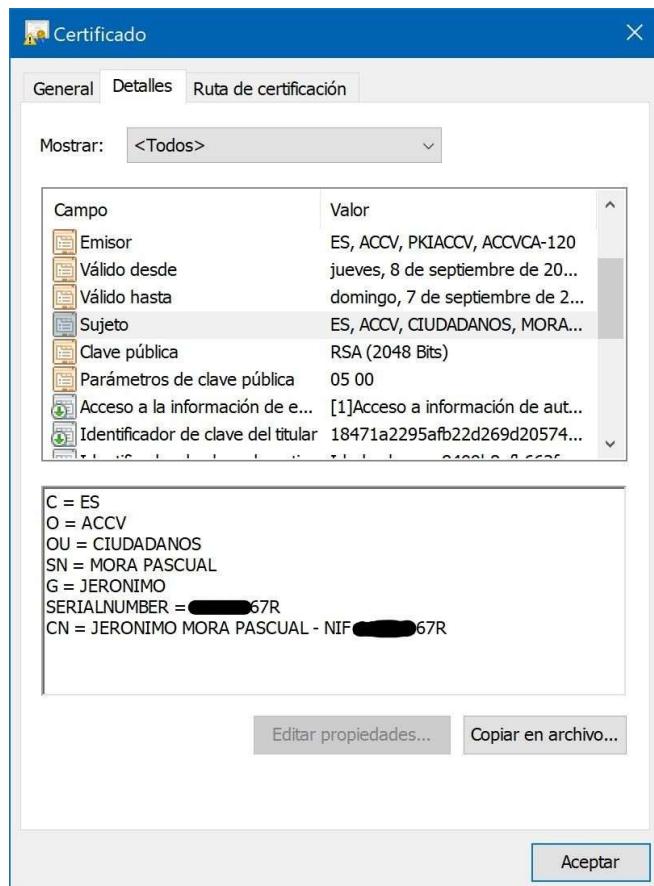
criptografía

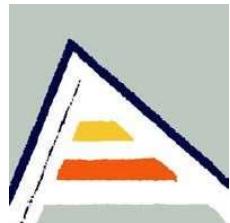
escenarios

autenticación y autorización

buenas prácticas

## Ejemplo: certificado personal (formato .p12)





Grado en Ingeniería Informática

## Sistemas distribuidos

# Seguridad



Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# AAA: introducción

---

introducción

amenazas

criptografía

escenarios

**autenticación  
y autorización**

buenas  
prácticas

## Autenticación (Authentication):

- La autenticación es el proceso por el que **se verifica la identidad** del usuario y se garantiza que es quien dice ser
- Evita accesos indeseados de usuarios ilegítimos al sistema

# autenticación

---

introducción

amenazas

criptografía

escenarios

**autenticación  
y autorización**

buenas  
prácticas

## Métodos:

- ***Login y password o certificados digitales:*** el más común de los métodos de autenticación.

# autenticación: contraseñas

introducción

amenazas

criptografía

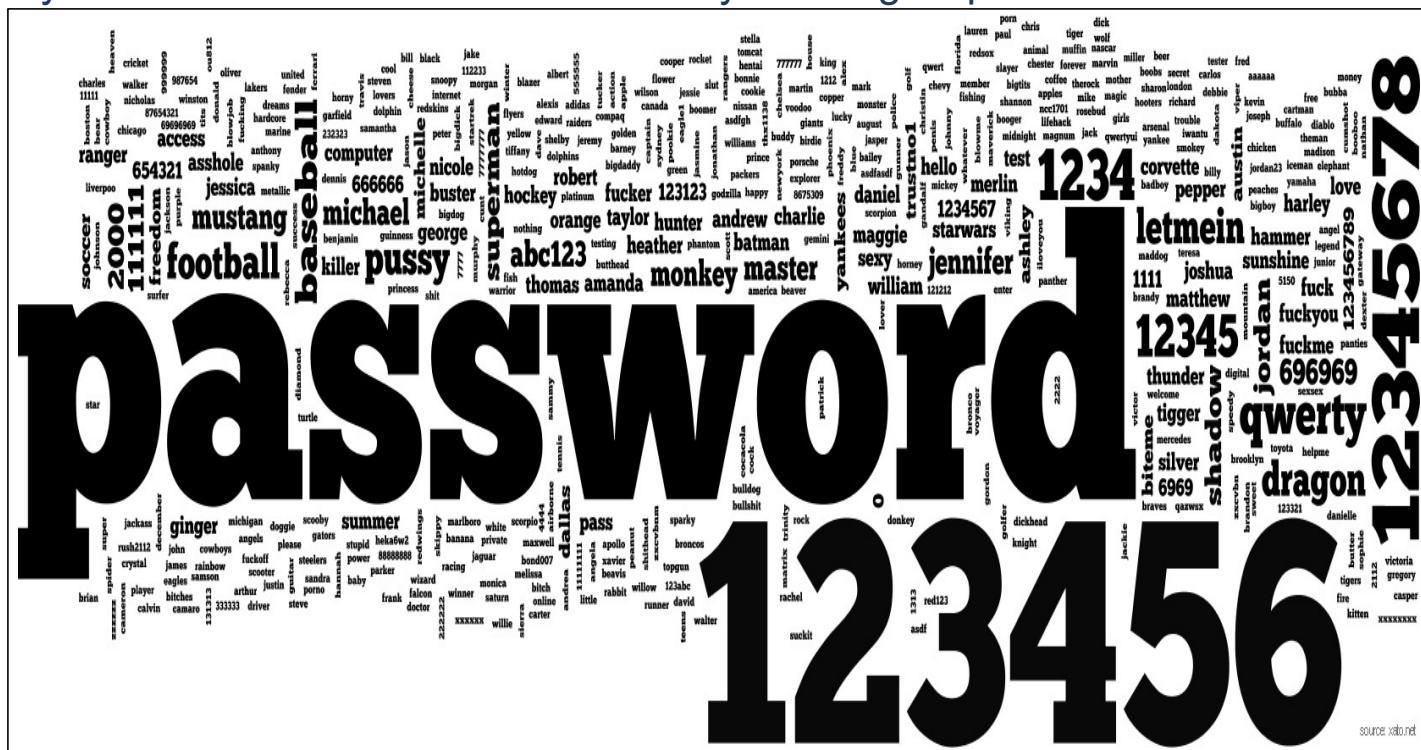
escenarios

autenticación  
y autorización

buenas  
prácticas

<https://xato.net/tagged/passwords>

Blog muy interesante sobre vulnerabilidades y estrategias para establecer contraseñas seguras



# autenticación: contraseñas

introducción

amenazas

criptografía

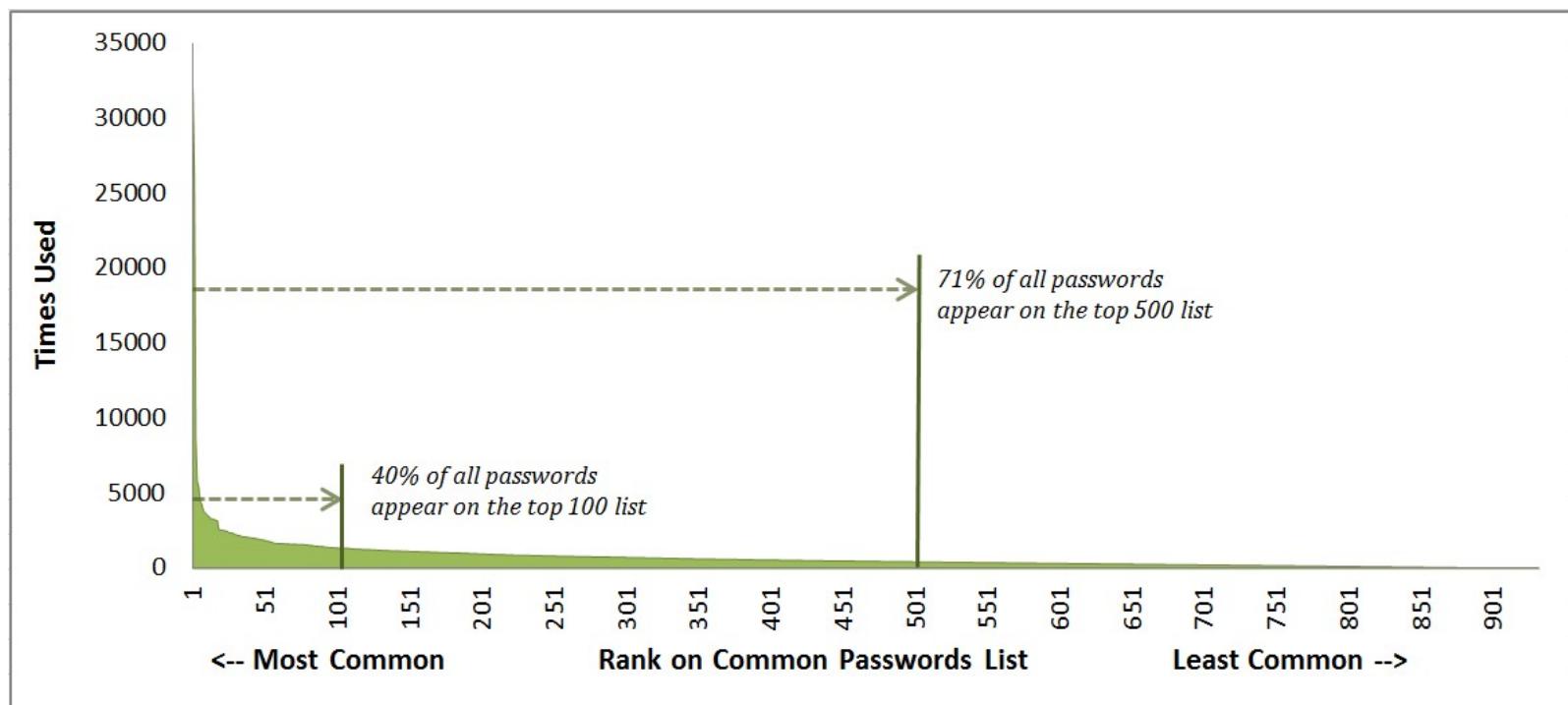
escenarios

**autenticación  
y autorización**

buenas  
prácticas

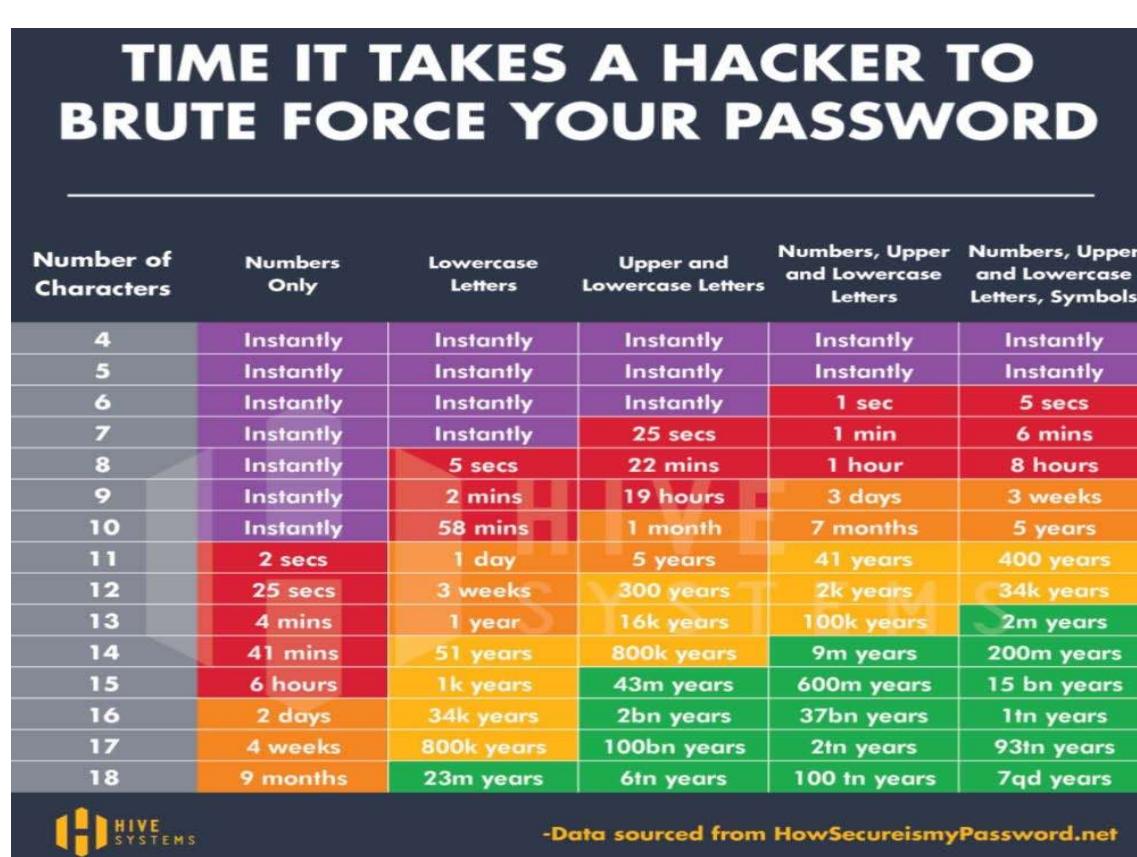
<https://xato.net/tagged/passwords>

Blog muy interesante sobre vulnerabilidades y estrategias para establecer contraseñas seguras



# autenticación: contraseñas

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas



### Política sencilla de contraseñas:

- 1.Al menos de 15 caracteres
- 2.La nueva contraseña que no se parezca a la última
- 3.No reutilizar contraseñas en otro lugar

# autenticación

---

introducción

amenazas

criptografía

escenarios

**autenticación  
y autorización**

buenas  
prácticas

## Métodos:

- **Login y password o certificados digitales:** el más común de los métodos de autenticación.
- **Pin de un solo uso:** Otorga acceso para una sola sesión o transacción y caduca en poco tiempo. Útil en 2FA (doble factor de autenticación)

# Autenticación: 2FA

---

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas



# autenticación

---

introducción

amenazas

criptografía

escenarios

**autenticación  
y autorización**

buenas  
prácticas

## Métodos:

- **Login y password o certificados digitales:** el más común de los métodos de autenticación.
- **Pin de un solo uso:** Otorga acceso para una sola sesión o transacción y caduca en poco tiempo. Útil en 2FA (doble factor de autenticación)
- **Biometría:** El usuario presenta una huella digital o un escaneo facial para obtener acceso al sistema.

# Autenticación: Biometría

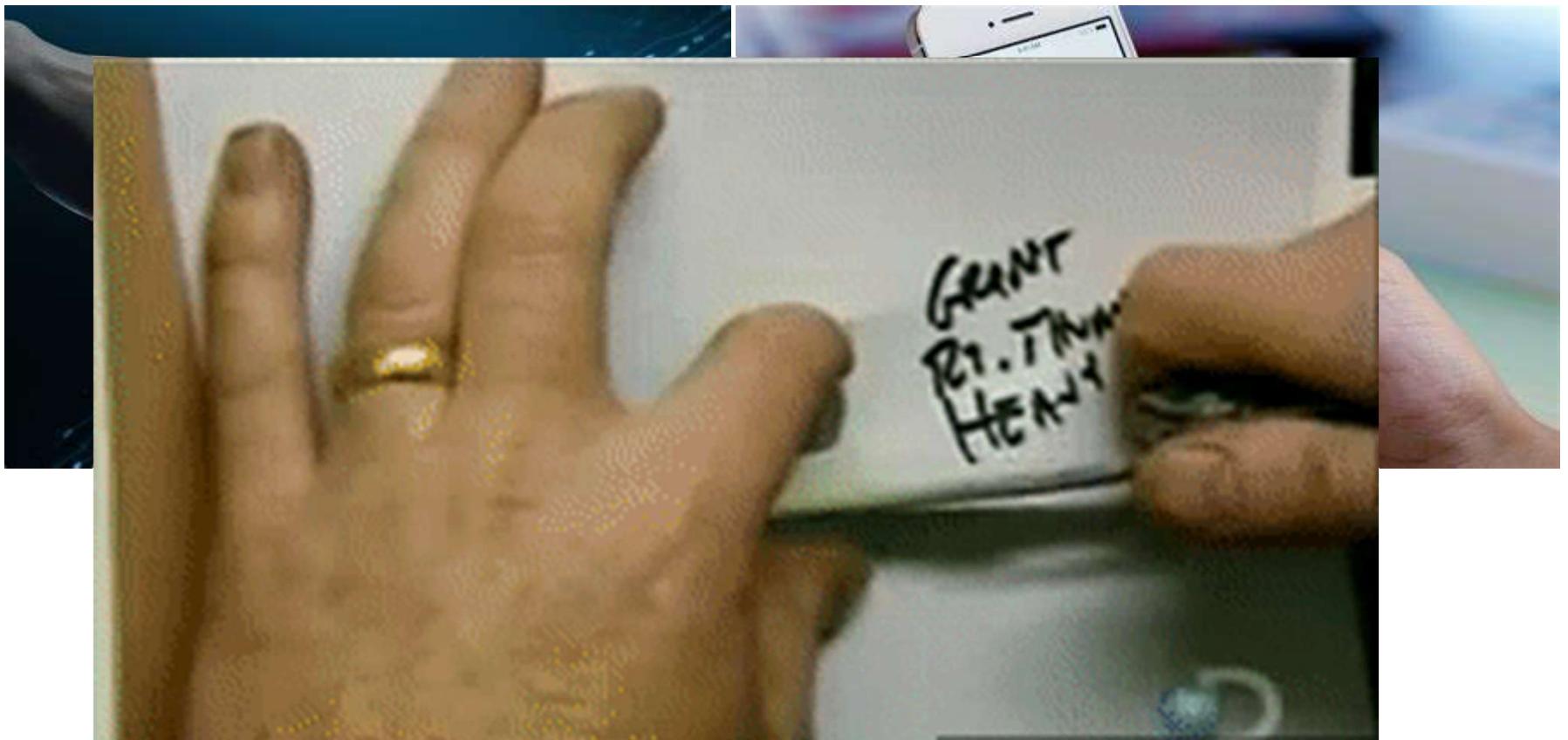
introducción  
amenazas  
criptografía  
escenarios  
**autenticación  
y autorización**  
buenas  
prácticas



Windows Hello

# Autenticación: Biometría

introducción  
amenazas  
criptografía  
escenarios  
**autenticación  
y autorización**  
buenas  
prácticas



# autenticación

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

## Métodos:

- **Login y password o certificados digitales:** el más común de los métodos de autenticación.
- **Pin de un solo uso:** Otorga acceso para una sola sesión o transacción y caduca en poco tiempo. Útil en 2FA (doble factor de autenticación)
- **Biometría:** El usuario presenta una huella digital o un escaneo facial para obtener acceso al sistema.
- **Aplicaciones externas:** Genera códigos de seguridad a través de una aplicación externa que otorga el acceso al sistema, p. ej. mediante redes sociales.

# Autenticación: Aplicaciones externas

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas



Log in to Overleaf



Log in with Google



Log in with ORCID



Log in with IEEE

► Log in with Twitter ...

## Iniciar sesión

Mantente al día de tu mundo profesional

Email o teléfono

Contraseña

mostrar

[¿Has olvidado tu contraseña?](#)

Iniciar sesión

Al hacer clic en «Continuar», aceptas las [Condiciones de uso](#), la [Política de privacidad](#) y la [Política de cookies](#).

Continue with Google

Iniciar sesión con Apple

Inicia sesión con un enlace de...

# Autenticación: Aplicaciones externas

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas



Log in to Overleaf



Log in with Google



Log in with ORCID



Log in with IEEE

► Log in with Twitter ...

## Iniciar sesión

Mantente al día de tu mundo profesional

Email o teléfono

Contraseña

mostrar

[¿Has olvidado tu contraseña?](#)

Iniciar sesión

Al hacer clic en «Continuar», aceptas las [Condiciones de uso](#), la [Política de privacidad](#) y la [Política de cookies](#).

Continue with Google

Iniciar sesión con Apple

Inicia sesión con un enlace de...

# autenticación

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

## Métodos:

- **Login y password o certificados digitales:** el más común de los métodos de autenticación.
- **Pin de un solo uso:** Otorga acceso para una sola sesión o transacción y caduca en poco tiempo. Útil en 2FA (doble factor de autenticación)
- **Biometría:** El usuario presenta una huella digital o un escaneo facial para obtener acceso al sistema.
- **Aplicaciones externas:** Genera códigos de seguridad a través de una aplicación externa que otorga el acceso al sistema, p. ej. mediante redes sociales.
- **Multiple Factor Autenticacion MFA**

# Autenticación: MFA

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas



# AAA: introducción

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas



## Autenticación (Authentication):

- La autenticación es el proceso por el que **se verifica la identidad** del usuario y se garantiza que es quien dice ser
- Evita accesos indeseados de usuarios ilegítimos al sistema

## Autorización (Authorization):

- La autorización es el proceso por el que **se valida a qué tiene acceso** el usuario autenticado
- Es el proceso por el que se controla el acceso de un usuario legítimo a determinados recursos del sistema

## Responsabilidad (Accounting):

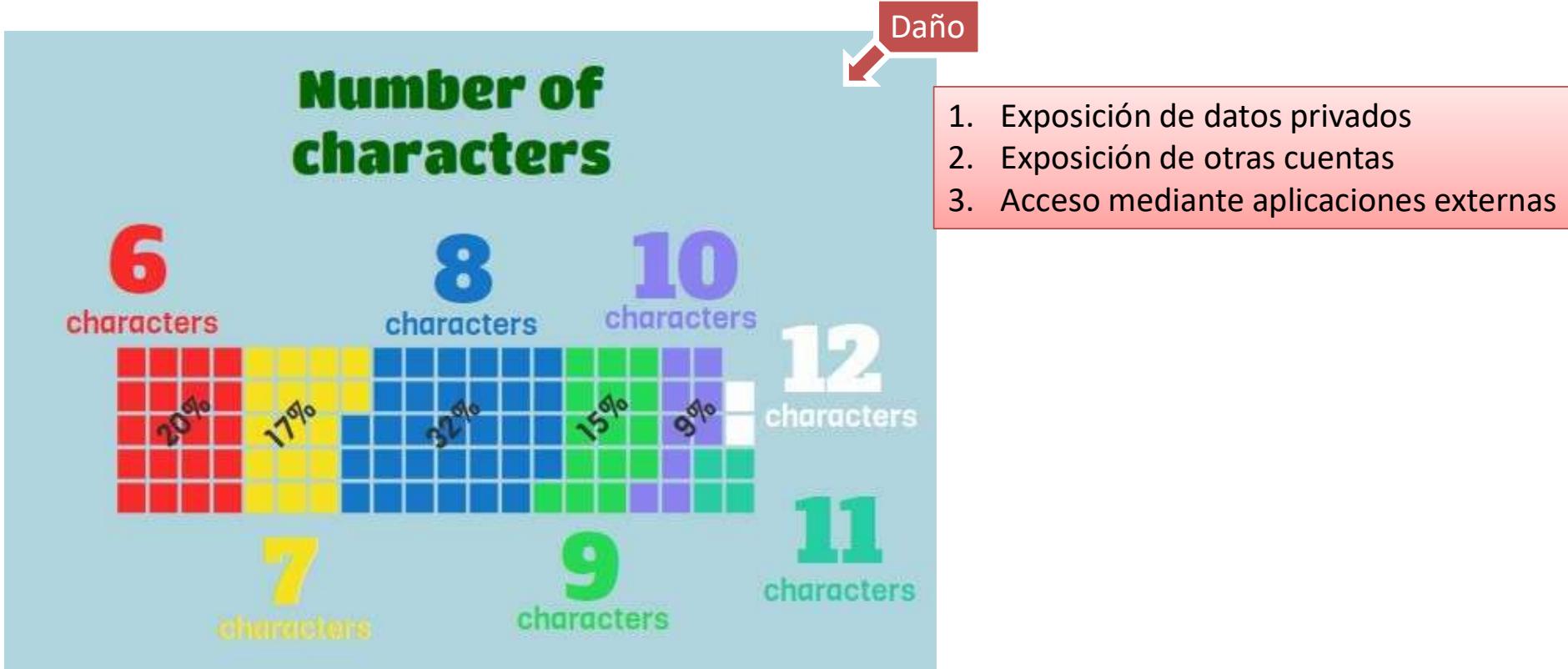
- Completa el proceso de seguridad almacenando información clave sobre el “quién”, “qué”, “cuándo” y “desde dónde” para analizar, a posteriori, incidentes de ciberseguridad

En ocasiones, se añade la **auditoría**, y resulta en **AAAA**.

# autenticación: auditoría

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas

Filtración de 6,5 millones de contraseñas: **Caso LinkedIn**



# autenticación: auditoría

introducción

amenazas

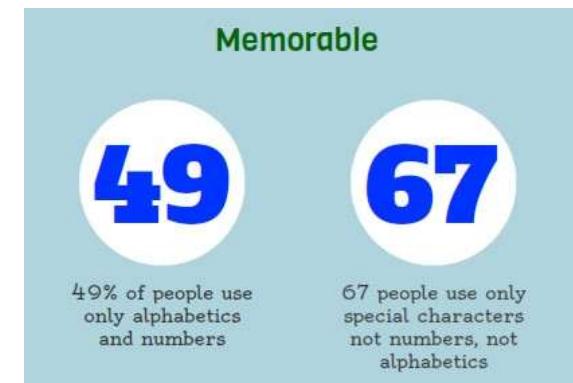
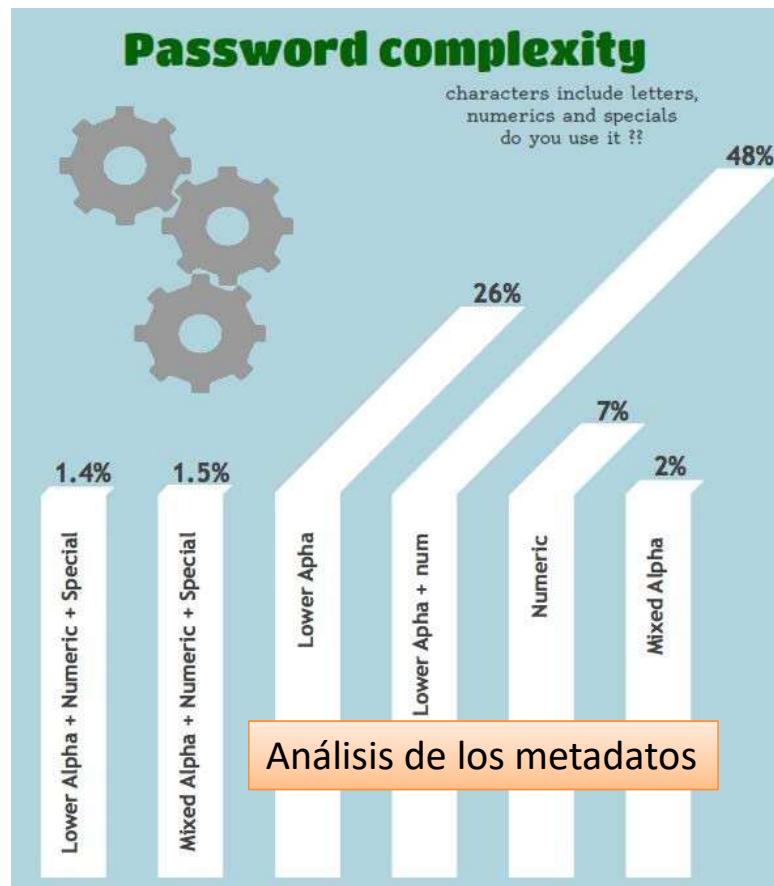
criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

Filtración de 6,5 millones de contraseñas: **Caso LinkedIn**



- Resultados de la auditoria**
- Errores cometidos:**
1. Sólo usaron HASH (sha-1)
  2. Sólo 1 iteración “sha-1”
  3. No usaron SAL, ni PIMENTA

# AAA: introducción

introducción  
amenazas  
criptografía  
escenarios  
autenticación y autorización  
buenas prácticas



## Responsabilidad (Accounting):

- Completa el proceso de seguridad almacenando información clave sobre el “quién”, “qué”, “cuándo” y “desde dónde” para analizar, a posteriori, incidentes de ciberseguridad

En ocasiones, se añade la **auditoría**, y resulta en **AAAAA**.

**QA!!**

## Comparison between QA, QC and Testing

### Quality Assurance

- Subset of SDLC
- Process oriented
- Ensure that processes and procedures are in place to achieve quality
- Focus on process to achieve required quality
- Prevent defects
- Whole team approach
- Proactive process

### Quality Control

- Subset of QA
- Product oriented
- Activities to ensure the product quality
- Focus on product to check for the required quality
- Find and fix defects
- Reactive process
- Testing team

### Testing

- Subset of QC
- Product oriented
- Validate the product against specifications
- Focus on actual testing of the product
- Find and fix defects
- Reactive process
- Testing team

# autenticación: Needham-Schroeder

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

**Algo de historia:** (ejemplo de autenticación mediante servidor → escenario 2)

- **En los primeros sistemas distribuidos (1974-84) era difícil proteger los servidores:**
  - P.e. contra ataques enmascarados sobre un servidor de ficheros
  - No había mecanismos de autenticación del origen de la petición
  - La criptografía de clave pública no estaba disponible
    - computadoras demasiado lentas para cálculos importantes
    - RSA no estaba disponible
- **Needham y Schroeder desarrollaron un protocolo de autenticación y distribución de claves para uso en red local:**
  - Supuso un primer ejemplo del cuidado en el diseño de protocolos de seguridad
  - Introdujeron varias ideas de diseño: p.e. **ocasiones**

# autorización: Needham-Schroeder

introducción

amenazas

criptografía

escenarios

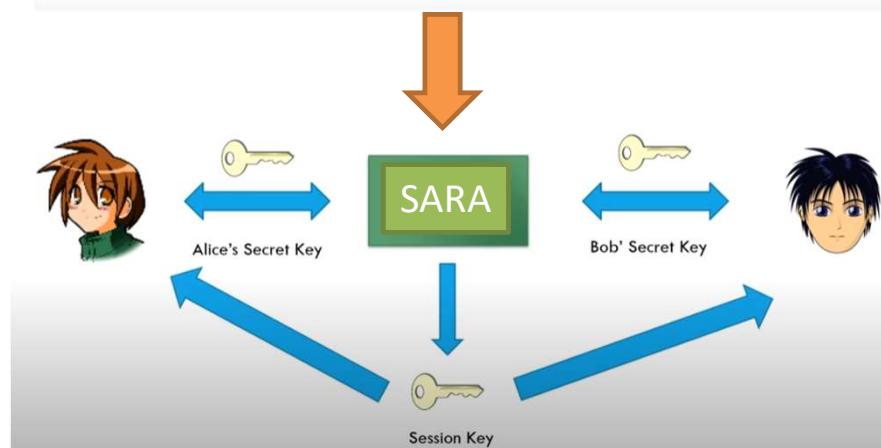
**autenticación  
y autorización**

buenas  
prácticas

## *Encabezado Mensaje*



## *Notas*



# autorización: Needham-Schroeder

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

<i>Encabezado Mensaje</i>	<i>Notas</i>
1. A->S: $A, B, N_A$	Ocasión A solicita una clave a S para comunicarse con B
2. S->A: $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{KB}\}_KA$	Token S devuelve un mensaje encriptado en la clave secreta de A, con una clave nueva $K_{AB}$ y un “ticket” encriptado en la clave secreta de B. La ocasión $N_A$ demuestra que el mensaje fue enviado en respuesta al anterior. A confía en que S envió el mensaje porque sólo S conoce la clave secreta de A
3. A->B: $\{K_{AB}, A\}_{KB}$	A envía el “ticket” a B
4. B->A: $\{N_B\}_{KAB}$	Ocasión B desencripta el “ticket” y utiliza la nueva clave $K_{AB}$ para encriptar otra ocasión $N_B$
5. A->B: $\{N_B - 1\}_{KAB}$	A demuestra a B que fue el emisor del mensaje anterior devolviendo una transformación acordada sobre $N_B$ .

# autorización: Needham-Schroeder

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

## Encabezado Mensaje

## Notas

1. A->S:  $A, B, N_A$

Ocasión

A solicita una clave a S para comunicarse con B

2. S->A:  $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{KB}\}_{KA}$

Token

S devuelve un mensaje encriptado en la clave secreta de A, con una clave nueva  $K_{AB}$  y un “ticket” encriptado en la clave secreta de B. La ocasión  $N_A$  demuestra que el mensaje fue enviado en respuesta al anterior. A confía en que S envió el mensaje porque sólo S conoce la clave secreta de A

3. A->B:  $\{K_{AB}, A\}_{KB}$

A envía el “ticket” a B

4. B->A:  $\{N_B\}_{KAB}$

B desencripta el “ticket” y utiliza la nueva clave  $K_{AB}$  para encriptar otra ocasión  $N_B$

5. A->B:  $\{N_B - 1\}_{KAB}$

A demuestra a B que fue el emisor del mensaje anterior devolviendo una transformación acordada sobre  $N_B$ .

# autorización: Needham-Schroeder

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas



1.  $ID_a \parallel ID_b \parallel N_1$
2.  $E(K_a, [K_s \parallel ID_b \parallel N_1 \parallel E(K_b, [K_s \parallel ID_a])])$
3.  $E(K_b, [K_s \parallel ID_a])$
4.  $E(K_s, N_2)$
5.  $E(K_s, f(N_2))$

If an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations of Bob.

Example :

Suppose an opponent, X , has been able to compromise an old session key

The opponent is able to observed what is happening at step 3 and recorded it.

X can impersonate Alice and trick Bob into using the old key by simply replaying step 3.

If X can intercept the handshake message in step 4, then it can impersonate Alice's response in step 5.

From this point on, X can send his responses to Bob that appear to Bob to come from Alice.

# autorización

---

introducción

amenazas

criptografía

escenarios

**autenticación  
y autorización**

buenas  
prácticas

Dado un usuario autenticado, establece a qué recursos tiene acceso legítimo

- **Kerberos**
- **OAuth 2.0**

# autenticación: Kerberos

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

## Algo de historia:

- Comunicación segura con servidores en una red local
  - Desarrollado en el MIT en 80s para ofrecer seguridad en la red del campus > 5000 usuarios
  - basado en Needham - Schroeder
- Estandarizado e incluido en muchos SO
  - Internet RFC 1510, OSF DCE
  - BSD UNIX, Linux, Windows 2000, NT, XP, etc.
  - Disponible en la web del MIT
- El servidor Kerberos crea una clave secreta compartida para cada servidor solicitado y la envía encriptada al computador del usuario
- El password del usuario es el secreto inicial para la autenticación en Kerberos
- Originalmente se usaba una estrategia basada en servidor de autenticación (escenario 2). En la actualidad se utiliza el **cifrado asimétrico** para el intercambio de claves

# autenticación: Kerberos

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

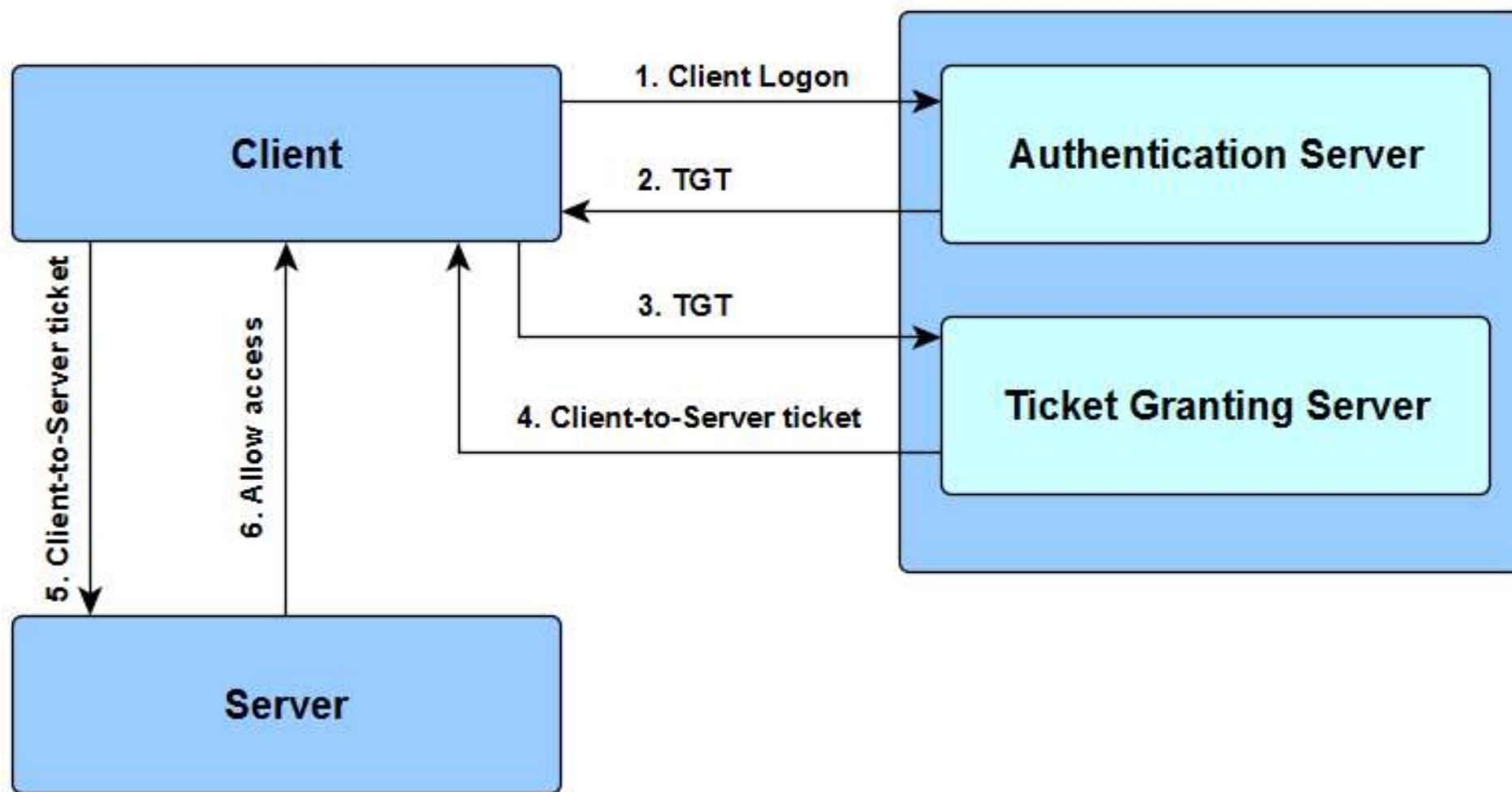
buenas  
prácticas

## Algo de historia:

- Comunicación segura con servidores en una red local
  - Desarrollado en el MIT en 80s para ofrecer seguridad en la red del campus > 5000 usuarios
  - basado en Needham - Schroeder
- Estandarizado e incluido en muchos SO
  - Internet RFC 1510, OSF DCE
  - BSD UNIX, Linux, Windows 2000, NT, XP, etc.
  - Disponible en la web del MIT
- El servidor Kerberos crea una clave secreta compartida para cada servidor solicitado y la envía encriptada al computador del usuario
- El password del usuario es el secreto inicial para la autenticación en Kerberos
- Originalmente se usaba una estrategia basada en servidor de autenticación (escenario 2). En la actualidad se utiliza el **cifrado asimétrico** para el intercambio de claves

# autorización: Kerberos

Key Distribution Center



introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

# autorización: Kerberos

introducción

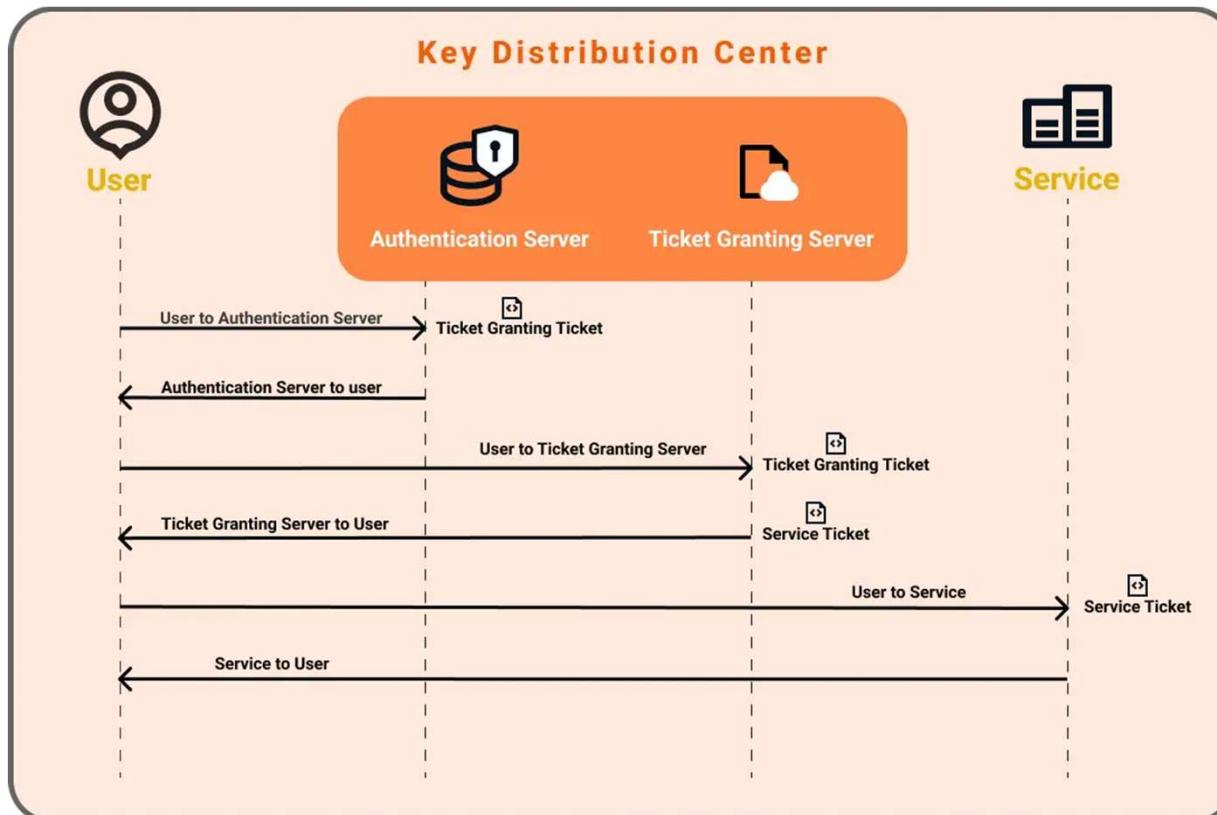
amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas



**Paso 1:** el usuario envía una solicitud al servidor de autenticación (SA) pidiendo acceder a un servicio

**Paso 2:** El SA valida al usuario y otorga un TGT: ticket-granting-ticket y lo devuelve al usuario cifrado mediante la clave pública del usuario (autenticación)

**Paso 3:** el usuario descifra el mensaje y envía el TGT con información adicional al servidor de concesión de tickets (TGS - Ticket Granting Server)

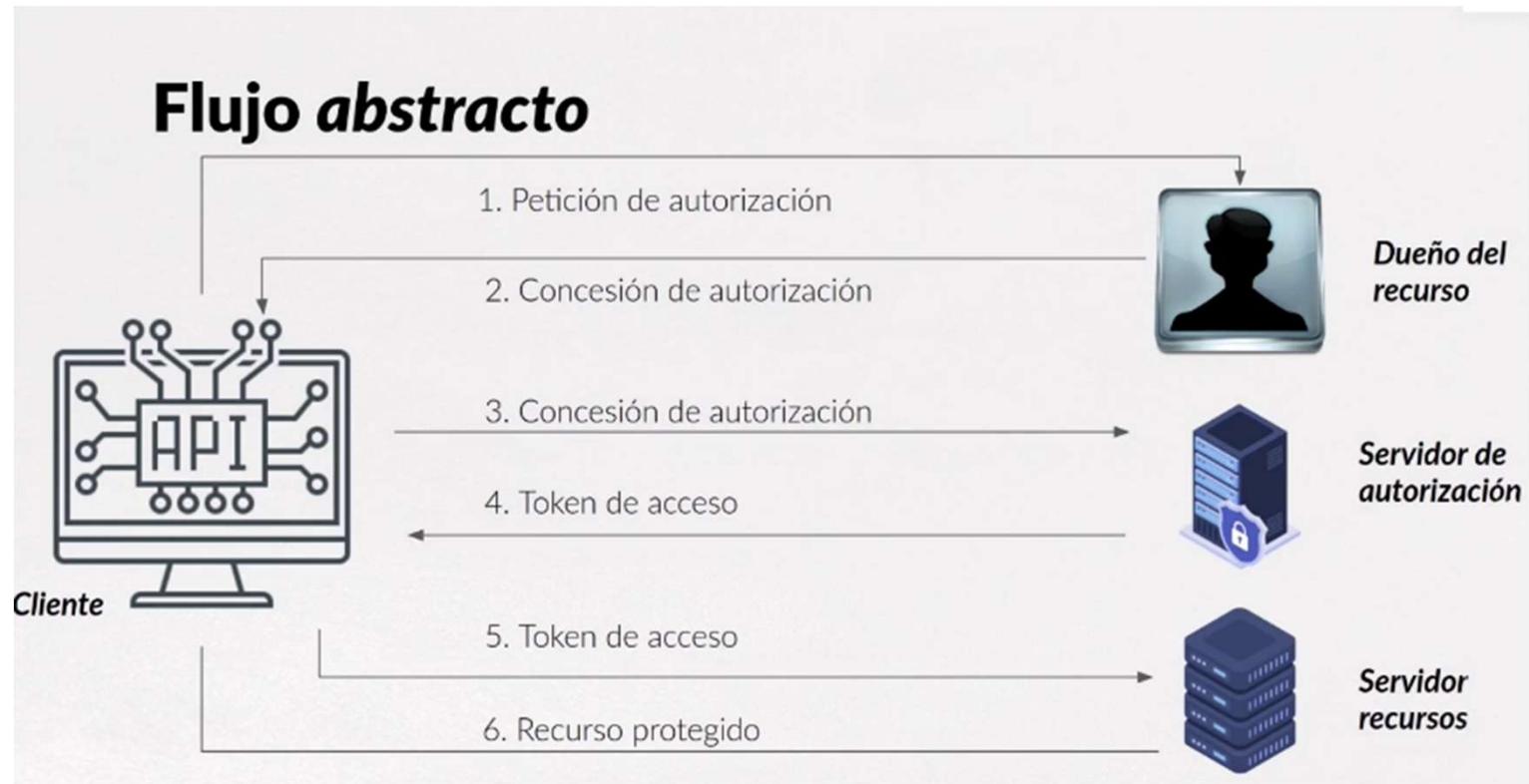
**Paso 4:** el TGS descifra el TGT, realiza las validaciones y crea un ticket de servicio (ST) y lo envía al usuario (autorización)

**Paso 5:** el usuario descifra el mensaje, crea un mensaje de autenticación y envía el autenticador de usuario y el ticket de autorización al Servicio

**Paso 6:** el servicio realiza su descifrado y validación y crea su propio mensaje de autenticación final con cifrado simétrico

# autenticación: OAuth2

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas



# autenticación: OAuth2

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

Framework de autorización estándar y de código abierto, que permite a una API obtener acceso limitado a las cuentas de usuario de servicios como Facebook, Twitter, Google, LinkedIn, etc.

- Delega la autenticación de usuario al servicio que gestiona las cuentas
- Provee el flujo para la autorización de aplicaciones web, aplicaciones móviles, etc.

Entidades involucradas en el flujo de OAuth2:

- **Propietario del recurso:** Usuario propietario de los datos personales que se usan para autorizar
- **Aplicación cliente:** Sitio web o aplicación que accederá a los recursos protegidos de un usuario con la autorización del mismo
- **Servidor de autorización:** Valida usuario y credenciales y genera tokens de acceso
- **Servidor de recursos:** Recibe peticiones de acceso a los recursos protegidos autorizando acceso, si el token es válido

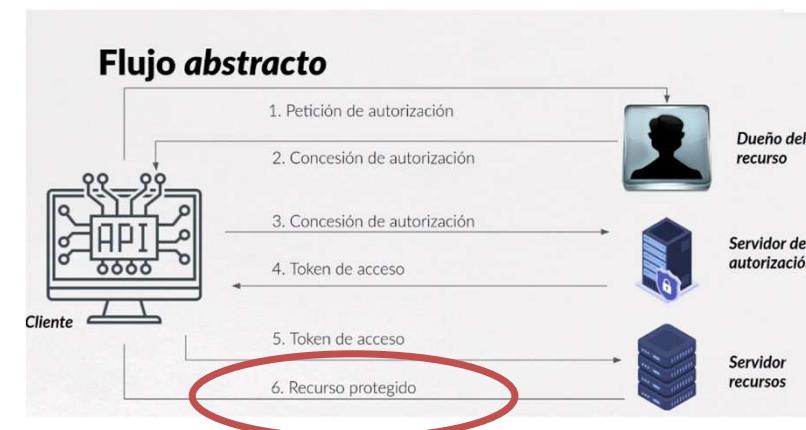
Casos de uso:

- Una persona con cuenta en Facebook o Twitter quiere publicar contenido a través de otra aplicación
- Una persona con cuenta en Facebook o Twitter quiere acceder a otra plataforma pero sin tener que registrarse

# autenticación: OAuth2

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas

1. La *aplicación* solicita autorización para acceder a los recursos de servicio del *usuario*
2. Si el *usuario* autoriza la solicitud, la *aplicación* recibe la autorización
3. La *aplicación* solicita al *servidor de autorización (API)*, presentando la autenticación de su identidad y la autorización otorgada. La aplicación solicita al servidor de autorización (API) **un token de acceso** presentando la autenticación de su propia identidad y la autorización otorgada
4. Si la identidad de la aplicación es autenticada y la autorización es válida, el **servidor de autorización (API)** **emite un token de acceso** a la aplicación. **La autorización finaliza**
5. La *aplicación* solicita el recurso al *servidor de recursos (API)* y presenta el token de acceso para autenticarse
6. Si el token de acceso es válido, el *servidor de recursos (API)* provee el recurso a la *aplicación*



# autenticación: ejemplo OAuth2 (parte I)

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

Supongamos la página de DISNEY que permite acceder a través de Facebook (se debe haber registrado la web con el servicio y dar permisos para su uso):

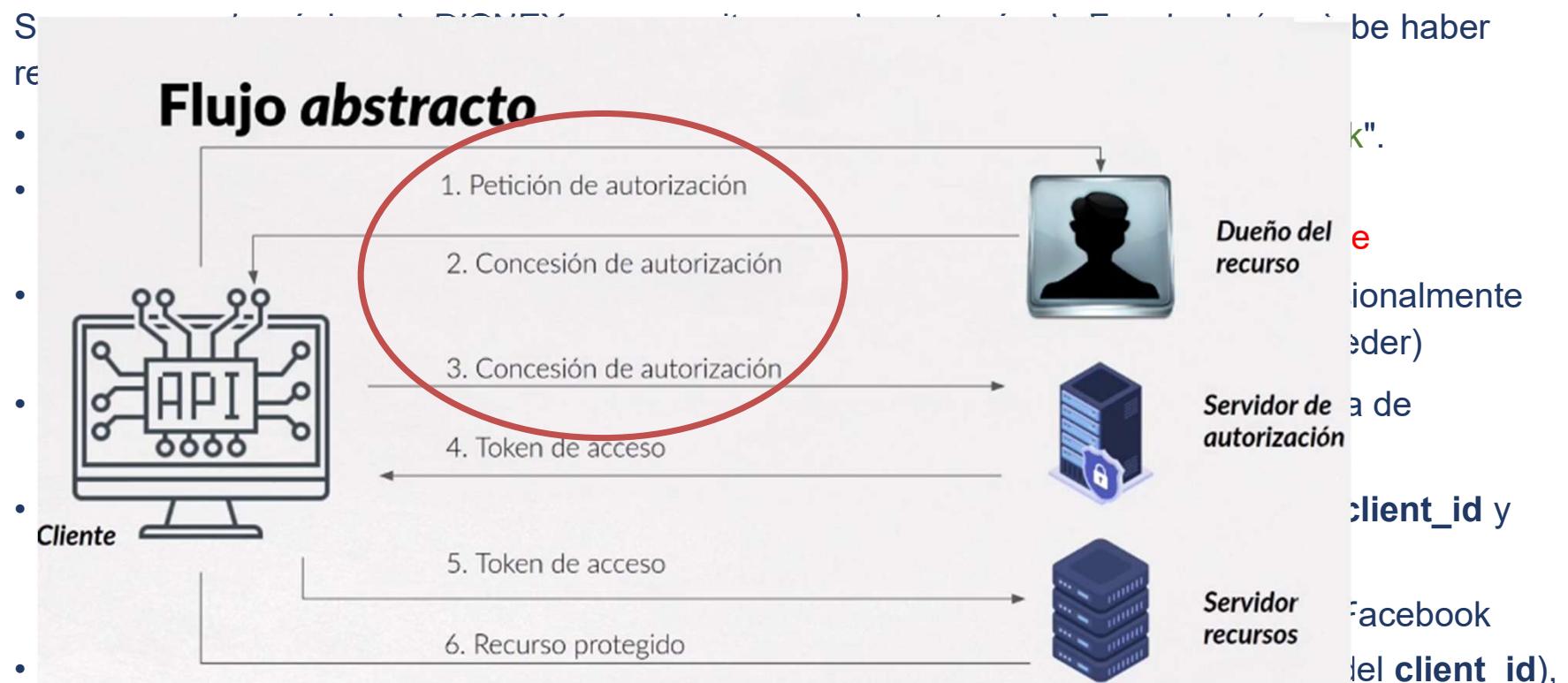
- El usuario entra en la página de DISNEY y hace clic en "Ingresar usando Facebook".
- La aplicación lo va a redirigir a una URL como la siguiente:  
`facebook.com/oauth2/auth?client_id=ABC&redirect_uri=disney.com/oauth_response`
- Esta URL contiene los siguientes parámetros: un **client\_id**, una **redirect\_uri** y opcionalmente un parámetro **scopes** (para indicar a qué información de Facebook queremos acceder)
- Facebook primero va a ver si nuestro **client\_id** es válido (comparándolo con la lista de **oauth\_client** permitidos)
- Si todo está correcto, entonces define una variable de sesión que guarda nuestro **client\_id** y **redirect\_uri**. y entonces:

Redirige al usuario a `facebook.com/login` o avanza si ya hay una sesión activa en Facebook

- Facebook muestra el logo de DISNEY y el nombre de la app (lo reconoce a partir del **client\_id**), indicando al usuario: "Esta app quiere acceder a tus datos de Facebook, ¿le das permiso?" (según el **scope** indicado previamente).

# autenticación: ejemplo OAuth2 (parte I)

introducción  
amenazas  
criptografía  
escenarios  
**autenticación y autorización**  
buenas prácticas



indicando al usuario: "Esta app quiere acceder a tus datos de Facebook, ¿le das permiso?"  
(según el **scope** indicado previamente).

# autenticación: ejemplo OAuth2 (parte II)

---

introducción

amenazas

criptografía

escenarios

autenticación  
y autorización

buenas  
prácticas

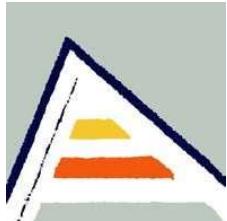
**Si se acepta lo anterior entonces ...**

- Facebook genera un código **client\_secret** (que tiene un sólo uso válido para DISNEY, el usuario y el **scope**). Facebook redirige al usuario según la **redirect\_uri** indicada al inicio
- Facebook envía el código generado hacia la aplicación DISNEY:  
[disney.com/oauth\\_response?code=aqui\\_un\\_codigo\\_extenso](http://disney.com/oauth_response?code=aqui_un_codigo_extenso)
- DISNEY toma el código que recibe de Facebook y vuelve a hacer una petición a Facebook, incluyendo ahora su **client\_secret**

Para probar su identidad, DISNEY hace una petición de la siguiente forma:

[facebook.com/oauth2/token?client\\_id=ABC&client\\_secret=XYZ&code=aqui\\_un\\_codigo\\_extenso](http://facebook.com/oauth2/token?client_id=ABC&client_secret=XYZ&code=aqui_un_codigo_extenso)

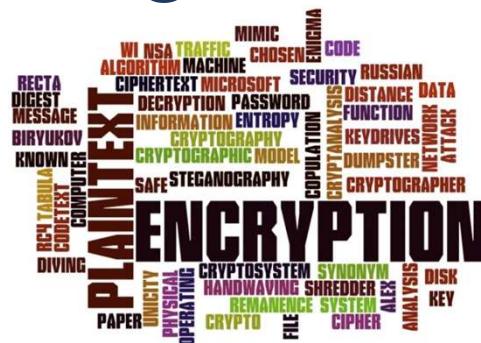
- Facebook verifica que el código sea válido y lo invalida en ese instante (ya que son de uso único)
- Por último, Facebook responde con un **AccessToken**, que DISNEY podrá usar (hasta que expire) para hacer peticiones a la API, en nombre del usuario que ha otorgado los permisos



# Grado en Ingeniería Informática

# Sistemas distribuidos

# Seguridad



Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# recomendaciones de seguridad

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

**1. Educación de los usuarios:** ninguna herramienta puede proteger de los errores de un usuario. Deben sensibilizarse sobre lo que puede ocurrir. El miedo o la prohibición no son soluciones.

Los programas de sensibilización deben tratar:

- **Política de seguridad corporativa**
- Establecimiento de contraseñas y renovación
- Comportamiento ante los virus y prevención
- **Uso y abuso del email:** puerta de entrada a infección
- **Acceso a Internet: un privilegio, no un derecho**
- Robo de dispositivos portátiles: proteger los datos
- **Ingeniería social**
- Importancia del control de acceso a las instalaciones

# recomendaciones de seguridad

introducción

amenazas

criptografía

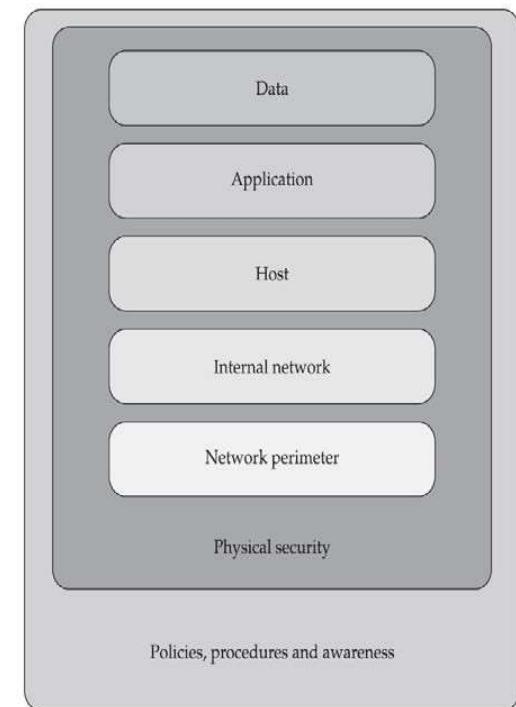
escenarios

autenticación y  
autorización

buenas  
prácticas

## 2. Defensa elástica: en vez de parar el ataque, se busca ralentizarlo al máximo

- Firewalls
- Filtros de paquetes
- Redes virtuales (VPN)
- Acceso temporizado
- Biometría
- Soft/hard no conectado exterior
- Auditoría y logs



# recomendaciones de seguridad

---

introducción

amenazas

criptografía

escenarios

autenticación y  
autorización

buenas  
prácticas

## **3. Robustecimiento del sistema:**

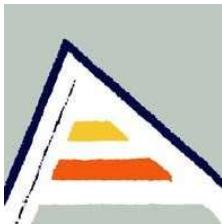
- eliminar utilidades y programas no esenciales
- Parar cualquier servicio innecesario
- Evitar arranques SO desde elementos externos
- TCP/IP único protocolo instalado
- Evitar compartición de ficheros e impresoras
- Eliminar cuentas “guest”. Renombrar a “root”
- Enjaulado de servicios (chroot)

## **4. Actualizaciones automáticas:** parches y mejoras

**5. Virtualización y contenedores:** se mejoran consumos y mantenimiento, recuperación ante desastres y procedimientos de seguridad

**6. Herramientas de control (unix):** rkhunter (compara hash de archivos con originales), chkrootkit (shell script), antivirus, detectores adware, etc.

**7. Registros externalizados,** guardando regularmente copias de seguridad de los mismos



Grado en Ingeniería Informática

## Sistemas distribuidos

# Seguridad



Departamento de Tecnología Informática y Computación

Curso 2023 - 2024



Grado en Ingeniería Informática

## **Sistemas distribuidos**

# **Sincronización de tiempo y coordinación**

## **Coulouris: Disitributed systems Ch14**

Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

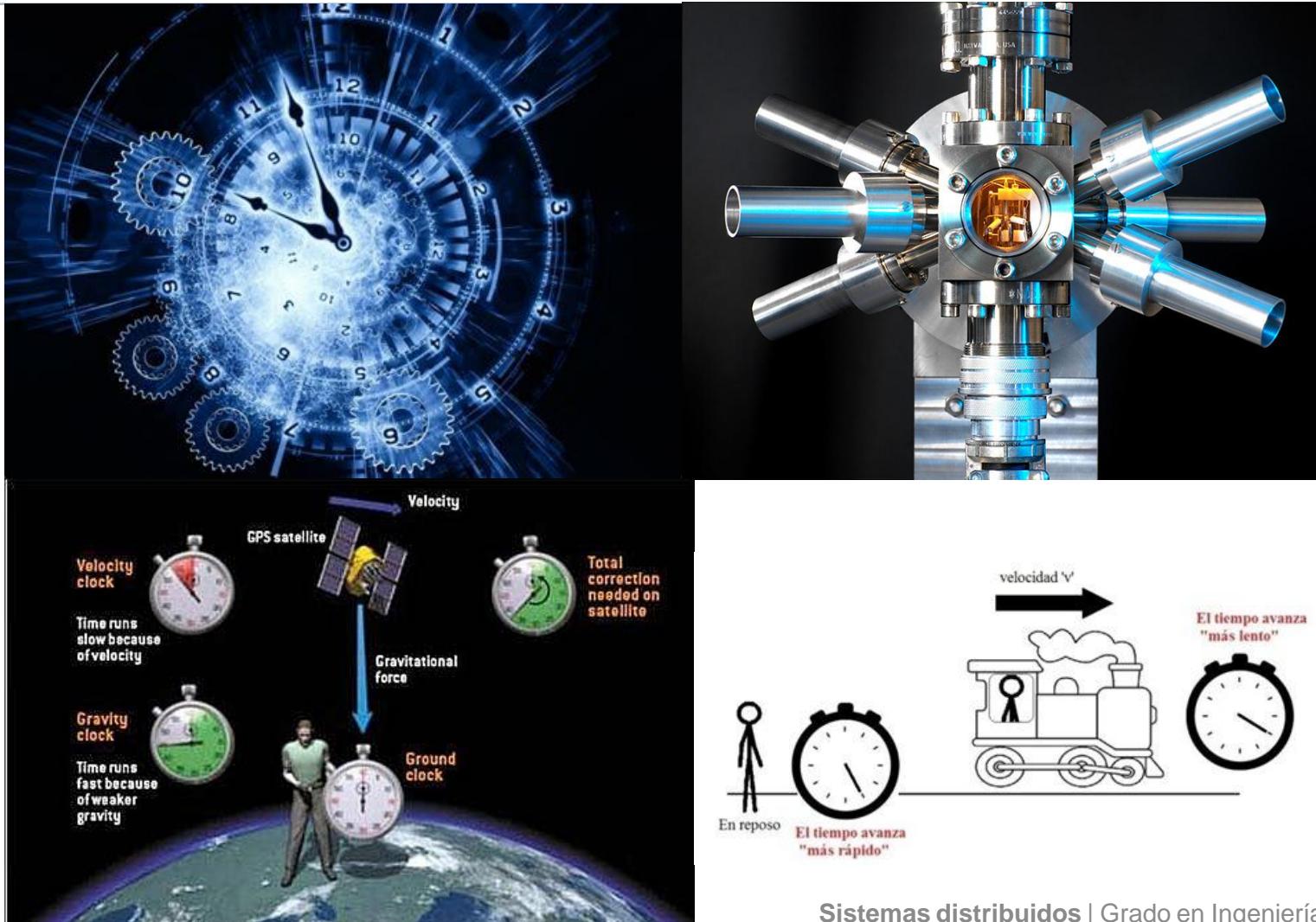
# sincronización en los sistemas distribuidos

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador



# sincronización en los sistemas distribuidos

---

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Para poder ejecutar tareas en entornos distribuidos de forma coherente es importante que el **orden** en las secuencias de las operaciones de procesos sea **estricto** y **universal**

Ejemplo:

Timestamps de comercio electrónico (facturas, orden de compra, ...)

# sincronización en los sistemas distribuidos

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Para poder ejecutar tareas en entornos distribuidos de forma coherente es importante que el **orden** en las secuencias de las operaciones de procesos sea **estricto** y **universal**
- La **sincronización** se define como **la forma de forzar un orden parcial o total para cualquier conjunto de eventos** y se utiliza para abordar tres problemas distintos y relacionados:
  - La sincronización entre un emisor y un receptor
  - La especificación y el control de la actividad común entre procesos cooperativos
  - La serialización de accesos concurrentes a objetos compartidos
- La **sincronización de relojes** en un sistema distribuido cobra especial importancia:
  - Se debe establecer la misma referencia de tiempo para todas las entidades
  - Se debe garantizar que los procesos se ejecuten de forma coordinada y respetando el orden original de los eventos, en la medida de lo posible

# introducción

---

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Monitorización y temporización de ejecuciones
- Necesidad de conocer cuando ocurrió un evento
  - Algoritmos de sincronización
  - Manteniendo de consistencia en transacciones
  - Protocolos de autenticación
  - ...
- ¿Existe un reloj universal de referencia?
  - Teoría Especial de la Relatividad de Einstein
    - Causa física y efecto físico
    - Temporización de la causa y el efecto
    - Tiempo físico absoluto de Newton

# introducción

---

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

Noción de tiempo es también problemática en un sistema distribuido:

- No existe un reloj global al sistema
- Cada computador de la red tiene su propio reloj interno:
  - usado por los procesos locales para obtener el valor del tiempo actual. P. ej. uso en el *make* de *unix*
  - procesos en computadores distintos pueden tener marcas de tiempo distintas
  - los relojes derivan con respecto al tiempo perfecto y las tasas de deriva también difieren entre ellos
- Aunque todos los relojes del SD se sincronicen, estos variarán significativamente con el tiempo

Aproximaciones a la solución:

- Algoritmos de sincronización de los relojes de los computadores
- Reloj lógicos y relojes vectoriales

# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- **Caracterización de un sistema distribuido (SD):**
  - Un SD se define como una colección P de N procesos  $p_i$  donde  $i = 1, 2, \dots, N$
  - Cada proceso  $p_i$  tiene un estado si formado por todas sus variables u objetos y que puede cambiar en ejecución
  - Se comunican a través de la red mediante mensajes
  - Las acciones que puede realizar un proceso son: enviar, recibir o cambiar estado
- **Evento:** ocurrencia de una acción que lleva a cabo un proceso al ejecutar:  
p.e. un envío, una recepción, un cálculo, ...
- Los eventos en el proceso  $p_i$ , pueden ordenarse de forma total por la relación  $\rightarrow_i$  “suceder antes en  $p_i$ ”

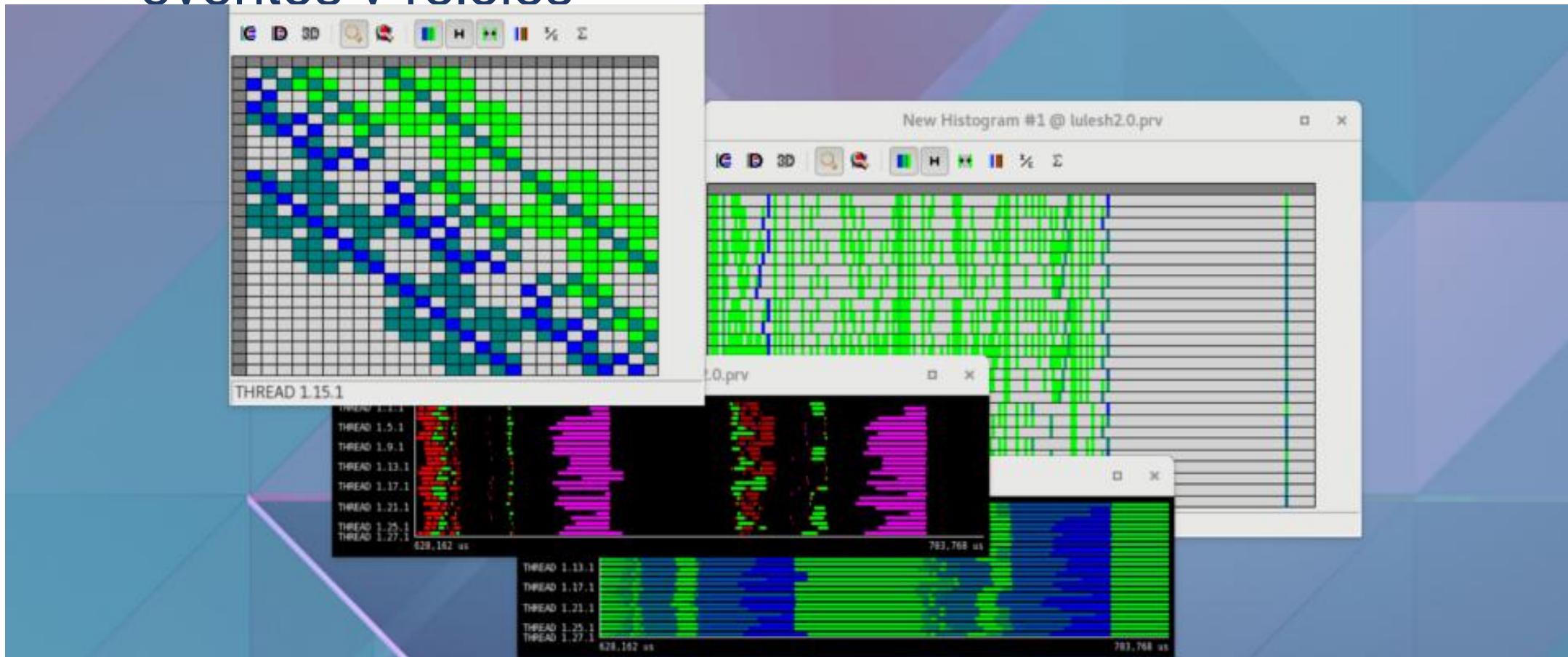
# eventos v relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador



- Los eventos en el proceso  $p_i$ , pueden ordenarse de forma total por la relación  $\rightarrow_i$  “suceder antes en  $p_i$ ”

# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Relojes

- Para establecer las marcas temporales se usa el reloj del computador
- En un instante  $t$  el SO lee el valor del reloj hardware del computador:  $H_i(t)$  y calcula el tiempo mediante software (reloj software):
$$C_i(t) = \alpha H_i(t) + \beta \text{ (es decir, escala y compensa)}$$
  - p.e. un número de 64-bit dando los nanosegundos desde un tiempo base
  - En general no es completamente exacto, pero si  $C_i$  se comporta suficientemente bien, puede ser usado como marcador de los eventos de  $p_i$
- **Resolución del reloj:** periodo entre dos actualizaciones consecutivas del reloj y debe ser menor que el intervalo de tiempo entre dos eventos consecutivos

# eventos y relojes

---

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Relojes

- Los relojes de un SD no siempre están en perfecto acuerdo
  - **Sesgo:** diferencia de tiempo entre dos relojes en un instante determinado
  - **Tasa de deriva:** diferencia por unidad de tiempo en que el reloj del computador difiere del reloj de referencia
    - Los relojes de cuarzo ordinarios derivan 1 seg. cada 11-12 días ( $10^{-6}$  seg/seg)
    - Los relojes de alta precisión derivan  $10^{-7}$  ó  $10^{-8}$  seg/seg
- ¿Qué puede influir en la tasa de deriva? p. ej. el estado de las baterías, ¿se te ocurren otras?

# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Relojes

- **Corrección de un reloj:** Se dice que un reloj hardware ( $H$ ) es correcto, si su límite de derivada es conocido  $\rho > 0$ . (p.e.  $10^{-6}$  sec/sec):
  - El error en la marca de tiempo de dos eventos en  $t$  y  $t'$  está acotado:  $(1 - \rho)(t' - t) \leq H(t') - H(t) \leq (1 + \rho)(t' - t)$ , donde  $t' > t$
  - **Se impide que se produzcan saltos traumáticos en el valor de tiempo**
- Se puede relajar la condición, exigiendo tan solo **monotonía**:
  - $t' > t \rightarrow C(t') > C(t)$  [p.e. exigencia del *make* de Unix]
  - se puede alcanzar la monotonía en un reloj hardware que a mayor frecuencia, ajustando los valores de  $\alpha$  y  $\beta$   
 $C_i(t) = \alpha H_i(t) + \beta$
- **Un reloj defectuoso** es aquel que no cumple ninguna de las condiciones de corrección
- **Un fallo de ruptura** de reloj: el reloj se para, no emite ticks
- **Un fallo arbitrario**: cualquier otro fallo... p. ej. efecto 2000 (**Y2K**) 2038 (**Y2K38**)

# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Reloj

- **Corrección de un reloj:** Se dice que un reloj hardware ( $H$ ) es correcto, si su límite de error es menor que el límite de tolerancia ( $\epsilon$ ).
  - $H \in [H_{min}, H_{max}]$
  - $|H - H_{true}| \leq \epsilon$
- **Binary:** 01111111 11111111 11111111 11110000
- **Decimal:** 2147483632
- **Date:** 2038-01-19 03:13:52 (UTC)
- **Date:** 2038-01-19 03:13:52 (UTC)
- **Un fallo de ruptura de reloj:** el reloj se para, no emite ticks
- **Un fallo arbitrario:** cualquier otro fallo... p. ej. efecto 2000 (**Y2K**) 2038 (**Y2K38**)

# eventos y relojes

introducción

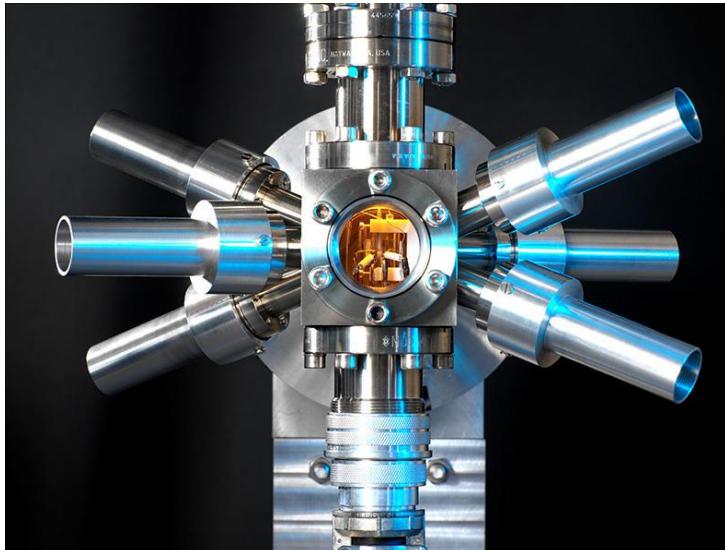
establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Tiempo Universal Coordinado (UTC)

- UTC es un estándar internacional de establecimiento y mantenimiento del tiempo transcurrido
- Basado en el tiempo atómico y ocasionalmente ajustado al tiempo astronómico\*. La referencia de tiempo la obtiene de un reloj con tasa de deriva de  $10^{-13}$  (aprox. de un seg. cada 300.000 años) que establece el Tiempo Atómico Internacional



# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Tiempo Universal Coordinado (UTC)

- UTC es un estándar internacional de establecimiento y mantenimiento del tiempo transcurrido
- Basado en el tiempo atómico y ocasionalmente ajustado al tiempo astronómico\*. La referencia de tiempo la obtiene de un reloj con tasa de deriva de  $10^{-13}$  (aprox. de un seg. cada 300.000 años) que establece el Tiempo Atómico Internacional
- La señal se difunde mediante estaciones de radio por tierra y mediante satélites, permitiendo que los relojes de los computadores se sincronizan con estas fuentes externas tan precisas, con receptores adecuados **¿factible? ¿útil?**
  - Las estaciones terrestres tienen una precisión entre 0.1-10 miliseg.
  - El GPS tiene una precisión de 1 microseg.

# eventos y relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Sincronización de los relojes

- **Sincronización externa:** un reloj  $C_i$  se sincroniza con una fuente UTC exacta  $S$ , si se cumple:
  - $|S(t) - C_i(t)| < D$  para  $i = 1, 2, \dots, N$  en un intervalo de tiempo
  - Los relojes  $C_i$  son precisos con el límite  $D$
- **Sincronización interna:** cualquier par de computadores están sincronizados, si sus relojes cumplen:
  - $|C_i(t) - C_j(t)| < Z$  para  $i, j = 1, 2, \dots, N$ ;  $i \neq j$  en un intervalo de tiempo
  - Los relojes  $C_i$  y  $C_j$  concuerdan con el límite  $Z$
- Relojas sincronizados internamente no necesariamente lo están externamente, puesto que pueden derivar juntos
- *Si el conjunto  $P$  está sincronizado externamente, ¿también lo está internamente?*

$$Z = 2D$$

# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

La sincronización de relojes en un SD depende de la naturaleza del mismo y determinará el algoritmo de sincronización más adecuado:

- SD síncrono
- SD asíncrono

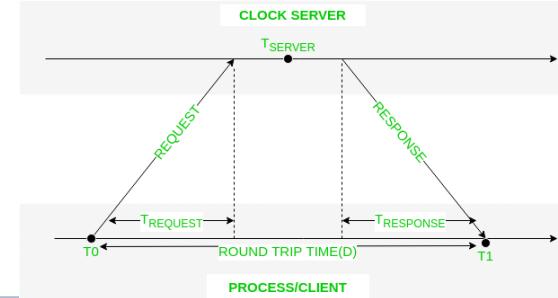
Un **sistema distribuido es síncrono** si están definidos o se pueden determinar los parámetros siguientes:

- Tiempo máximo y mínimo para ejecutar cada instrucción de un proceso
- Tiempo máximo y mínimo de recepción de un mensaje
- Los límites de deriva de cada reloj local donde se ejecuta cada proceso son conocidos

Un **sistema distribuido es asíncrono** si no se pueden determinar los parámetros anteriores (es decir, **si no es síncrono**)

¿Ejemplos de uno y otro? ¿computadores en una LAN? ¿Internet?

# sincronización de relojes



## Método de Cristian (para SD síncronos y sincronización externa)

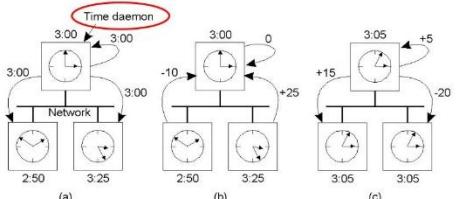
Un servidor de tiempo S recibe señales UTC

- El proceso p solicita el tiempo en un mensaje  $m_r$  y recibe t en  $m_t$  de S
- p establece su tiempo a  $t + T_{round}/2$  [ $T_{round}$  es el tiempo de ida y vuelta]
- Precisión:  $\pm (T_{round}/2 - min)$  [min es el mínimo estimado de transmisión]
  - El momento más temprano en que S pone t en  $m_t$  es min después de que p enviara  $m_r$ .
  - El momento más tardío es min antes de que  $m_t$  llegue a p
  - El tiempo de S cuando  $m_t$  llega está en el rango:  $[t + min, t + T_{round} - min]$

Problemas:

- Es probabilístico
- Se soporta mediante un único servidor de tiempo → No tolerante a fallos

Solución a la tolerancia a fallos: tener varios servidores de tiempo y aceptar el primer mensaje válido



a)  
b)  
c)  
The time daemon asks all the other machines for their clock value discrepancies  
The machines answer  
The time daemon tells everyone how to adjust their clock to the average  
no Universal Coordinated Time available

# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Método de Berkeley (para SD síncronos y sincronización interna)

- Un maestro consulta y recoge valores de reloj del resto de computadores, esclavos
- El maestro utiliza los tiempos de ida y vuelta de los mensajes para estimar el valor de los relojes esclavos (similar a **Cristian**)
- Promedia los resultados incluyéndose y eliminando cualquier valor que no sea consistente
- Envía la magnitud de ajuste de cada reloj, puede ser positivo o **negativo**

### Experimentos:

- 15 computadoras, tiempo de sincronización 20-25 milisegs. Tasa de deriva de relojes locales  $< 2 \times 10^{-5}$

Problema: si se produce el fallo del maestro

Solución: activar algoritmo de elección para establecer un nuevo maestro

# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

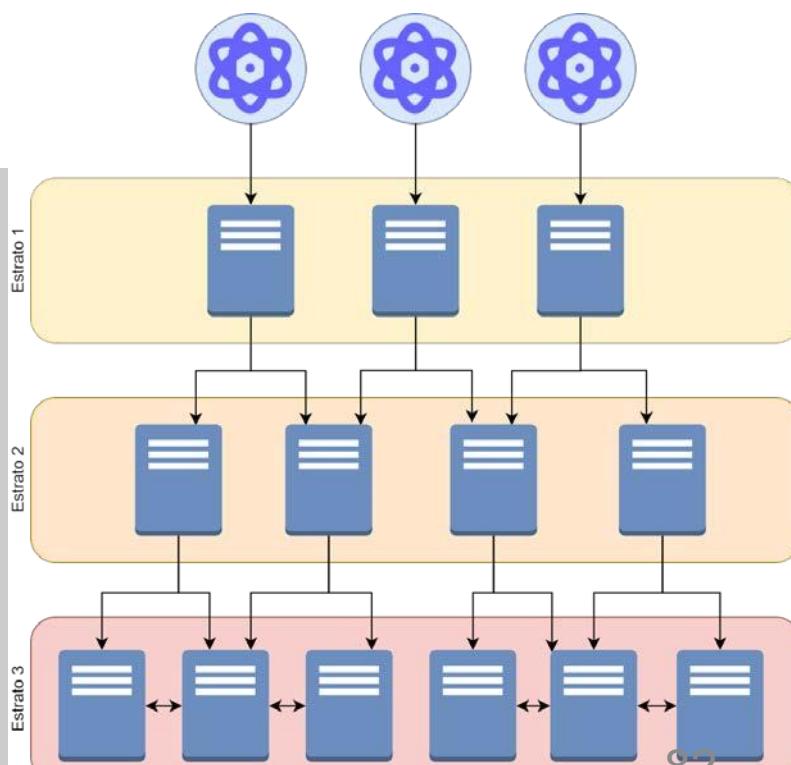
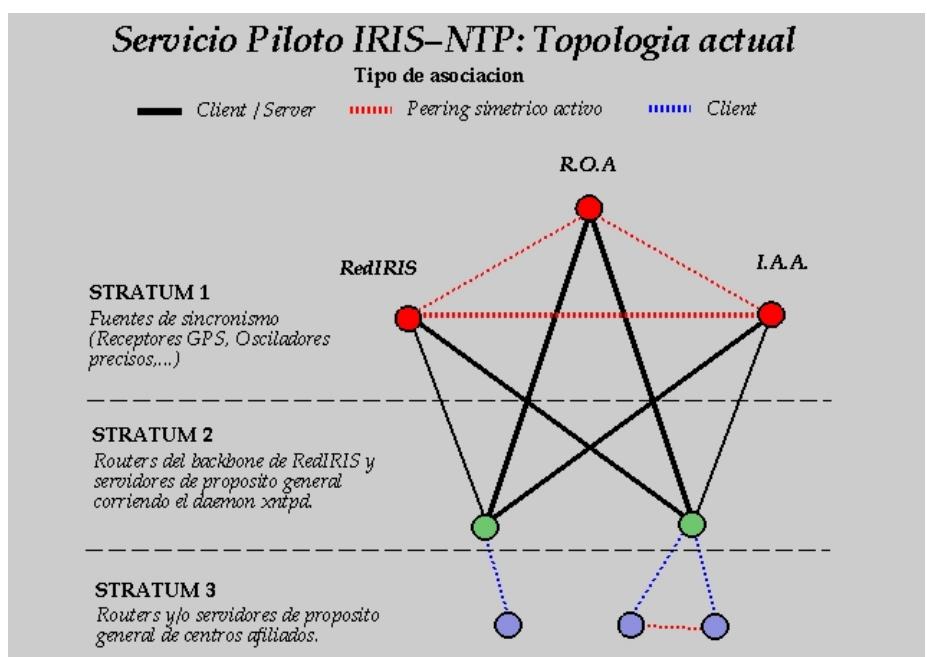
elección  
coordinador

## Protocolo de Tiempo de Red – NTP (para SD asíncronos y sincronización externa)

Servicio de sincronización de tiempo en Internet. Sincroniza a los clientes con UTC

Es un servicio fiable, redundante, reconfigurables si alguno cae, escalable y permite autenticación de las fuentes de tiempo

Este sistema conforma una jerarquía lógica **llamada subred de sincronización**



# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador



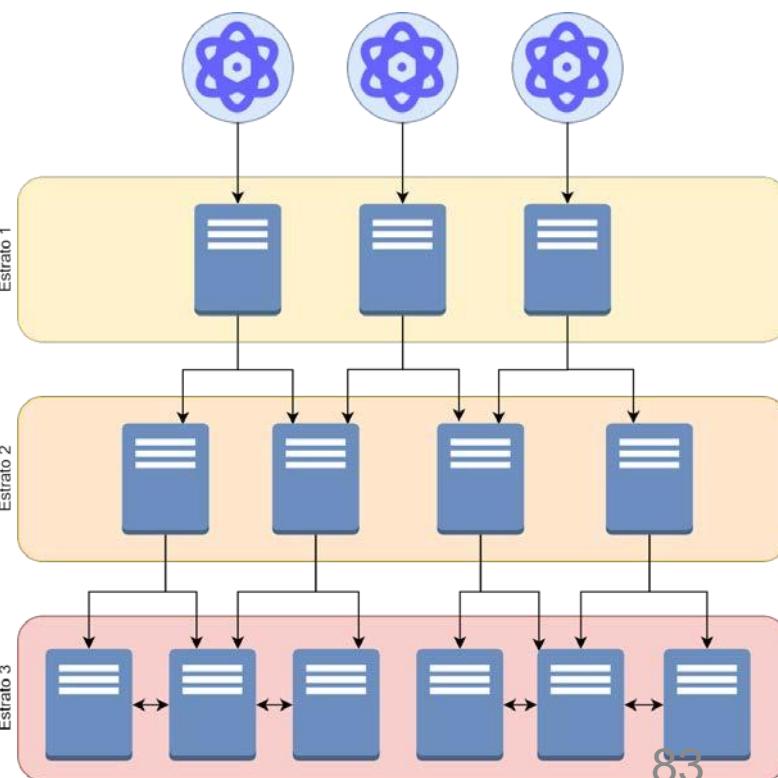
## Protocolo de Tiempo de Red – NTP (para SD asíncronos y sincronización externa)

Servicio de sincronización de tiempo en Internet. Sincroniza a los clientes con UTC

Es un servicio fiable, redundante, reconfigurables si alguno cae, escalable y permite autenticación de las fuentes de tiempo

Este sistema conforma una jerarquía lógica **llamada subred de sincronización**

- Los relojes UTC están en el estrato 0.
- Los servidores primarios ocupan el estrato 1 y están conectados directamente a fuentes UTC
- Los servidores secundarios están sincronizados directamente con los primarios
- Los servidores hoja o de nivel más bajo se ejecutan en las máquinas de trabajo de los usuarios



# sincronización de relojes

introducción

establecimiento  
de tiempo

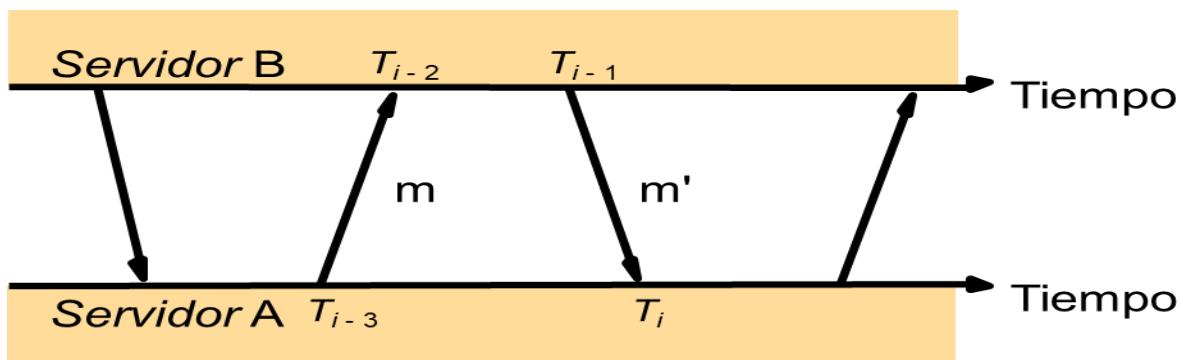
exclusión mutua

elección  
coordinador

## Protocolo de Tiempo de Red – NTP: Funcionamiento

Intercambio de mensajes entre pares de servidores

- Se usan mensajes UDP
- Cada mensaje lleva marcas de tiempo de los eventos recientes:
  - Tiempos locales de Envío y Recepción del mensaje anterior m
  - Tiempo local de envío del mensaje actual m'
- El receptor anota el tiempo local  $T_i$  cuando al recibir (o sea, tenemos  $T_{i-3}, T_{i-2}, T_{i-1}, T_i$ )
- Puede haber un retraso entre la llegada de un mensaje y el envío del siguiente y se pueden perder mensajes...
  - Pares de servidores se intercambian mensajes conteniendo información de tiempo
  - Usado en los casos en que se necesita muy alta precisión (p.ej. en primeros niveles)



# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Protocolo de Tiempo de Red – NTP: Precisión y calidad de los valores estimados

- Para cada par de mensajes entre servidores, NTP estima una compensación  $o$ , entre los dos relojes y un retardo  $d_i$  (tiempo total de transmisión para los dos mensajes  $t$  y  $t'$ )

$$T_{i-2} = T_{i-3} + t + o \text{ y } T_i = T_{i-1} + t' - o$$

- Sumando las ecuaciones:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

- Restando las ecuaciones:

$$o = o_i + (t' - t)/2 \text{ donde } o_i = (T_{i-2} - T_{i-3} - T_i + T_{i-1})/2$$

- Como  $t, t' > 0$  se puede ver que:

$$o_i - d_i/2 \leq o \leq o_i + d_i/2$$

por tanto  $o_i$  es una estimación de la deriva y  $d_i$  es una medida de la precisión

Los servidores NTP mantienen pares del tipo  $\langle o_i, d_i \rangle$ , estimando la fiabilidad de las variaciones y permitiendo cambiar el propio par

Ej.: precisión del orden de decenas de mseg. sobre Internet y de 1 mseg. sobre LAN

# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Protocolo de Tiempo de Red – NTP: Sincronización de servidores

- La subred de sincronización se puede reconfigurar si se produce un fallo:
  - un primario que pierde su conexión con UTC puede pasar a secundario
  - un secundario que pierde a su primario puede elegir otro primario
- Modos de sincronización:
  - Multidifusión (multicast)
    - En LAN de alta velocidad. Un servidor reparte el tiempo al resto que establecen su tiempo asumiendo un retraso de transmisión (no preciso)
  - Llamada a procedimiento
    - Similar a de Cristian. El servidor acepta peticiones. Precisión más alta
  - Simétrica
    - Pares de servidores se intercambian mensajes conteniendo información de tiempo
    - Usado en los casos en que se necesita muy alta precisión (p.e. en primeros niveles)

# sincronización de relojes

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Protocolo de Tiempo de Red – NTP: Ejemplo en Java de obtención de la fecha y hora

```
import java.io.*;
import java.net.*;
public class AskTime {

    public static void main(String args[]) throws Exception {

        String machine = "time-nw.nist.gov";
        final int daytimeport = 13;
        Socket so = new Socket(machine, daytimeport);
        BufferedReader br =
            new BufferedReader( new InputStreamReader( so.getInputStream() ) );
        String timestamp = br.readLine();
        System.out.println( "Según el servidor UTC:" + machine + " la fecha y hora es: " + timestamp );
    }
}
```

# relojes lógicos

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- No se sincronizan relojes físicos, si no que se pretende establecer una ordenación coherente de los eventos del sistema distribuido
  - Relación de orden parcial → “suceder antes”
- Se basan en el establecimiento de contadores software que garantizan que el comportamiento será siempre monótono creciente (alternativa a la corrección de un reloj)
- Estudiaremos dos tipos de relojes lógicos:
  - Reloj de Lamport
  - Reloj vectorial

# relojes de Lamport

introducción

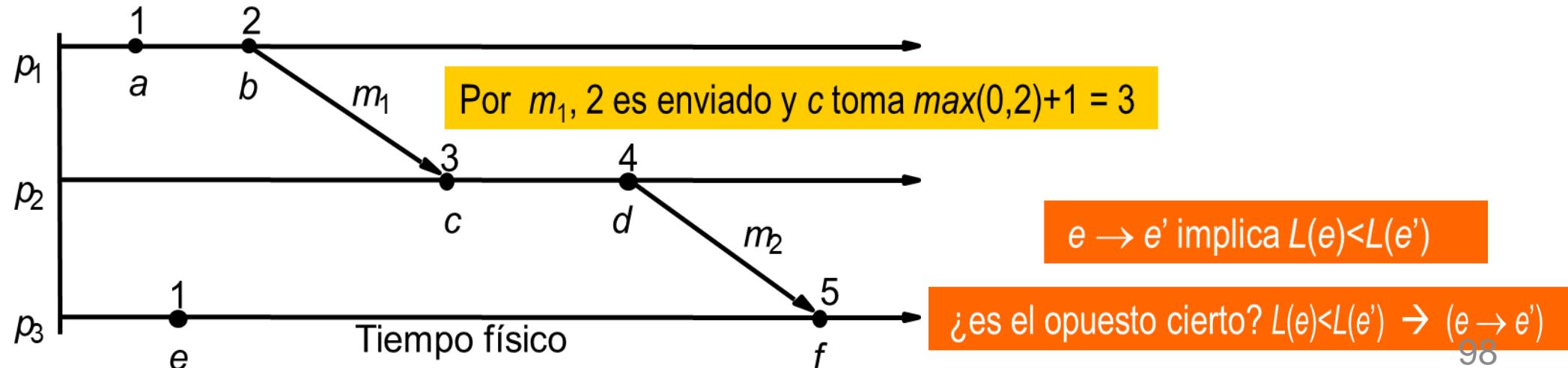
establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

Desarrollados por Lamport en 1978

- Cada proceso  $p_i$  tiene su reloj lógico ( $L_i$ ) inicializado a 0 que se utiliza para fijar las marcas temporales a los eventos según las siguientes reglas:
  - **R1:**  $L_i$  se incrementa en 1 antes de cada evento propio de  $p_i$
  - **R2:** cuando  $p_i$  envía ( $m$ ), adjunta al mensaje el valor propio  $t = L_i$
  - **R3:** cuando  $p_j$  recibe ( $m, t$ ) establece  $L_j := \max(L_j, t)$  y aplica **R1**, antes de establecer la nueva marca de tiempo



# relojes vectoriales

introducción

establecimiento  
de tiempo

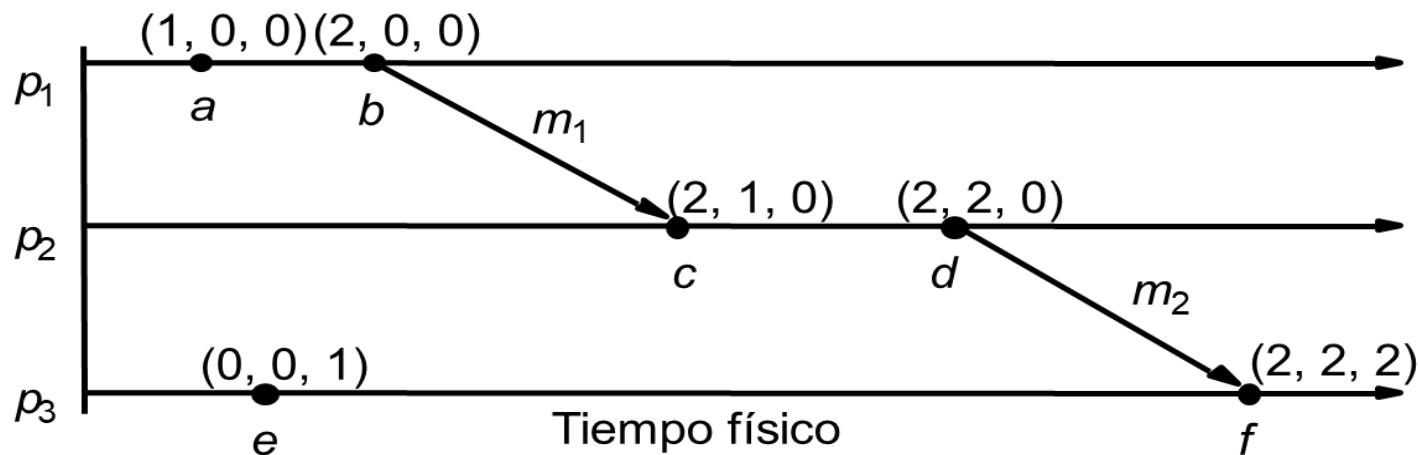
exclusión mutua

elección  
coordinador

Mattern y Fidge (1989-91) los desarrollan para superar la deficiencia de los relojes lógicos de Lamport:  **$L(e) < L(e')$  no implica  $e \rightarrow e'$**

- Cada proceso  $p_i$  tiene un vector  $V_i$  con N componentes, tantos como procesos en el sistema. Las reglas de actualización son las siguientes:

- **R1:** Antes de marcar un nuevo evento  $p_i$  incrementa  $V_i[i] := V_i[i] + 1$
- **R2:** cuando  $p_i$  envía ( $m$ ), adjunta al mensaje el vector propio  $t = V_i$
- **R3:** cuando  $p_j$  recibe ( $m, t$ ) establece establece  $V_i[j] := \max(V_i[j], t[j])$   $j = 1, 2, \dots, N$  y se aplica R1



$$\begin{aligned}V = V' &\Leftrightarrow V[j] = V'[j] \quad j=1, 2, \dots, N \\V \leq V' &\Leftrightarrow V[j] \leq V'[j] \quad j=1, 2, \dots, N \\V < V' &\Leftrightarrow V \leq V' \wedge V \neq V'\end{aligned}$$

- $V(e) < V(e')$  implica  $e \rightarrow e'$
- $c$  y  $e$  paralelos, ni  $V(c) \leq V(e)$  ni  $V(e) \leq V(c)$



Grado en Ingeniería Informática

## **Sistemas distribuidos**

# Sincronización de tiempo y coordinación

Coulouris: Disitributed systems Ch15

Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# coordinación distribuida: introducción

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Los procesos distribuidos necesitan a menudo coordinar sus actividades
  - sistemas síncronos y asíncronos
  - tolerancia a fallos
  - exclusión mutua de los procesos distribuidos
  - ejemplo: reservas de billetes de avión
- En los SD para solucionar el problema de la exclusión mutua, no se pueden utilizar:
  - ni variables compartidas
  - ni facilidades dadas por un único núcleo central

→ soluciones basadas en el **paso de mensajes**

# coordinación distribuida: introducción

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Dado un conjunto de procesos en un sistema distribuido se necesita:
  - Coordinar sus acciones
  - Llegar a un acuerdo en uno o más valores
- Formas de coordinación y acuerdo:
  - Acceso a recursos: exclusión mutua distribuida
  - Selección de valores: algoritmos de elección
  - Comunicación distribuida: algoritmos de multidifusión
  - Toma de decisiones: algoritmos de consenso

# algoritmos de exclusión mutua

---

introducción

establecimiento  
de tiempo

**exclusión mutua**

elección  
coordinador

Los algoritmos de exclusión mutua distribuida deben cumplir las siguientes propiedades:

- EM1: **Seguridad** → en todo momento, como máximo hay un solo proceso ejecutando la región crítica
- EM2: **Pervivencia** → a todo proceso que lo solicita se le concede la entrada/salida en la región crítica en algún momento:
  - Debe evitar el abrazo mortal (*deadlock*) y la inanición (*starvation*)
- EM3: **Ordenación** → la entrada en la región crítica debe concederse según la relación “suceder – antes”

# algoritmos de exclusión mutua

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo basado en servidor central

- El servidor central concede permisos en forma de ***token*** que concede acceso a la sección crítica (SC) y al salir de la SC, el proceso devuelve el ***token*** al servidor
- Suponiendo que no hay caídas y no se pierden mensajes:
  - se cumplen EM1 y EM2
  - EM3 está asegurada en el orden de llegada de los mensajes al servidor

## Problemas

- todas las solicitudes se envían al servidor (recurso crítico) → cuello de botella
- caída o fallo del servidor → elección de nuevo servidor → EM3 no asegurada
- caída o fallo de un proceso en la SC → se detecta con temporizadores → se requisa el *token*

## Complejidad del algoritmo

- 2 mensajes para la entrada en la SC
- 1 mensaje para salir de la SC

# algoritmos de exclusión mutua

introducción

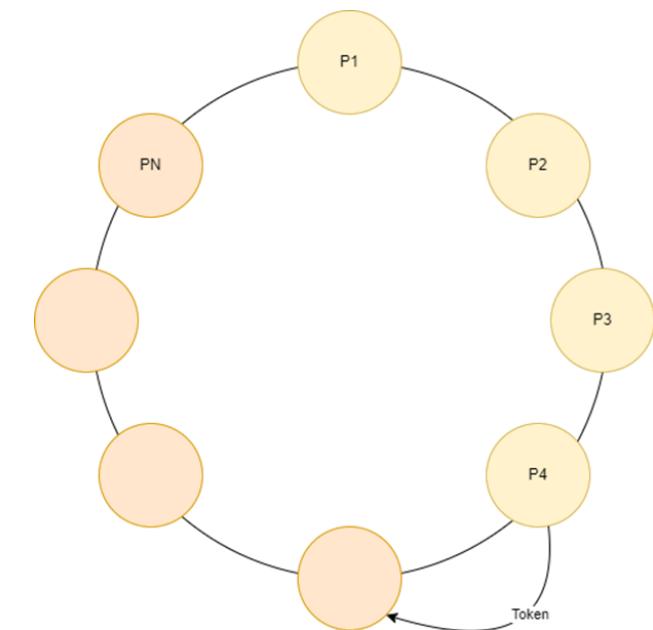
establecimiento  
de tiempo

**exclusión mutua**

elección  
coordinador

## Algoritmo del anillo

- Se da a cada proceso la dirección de su vecino
- El *token* siempre está circulando por el anillo
- Cuando un proceso recibe el *token*:
  - Si no quiere entrar en la sección crítica lo envía a su vecino
  - Si quiere entrar en la sección crítica lo retiene y al terminar lo devuelve al vecino
- Tolerancia a fallos
  - Pérdida del token: detección y regeneración (temporizadores)
  - Caída de un proceso del anillo: reconfiguración del anillo



# algoritmos de exclusión mutua

introducción

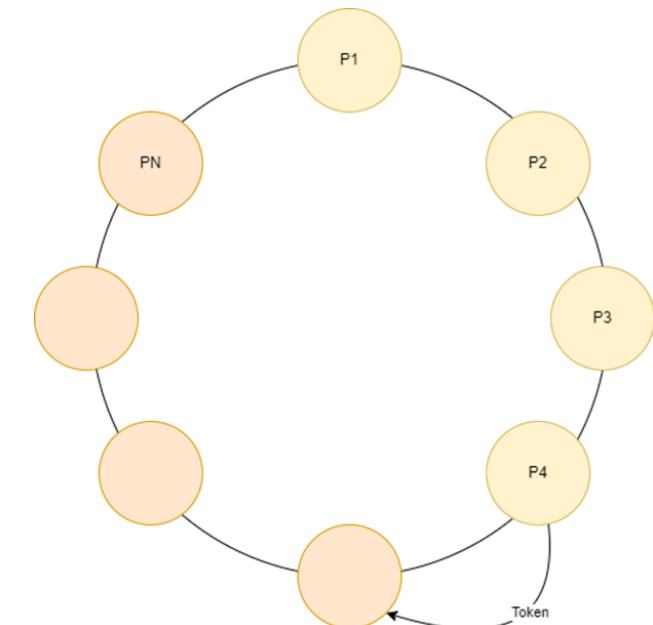
establecimiento  
de tiempo

**exclusión mutua**

elección  
coordinador

## Algoritmo del anillo

- Se da a cada proceso la dirección de su vecino.
- Problemas
  - se sobrecarga la red aun cuando ningún proceso quiera entrar en la SC
  - si un proceso cae → necesita reconfiguración
  - si además tenía el testigo → elección para regenerar el testigo
  - asegurarse de que el proceso ha caído → varios testigos
  - si se produce desconexión o ruptura de la red



# algoritmos de exclusión mutua

introducción

establecimiento  
de tiempo

**exclusión mutua**

elección  
coordinador

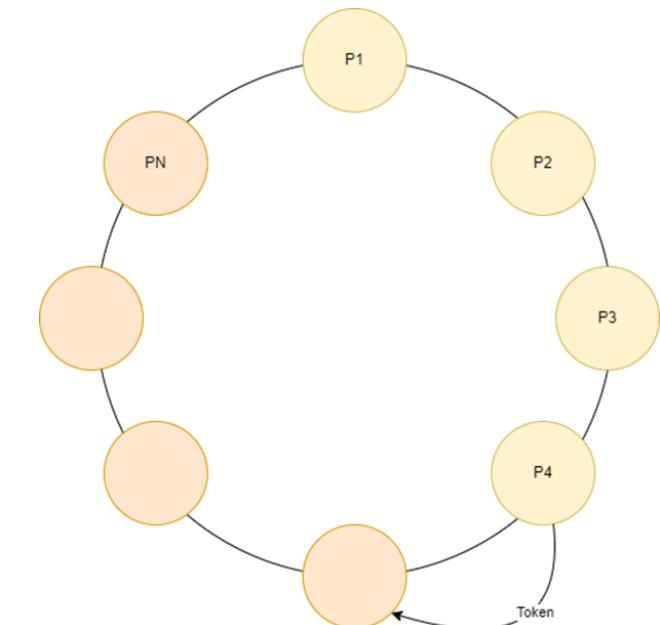
## Algoritmo del anillo

### Cumplimiento de propiedades

- EM1: Sí, el servidor se encarga de ello a través de un token único
- EM2: Sí, todas las peticiones se registran en la cola
- EM3: No, cada proceso recibe el *token* en el orden establecido por el anillo

### Complejidad del algoritmo

- Acceso a SC: entre 0 y N mensajes
- Acceso SC: entre 1 y  $(N - 1)$  mensajes



# algoritmos de exclusión mutua

introducción

establecimiento  
de tiempo

**exclusión mutua**

elección  
coordinador

## Algoritmo Ricart - Agrawala

- Algoritmo basado en relojes lógicos y multidifusión:
  - Cada proceso conoce la dirección de los demás procesos
  - Basado en relojes de Lamport (cada proceso posee un reloj lógico)
  - Algoritmo descentralizado: evita cuellos de botella
  - Pretende asegurar EM1, EM2 y EM3 (*¿lo consigue?*)
- Idea básica del algoritmo
  - Cuando un proceso quiere entrar en la sección crítica (SC) → les pregunta a los demás si puede entrar
  - Cuando todos los demás le contesten afirmativamente → entra
  - El acceso a la SC se obtiene a través de un **token**
- Cada proceso guarda el estado en relación a la SC: liberada, buscada o tomada
  - Cola de solicitudes en cada proceso
- Estructura de los mensajes → Tupla  $\langle T_i, P_i, SC_i \rangle$  (*en lo sucesivo nos referiremos solo una SC*)

# algoritmos de exclusión mutua

introducción

establecimiento  
de tiempo

exclusión mutua

elección y  
consenso

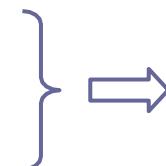
## Algoritmo Ricart – Agrawala (pseudocódigo)

### Inicialización

- estado SC := LIBERADA

### Para entrar en la sección crítica

- estado SC := BUSCADA
- Multitransmite petición a todos los procesos
- T := marca temporal de la petición propia
- espera hasta que número de respuestas recibidas =  $N - 1$
- estado SC := TOMADA;



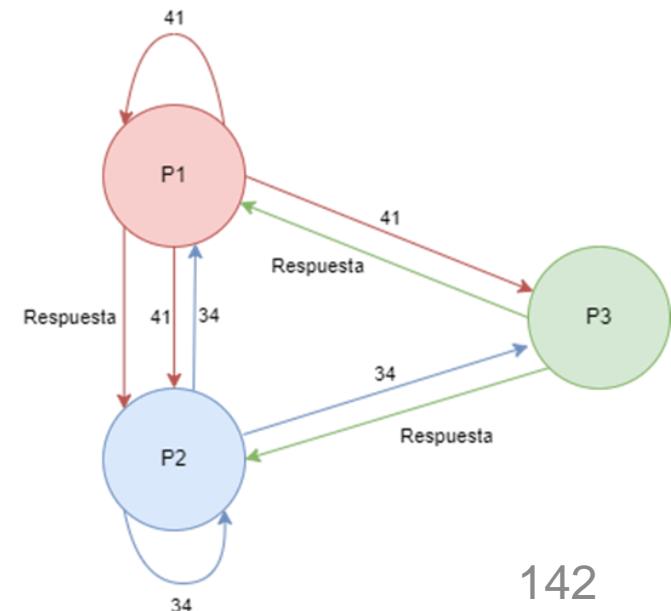
Si el proceso desea entrar en la SC,  
aplaza el procesamiento del resto de  
peticiones recibidas

### Al recibir una petición $\langle T_i, p_j \rangle$ en el proceso $p_i$ ( $i \neq j$ )

- **si** (estado = TOMADA o (estado = BUSCADA y  $(T_i, p_j) < (T_i, p_i)$ )) **entonces**  
    pone en la cola la petición por parte de  $p_i$  sin responder
- **sino**  
    responde inmediatamente a  $p_i$
- **fin si**

### Para salir de la sección crítica

- estado := LIBERADA
- responde a todas y cada una de las peticiones en la cola propia



# algoritmos de exclusión mutua

introducción

establecimiento  
de tiempo

**exclusión mutua**

elección y  
consenso

## Algoritmo Ricart - Agrawala

Complejidad del algoritmo

Número de mensajes necesarios para obtener el recurso:

- sin soporte multicast:  $2(n-1)$
- con soporte multicast: n

Problemas:

- Algoritmo más costoso que el del servidor central
- Pese a ser algoritmos distribuidos, el fallo de cualquier proceso bloquea el sistema
- Los procesos implicados reciben y procesan cada solicitud:
  - igual o peor congestión que el servidor central

# elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

- Procedimiento para elegir a un proceso dentro de un grupo para desempeñar un rol determinado
- Se activa tanto en la inicialización del SD, como si no se detecta la presencia de coordinador (¿caída?)
- Una vez elegido el coordinador, los demás miembros del grupo lo asumen
- *Ejemplo:* elección del “maestro” en el algoritmo de Berkeley de sincronización de relojes

Los algoritmos de elección deben cumplir las propiedades siguientes:

- E1 → Seguridad: Elección única de coordinador, aunque varios procesos inicien la elección
- E2 → Vivacidad: Si un proceso solicita coordinador, lo obtendrá en algún momento

Tres algoritmos:

- Algoritmo basado en anillo: Chang y Roberts (1979)
- Algoritmo del matón (*bully*): H. García-Molina (1982)
- Algoritmo de invitación: H. García-Molina (1982)

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo de elección basado en anillo

*Objetivo:* Elegir como coordinador al proceso con identificador más alto

Los procesos conforman un anillo lógico (cada uno conectado con su vecino)

- Inicialmente todos los procesos son no-candidatos → cualquiera puede empezar una elección:
  - (1) se marca como candidato y (2) envía mensaje de elección con su identificador al vecino
- Cuando un proceso recibe un mensaje de elección:
  - si identificador del mensaje es mayor que el suyo → lo reenvía a sus vecinos
  - si es menor:
    - si es no-candidato → (1) sustituye el identificador por el suyo, (2) lo envía al vecino y (3) se marca como candidato
    - si es el suyo → (1) se marca como **elegido** y (2) envía mensaje de **elegido** a su vecino añadiendo su identidad
- Cuando un proceso recibe un mensaje de elegido:
  - (1) se marca como no-candidato y (2) lo envía a su vecino

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo de elección basado en anillo

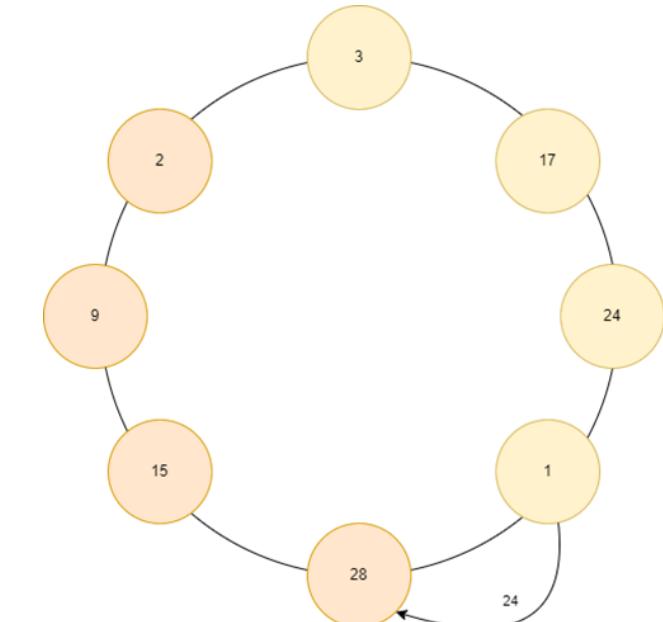
No detecta fallos, se supone procesos estables durante la elección

### Cumplimiento de propiedades

- E1: Sí, se garantiza que el proceso activo con identificador más alto
- E2: Sí, la última vuelta garantiza la propagación del nuevo coordinador a todos

### Complejidad del algoritmo

- peor caso: lanza elección sólo el vecino al futuro coordinador →  $3n-1$  mensajes
- mejor caso: lanza elección el futuro coordinador →  $2n$  mensajes



# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo de elección del **abusón** o **bully**

- Todos los procesos deben conocer el identificador (ID) y la dirección del resto de procesos
- Se presupone una comunicación fiable
- El algoritmo selecciona al proceso superviviente con **mayor ID / prioridad**
- Permite la caída de procesos durante la elección: utiliza **temporizadores** para detectar fallos de procesos

Existen 3 tipos de mensajes:

- Mensaje de **elección**: para anunciar una elección
- Mensaje de **respuesta**: respuesta a un mensaje de elección
- Mensaje de **coordinador**: para anunciar el ID del nuevo coordinador

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo de elección del **abusón** o **bully**

- Un proceso inicia una elección al darse cuenta de que el coordinador ha caído:
  - envía mensaje de elección a los procesos con ID mayor que el suyo
  - espera algún mensaje de respuesta:
    - si vence temporizador (sin respuesta) → el proceso se erige como coordinador y envía mensaje de coordinador a todos los procesos con IDs más bajos
  - si recibe alguna respuesta → espera mensaje de coordinador
    - si vence temporizador → lanza una nueva elección
- Si un proceso recibe un mensaje de elección:
  - contesta con un mensaje de respuesta y lanza una elección (si no ha lanzado ya antes una)
- Si un proceso recibe un mensaje de coordinador:
  - guarda el ID y trata a ese proceso como nuevo coordinador
- Cuando un proceso se reinicia:
  - lanza una elección a menos que sea el de ID más alto (en cuyo caso se erigiría como nuevo coordinador → **abusón**)

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Algoritmo de elección del **abusón** o **bully**

Cumplimiento de propiedades:

- E1: Sí, salvo partición de la red
- E2: Sí, en último extremo, el que detecta el fallo del coordinador, acaba siéndolo

Complejidad del algoritmo

- caso mejor: se da cuenta el segundo ID más alto →  $n - 2$  mensajes
- caso peor: se da cuenta el ID más bajo →  $O(n^2)$  mensajes

Discusión

¿Qué sentido puede tener que después de elegir nuevo coordinador, si se reinicia el de mayor ID se vuelva a asignar como coordinador?

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

exclusión mutua

elección  
coordinador

## Problemática de los algoritmos de elección anteriores

- Se basan en **temporizadores**: Retrasos de transmisión pueden causar la elección de múltiples líderes.
- La perdida de conexión entre dos grupos de procesadores puede aislar permanentemente los nodos (p. ej. la partición de una red formando grupos de nodos aislados)

Los algoritmos de **invitación** pueden solventar los problemas

## Características del algoritmo de **invitación**:

- Definición de grupos de procesadores con líder único
- Detección y agregación de grupos
- Reconocimiento por parte del líder de los miembros del grupo

# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

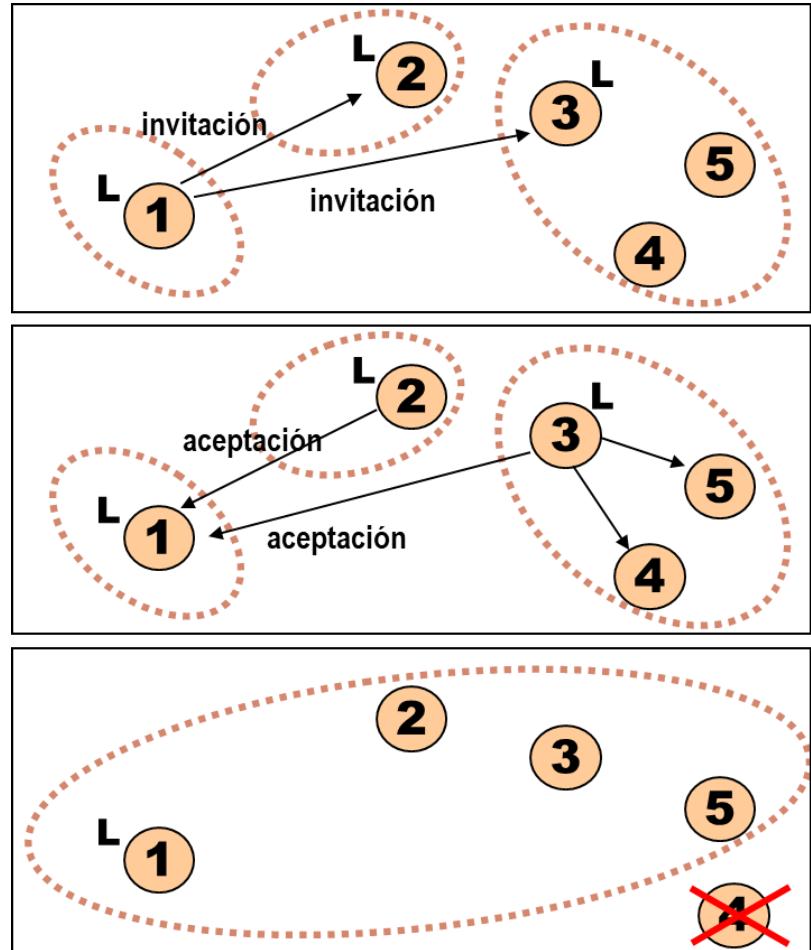
exclusión mutua

elección  
coordinador

## Algoritmo de elección por **invitación**

### Funcionamiento

- Si un procesador detecta la pérdida del líder, entonces se declara líder y forma su propio grupo
- Periódicamente el líder de cada grupo invita a líderes de otros grupos
- Dos grupos se unen por medio de mensajes de aceptación:
  - De forma explícita
  - Como respuesta a mensajes de invitación (a líderes más prioritarios)



# algoritmos de elección de coordinador

introducción

establecimiento  
de tiempo

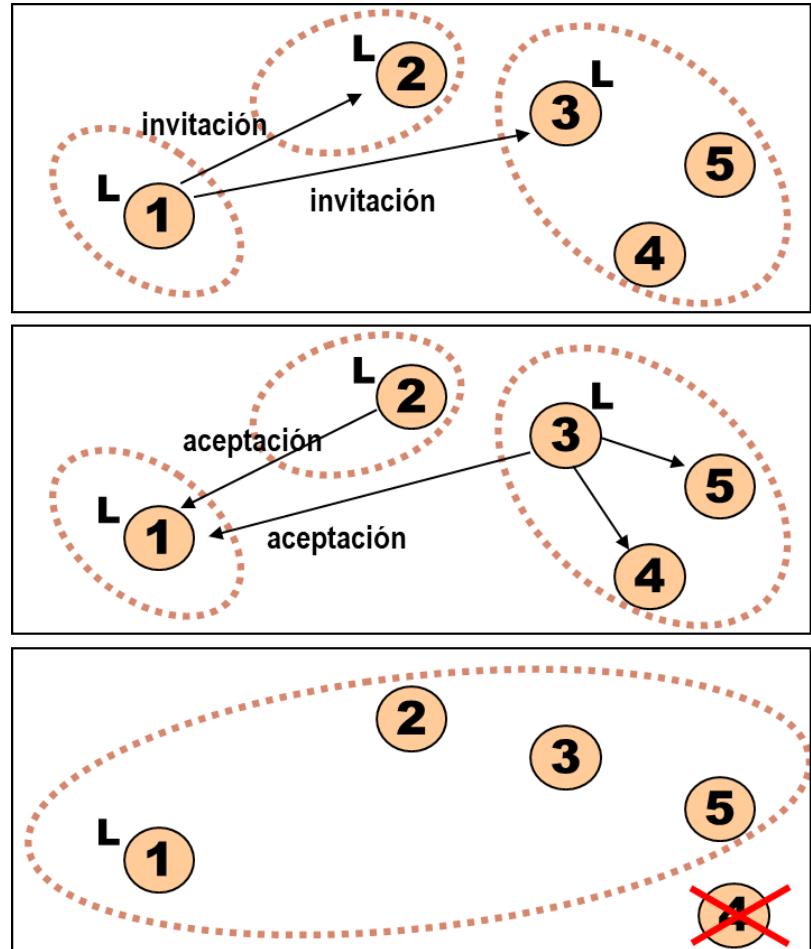
exclusión mutua

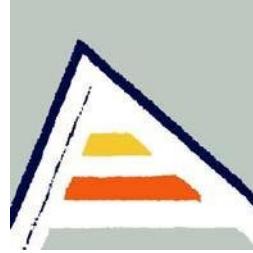
elección  
coordinador

## Algoritmo de elección por **invitación**

### Cumplimiento de propiedades

- E1: Sí, al final del proceso se tiende a la existencia de un solo líder (tras desparticionarse el sistema y volver la conectividad)
- E2: Sí, todos los grupos tienen líder (incluso los de un solo miembro)





Grado en Ingeniería Informática

## **Sistemas distribuidos**

# **Sistema de ficheros distribuido y DLT**

Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# Sistema de ficheros distribuido

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

El principal objetivo de un sistema distribuido es compartir recursos  
Compartir información (archivos) es quizás el mas relevante.

- Servidores con acceso restringido a datos almacenados internamente
  - Sub-net local
  - Internet

El diseño de sistemas de ficheros distribuidos a gran escala presentan problemas:

- Balanceo de carga
- Fiabilidad
- Disponibilidad
- Seguridad

Algunos de estos problemas son los que tiene que hacer frente sistemas como:

- P2P (nodos no disponibles siempre)
- Stream Media Content servers (Distribución “multicast” tiempo real)

# Sistema de ficheros distribuido

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Sub-net locales se busca la distribución actualizada de

- Persistencia de datos:
  - nextCloud, Dropbox, SVN, GIT, NFS, ...
- Programas y/o servicios
  - Overleaf, collabora office, onlyoffice, office365, ...

Se busca emular la funcionalidad de un sistema no-distribuido

- Los cliente usan maquinas “tontas”

Nótese la diferencia en requerimientos con los servidores de contenido multimedia:

- Bandwidth
- Tiempo real.

# Sistema de ficheros (centralizados)

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Introducción (recuerdos de SO):

Los sistemas de archivos fueron desarrollados para sistemas centralizados como:

- Servidores
- Ordenadores personales (desktop or laptop)

Estos proveen una serie de servicios:

- API de programación
- Control de Acceso
- Locking (compartición)

## introducción

## modelos DFS

## implementación DFS

## Distributed Ledger Tech (DLT)

# Conceptos básicos

El SO es el software encargo de la gestión del SAS.  
Dispone de un módulo que realiza un segundo nivel de gestión de dispositivos periféricos:

Abstacta las propiedades físicas de los dispositivos de almacenamiento  
*hardware interfaces*

Proporciona una interfaz a los usuarios de acceso a la información  
*user space interfaces*

Administra el espacio libre  
*virtual filesystem*

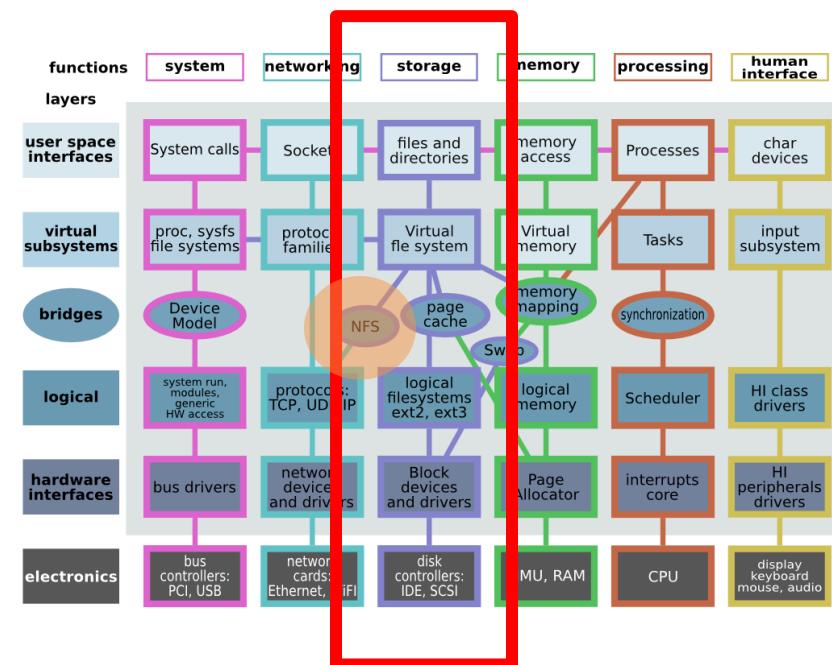
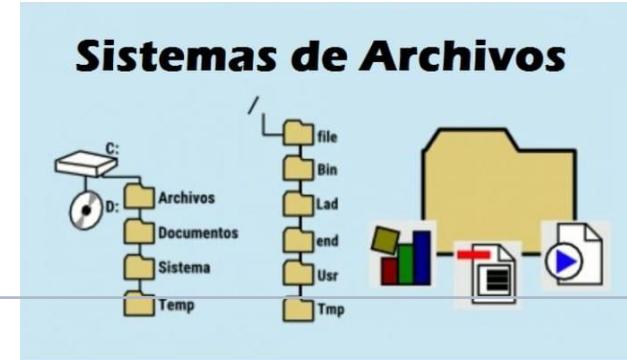
Realiza la asignación del almacenamiento  
*logical filesystem*

El sistema de archivos está formado por:

Colección de archivos o ficheros.  
Cada uno de los cuales contiene datos relacionados.

Estructura de directorios.  
Organiza todos los archivos del sistema  
Proporciona información sobre ellos.

Particiones.  
Permite separar grandes colecciones de directorios.



# Interfaz con el Sistema de Archivos

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Fichero: Atributos

*Nombre del archivo:*

En algunos sistemas operativos pueden existir más de uno, distinguen entre mayúsculas y minúsculas.

*Tipo de archivo:*

Información necesaria en sistemas que soportan más de un tipo.

*Ubicación:*

Información sobre el dispositivo y bloques de memoria secundaria que le dan soporte físico al archivo.

*Tamaño:*

El número de bytes de la información que contiene el archivo.

*Propietario:*

Identificador del creador del archivo.

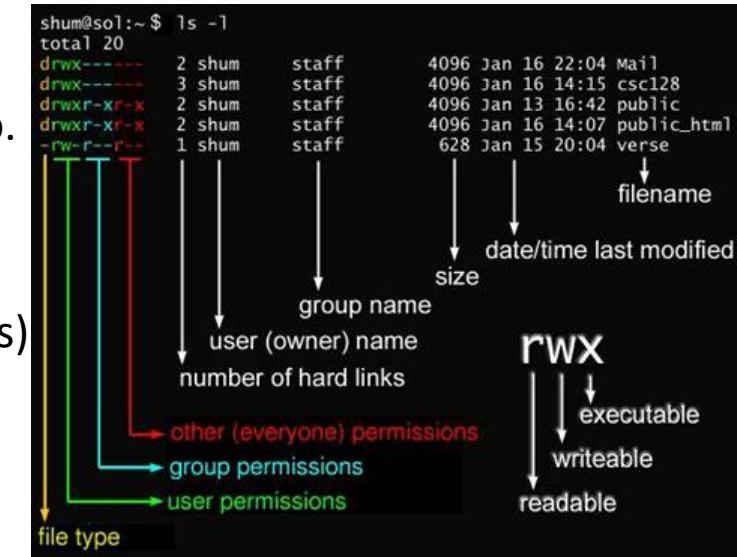
*Protección:*

Información sobre quien pueden acceder al archivo (permisos)

*Fechas de creación, último acceso, última modificación*

*Información:*

Es el contenido del archivo.



# Interfaz con el Sistema de Archivos

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Fichero: Operaciones

Crear archive / Truncar archivo:

*fd=creat (filename, mode)*

Leer archivo:

*read (fd, buffer, nbytes)*

Rebobinado de archivo:

*lseek (fd, offset, where)*

Borrar archivo:

*unlink (filename)*

Escribir en archivo:

*write (fd, buffer, nbytes)*

Obtener atributos:

*fd=fstat (filename, starbuf)*

Utilizando y combinando las llamadas al sistema básicas, se pueden realizar otras operaciones más complejas:

*Copiar un archivo (cp)*

*Concatenar archivos (cat)*

*Renombrar archivos (mv)*



# Interfaz con el Sistema de Archivos

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Ficheros: tipos de fichero

Para cada SO se puede establecer una clasificación de los archivos desde el punto de vista funcional. Por ejemplo, para UNIX se distinguen los siguientes archivos:

### Archivos ordinarios o regulares.

Son archivos que contienen información introducida por un usuario:  
programa de aplicación / utilidad del sistema.

### Archivos directorios.

Contienen una lista de nombres de archivo y punteros a **nodos-i** asociados.

Son archivos **ordinarios especiales**

tienen unos privilegios especiales de protección; solo el sistema de archivos puede escribir en ellos.

### Archivos especiales.

Usados para acceder a los dispositivos. Se pueden distinguir dos tipos:

**Caracteres.** Permiten modelar dispositivos orientados a caracteres (impresoras, redes)

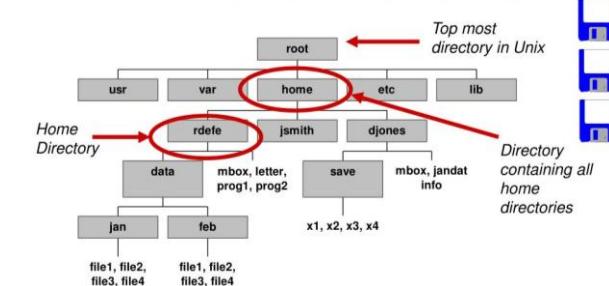
**Bloques.** Permiten modelar dispositivos orientados a bloques (discos, cd-rom, etc.)

### Pseudoarchivos.

Son archivos que modelan los mecanismos de UNIX para la comunicación entre procesos:

**tuberías**

**sockets**



```
/dev# ls -l | grep ^l
 3 Jul 18 10:47 cdrom -> sr0
 3 Jul 18 10:47 cdrw -> sr0
 11 Jul 18 10:45 core -> /proc/kcore
 3 Jul 18 10:47 dvd -> sr0
 3 Jul 18 10:47 dvdrw -> sr0
 13 Jul 18 10:45 fd -> /proc/self/fd
 12 Jul 18 10:45 initctl -> /run/initctl
 28 Jul 18 10:45 log -> /run/systemd/journal

 4 Jul 18 10:47 rtc -> rtc0
 15 Jul 18 10:45 stderr -> /proc/self/fd/2
 15 Jul 18 10:45 stdin -> /proc/self/fd/0
 15 Jul 18 10:45 stdout -> /proc/self/fd/1
/dev#
```

# Sistemas de almacenamiento

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Figure 12.1 Storage systems and their properties

	<i>Sharing</i>	<i>Persistence</i>	<i>Distributed cache/replicas</i>	<i>Consistency maintenance</i>	<i>Example</i>
Main memory	✗	✗	✗	1	RAM
File system	✗	✓	✗	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	✗	Web server
Distributed shared memory	✓	✗	✓	✓	Ivy (DSM, Ch. 6)
Remote objects (RMI/ORB)	✓	✗	✗	1	CORBA
Persistent object store	✓	✓	✗	1	CORBA Persistent State Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)



EXTFS/UFS

NFS, AFS

Cache/proxy

JAVA

Khazana

Block-Chain

Types of consistency:

1: strict one-copy ✓: slightly weaker guarantees 2: considerably weaker guarantees

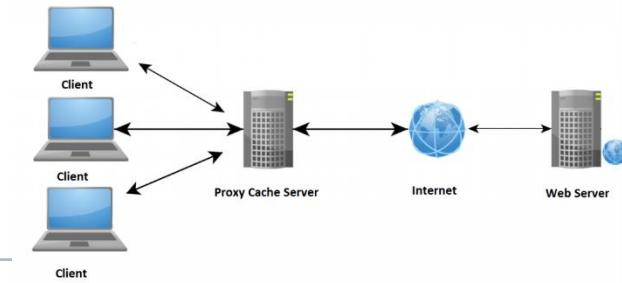
# Sistemas de almacenamiento

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

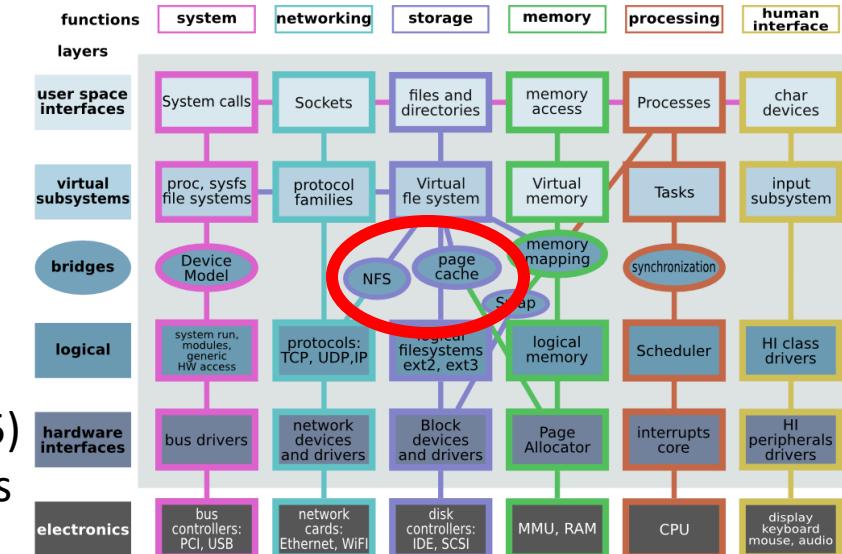


**Consistencia** es el mecanismo para mantener sincronizadas múltiples copias de un mismo dato cuando hay una actualización en alguna de ellas.

- El problema esta en la parte de cache (acceleration)

## En sistemas distribuidos

- Notificar un cambio no es instantaneo
- Consistencia estricta es mas dificil de lograr.
- AFS y NFS solo envía porciones de los archivos.
  - Manejo similar a SVN o GIT
- En servidores WEB es bajo demanda (RO)
  - Solo cuando se pide por parte del cliente (F5)
  - No es posible realizar acciones cooperativas
- En servicios basados en CORBA
  - Solo se permite una copia obtenida bajo demanda (lock)
- En servicios basados en Khazana
  - Se permiten multiples replicas
  - Mecanismo relajado de consistencia similar a Distributed Shared Memory.



# sistema de ficheros distribuido

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## ¿Qué es un sistema de ficheros distribuido (DFS)?

- Tiene las mismas funciones que el sistema de ficheros de un SO convencional, pero más complejo
- Los usuarios y los dispositivos de almacenamiento se encuentran dispersos por la red
- Se accede a los datos y se procesan como si estuviesen almacenados en local.
- Un DFS debe permitir:
  - Compartir información remotamente, de forma controlada y autorizada
  - Movilidad de los usuarios
  - Disponibilidad
  - Tolerancia a fallos

# arquitectura de un DFS

introducción

modelos DFS

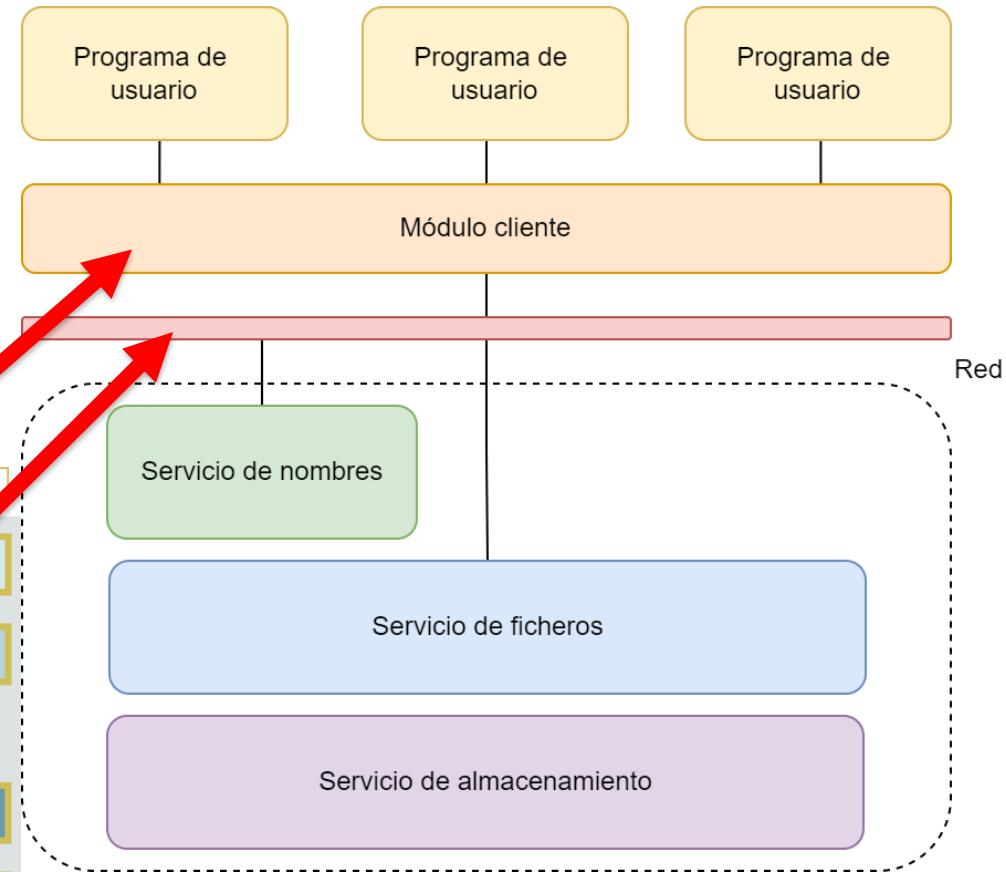
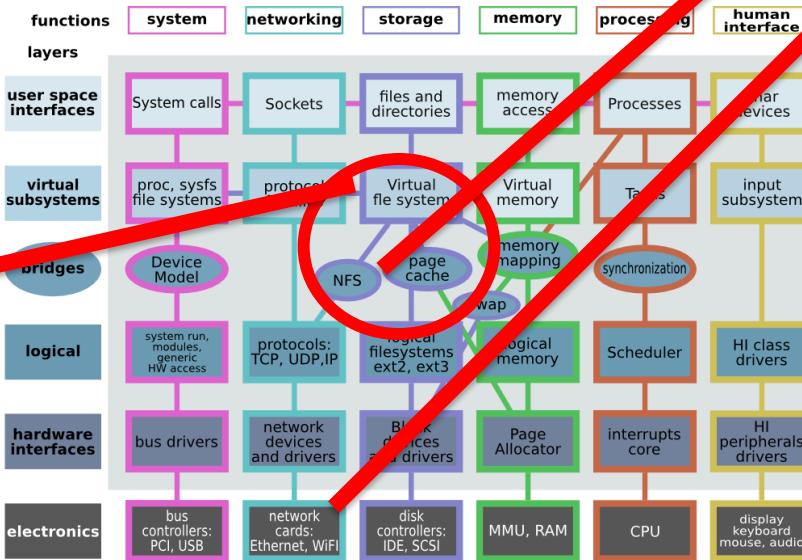
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

```
shum@shum:~$ ls -l
total 20
drwxr-x--- 2 shum staff 4096 Jan 16 14:15 Mail
drwxr-xr-x 2 shum staff 4096 Jan 13 16:42 csc128
drwxr-xr-x 2 shum staff 618 Jan 15 20:04 verse
-rw-r--r-- 1 shum staff 4096 Jan 16 14:07 public_html
               filename
               size
               date/time last modified
               user (owner) name
               group name
               number of hard links
               rwx
               executable
               writeable
               readable
               file type
               other (everyone) permissions
               group permissions
               user permissions
```

## Componentes de un DFS:

- Servicio de almacenamiento
- Servicio de ficheros
- Servicio de nombres o de directorio
- Módulo cliente: biblioteca de clases o API



# Requerimientos de diseño de un DFS

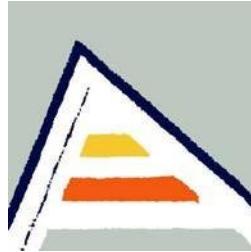
introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

- Transparencia
  - de **acceso**: los programas no tienen que notar si es local o remoto el archivo al que acceden.
  - de **ubicación**: los programas deben de ver el sistema de archivos de manera uniforme, incluso con realocatación.
  - de **movilidad**: las tablas de contenidos (directorios) no deben modificarse cuando se modifican/mueven en remoto.
  - de **prestaciones**, el rendimiento debe de ser aceptable incluso con una congestión alta del servicio.
  - de **escala**: el servicio debe tolerar la ampliación para dar soporte a nuevas realidades de red o de clientes.
- Movilidad de los usuarios: acceso mediante distintos dispositivos y protocolos (net-fs, web-services, ...)
- Rendimiento: DFS debe ofrecer un rendimiento similar a un sistema centralizado
- Alta disponibilidad: Replicación de archivos para realizar balanceo de carga entre servidores
- Tolerancia a fallos: Solucionar problemas de comunicación en red, servidor transaccional
- Conurrencia: Cambios en los archivos realizados por un usuario no deben afectar a otros.
- Fiabilidad e integridad de la información: propagación de los cambios en un sistema con replicación
- Seguridad: control de acceso basados en ACL, certificados y encriptación de cada transacción
- Heterogeneidad del HW's y SO's: debe funcionar en cualquier OS



Grado en Ingeniería Informática

## **Sistemas distribuidos**

# **Sistema de ficheros distribuido y DLT**

Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# Modelos de acceso DFS

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Existen dos modelos básicos:

- modelo de **caché de datos**
  - los ficheros se acceden de forma local
  - aumento del rendimiento
  - problemas de consistencia
- modelo de **servicio remoto**
  - las operaciones se realizan en los servidores
  - problemas de eficiencia

Se pueden combinar ambos modelos

- ejemplos: NFS, Dropbox, OneDrive, ...

# Técnicas de implementación DFS

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Los modelos anteriores se implementan:

- **Caché**: asociada a los clientes y orientada a estructuras de datos a nivel de bloque
- **Replicación de ficheros**: asociada a los servidores y orientada a estructuras de datos a nivel de fichero completo

# Técnicas basadas en caché

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Localización:

- Localización de la caché **en el disco del servidor**
  - No se aconseja la caché en disco → **demasiado costoso**
- Localización de la caché **en la memoria del servidor**
  - Ventajas: (1) **transparente para los clientes**; (2) los SO tradicionales suelen utilizar este esquema (NFS)
  - Inconvenientes: (1) **coste de las transferencias por la red**
- Localización de la caché **en el disco del cliente**
  - Ventajas: (1) **fiabilidad**; (2) permite gran tamaño de la caché
  - Inconveniente: (1) hay que acceder a disco
- Localización de la caché **en la memoria del cliente**
  - Ventajas: (1) se eliminan los costes de transmisión por la red y acceso a disco; (2) máximo rendimiento en caso de acierto
  - Inconvenientes: (1) problema de **inconsistencia de la información** en la caché

# Técnicas basadas en caché

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Tratamiento de **inconsistencias y propagación** de modificaciones:

- considerar cómo verificar la validez de los datos en la caché
- considerar cuándo propagar al fichero original del servidor las modificaciones de caché

El contenido de una caché se vuelve **inválido** cuando otro cliente modifica la copia principal:

- es necesario comprobar si la caché de un cliente es consistente respecto a la copia principal
- si no es consistente → es necesario invalidar la caché y actualizar los datos
- dos estrategias básicas: (1) **iniciada por el cliente** e (2) **iniciada por el servidor**

Existen tres esquemas básicos para propagar las modificaciones:

- escritura inmediata (*write-through*)
- escritura retrasada (*delayed-write*)
- escritura al cerrar (*write-on-close*)

# Técnicas de implementación DFS

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Los modelos anteriores se implementan:

- **Caché**: asociada a los clientes y orientada a estructuras de datos a nivel de bloque
- **Replicación de ficheros**: asociada a los servidores y orientada a estructuras de datos a nivel de fichero completo

# Técnicas basadas en replicación de ficheros

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

La **replicación de ficheros** consiste en la existencia de varias copias, cada una ubicada en un servidor diferente

La replicación de ficheros aporta las siguientes ventajas:

- aumenta la disponibilidad y la escalabilidad
- aumenta la fiabilidad
- mejora el tiempo de respuesta y el rendimiento (throughput)
- permite trabajar en modo de operación desconectada

Cuestiones a resolver:

- que la replicación sea **transparente** a los usuarios (¿denominación de réplicas?)
- cómo **actualizar** las réplicas en el caso de modificaciones en una de ellas:  
(1) explícita; (2) implícita: replicación perezosa y comunicación a grupos

# Técnicas basadas en replicación de ficheros

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Protocolos de actualización de réplicas

- Replicación de **sólo lectura**
  - se aplica a ficheros inmutables
- Protocolo **escribir en todos - leer de cualquiera**
  - las escrituras son costosas
  - problemas si un servidor cae
- Protocolo basado en la **disponibilidad de copias**
  - problemas de inconsistencias en caso de partición de la red
- Protocolo de **copia primaria**
  - se escribe en una y se lee de cualquiera
  - problemas si cae el servidor primario
- Protocolo basado en **quorum**
  - los clientes requieren un quorum de  $N_r$  servidores para leer y  $N_w$  para escribir
    - donde  $N_r + N_w > N$  ( $N$  = número de réplicas)

# Ejemplos de sistemas ficheros distribuidos

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Nombre	Info
9P	
Amazon S3	
<b>Andrew File System (AFS)</b>	Escalable e independiente de localización. Usa cache cliente y Kerberos para autenticación.
OpenAFS	
AvereOS	
DCE Distributed File System (DCE/DFS)	Similar a AFS. Se centra en full POSIX file system semantics y high availability.
File Access Listener (FAL)	
Magma	
MapR FS	
Microsoft Office Groove shared workspace	
NetWare Core Protocol (NCP)	
<b>Network File System (NFS)</b>	Estandar en UNIX-based networks. NFS usa Kerberos authentication y client cache.
Self-certifying File System (SFS)	global network file system diseñado para permitir el acceso de forma segura a dominios con separación de privilegios administrativos.
<b>Server Message Block (SMB)</b>	Original de IBM, pero la version mas famosa es de Microsoft conocida tambien como Common Internet File System (CIFS). SMB usa Kerberos authentication.

## introducción

## modelos DFS

## implementación DFS

## Distributed Ledger Tech (DLT)

Client	Written in	License	Access API	High availability	Shards	Efficient Redundancy	Redundancy Granularity	Initial release year	Memory requirements (GB)
Alluxio (Virtual Distributed File System)	Java	Apache License 2.0	HDFS, FUSE, HTTP/REST, S3	hot standby	No	Replication <sup>[1]</sup>	File <sup>[2]</sup>	2013	
Ceph	C++	LGPL	librados (C, C++, Python, Ruby), S3, Swift, FUSE	Yes	Yes	Pluggable erasure codes <sup>[3]</sup>	Pool <sup>[4]</sup>	2010	1 per TB of storage
Coda	C	GPL	C	Yes	Yes	Replication	Volume <sup>[5]</sup>	1987	
GlusterFS	C	GPLv3	libglusterfs, FUSE, NFS, SMB, Swift, libgfapi	mirror	Yes	Reed-Solomon <sup>[6]</sup>	Volume <sup>[7]</sup>	2005	
HDFS	Java	Apache License 2.0	Java and C client, HTTP, FUSE <sup>[8]</sup>	transparent master failover	No	Reed-Solomon <sup>[9]</sup>	File <sup>[10]</sup>	2005	
IPFS	Go	Apache 2.0 or MIT	HTTP gateway <sup>[11]</sup> , FUSE <sup>[12]</sup> , Go client <sup>[13]</sup> , Javascript client <sup>[14]</sup> , command line tool <sup>[15]</sup>	Yes	with IPFS Cluster <sup>[16]</sup>	Replication <sup>[11]</sup>	Block <sup>[17]</sup>	2015 <sup>[18]</sup>	
JuiceFS ↗	Go	Apache License 2.0	POSIX, FUSE, HDFS, S3	Yes	Yes	Reed-Solomon	Object	2021	
Kerish-DFS ↗	Go	GPLv3	HTTP(REST), CLI, C# Client, Go Client	Yes		Replication		2020	
LizardFS	C++	GPLv3	POSIX, FUSE, NFS, Ganesh, Ceph FSAL (via libcephfs)	master	No	Reed-Solomon <sup>[19]</sup>	File <sup>[16]</sup>	2013	
Lustre	C	GPLv2	POSIX, NFS-Ganesha, NFS, SMB	Yes	Yes	No redundancy <sup>[10][17]</sup>	No redundancy <sup>[18][19]</sup>	2003	
MinIO	Go	AGPL3.0	AWS S3 API, FTP, SFTP	Yes	Yes	Reed-Solomon <sup>[20]</sup>	Object <sup>[21]</sup>	2014	
MooseFS	C	GPLv2	POSIX, FUSE	master	No	Replication <sup>[22]</sup>	File <sup>[23]</sup>	2008	
OpenAFS	C	IBM Public License	Virtual file system, Installable File System			Replication	Volume <sup>[24]</sup>	2000 <sup>[25]</sup>	
OpenIO <sup>[26]</sup>	C	AGPLv3 / GPLv3	Native (Python, C, Java), HTTP/REST, S3, Swift, FUSE (POSIX, NFS, SMB, FTP)	Yes		Pluggable erasure codes <sup>[27]</sup>	Object <sup>[28]</sup>	2015	0.5
Ori <sup>[29]</sup>	C, C++	MIT	libori, FUSE C++ client, FUSE (C++ server: MetaServer and ChunkServer are both in C++)			Replication	Filesystem <sup>[30]</sup>	2012	
Quantcast File System	C	Apache License 2.0	FUSE, SMB, NFS, key/value	master	No	Reed-Solomon <sup>[31]</sup>	File <sup>[32]</sup>	2012	
RozoFS	C, Python	GPLv2	HTTP (REST), POSIX, FUSE, S3, HDFS	Yes		Mojette <sup>[33]</sup>	Volume <sup>[34]</sup>	2011 <sup>[35]</sup>	
SeaweedFS ↗	Go, Java	Apache License 2.0	HTTP (REST), POSIX, FUSE, S3, HDFS	requires CockroachDB, undocumented config		Reed-Solomon <sup>[36]</sup>	Volume <sup>[37]</sup>	2015	
Storj ↗	Go	Apache License 2.0, After General Public License v3	HTTP (REST), S3, Native (Go, C, Python, Java)	Yes		Reed-Solomon <sup>[38]</sup>	Object <sup>[39]</sup>	2018	
Tahoe-LAFS	Python	GNU GPL <sup>[39]</sup>	HTTP (browser or CLI), SFTP, FTP, FUSE, via SSHFS, pyfilesystem			Reed-Solomon <sup>[40]</sup>	File <sup>[41]</sup>	2007	
XtreemFS	Java, C++	BSD License	libxtreemfs (Java, C++), FUSE			Replication <sup>[42]</sup>	File <sup>[43]</sup>	2009	

# sistemas ficheros distribuidos

Name	Run by	Access API
Amazon S3	Amazon.com	HTTP (REST/SOAP)
Google Cloud Storage	Google	HTTP (REST)
SWIFT (part of OpenStack)	Rackspace, Hewlett-Packard, others	HTTP (REST)
Microsoft Azure	Microsoft	HTTP (REST)
IBM Cloud Object Storage	IBM (formerly Cleversafe) <sup>[48]</sup>	HTTP (REST)

Client	Written in	License	Access API
BeeGFS	C / C++	FRAUNHOFER FS (FhGFS) EULA, <sup>[44]</sup> GPLv2 client	POSIX
ObjectiveFS <sup>[45]</sup>	C	Proprietary	POSIX, FUSE
Spectrum Scale (GPFS)	C, C++	Proprietary	POSIX, NFS, SMB, Swift, S3, HDFS
MapR-FS	C, C++	Proprietary	POSIX, NFS, FUSE, S3, HDFS, CLI
PanFS ↗	C, C++	Proprietary	DirectFlow, POSIX, NFS, SMB/CIFS, HTTP, CLI
Infini <sup>[46]</sup>	C++	Proprietary (to be open sourced) <sup>[47]</sup>	FUSE, Installable File System, NFS/SMB, POSIX, CLI, SDK (libinfini)
Isilon OneFS	C/C++	Proprietary	POSIX, NFS, SMB/CIFS, HDFS, HTTP, FTP, SWIFT Object, CLI, Rest API
Qumulo	C/C++	Proprietary	POSIX, NFS, SMB/CIFS, CLI, S3, Rest API
Scality	C	Proprietary	FUSE, NFS, REST, AWS S3
Quobyte ↗	Java, C++	Proprietary	POSIX, FUSE, NFS, SMB/CIFS, HDFS, AWS S3, TensorFlow Plugin, CLI, Rest API

# Sistemas ficheros distribuidos

## AFS: Historia

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

El Sistema AFS proporciona un sistema de archivos distribuido que pudiese escalar para proporcionar acceso a archivos a un gran número de clientes a través de un campus universitario.

- AFS comparte algunas de las consideraciones de diseño de NFS:
  - Soportar la semántica del sistema de ficheros UNIX
  - las aplicaciones no necesitaran ser modificadas.
  - Almacena en caché de los clientes los datos de los archivos de forma agresiva
  - Confía en las llamadas al servidor para mantener la consistencia de la caché.
  - Soporta un espacio de nombres:
    - Compartido
    - Independiente de la ubicación
    - Un único directorio raíz - **los clientes no montan sistemas de archivos individualmente.**
  - Autenticación mediante Kerberos.

# Sistemas ficheros distribuidos

## AFS: File Caching

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Cuando un cliente AFS abre un fichero por primera vez, el gestor de caché del cliente copia los datos del archivo desde el servidor a una caché de disco local.

- AFS almacena en caché
  - Datos de ficheros (en trozos de 64 KB)
    - Las primeras versiones de AFS en versiones de AFS necesitaban copiar el fichero entero
    - La transferencia del archivo retrasaba el acceso de la aplicación a los datos del archivo
  - Atributos de ficheros
  - Directorios
  - Enlaces simbólicos.
- Una vez que un archivo localmente, después de abrir un archivo en el caché, el cliente puede leer y escribir el archivo sin comunicación con el servidor.
- Los cambios en un archivo almacenado en caché no se transmiten al servidor hasta después de que se cierra el archivo.
- El cliente sólo necesita contactar con el servidor para escribir los cambios.
  - Si varios clientes actualizan simultáneamente copias en caché del mismo archivo, los cambios del último cliente sobrescribirán los cambios de los otros clientes.

# Sistemas ficheros distribuidos

## AFS: File Caching

introducción

modelos DFS

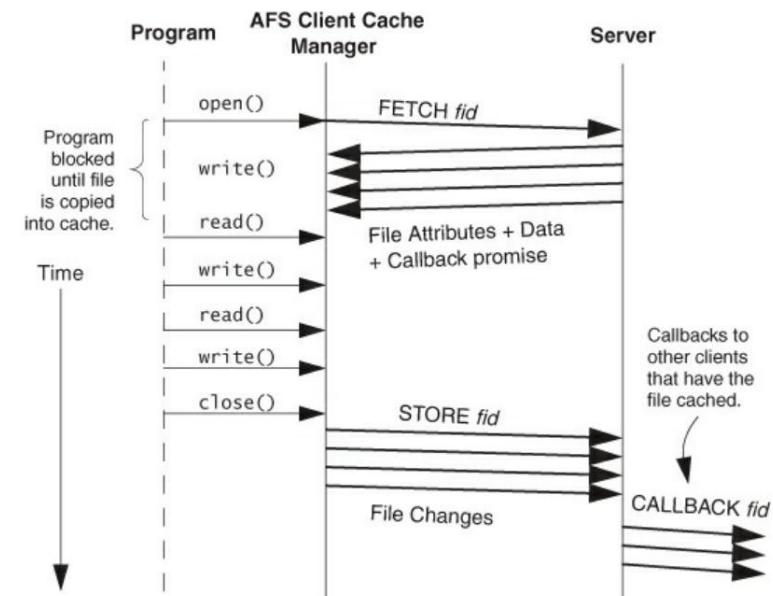
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Cuando se almacenan datos en caché del servidor, el gestor de caché del cliente obtiene una promesa de callback del servidor.

Si un archivo o directorio cambia en el servidor, el servidor enviará un mensaje a cada cliente con copia en caché, indicando que los datos en caché ya no son válidos.

El mecanismo permite al cliente operar con datos almacenados en caché durante largos períodos de tiempo sin contactar con el servidor.



# Sistemas ficheros distribuidos

## AFS: Fault tolerance and Reliability

introducción

modelos DFS

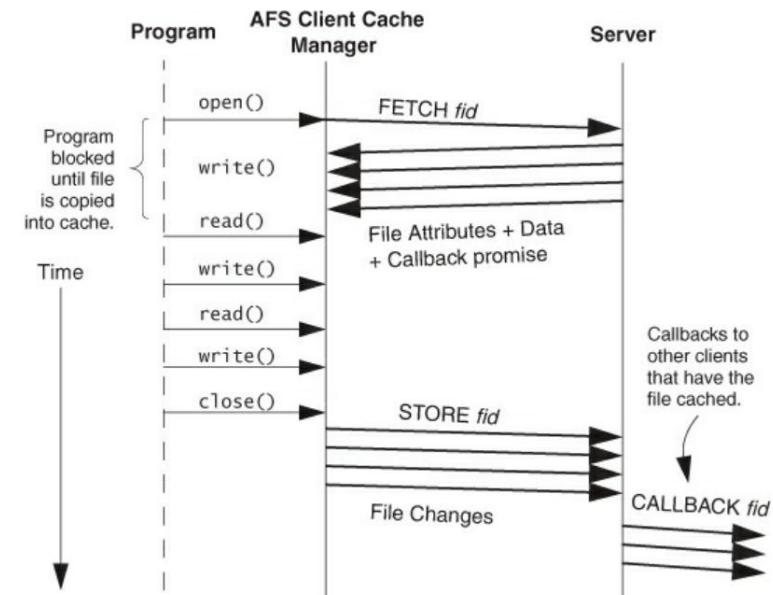
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Es posible que durante un problema de red impida que los mensajes lleguen a uno o más clientes, dejando al cliente trabajando con datos obsoletos almacenados en caché.

El cliente limita este periodo de posible incoherencia enviando un mensaje de sondeo al servidor cada 10 minutos. La respuesta al cliente permite restablecer la coherencia de la caché.

Como el servidor AFS necesita registrar el estado del cliente, se mantiene este estado en almacenamiento estable para que no se pierda si el servidor se bloquea y se reinicia.



# Sistemas ficheros distribuidos

## AFS: Alta disponibilidad

introducción

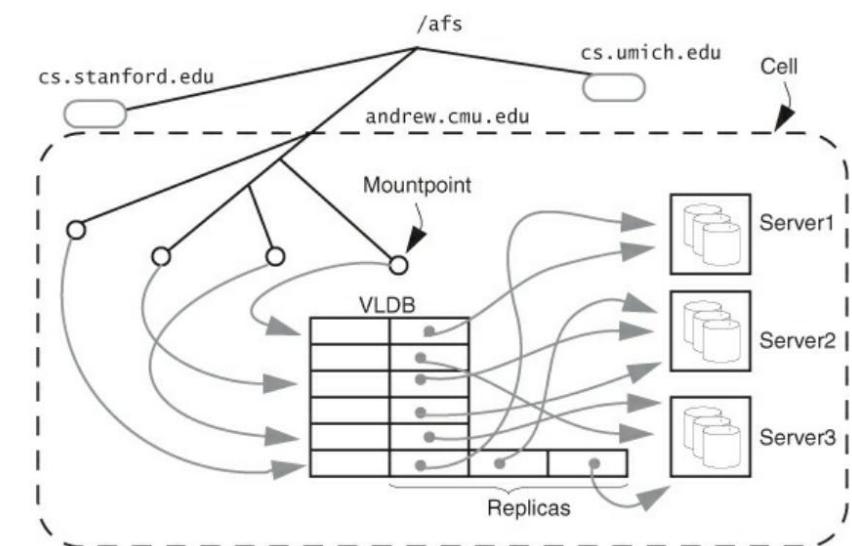
modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Dado que los clientes acceden a los datos de forma independiente de la ubicación a través del espacio de nombres compartido, los administradores del sistema son libres de reubicar los volúmenes de almacenamiento de un servidor a otro mientras los volúmenes están en uso.

La independencia de la ubicación facilita la replicación de sistemas de archivos de sólo lectura, lo que permite distribuir la carga del servidor y aumentar la disponibilidad de los datos.



# Sistemas ficheros distribuidos

## SMB/CIFS: Historia

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Microsoft desarrolló el protocolo SMB, posteriormente renombrado CIFS, como una forma de compartir recursos en una red.

- Inicialmente, se desarrolló en IBM con el objetivo de convertir el acceso a archivos locales de la interfaz de programación de aplicaciones (API) de DOS en un sistema de archivos en red.
- Más tarde, Microsoft añadió muchos cambios al protocolo y lo fusionó con el protocolo LAN Manager.
- Microsoft siguió añadiendo características al protocolo cuando se lanzó con Windows para grupos de trabajo.
- SMB se diseñó originalmente para ejecutarse sobre la API NetBIOS, pero desde Windows 2000 se ejecuta por defecto sobre TCP.

# Sistemas ficheros distribuidos

## SMB/CIFS: Evolución

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

- SMB 1.0 era muy "ruidosa", hacía un uso ineficiente de los recursos de red.
- SMB 2.0 menos "ruidosa", mejora del rendimiento.
  - Incluía soporte para enlaces simbólicos,
  - Almacenamiento en caché de las propiedades de los archivos
  - Mejoras escalabilidad
    - aumentando el número de usuarios
    - recursos compartidos
    - archivos abiertos por servidor.
- SMB 2.1 fue una actualización menor del protocolo.
- SMB 3.0 añadió nuevas funcionalidades (seguridad) y mejoró el rendimiento general.
  - Emular un BDC de Windows NT.
  - Crear un servidor miembro o un servidor autónomo
  - No es posible emular un controlador de dominio de Active Directory.
- SMB 4.0 emular el funcionamiento de un controlador de dominio de Active Directory.
  - Un servidor LDAP.
  - Un servidor Kerberos.
  - Un servidor DNS dinámico.
  - Llamadas a procedimientos remotos (RPC).

# Sistemas ficheros distribuidos

## SMB/CIFS: Architecture

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Según el papel que desempeñe un ordenador en la red, podemos dividirlos en diferentes grupos.

La clasificación más sencilla sería dividir los ordenadores entre:

- Servidores: proporcionan recursos compartidos al resto de ordenadores
- Clientes: ordenadores independientes
- Grupo de trabajo: una red con cierto numero de ordenadores independientes donde cada uno de ellos gestiona sus propios usuarios locales.
- Servidor de dominio: centraliza la gestión los usuarios localmente en cada ordenador.

Si estamos trabajando en:

- un grupo de trabajo: el servidor Samba será un servidor independiente.
- un dominio: el servidor Samba será un servidor miembro del dominio.
  - Podrá obtener la lista de usuarios del dominio
  - No mantendrá una copia local de los usuarios del dominio.

Por último, Samba puede ejercer de controlador de dominio usando Active Directory.

# Sistemas ficheros distribuidos

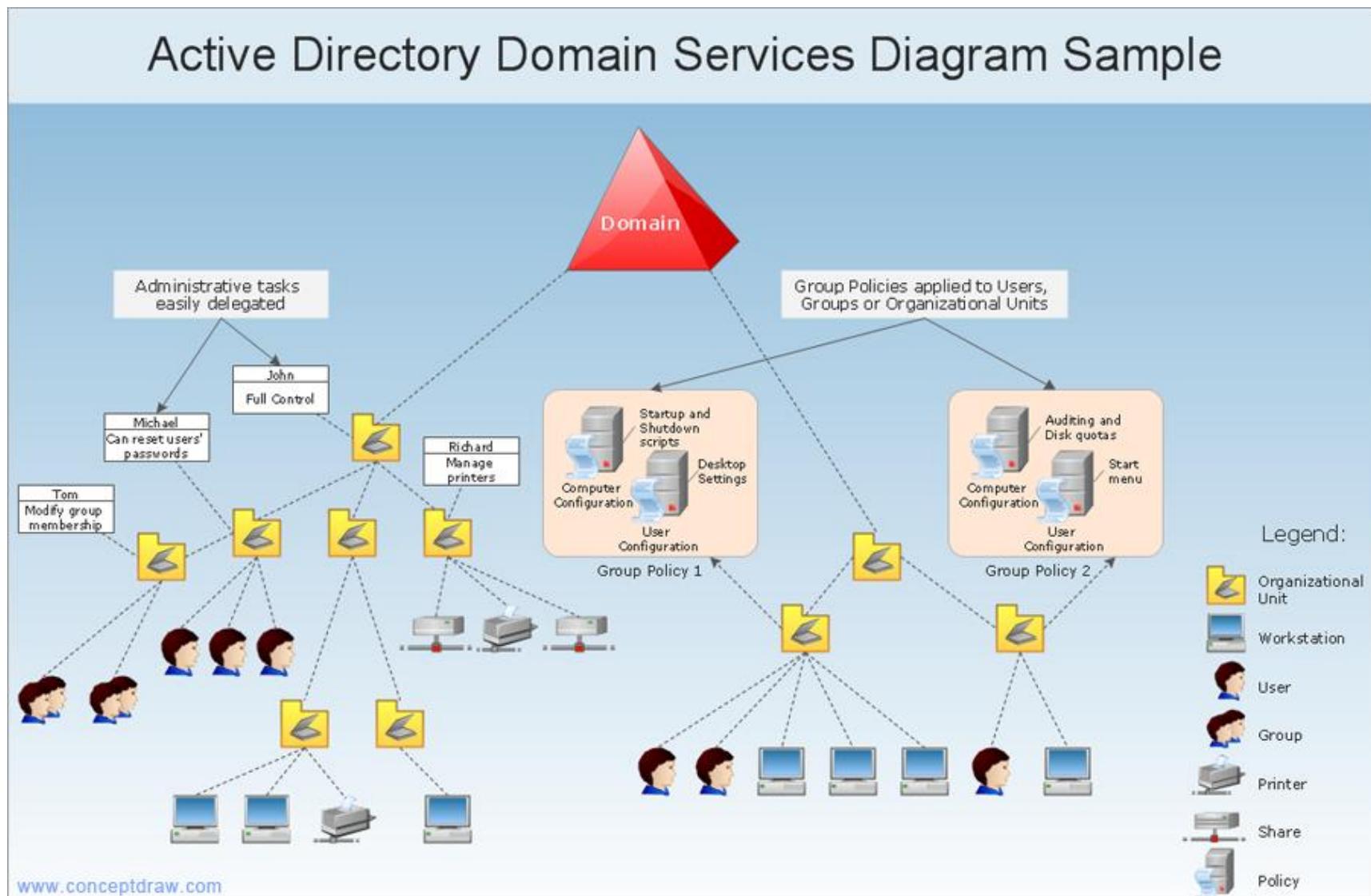
## SMB/CIFS: Architecture

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)



# Sistemas ficheros distribuidos

## SMB/CIFS: Características

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Los programas DOS y Windows son únicos en su asociación de bloqueos de archivos con todos los accesos a archivos.

- Cuando un programa DOS o Windows abre o crea un archivo, debe especificar:
  - el tipo de acceso que necesita (lectura o escritura),
  - "modo de denegación": el tipo de acceso que de otros programas mientras esté abierto
- Además, el protocolo CIFS admite el bloqueo de rangos de bytes dentro de un archivo.
  - Se reconocen bloqueos:
    - compartidos (no exclusivos)
    - Exclusivos

Un cliente CIFS que desee almacenar en caché los datos de un archivo debe solicitar un bloqueo (oplock) cuando se abre el archivo.

- Se concederá un oplock exclusivo si ningún otro cliente tiene el archivo abierto.
- Si otro cliente ya tiene el archivo abierto:
  - no se concederá el oplock exclusivo
  - el cliente realizará las operaciones de forma sincrónica con el servidor.

# Sistemas ficheros distribuidos

## SMB/CIFS: Características

introducción

modelos DFS

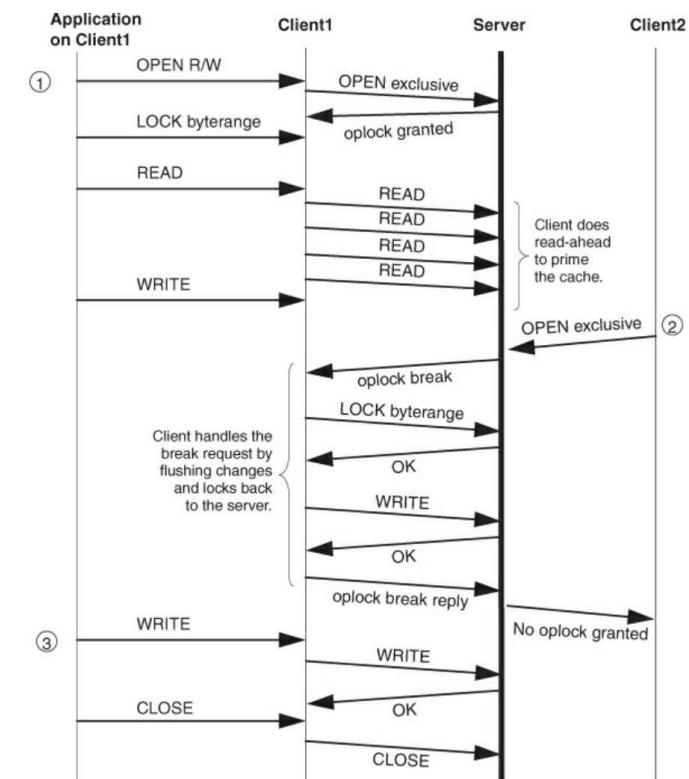
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Mientras se mantiene el oplock, las aplicaciones en el cliente pueden leer datos de la caché, cambiar el archivo o aplicar bloqueos.

Si otro cliente intenta abrir el archivo mientras se mantiene el oplock:

- El servidor enviará un callback de ruptura de oplock para romper el oplock.
  - Si el cliente ya no necesita almacenar el archivo en caché
    - Responderá cerrando el archivo.
  - Si el fichero aún está en uso, el cliente debe reenviar:
    - el bloqueo de fichero al servidor
    - así como cualquier escritura pendiente en la caché.
    - El servidor concede acceso al fichero a otro cliente.



# Sistemas ficheros distribuidos

## SMB/CIFS: Pitfalls

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

In 2016, **Badlock** (assigned [CVE-2016-2118](#), rated **Critical**) was disclosed to Windows and Samba, where the substitution augmentation, modification, and redefinition (SAMR) and local security authority domain (LSAD) protocols could be abused for man-in-the-middle (MiTM) attacks.

In 2017, a remote code execution gap was found in Samba and named **EternalRed** or SambaCry (assigned [CVE-2017-7494](#), rated **Important**), which affected all versions since 3.5.0. [NamPoHyu](#) was amongst the ransomware families that exploited this gap.

In 2020, a proof of concept (PoC) for a Netlogon vulnerability called **Zerologon** (assigned [CVE-2020-1472](#), rated **Critical**) was identified. The flaw allowed an attacker to elevate privileges by establishing a vulnerable Netlogon secure channel connection to a domain controller using the Netlogon Remote Protocol (MS-NRPC). Federal agencies using the software were **ordered** to install the patches released in August 2020.

[CVE-2021-44142](#) is a vulnerability that allows remote attackers to execute arbitrary code on affected installations of Samba. The specific gap exists in the parsing of the EA metadata in the server daemon *smbd* when opening a file. An attacker can abuse this vulnerability to execute code in the root context even without authentication.

# Sistemas ficheros distribuidos

## NFS

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

El protocolo NFS permite nombrar los archivos mediante una secuencia de nombres que forman una ruta.

El protocolo tiene cuidado de no requerir que los nombres de ruta sean soportados como entidades dentro del propio protocolo.

Un nombre de ruta se evalúa con una secuencia de peticiones LOOKUP.

La evaluación componente a componente hace innecesario que el protocolo reserve un carácter para **separar los componentes de un nombre de ruta.**



- In Win32 namespace: any UTF-16 code unit (case-insensitive) except \.\*"?<>| as well as NUL<sup>[8]</sup>
- In POSIX namespace: any UTF-16 code unit (case-sensitive) except / as well as NUL
- Trailing spaces are not allowed and will be removed<sup>[9]</sup>



introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

# Sistemas ficheros distribuidos

## NFS



IT'S OPEN

By Swapnil Bhartiya

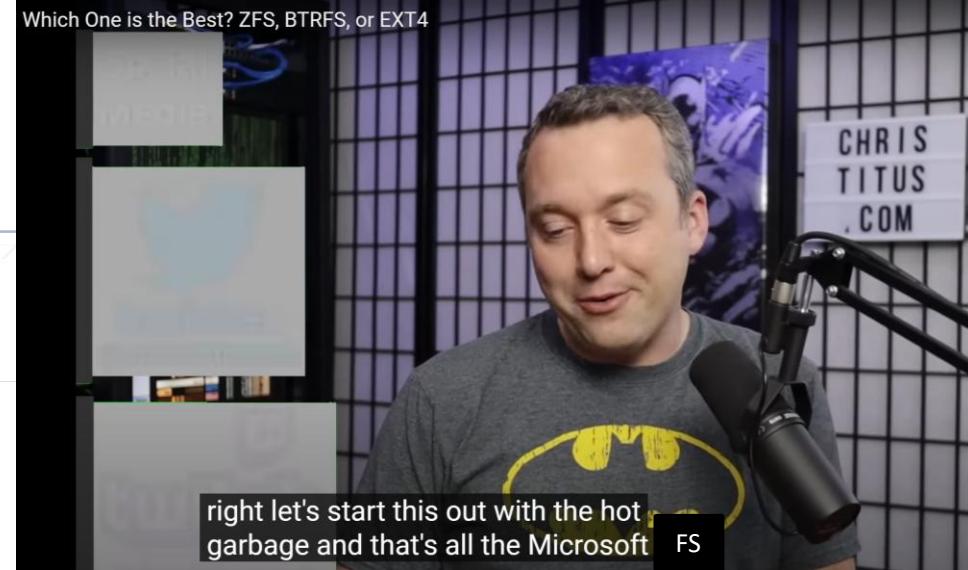
FOLLOW ★ Advisor

## Linus Torvalds: Apple's HFS+ is probably the worst file system ever



Linus Torvalds at LinuxCon NA, 2014 Credit: Swapnil Bhartiya

It's been a long time since we heard a good rant from Linus Torvalds. He doesn't rant much, but when he does he hits the nail on the head and he doesn't mince words.



Which One is the Best? ZFS, BTRFS, or EXT4

right let's start this out with the hot garbage and that's all the Microsoft FS

### MORE GOOD READS



11 technologies that tick off Linus Torvalds

In December 2014, a critical vulnerability was found in Git that affected Mac OS X and Windows users. Even if the vulnerability didn't affect Linux users, it could harm users who work on Windows or Mac systems.

A patch was released immediately, but that wasn't soon enough to keep Linus from yelling out loud about how horrible Apple's HFS+ file system is. As an Apple user, and as much as I love their hardware, I am not a fan of their software.

[ Next read: [11 technologies that tick off Linus Torvalds](#) ]

So what's the basic problem with HFS+? Both NTFS and HFS+ are case insensitive, which means if you have a folder named 'Linux' or 'linux' they will treat them as the same folder, which understandably causes a lot of problems.

# Sistemas ficheros distribuidos

## NFS

The true horrors of HFS+ are not in how it's not a great filesystem, but in how it's actively designed to be a bad filesystem by people who thought they had good ideas.

The case insensitivity is just a horribly bad idea, and Apple could have pushed fixing it. They didn't. Instead, they doubled down on a bad idea, and actively extended it – very very badly – to unicode. And it's not even UTF-8, it's UCS2 I think.

Ok, so NTFS did some of the same. But apple really took it to the next level with HFS+.

There's some excuse for case insensitivity in a legacy model ("We didn't know better"). But people who think unicode equivalency comparisons are a good idea in a filesystem shouldn't be allowed to play in that space. Give them some paste, and let them sit in a corner eating it. They'll be happy, and they won't be messing up your system.

And then picking NFD normalization – and making it visible, and actively converting correct unicode into that absolutely horrible format, that's just inexcusable. Even the people who think normalization is a good thing admit that NFD is a bad format, and certainly not for data exchange. It's not even "paste-eater" quality thinking. It's actually actively corrupting user data. By design. Christ.

And Apple let these monkeys work on their filesystem? Seriously?

There are lots of good reasons to not move to ZFS (cough-Oracle-cough), but they could have pushed people to case-sensitive HFS+, which would have then made it much easier to (in the long run) migrate to anything else saner. But no. There is a case sensitive option, but Apple actively hides it and doesn't support it.

The stupidity, it burns.

So you had all these people who made really bad decisions and actively coded for them. And I find that kind of "we actively implement shit" much more distasteful than just the "ok, we don't implement a lot of clever things" that John complained about.

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

# Sistemas ficheros distribuidos

## NFS: Continúan los problemas (OS independent)

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

DOS admite sólo un pequeño conjunto de atributos de archivo en comparación con el conjunto POSIX de aproximadamente 13 atributos de archivo proporcionados en la estructura por la solicitud GETATTR.

- Algunos de los atributos de NFS, como el tamaño del archivo y el tiempo m, tienen equivalentes en DOS.
- Otros no tienen ningún mapeo o tienen un mapeo indirecto.

El protocolo NFS maneja el contenido del archivo como una secuencia de bytes no interpretada. **La insensibilidad de NFS al contenido del archivo es una característica deseable para la mayoría de los formatos de archivo.** Sin embargo, en el caso de los archivos de texto ASCII, existen algunas diferencias menores en la representación del carácter de fin de línea que pueden tener efectos inesperados.

- DOS utiliza dos caracteres: retorno de carro y nueva línea <CR><LF>.
- UNIX utiliza sólo el carácter de nueva línea <LF>
- MAC utilizan sólo el carácter de retorno de carro <CR>.

# Sistemas ficheros distribuidos

## NFS: Continúan los problemas (OS independent)

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

El protocolo NFS admite enlaces simbólicos de estilo UNIX.

**Un enlace simbólico es un archivo especial que contiene sólo una ruta de acceso.**

- Cualquier proceso que intente acceder al enlace simbólico como un archivo o directorio será redirigido automáticamente al destino de la ruta en el enlace simbólico.
- El nombre de ruta en un enlace simbólico puede ser cualquier nombre de ruta que sea válido en el servidor.
- Se evaluará un nombre de ruta relativo en relación con el directorio que contiene el enlace simbólico.

Dado que los enlaces simbólicos UNIX no son compatibles con los sistemas operativos de PC como Windows, DOS u OS/2, el software cliente PC-NFS debe evaluar por sí mismo cualquier enlace simbólico que se encuentre en el servidor.

- Para hacer esto, debe seguir la convención de nombre de ruta de UNIX:
  - Forward Slash
  - Big names
  - Case sensitive

# Sistemas ficheros distribuidos

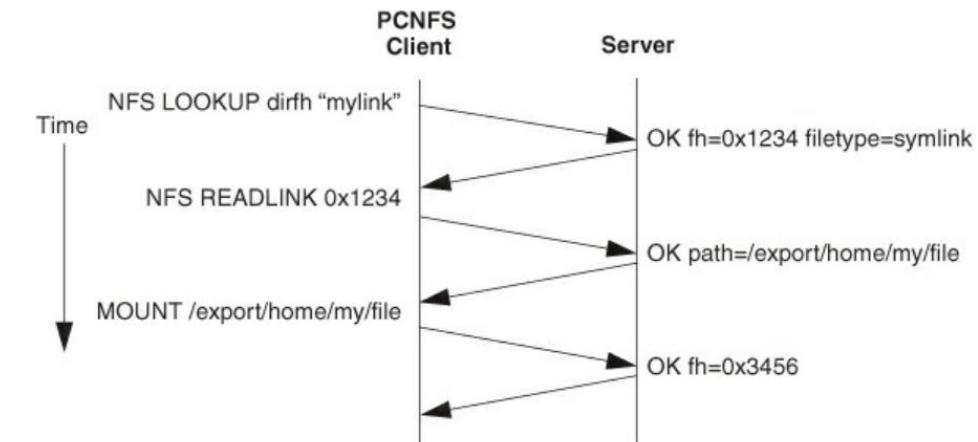
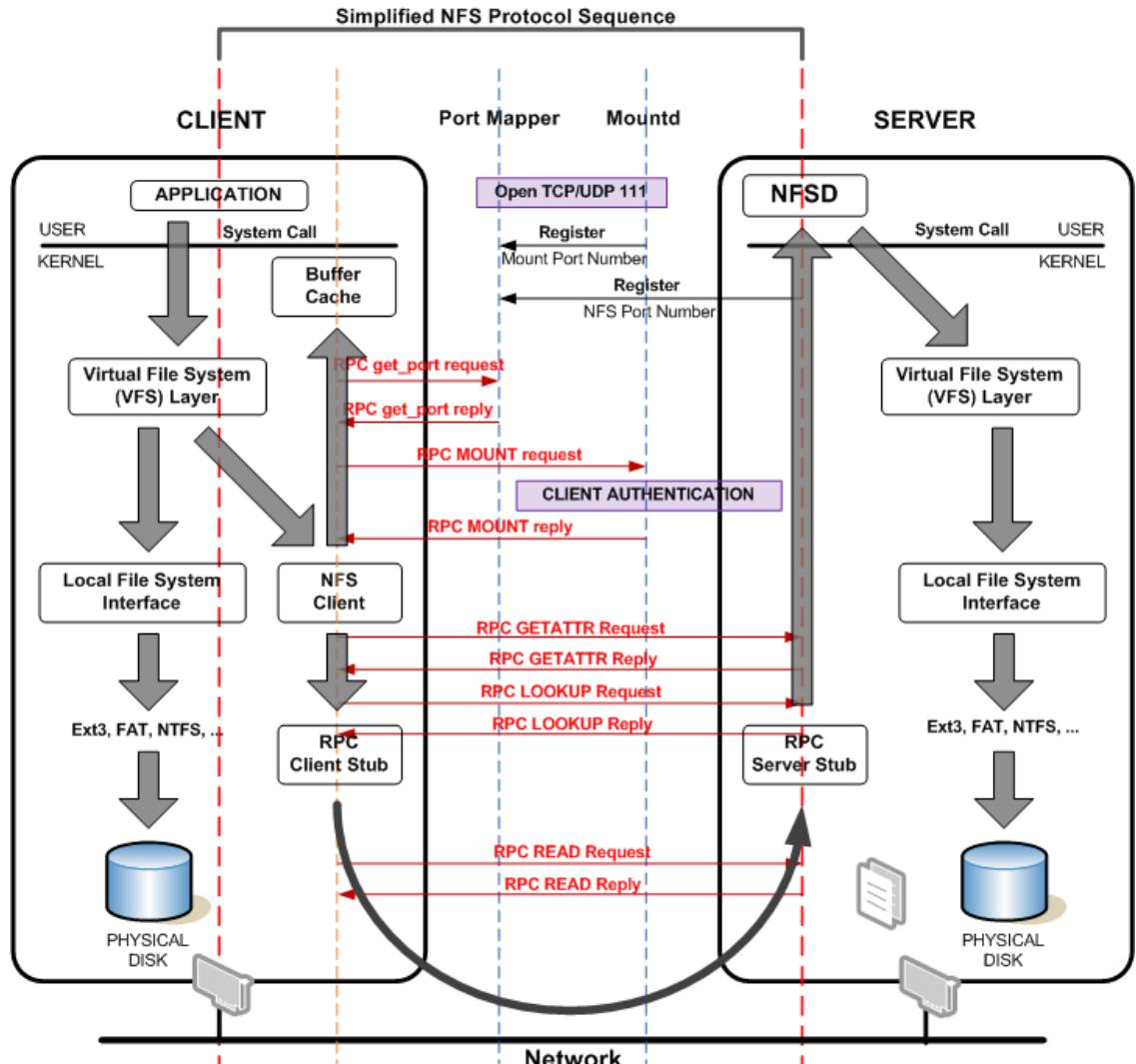
## NFS: Capas y protocolo

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)



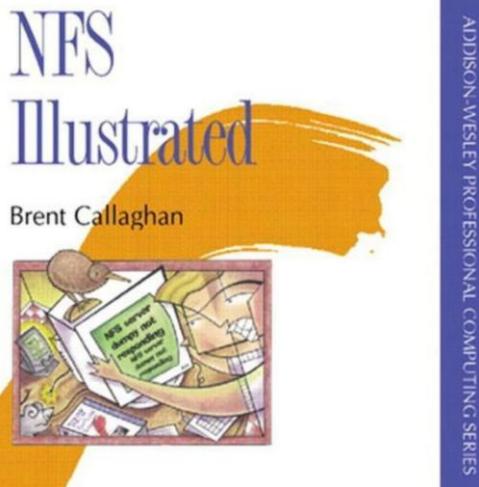
# Sistemas ficheros distribuidos

introducción

modelos DFS

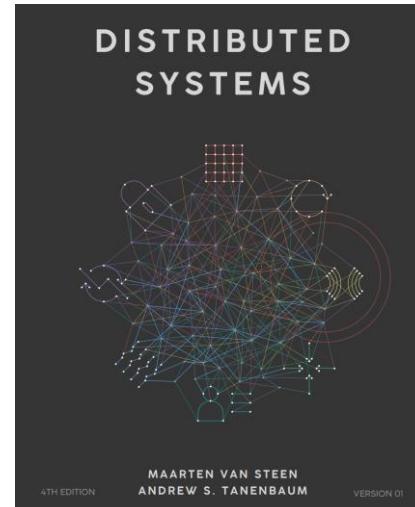
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

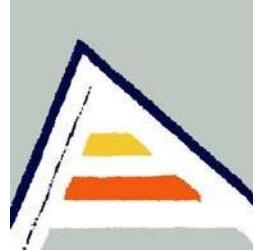


<b>13 Other Distributed Filesystems</b>
13.1 Remote File Sharing
13.1.1 Remote System Calls
13.1.2 RFS Naming
13.1.3 Security and UID/GID Mapping
13.1.4 Summary
13.2 Andrew File System
13.2.1 File Caching
13.2.2 Shared Namespace
13.2.3 Volume Movement
13.2.4 Read-only Replication
13.2.5 Security
13.2.6 Summary
13.3 DCE/DFS
13.3.1 Cache Consistency with Tokens
13.3.2 DFS Namespace
13.3.3 Episode File System
13.3.4 Summary
13.4 SMB File Access
13.4.1 Namespace
13.4.2 Session Setup
13.4.3 PC File Semantics
13.4.4 Batched Requests
13.4.5 File Locking
13.4.6 Opportunistic Locks
13.4.7 The Samba Server
13.4.8 Summary

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



<b>8 Fault tolerance</b>
8.1 Introduction to fault tolerance . . . . .
8.1.1 Basic concepts . . . . .
8.1.2 Failure models . . . . .
8.1.3 Failure masking by redundancy . . . . .
8.2 Process resilience . . . . .
8.2.1 Resilience by process groups . . . . .
8.2.2 Failure masking and replication . . . . .
8.2.3 Consensus in faulty systems with crash failures . . . . .
8.2.4 Example: Paxos . . . . .
8.2.5 Consensus in faulty systems with arbitrary failures . . . . .
8.2.6 Consensus in blockchain systems . . . . .
8.2.7 Some limitations on realizing fault tolerance . . . . .
8.2.8 Failure detection . . . . .
8.3 Reliable client-server communication . . . . .
8.3.1 Point-to-point communication . . . . .
8.3.2 RPC semantics in the presence of failures . . . . .
8.4 Reliable group communication . . . . .
8.4.1 Introduction . . . . .
8.4.2 Scalability in reliable multicasting . . . . .
8.4.3 Atomic multicast . . . . .
8.5 Distributed commit . . . . .
8.6 Recovery . . . . .
8.6.1 Introduction . . . . .
8.6.2 Checkpointing . . . . .
8.6.3 Message logging . . . . .
8.7 Summary . . . . .



Grado en Ingeniería Informática

## **Sistemas distribuidos**

# **Sistema de ficheros distribuido y DLT**

Departamento de Tecnología Informática y Computación

Curso 2023 - 2024

# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

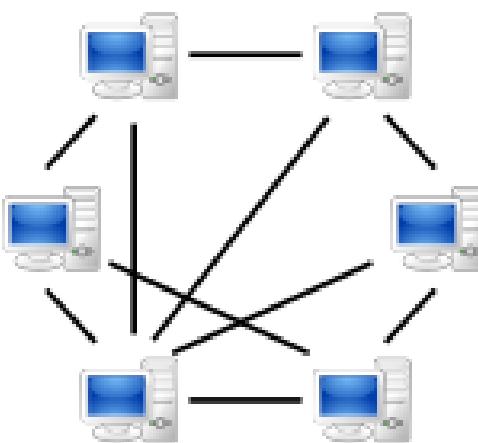
Distributed  
Ledger Tech  
(DLT)

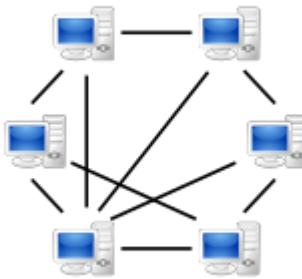
## **Red peer-to-peer (*P2P*), red de pares, red entre iguales o red entre pares**

- Red de ordenadores en la que todos o algunos aspectos se comportan como iguales entre sí (cliente y servidor al mismo tiempo).
- Construida sobre la capa OSI de aplicación de Internet

Las redes *P2P* aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos

- obtienen más rendimiento que con algunos métodos convencionales





# Sistemas ficheros distribuidos: P2P

introducción

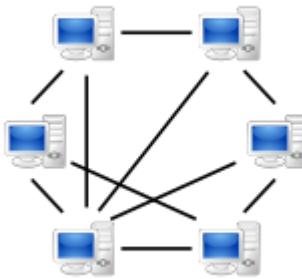
modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Redes aplicación conocida:

- Intercambio y búsqueda de ficheros DHT.
  - BitTorrent o eMule (eDonkey2000), Napster, Gnutella, KaZaA, Morpheus, ...
  - Compañías como Warner Bros o la BBC, empezaron a ver el P2P como alternativa a la distribución de sus contenidos a través de tecnologías como la de BitTorrent
- Sistemas de ficheros distribuidos
  - **CFS o Freenet.**
  - Cálculos científicos que procesen **enormes bases de datos** (bioinformáticos)
- Sistemas para proporcionar anonimato (**peer-to-peer anónimo**)
  - **i2p**, Tarzan P2P o MorphMix, **ZeroNet**.
  - Este tipo de tecnologías forman parte de la llamada DARKNET
- Sistemas de telefonía por Internet
  - versiones antiguas de Skype.
- Monedas virtuales para transacciones entre partes.
  - **Bitcoin**
- Grabadores de sistemas de CCTV que transmiten las imágenes a usuarios conectados desde celulares y computadores en ISP con puertos bloqueados.



# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Escalabilidad

- Cuantos más nodos mejor será su funcionamiento y mayor los recursos totales del sistema aumentan.

## Robustez.

- En caso de fallos en las réplicas, los peers encuentran la información sin hacer peticiones a ningún servidor centralizado de indexado; no hay ningún punto singular de falla en el sistema.

## Descentralización.

- Por definición, descentralizadas y todos los nodos son iguales.
- No existen nodos con funciones especiales o imprescindible para el funcionamiento de la red.

## Distribución de costes entre los usuarios.

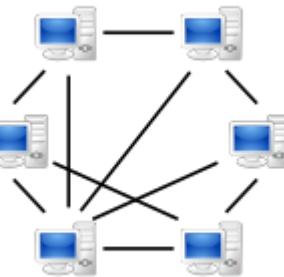
- Se comparten o donan recursos a cambio de recursos. Según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.

## Anonimato.

- Es deseable que quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo, siempre que así lo necesiten los usuarios.
- Son incompatibles con la industria (DRM) para limitarlo.

## Seguridad.

- Identificar y evitar los nodos o contenido maliciosos o infectado.
- Creación de grupos seguros de nodos dentro de la red



# Sistemas ficheros distribuidos: P2P

introducción

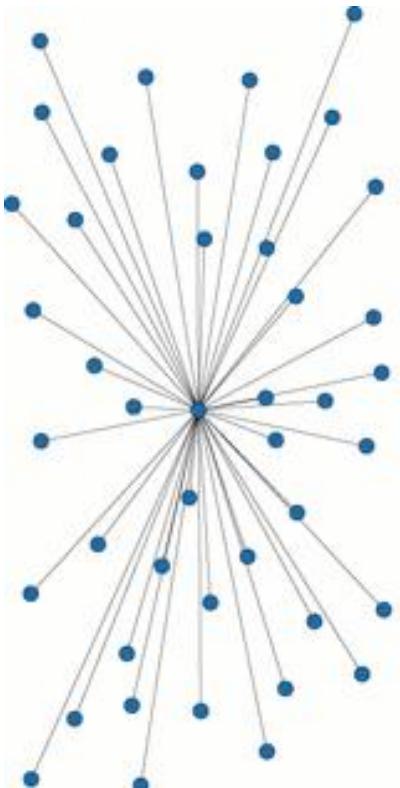
modelos DFS

implementación  
DFS

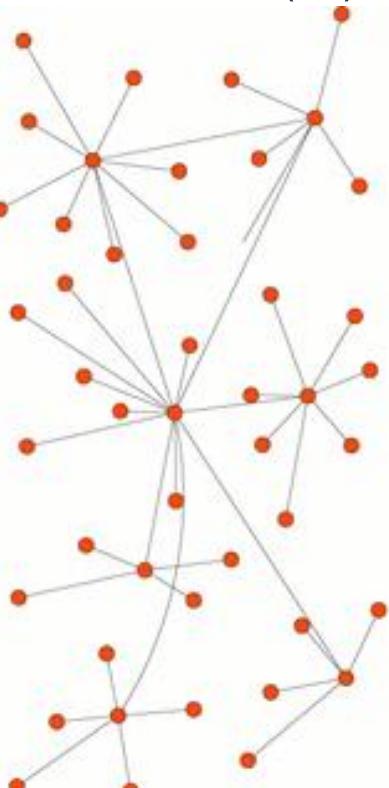
Distributed  
Ledger Tech  
(DLT)

## Problemas:

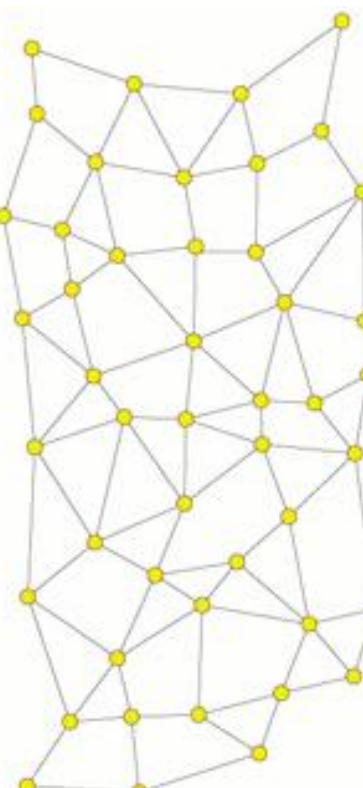
- Descentralizado y Anónimo VS Seguridad
- Conexiones e identificadores (IP) no permanentes



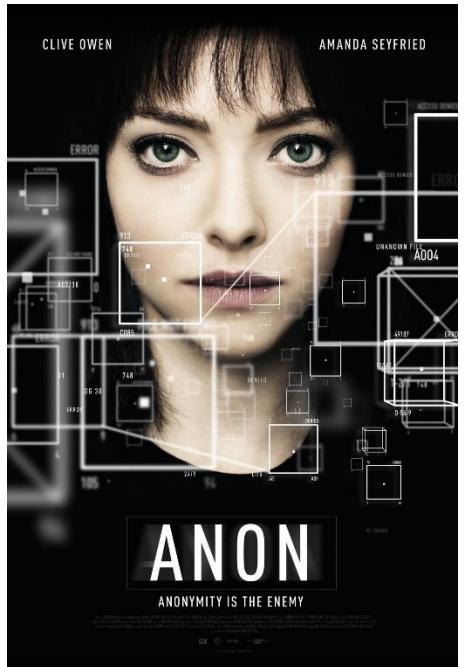
RED CENTRALIZADA

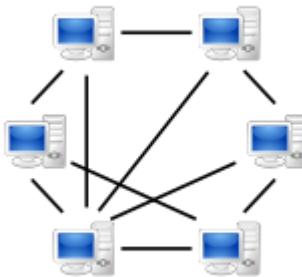


RED DESCENTRALIZADA



RED DISTRIBUIDA





# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Seguridad: Confidencialidad e integridad

La mayoría de las aplicaciones de tratan con datos personales en la red con peers que no son especialmente fiables.

Por tanto, los datos deben protegerse mientras se transmiten y almacenan.

La confidencialidad y la integridad de los datos almacenados se garantizan con los medios criptográficos habituales, como métodos de cifrado y sumas de comprobación.

 Physical Security

**Protection of your downloads, uploads and Freenet browsing cache**

How concerned are you about people stealing or seizing your computer?

We strongly recommend you encrypt your hard disk using e.g. *Truecrypt*. This will provide the best protection, especially if you use third party apps and plugins such as Sone and FMS, but may take some time to set up. Otherwise, you can set a password for your downloads, uploads, and cache of recently visited Freenet websites.

Also, you should encrypt your swapfile if possible (this is a temporary file used by Linux to provide extra space if it runs out of memory). This may be easier to set up than full disk encryption.

**NONE:** Don't encrypt anything. (Choose only if you have Truecrypt etc installed)

**LOW:** Protect the downloads, uploads, and Freenet browsing cache, with a persistent encryption key so they can be wiped quickly.

**HIGH:** Protect the downloads, uploads, and Freenet browsing cache with a password.

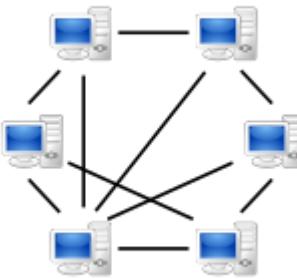
Set the password:

**MAXIMUM:** Use a temporary encryption key, so that downloads, uploads and Freenet browsing cache are wiped on restarting Freenet.

You should follow standard security precautions (keep updated, run an anti-virus, don't open files you don't trust etc), and note that if they capture your computer while it is switched on and running Freenet, no amount of encryption will help you!

[Back](#) [Next](#) 

**MAXIMUM:** Use a temporary encryption key, so that downloads, uploads and Freenet browsing cache are wiped on restarting Freenet.



# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

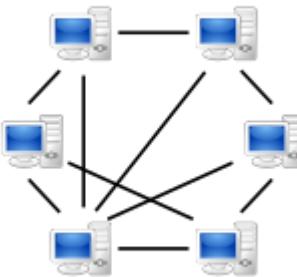
## Control de acceso

El cifrado es un mecanismo básico para el control de acceso en las operaciones de lectura.

La falta de autenticación puede superarse mediante la distribución de las claves necesarias para acceder a los datos almacenados a un subconjunto de pares privilegiados.

Las listas de control de acceso también pueden ser asignadas a los datos mediante certificados firmados.

El control de acceso en las operaciones de borrado es critico debido a su resultado devastador en caso de error o ataque.



# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

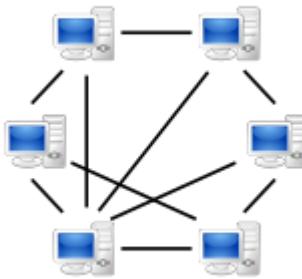
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## **IPFS** es un sistema de archivos distribuidos punto a punto

- Conectar todos los dispositivos con el mismo sistema de archivos.
- Similar a la WWW, pero en modo enjambre
- Proporciona un modelo de almacenamiento en bloques de alto rendimiento y contenido direccionado, con hipervínculos dirigidos al contenido.
- La distribución de contenido descentralizada ahorra ancho de banda y previene ataques DDoS, contra lo que HTTP tiene dificultades.
- Los archivos se identifican por sus valores hash
- Tiene un servicio de nombres llamado IPNS, un espacio de nombres global basado en PKI.





# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

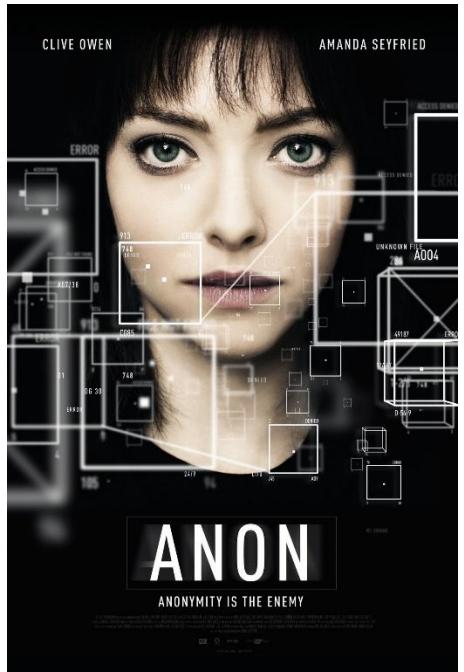
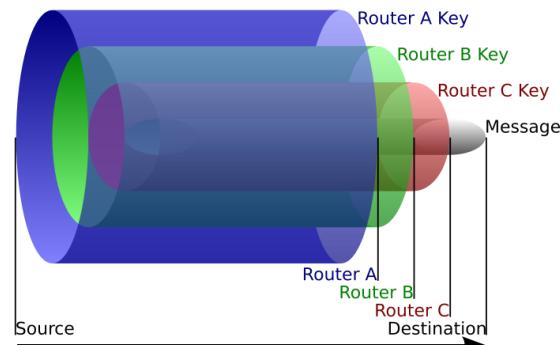
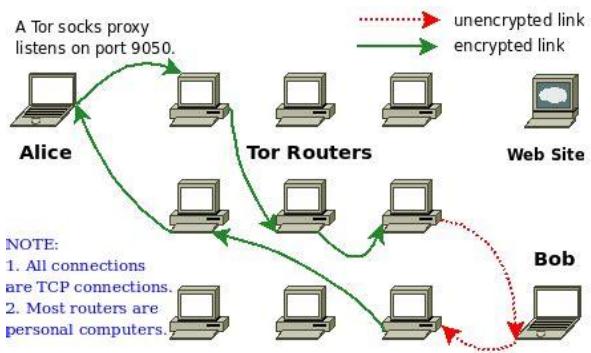
Distributed  
Ledger Tech  
(DLT)

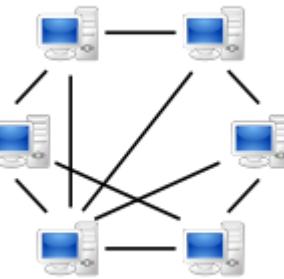


## Anonimato

El anonimato permite evitar ataques en los que los datos de un usuario determinado son el objetivo específico para destruirlos del sistema.

Los sistemas que pretenden proporcionar anonimato suelen emplear infraestructuras para proporcionar capas de conexión anónimas





# Sistemas ficheros distribuidos: P2P

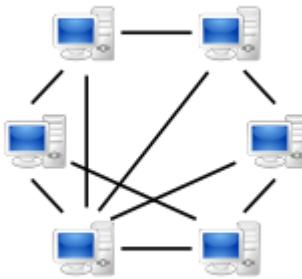
introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)





# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

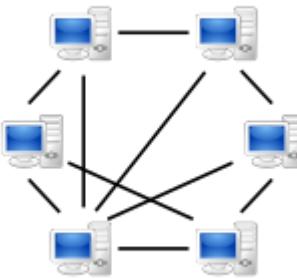
Distributed  
Ledger Tech  
(DLT)

Fiabilidad de los datos

La técnica habitual para lograr la fiabilidad de los datos se basa en la redundancia de los datos en varias ubicaciones de la red.

Los datos pueden replicarse simplemente con un factor de redundancia determinado.  
El rejuvenecimiento de los datos puede realizarse de forma periódica o por eventos.





# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

implementación  
DFS

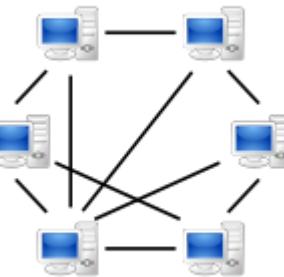
Distributed  
Ledger Tech  
(DLT)

## Protocolos de verificación

Prueba de conocimiento en la que se intenta convencer a un verificador de que posee unos datos, lo que se demuestra respondiendo correctamente a consultas que requieren computación sobre los propios datos.

Se distinguen dos categorías de verificación (Probabilísticos vs Deterministas):

- Homomorphic Hash Functions
- Storage Enforcing Commitmen
- Deterministic Remote Integrity Check
- Incremental Cryptography
- Algebraic Signatures
- Data Chunk Recovery
- Remote Integrity Check
- Compact Proofs of Retrievability
- Proof of Retrievability
- Provable Data Possession
- Authenticator



# Sistemas ficheros distribuidos: P2P

introducción

modelos DFS

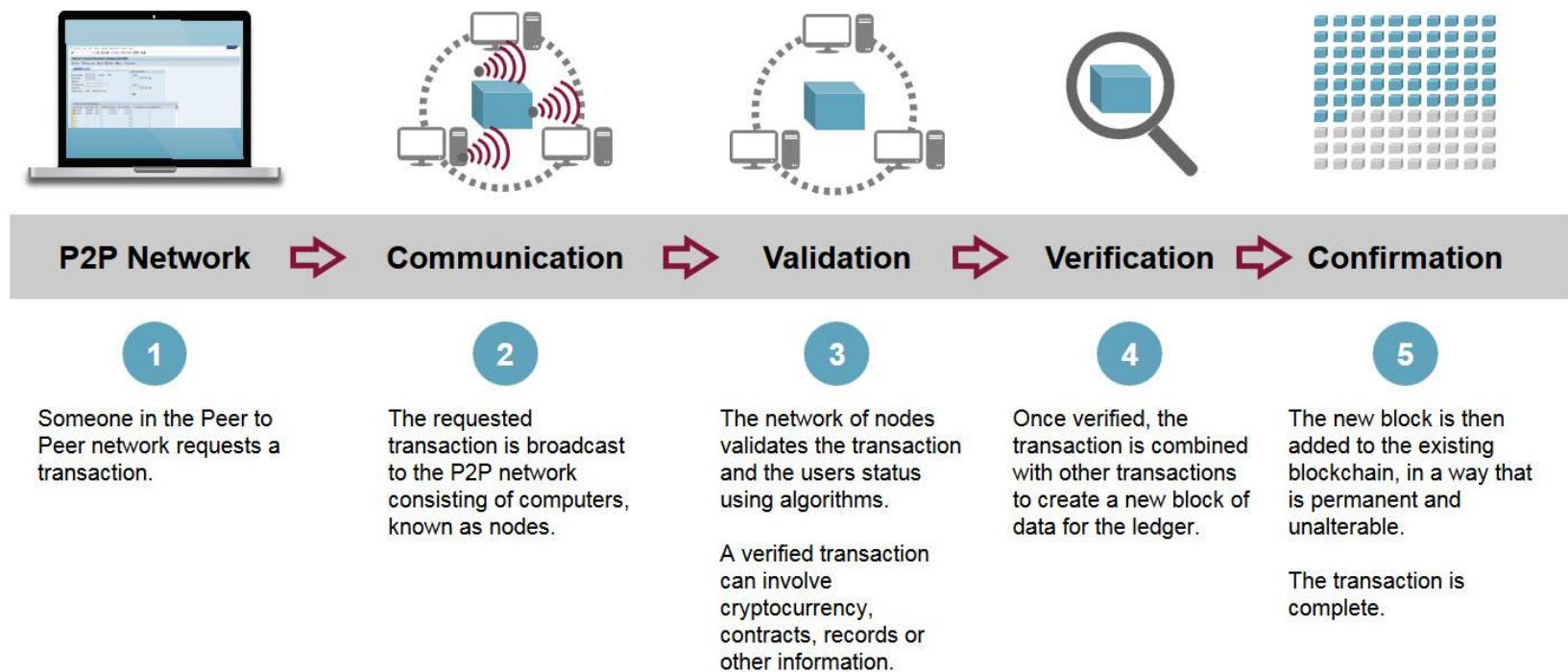
implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Ficheros mutables vs Inmutables (registros)

- Blockchain

## Blockchain Process Steps



# ¿qué es blockchain?

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

- **DLT más popular:** de su nombre en inglés se deduce que es una “cadena de bloques” (DLT == *Tecnología de libro mayor distribuido*)
- Se puede entender como un “libro de registro en formato digital” (como el del Registro Civil, Registro Mercantil, Registro de la Propiedad, ...)
- Pero con algunos cambios (y limitaciones):
  - No es un libro físico → una cadena DIGITAL en la que cada eslabón/BLOQUE representa una “HOJA” del libro
  - Cada BLOQUE está anclado criptográficamente al anterior → cada bloque contiene el código HASH verificable del bloque anterior
  - No existe una sola copia del libro entero, sino MÚLTIPLES COPIAS → una copia por usuario de *blockchain* (que guarda la cadena entera)
  - No se puede borrar, ni eliminar, ni modificar NINGÚN eslabón una vez incorporado → SÓLO AÑADIR

# ¿qué es blockchain?

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Estructura de la cadena de bloques:



El orden de los bloques es importante, igual que lo es el número de página de un libro

# blockchain: aspectos clave

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## No requiere de intermediarios, ni responsables únicos

- No hay una copia principal
- El control y verificación -> COMUNIDAD (que posee una copia inmutable)

## Transparente

- Cualquiera puede hacer consultas a la cadena (sólo leer)

## Segura ¿qué significaría cambiar un eslabón fraudulentamente?

- Que tu cadena tiene un eslabón distinto a las miles de copias existentes de esa cadena
  - El código HASH no coincide con el bloque anterior y siguiente
- El CONSENSO (más del 50%) de los usuarios detecta el fraude y rechaza el cambio (y te echa del sistema por mala praxis)
- ¿Se puede CONTROLAR al 51% de los poseedores de una copia?
  - debería generar una cadena NUEVA desde CERO y poseer el 51% de los NODOS

## Control no determinista: Power of Work (PoW) y Power of Stake (PoS)

- Para añadir bloques a la cadena → Resolver desafío matemático → ganador no determinista → obtiene recompensa (MINE)
- En Bitcoin el tiempo medio de resolución es de 10 mins., en Ethereum de 15 segs.

# blockchain: ejemplo *Bitcoin*

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Cómo funciona blockchain



Fuente: FT

INSIDER<sup>PRO</sup>

<https://blockstream.info/>

# blockchain: ejemplo *Bitcoin*

introducción

modelos DFS

implementación  
DFS

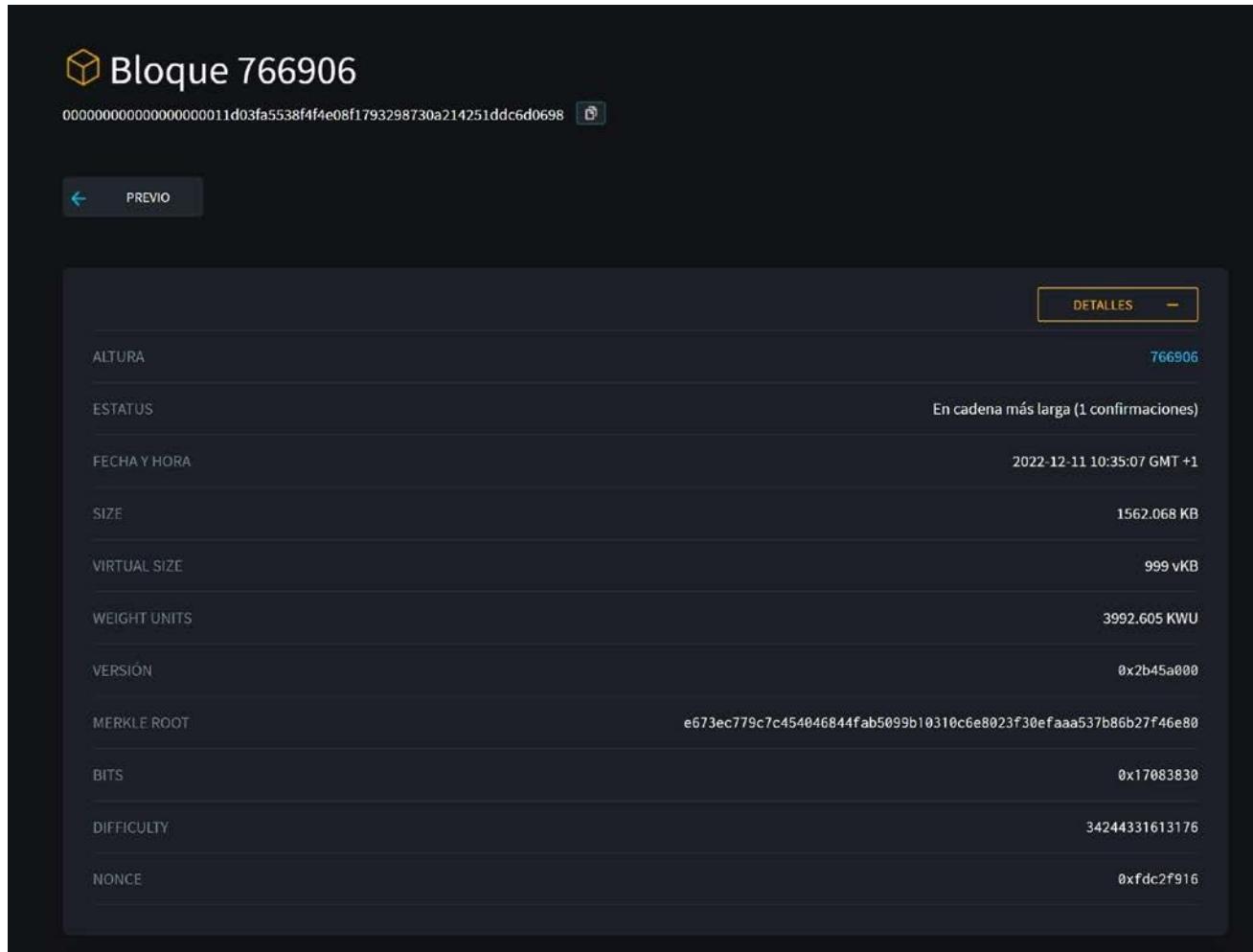
Distributed  
Ledger Tech  
(DLT)

The screenshot shows the Blockstream Explorer interface. At the top, there are tabs for 'Dashboard', 'Blocks' (which is selected), and 'Transactions'. There are also buttons for 'Bitcoin' (highlighted in yellow) and 'Liquid'. A search bar at the top right allows searching by block height, hash, transaction, or address. Below the search bar, a section titled 'Latest Blocks' displays a table of ten recent blocks. The columns in the table are 'Altura' (Height), 'Fecha y Hora' (Date and Time), 'Transacciones' (Transactions), 'Tamaño (KB)' (Size KB), and 'Peso (KWU)' (Weight KWU). The data for the ten blocks is as follows:

Altura	Fecha y Hora	Transacciones	Tamaño (KB)	Peso (KWU)
766906	2022-12-11 10:35:07	3248	1562.068	3992.605
766905	2022-12-11 10:09:17	1072	1093.294	2543.521
766904	2022-12-11 10:04:04	1087	1196.445	3993.363
766903	2022-12-11 10:01:14	1699	1329.317	3993.371
766902	2022-12-11 09:41:33	2583	1706.643	3993.48
766901	2022-12-11 09:20:34	2161	1498.649	3993.071
766900	2022-12-11 09:03:41	1376	980.117	2323.433
766899	2022-12-11 08:54:34	1910	1142.784	2931.666
766898	2022-12-11 08:37:14	1489	1148.135	2992.964
766897	2022-12-11 08:25:22	1421	982.75	2467.879

<https://blockstream.info/>

# blockchain: ejemplo *Bitcoin*



<https://blockstream.info/>

# blockchain: *smart contracts*

introducción

modelos DFS

implementación  
DFS

**Distributed  
Ledger Tech  
(DLT)**

## How does a **SMART CONTRACT WORK?**



### Pre-Defined Contract

Terms and conditions are agreed by all the parties involved.



### Events

Execution of the contract is triggered by an event.



### Execution

The smart contract is executed automatically.



### Settlement

All the settlements are executed quickly and efficiently.

# blockchain: *smart contracts*

---

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Contratos inteligentes (de la traducción del inglés)

- Son algoritmos preestablecidos: Generan ACCIONES automáticamente, si se cumplen determinadas CONDICIONES en la blockchain
- Ni son contratos → No es fuente de obligaciones, sino MECANISMO DE EJECUCIÓN de obligaciones (automáticamente)
- Ni son inteligentes → No tienen capacidad de “pensar”, ni de “inferir” → No valoran el contexto y son inflexibles en la aplicación (no permiten la negociación a posteriori)
- Pueden reducir los trámites y esperas → en casos muy estructurados
- Suele requerir del uso de “**oráculos**” como fuentes externas “fiables” de información

## Casos de uso

- *Seguros*: Devolución inmediata de importe de viaje, en caso de retraso en la llegada > 1h
- *Testamentos*: En caso de fallecimiento, reparto automático de bienes según testamento
- *Trazabilidad* y control caducidad de los alimentos: valor añadido para el cliente
- *Compliance*: auditoría de procesos, buenas prácticas, RGPD, ...

# tipos de blockchain



introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## • Públicas

- No son necesarios permisos para acceder
- Transacciones visibles y generables por cualquiera

	Públicos Bitcoin, Ethereum, Litecoin	Privados Hyperledger, Corda, Quorum	Federados Hyperledger, Corda, Quorum	Blockchain as a Service IBM, Microsoft, Amazon
Cualquiera puede participar	✓	✗	✗	NA
Los participantes actúan, en general, como nodos	✓	✗	✗	NA
Transparencia	✓	~~	~~	NA
Hay un único administrador	✗	✓	✗	NA
Hay más de un administrador	✗	✗	✓	NA
No hay administradores	✓	✗	✗	NA
Ningún participante tiene más derechos que los demás	✓	✗	✗	NA
Se pueden implementar Smart Contracts	✓	✓	✓	NA
Existe recompensa por minado de bloques	~~	✗	✗	NA
Soluciona problema de falta de confianza	✓	✗	~~	NA
Seguridad basada en protocolos de consenso	✓	✗	~~	NA
Seguridad basada en funciones hash	✓	~~	~~	NA
Provee servicios en la nube	NA	NA	NA	✓



Sí



No



A veces

NA No Aplica

# blockchain: *non-fungible-tokens (NFT)*

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

Blockchain como registro de:

- certificado de autenticidad
- propietario de un activo digital (fotos, diseños, videos, música, ...)



<https://opensea.io/collection/cryptopunks>



jack @jack  
just setting up my twttr  
9:50 PM · Mar 21, 2006  
165.1K 9.2K ⌂ Copy link to Tweet



# tecnología disruptiva

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

## Insurtech

Liquidación entre aseguradores, IoT e identidad digital para reducir costos de aseguramiento, contratos inteligentes aplicados a este campo

## Salud

Envío de información encriptada de pacientes a través de Blockchain, cumpliendo con regulación de protección de datos

## Servicios Financieros

Securización, tokenización de activos, liquidaciones más baratas, trazabilidad de transacciones, transparencia

## Compliance

Blockchain podría ahorrar millones mejorando procesos de cumplimiento y quitando duplicidades entre entidades, identidad digital podría ser relacionada a AML/KYC, privacidad de data y políticas FATCA

## Legal/Digitalización de contratos

Es un protocolo especialmente creado para programar acuerdos entre dos o más partes sin depender de intermediarios para garantizar su correcta ejecución

## Identidad Digital

Acelerar el onboarding digital, puede ser utilizada para materia de compliance, la identidad digital como llave del IoT, Blockchain habilita compartición de información

## Registro

Blockchain permite utilizar marcas de tiempo, "pruebas de existencia" y notarizar cada transacción

## Internet de las Cosas

Propiedad fraccionada, registro de propiedades, inclusión de objetos en canales de pago, Blockchain permite contratos persona-objeto



# reflexiones finales

introducción

modelos DFS

implementación  
DFS

Distributed  
Ledger Tech  
(DLT)

- Blockchain no sirve para todo → Para casos en que se requiera un registro inmutable, no manipulable, accesible 24/7, mantenido por una COMUNIDAD de miembros no necesariamente de confianza
- Blockchain se vincula ERRÓNEAMENTE con actividades ilegales → los registros/transacciones SON VISIBLES por todos en la cadena origen y destino → ¿y los pagos con dinero en metálico...?
- ¿En peligro abogados, notarios, jueces... legisladores?
  - En absoluto: va a ser necesaria la colaboración estrecha entre juristas e ingenieros
    - codificación de los Smart Contracts
    - Decidir qué redes de “blockchain” tienen la credibilidad de lo que registran
- Problemas jurisdiccionales al tratarse de sistemas que operan de forma global