

# Laboratorio de Computación IV

## Clase 6

*Andrés Fortier*

# ¿Consultas?

---

- Comando: ssh.
- Contenidos web: PAAS (Openshift).
- Herramienta: github (+remotes +issues).

# Tarea



- Jugar un poco con openshift
  - Ahora pueden romper tranquilos (crear/borrar aplicaciones, repos git, etc).
  - <https://developers.openshift.com/en/getting-started-overview.html>
- Configurar su *key ssh* para github.
- Leer un poco sobre markdown.
- Leer sobre github.

# Comandos del día: mkdir + rmdir

- mkdir == *make directory*.
- rmdir == *remove (empty) directory*.

```
$ mkdir prueba
```

```
$ ls -lh | grep prueba
```

```
drwxrwxr-x    2 andres andres 4,0K abr  4 18:41 prueba
```

```
$ rmdir prueba
```

```
$ ls -lh | grep "prueba"
```

```
$
```

```
$ mkdir contenedor/contenido
```

```
mkdir: cannot create directory 'contenedor/contenido':  
No such file or directory
```

# Comandos del día: mkdir + rmdir

-p : Crear los directorios intermedios necesarios

```
$ mkdir -p contenedor/contenido
```

```
$ ls -lh | grep cont
```

```
drwxrwxr-x 3 andres andres 4,0K abr  4 18:48 contenedor
```

```
$ rmdir contenedor/
```

```
rmdir: failed to remove 'contenedor/': Directory not empty
```

```
$ rmdir contenedor/contenido/
```

```
$ rmdir contenedor/
```

```
$ ls -lh | grep cont
```

# Comandos del día: mkdir + rmdir


**-v : Muestra los directorios creados**

```
$ mkdir -pv contenedor/contenido  
mkdir: created directory 'contenedor'  
mkdir: created directory 'contenedor/contenido'
```

```
$ ls -lh | grep cont  
drwxrwxr-x 3 andres andres 4,0K abr  4 18:49 contenedor
```

```
$ rmdir contenedor/contenido/  
$ rmdir contenedor/  
$ ls -lh | grep cont
```

# Seguridad en la web



- La seguridad informática es un mundo **gigante**.
- No soy un experto en el tema; esto es simplemente una introducción.
- Conceptos básicos de seguridad
  - Canal de comunicación seguro (HTTPS).
  - Manejo de passwords.

# HTTPS



- Técnicamente no es un protocolo en si mismo
  - HTTP + SSL/TSL
- Dos grandes beneficios
  - Autenticar la aplicación/servidor
    - *Man-in-the-middle attack.*
  - Encriptar la información que se comparte entre el cliente y el servidor
    - *Eavesdropping*
    - *Tampering*



# HTTPS



- Conexión HTTPS
  - Cliente inicia con “*ClientHello*”. Se setean parámetros de sesión SSL.
  - El servidor provee su certificado SSL. El cliente puede asumir que es válido o validarlo de alguna forma.
  - Intercambio de claves de encriptación simétricas.
  - El resto es HTTP estándar, pero sobre un canal seguro.

# HTTPS



- Validación de certificados
  - *Digital signatures*
    - Los certificados SSL poseen un *public/private key pair*.
    - *Public key* se entrega con el certificado.
    - El cliente encripta, pero sólo el servidor puede des-encriptar.
  - *Certificate Authorities*
    - ej. Symantec, Comodo, GoDaddy.
    - Encriptan con su clave privada (sólo ellos lo pueden hacer).
    - Ofrecen la clave pública para des-encriptar.

# HTTPS



---

- Certificados *self-signed*
  - Sirven para probar el sitio hasta que salga en producción.
  - No son confiados por ningún browser.

# HTTPS



- Evita “*man-in-the-middle*”
  - Servidor ABC copia el certificado del sitio XYZ.
  - Cliente es engañado para visitar ABC.
  - ABC da el certificado de XYZ.
  - Las validaciones de CA salen ok.
  - El cliente encripta la llave que se va a usar para encriptar el resto de la comunicación.
  - ABC no puede des-encriptarla.

# HTTPS



- Evita “*eavesdropping*”
  - Ver los bytes que se intercambian no dicen nada del contenido actual.
  - Sin embargo seguimos usando HTTP; cuidado con las redirecciones.

# Autenticación



- Proceso por medio del cual se verifica que una persona es quién dice ser.
- Forma básica: usuario/email + password.
- Ya vimos que usando HTTPS podemos intercambiar datos en forma segura.
- ¿Cómo guardamos las credenciales en el servidor?

# Autenticación

---

- Texto plano
  - Si nos roban la BD acceden a todas las cuentas.
  - Muchos usuarios usan el mismo password para otros sitios.

# Autenticación



- Encriptación del password
  - Reversible.
  - Si la clave para encriptar se filtra, se deducen todos los passwords.
  - 1 clave por password
    - ¿dónde se guarda?
    - ¿derivada?
  - Dependiente del algoritmo => Se puede filtrar el código

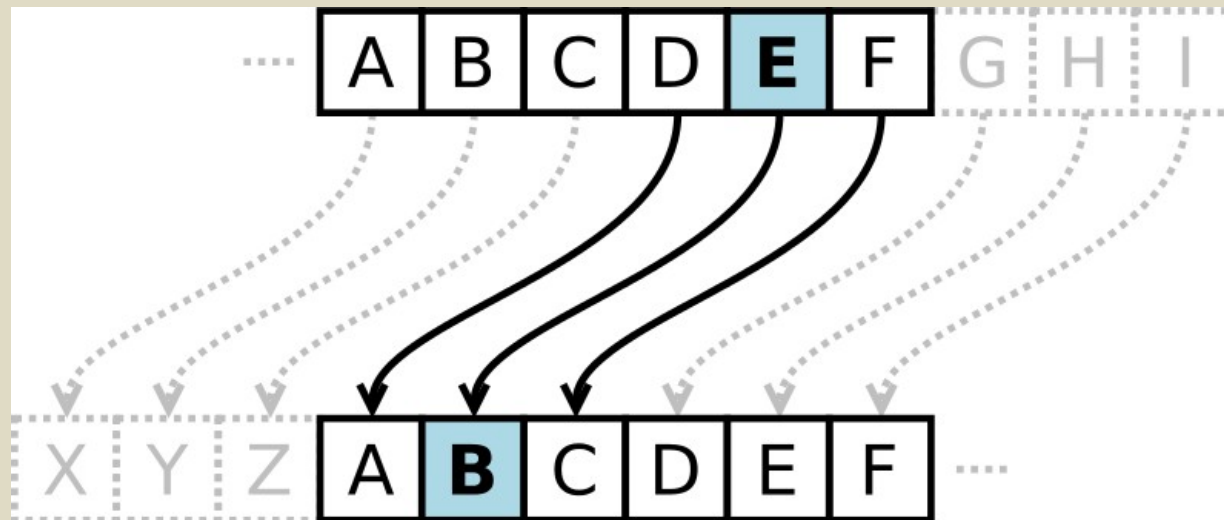


# Ejemplo de encriptación

- *Caesar cipher*

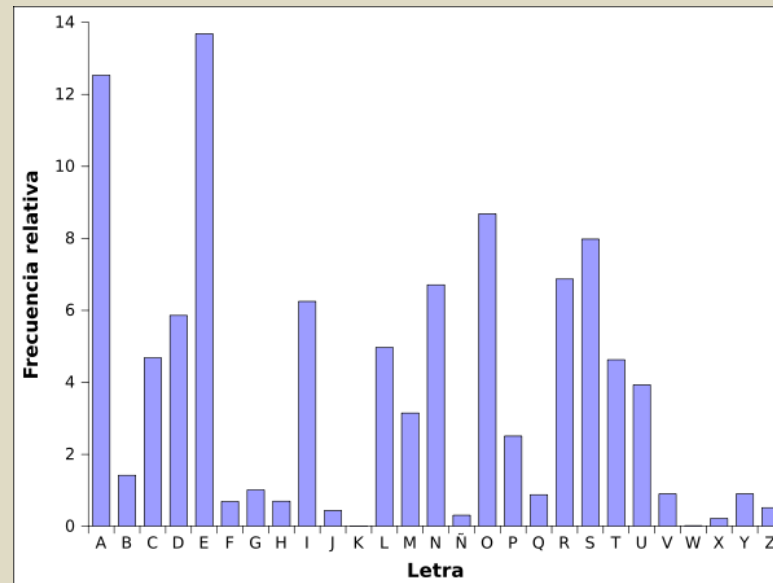
- Utilizado por Julio Caesar para su correspondencia.
- “Sumaba” 3 letras

Sin codificar:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Codificado	DEFGHIJKLMNOPQRSTUVWXYZABC



# Ejemplo de encriptación

- Fácil de quebrar
  - Fuerza bruta (probar las 27 combinaciones).
  - Análisis de frecuencia de letras.



- Múltiples encriptaciones no lo hacen mas seguro.

# Hashing

- Hash criptográfico (*Cryptographic hash*).
- Función
  - Convierte un mensaje (ej. password) en bytes (*digest*).

ejemplo →

```
$2a$10$H7HnvDAfNyKo1mnj.cuBoOxWOGshDpgOnmPlgXpCitIFmwo.s6b10
```

# Hashing



- Hash criptográfico (*Cryptographic hash*).
- Función
  - Convierte un mensaje (ej. password) en bytes (*digest*).
  - Simple computar el hash.
  - Baja probabilidad de generar el mensaje a partir del hash (no-reversibles).
  - Baja probabilidad de modificar el mensaje sin modificar el hash.
  - Baja probabilidad de colisiones.

# Hashing y fuerza bruta

- No cualquier función de hash
  - Fáciles de romper por fuerza bruta (ej. MD5 o SHA1).
  - Preferentemente *adaptive hashing*
    - *Procesador: PBKDF2 y Bcrypt*
    - *Memoria: Scrypt*

	Digest por segundo
NTLM	350,000,000,000
MD5	180,000,000,000
SHA1	63,000,000,000
SHA512Crypt	364,000
Bcrypt	71,000

# Hashing y diccionarios



- Lista de palabras (o claves) usuales.
- Se computa y comparan los hash; si hay coincidencia, tenemos la clave.
- No garantizan encontrar la clave.
- Sensibles al “tamaño” de la clave; mas caracteres, mas combinaciones.

# Hashing y *lookup tables*

- Pre-computar tablas (ya sean todas las combinaciones o por diccionario)
  - Ej. <clave>, bcrypt(<clave>)
- Luego se comparan los hash; si hay coincidencia, tenemos la clave.
- Sensibles al “tamaño” de la clave; tiempo vs espacio.
- *Rainbow tables*. Estructuras de datos que balancean tiempo vs espacio.

# Hashing y *salt*

- Tablas pre-computadas
    - Dos usuarios con el mismo password tienen el mismo hash.
    - Permite paralelizar la búsqueda.
  - *Salting*
    - Generar un string aleatorio por usuario
- ```
hash = bcrypt(password + random_salt)
```
- Dos usuarios con mismo password, distinto hash.
  - Evita ataques por tablas (no es pre-computable).



# Hashing - iteraciones

- Iterar sobre el password generado

```
hash = bcrypt(password + random_salt)
for (i = 0; i < 100; i++) {
    hash = bcrypt(password + random_salt + hash)
}
```

- Toma mas tiempo.
- No se encuentran en diccionarios.

# Hashing - recuperar password

- Un hash criptográfico no es reversible.
  - No se puede recuperar el password.
- Sitio web
  - Link “perdí mi password”.
  - Mail al usuario con un *token*
    - <http://misitio.com?recover=SDCACJVJERVNV...>
  - El *token* expira en un lapso de tiempo predefinido.
  - El *token* es válido para un ingreso.

# Entrega 1



- Modificar el programa en ``cmd.rb`` para que autentique usuarios.
- Implementar las tres estrategias vistas
  - Texto plano.
  - *Caesar cipher*.
  - *Bcrypt*.
    - <https://rubygems.org/gems/bcrypt/versions/3.1.10>
- Algunos casos de uso

# Entrega 1

- Salir del programa

```
> Seleccione una accion:  
1- Login  
2- Logout  
3- Estado  
4- Salir  
? 4  
> Adios, vuelva pronto!  
$
```

# Entrega 1

- Estado de persona no logueada

```
> Seleccione una accion:  
1- Login  
2- Logout  
3- Estado  
4- Salir  
? 3  
> Usted no se encuentra logueado  
> Seleccione una accion:  
...
```

# Entrega 1

- Estado de persona logueada

```
> Seleccione una accion:  
1- Login  
2- Logout  
3- Estado  
4- Salir  
? 3  
> Usted está logueado como "Pepe"  
> Seleccione una accion:  
...
```

# Entrega 1

- Log in exitoso

```
> Seleccione una accion:
1- Login
2- Logout
3- Estado
4- Salir
? 1
> Usuario: "Pepe"
> Password: *****
> Usted se ha logueado exitosamente!
> Seleccione una accion:
1- Login
2- Logout
3- Estado
4- Salir
? 3
> Usted está logueado como "Pepe"
> Seleccione una accion:
...
```

# Entrega 1

- Log in fallido

```
> Seleccione una accion:
1- Login
2- Logout
3- Estado
4- Salir
? 1
> Usuario: "Pepe"
> Password: *****
> Nombre de usuario o contraseña incorrecta
> Seleccione una accion:
1- Login
2- Logout
3- Estado
4- Salir
? 3
> Usted no se encuentra logueado
> Seleccione una accion:
...
```



# Entrega 1

- Logout de un usuario logueado

```
> Seleccione una accion:  
1- Login  
2- Logout  
3- Estado  
4- Salir  
? 2  
> Usted se ha deslogueado en forma exitosa  
> Seleccione una accion:  
...
```

# Entrega 1

- Logout de un usuario no logueado

```
> Seleccione una accion:  
1- Login  
2- Logout  
3- Estado  
4- Salir  
? 2  
> Usted no se encuentra logueado  
> Seleccione una accion:  
...
```

# Requerimientos mínimos



- Nombre de usuario y password válido predefinido.
- Seleccionar la estrategia a usar en forma sencilla
  - Por medio de ``require`` (o ``require_relative``).
  - Comentando/des-comentando alguna línea de código.

# Evaluación



- Correcto funcionamiento.
- Diseño
  - Separar el modelo de la vista/controlador.
  - **POLIMORFISMO.**
- Tests.
- Uso de git.
- [Bonus] Uso de *issues*.
  - Planificación del trabajo.
  - Investigación.

# Bonus track

- Sólo mostrar “Logout” si el usuario está logueado.
- Mecanismo de autenticación
  - Agregar una opción para cambiarlo en tiempo de ejecución.
  - Pasar un parámetro al programa

```
$ ruby cmd.rb -auth=caesar
```

- Ver por ejemplo <http://www.sitepoint.com/ruby-command-line-interface-gems/>

# Bonus track



- Agregar la opción de registrar usuarios (dar de alta usuario y password)
  - Sin persistir, sólo en memoria.
- *Coverage* de los tests
  - <https://github.com/colszowka/simplecov>
  - [https://shvets.github.io/blog/2013/10/19/configure\\_simplecov.html](https://shvets.github.io/blog/2013/10/19/configure_simplecov.html)

# Cosas importantes



- Toda información sensible debe ir por HTTPS
  - Si aplica, deshabilitar HTTP por completo.
- No diseñen sus protocolos de seguridad.
  - Estén al día con las implementaciones.
  - Usen librerías conocidas y probadas.

# Links



- <http://security.stackexchange.com/questions/211/how-to-securely-hash-passwords>
- <http://codahale.com/how-to-safely-store-a-password/>
- <http://chargin.matasano.com/chargin/2015/3/26/enough-with-the-salts-updates-on-secure-password-schemes.html>
- <http://chargin.matasano.com/chargin/2007/9/7/enough-with-the-rainbow-tables-what-you-need-to-know-about-secure-password-schemes.html>



# Links



- <http://blog.codinghorror.com/speed-hashing/>
- <https://crackstation.net/hashing-security.htm>
- <http://blog.moertel.com/posts/2007-02-09-dont-let-password-recovery-keep-you-from-protecting-your-users.html>
- <http://blog.codinghorror.com/youre-probably-storing-passwords-incorrectly/>
- <http://plaintextoffenders.com/faq/devs>