Encriptación

Antonio Espín Herranz

Encriptación

- Activar la extensión para la encriptación:
- Create extension **pgcrypto**;

- Disponemos de las funciones encrypt / decrypt

- Tenemos funciones para cifrado básico, simétrico y asimétrico.
  - Cifrado: La información se cifra para que no se pueda recuperar
    - Por ejemplo, podemos cifrar una password.
  - C.Simétrico: Se encripta y desencripta a través de una password privada.
  - C.Asimétrico: Se utiliza un par de claves públicas y una password privada.

Pgcrypto

- Tenemos que comprobar que módulos tenemos disponibles en postgre y los que tenemos instalados, por defecto sólo viene instalado: **psql**

- Podemos hacer una consulta a:
  - Select * from **pg_available_extensions**;

- También:
  - Select * from **pg_available_extensions** where name='pgcrypto';

pgcrypto

- La primera vez que lo utilizamos hay que instalarlo:
- Create extension pgcrypto;

- Estas operaciones se suelen realizar desde psql

- Cuando lanzamos el comando anterior nos devuelve:
  - CREATE EXTENSION

- Esto indica que ya está instalado

- Comprobarlo de nuevo con pg_available_extensions

pgcrypto

- Dentro de las funciones de encriptado tenemos los hashes, por ejemplo: la función **md5**

- Para utilizarla hacemos:
  - Select md5(‘mensaje a encriptar’);

- Un hash es como una firma digital de un contenido.
- Coincidirá en longitud, pero el texto cambia con un pequeño cambio en el texto.
- No se pueden descifrar, no hay vuelta al texto original.

pgcrypto

- **Digest**
  - Ejecuta un hash de un contenido y especificamos el tipo de algoritmo de encriptado.
  - Select digest(‘contenido mensaje’, ‘md5’);

  - Devuelve una cadena de bytes. Si identifica porque empieza con \x
  - Se puede obtener una cadena aplicando la función encode y la codificación hexadecimal.
  - Select **encode**(digest(‘mensaje’, ‘md5’), ‘hex’);

pgcrypto

- Con **digest** podemos cambiar el algoritmo y obtener hashes más largos.
- Al añadir más longitud serán más difíciles de desencriptar.
- Por ejemplo: **sha1**

- Select encode(digest(‘contenido mensaje’, ‘sha1’), ‘hex’);

pgcryto

- En la misma línea tenemos los hashes: **sha256** o **sha512**
- Que devuelve cadenas más largas.
- Normalmente los **password** se almacenan con **md5**.

pgcrypto

- Otra función que genera hashes más seguros: es la función hmac
- A esta función se le pasa el mensaje, un texto que hace las veces de clave y el nombre del algoritmo.
- Select **hmac**(‘contenido mensaje’, ‘la semilla’, ‘md5’);
- Si cambiamos la clave, el hash va a cambiar, aunque se mantenga el mismo mensaje.
- También se puede cambiar el algoritmo: sha1, sha128 y sha256

pgcrypto

- Función **crypt** genera un encriptado más difícil de romper.

- Y le pasamos en algoritmo a través de otra función (**gen_salt**) que genera cadenas de caracteres muy desordenadas (como cuando estamos creando una cuenta y nos ofrecen una contraseña aleatoria).

- Select **crypt**(‘mensaje a cifrar’, **gen_salt**(‘md5’));

- También se le puede pasar el algoritmo **bf** en vez de **md5,** saldrá una cadena encriptada más larga.

- Algoritmos con texto más corto: **xdes** devuelve **20 chars**.
- Y el **des** que devuelve **13 chars**.

Cifrado Simétrico
Encriptado/Desencriptado con clave

- Encripta una cadena de forma simétrica:
- pgp_sym_encrypt(mensaje, clave) → devuelve bytes
- pgp_sym_decrypt(mensaje_encriptado, clave) → devuelve el mensaje original

- Se puede añadir el factor de compresión (por defecto es 0):
  - 0 - no compression
  - 1 - ZIP compression
  - 2 - ZLIB compression

- Y el tipo de algoritmo: cipher-algo (por defecto: aes128)
- bf, aes128, aes192, aes256, 3des, cast5

Cifrado Simétrico
Encriptado/Desencriptado con clave

- Ejemplo, pasando el nombre del algoritmo y el factor de compresión:

```
select pgp_sym_decrypt(pgp_sym_encrypt('mi mensaje', '12345', 'compress-
algo=1, cipher-algo=aes256'),  '12345','compress-algo=1, cipher-algo=aes256');
```

Conversión bytes -> string

- La función decrypt nos devuelve bytes para convertir estos en una cadena legible aplicamos la función encode(bytes, ‘escape’);

- Esta función se aplica a los datos devueltos por decrypt:

  Select **encode**(decrypt(encrypt(‘mi mensaje’, ‘clave’, ‘algoritmo’),
  ‘clave’, ‘algoritmo’), **‘escape’);**

- También podemos utilizar la función convert_from:
- select convert_from(decrypt('\x34591627f9c8eae417fc7cbbf458592c','1234','aes'),'SQL_ASCII');

Cifrado Asimétrico

- Generación de claves:
  - Necesitamos el programa gpg (disponible en Linux)
  - Para Windows: https://www.gpg4win.org/get-gpg4win.html

- Para generar las claves:
- Desde la consola: … program files (x86)\bin\ →
  - **gpg --full-generate-key**
  - Nos realiza una serie de preguntas

Secuencia programa gpg 1 de 2

```
Por favor seleccione tipo de clave deseado:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sólo firmar)
  (14) Existing key from card
Su elección: 1
las claves RSA pueden tener entre 1024 y 4096 bits de longitud.
¿De qué tamaño quiere la clave? (3072) 2048
El tamaño requerido es de 2048 bits
Por favor, especifique el período de validez de la clave.
         0 = la clave nunca caduca
      <n>  = la clave caduca en n días
      <n>w = la clave caduca en n semanas
      <n>m = la clave caduca en n meses
      <n>y = la clave caduca en n años
¿Validez de la clave (0)?
La clave nunca caduca
¿Es correcto? (s/n) s

GnuPG debe construir un ID de usuario para identificar su clave.

Nombre y apellidos: server postgresql
Dirección de correo electrónico: server@gmail.com
Comentario: coment
Ha seleccionado este ID de usuario:
```

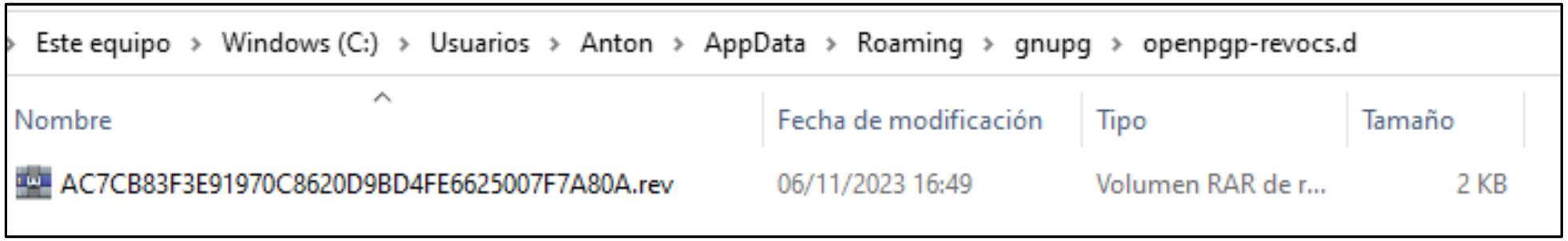
Secuencia programa gpg 2 de 2

```
Nombre y apellidos: server postgresql
Dirección de correo electrónico: server@gmail.com
Comentario: coment
Ha seleccionado este ID de usuario:
    "server postgresql (coment) <server@gmail.com>"

¿Cambia (N)ombre, (C)omentario, (D)irección o (V)ale/(S)alir? ok
¿Cambia (N)ombre, (C)omentario, (D)irección o (V)ale/(S)alir? V
Es necesario generar muchos bytes aleatorios. Es una buena idea realizar
alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usar
la red y los discos) durante la generación de números primos. Esto da al
generador de números aleatorios mayor oportunidad de recoger suficiente
entropía.
Es necesario generar muchos bytes aleatorios. Es una buena idea realizar
alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usar
la red y los discos) durante la generación de números primos. Esto da al
generador de números aleatorios mayor oportunidad de recoger suficiente
entropía.
gpg: creado el directorio 'C:\\Users\\Anton\\AppData\\Roaming\\gnupg\\openpgp-revocs.d'
gpg: certificado de revocación guardado como 'C:\\Users\\Anton\\AppData\\Roaming\\gnupg\\openpgp-revocs.d\\AC7CB83F3E919
70C8620D9BD4FE6625007F7A80A.rev'
claves pública y secreta creadas y firmadas.

pub   rsa2048 2023-11-06 [SC]
      AC7CB83F3E91970C8620D9BD4FE6625007F7A80A
uid                      server postgresql (coment) <server@gmail.com>
sub   rsa2048 2023-11-06 [E]


D:\Program Files (x86)\GnuPG\bin>
```

Pide la frase secreta (recordarla)

Clave generada

- Se almacena en:



- Cifrado asimétrico:
  - https://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica

En la Base de datos

- La BD protege la clave privada y la frase.
- Y a los clientes que puedan ver los datos se les da la clave pública.
- Podemos listas las claves con **gpg --list-secret-keys**

```
D:\Program Files (x86)\GnuPG\bin>gpg --list-secret-keys
[keyboxd]
---------
sec   rsa2048 2023-11-06 [SC]
      AC7CB83F3E91970C8620D9BD4FE6625007F7A80A
uid           [  absoluta ] server postgresql (coment) <server@gmail.com>
ssb   rsa2048 2023-11-06 [E]
```

Exportar la clave

- **gpg -a --export “server postgresql” > public.key** → El nombre del usuario.
- Se puede indicar el path al archivo y lo guarda en un fichero de texto:

```
D:\Program Files (x86)\GnuPG\bin>gpg -a --export "server postgresql" > \antonio\TRABAJO\CURSOS\BASES DE DATOS\PostgreSQL
\claves\public.key
```

- Después se puede exportar la clave privada:
- **gpg -a --export-secret-keys “server postgresql” > secret.key**
- Para exportar nos va a pedir la frase secreta → “antonio”

En postgresql

- Creamos la BD
- Activamos la extensión pgcrypto:
  - **create extension pgcrypto**

- **Las claves no se deberían guardar en la BD.**

- Dentro de la BD se crean dos tablas:
  - Create table publica(llave text)
  - Create table privada(llave text)

En postgresql

- En cada tabla se guarda su clave correspondiente:

```
Server [localhost]:
Database [postgres]: cifrado
Port [5432]:
Username [postgres]:
Contraseña para usuario postgres:
psql (16.0)
ADVERTENCIA: El código de página de la consola (850) difiere del código
             de página de Windows (1252).
             Los caracteres de 8 bits pueden funcionar incorrectamente.
             Vea la página de referencia de psql «Notes for Windows users»
             para obtener más detalles.
Digite «help» para obtener ayuda.

cifrado=# create extension pgcrypto;
CREATE EXTENSION
cifrado=# Create table publica(llave text)
cifrado-# ;
CREATE TABLE
cifrado=# Create table privada(llave text);
CREATE TABLE
cifrado=# copy publica from 'D:\antonio\TRABAJO\CURSOS\BASES DE DATOS\PostgreSQL\claves\public.key';
COPY 30
cifrado=# copy privada from 'D:\antonio\TRABAJO\CURSOS\BASES DE DATOS\PostgreSQL\claves\secret.key';
COPY 59
cifrado=#
```

En la BD

- Se define una función para poder **cifrar**: devuelve un array de bytes: **bytea**
- Concatena la llave.
- Utilizamos la función: **pgp_pub_encrypt**(texto, dearmor(key))
- dearmor → para que el sistema entienda la llave
- Crear la función y añadirla a la BD
- Uso de la función: select cifra(‘texto cifrado’);
- Veremos la llave y el array de bytes que devuelve el mensaje encriptado.

La función cifra

```
create or replace function cifra(texto text) returns bytea as $$
declare
        linea text;
        key text;
        cifrado text;
begin
        key = '';
        for linea in select llave from publica loop
                key = key || linea || E'\n'; -- Hay que añadir un salto de línea para reconstruir la clave
        end loop;
        raise notice 'LLAVE:';
        raise notice '======';
        raise notice '%', key;
        cifrado = pgp_pub_encrypt(texto, dearmor(key));
        return cifrado;
end;
$$ language plpgsql;
```

La función descifra

```
create or replace function descifra(texto bytea) returns text
as $$
declare
        linea text;
        key text;
        descifrado text;
begin
        key = '';
        for linea in select llave from privada loop
                key = key || linea || E'\n';
        end loop;
        -- Tenemos que pasar la frase.
        descifrado = pgp_pub_decrypt(texto, dearmor(key),'antonio');
        return descifrado;
end;
$$ language plpgsql;
```

Enlaces

- Cifrado hash, básico y simétrico
  - https://www.youtube.com/watch?v=M7K_i2wdBCI

- Cifrado asimétrico:
  - https://www.youtube.com/watch?v=4TezzdzAlXA