

MEETUP #3 – CRIPTOGRAFIA COM DBMS_CRYPTO

PL/SQL CAMP – 05/09/2018

PACKAGE DBMS_CRYPTO

- Possui interface para criptografar e descriptografar CLOB, BLOB e RAW
 - VARCHAR2 deve ser transformado em RAW antes de ser criptografado.
- Suporta vários algoritmos de criptografia
 - AES - Adotado como padrão de criptografia pelo governo dos Estados Unidos.
- Possibilita a geração de HASH e MAC

RESUMO DA PACKAGE DBMS_CRYPTO

Package Feature	DBMS_CRYPTO
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY
Padding forms	PKCS5, zeroes
Block cipher chaining modes	CBC, CFB, ECB, OFB
Cryptographic hash algorithms	MD5, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), MD4
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1, HMAC_SH256, HMAC_SH384, HMAC_SH512
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER
Database types	RAW, CLOB, BLOB

Fonte: https://docs.oracle.com/database/121/ARPLS/d_crypto.htm#ARPLS65670

```

1 SELECT rawtohex(
2     utl_i18n.string_to_raw(
3         'ABCDEF',
4         'AL32UTF8'
5     )) varchar_to_raw,
6
7     utl_i18n.raw_to_char(
8         hextoraw('414243444546'),
9         'AL32UTF8'
10    ) raw_to_varchar2
11 FROM dual

```

```

SQL> ed
Wrote file afiedt.buf

```

```

1 SELECT rawtohex(
2     utl_i18n.string_to_raw(
3         'ABCDEF',
4         'AL32UTF8'
5     )) varchar_to_raw,
6     utl_i18n.raw_to_char(
7         hextoraw('414243444546'),
8         'AL32UTF8'
9     ) raw_to_varchar
10* FROM dual
SQL> /

```

VARCHAR_TO_RAW	RAW_TO_VARCHAR
414243444546	ABCDEF

VARCHAR2:

VARCHAR2 não pode ser criptografado diretamente.

Passos para criptografar:

1. Converter o VARCHAR2 para o charset AL32UTF8
2. Converter para o tipo RAW

Passos para descriptografar

1. Converter o RAW para VARCHAR2 no charset AL32UTF8
2. Converter para o charset desejado.

QUANDO USAR HASH OU MAC (MESSAGE AUTHENTICATION CODE)

- Ambos são algoritmos de via única, ou seja partindo da informação final (Hash ou MAC) não é possível voltar para a original.
- HASH é usado para assegurar integridade, ou seja, ao receber um dado é possível calcular seu HASH e compará-lo com o fornecido pelo emissor.
 - EX: HASH em MD5 fornecido para o Apache HTTP Server
 - (818adca52f3be187fe45d6822755be95 *httpd-2.4.34.tar.bz2)
 - Fonte: <https://httpd.apache.org/download.cgi>
- MAC é similar ao HASH mas para que o receptor possa calculá-lo é necessário possuir a CHAVE usada pelo emissor.
 - Pode ser usado, por exemplo, para validar troca de mensagens ou arquivos entre sistemas que conheçam a chave um do outro.

GERAÇÃO E ARMAZENAMENTO DAS CHAVES

- DBMS_CRYPTO possui mecanismos de geração de chaves aleatórias. (Uma chave fraca compromete a criptografia)
- O armazenamento e manutenção da chave é responsabilidade do desenvolvedor.
- A criptografia gerada pelo DBMS_CRYPTO ocorre no servidor, não no cliente.
 - A conexão precisa ser protegida, caso contrário a informação pode ser capturada no “caminho”.
- Se um sistema possui uma package que gerencia as chaves, ela pode ser ofuscada pelo utilitário WRAP.
 - As chaves armazenadas no banco devem ser criptografadas
- OBS: Não usar DBMS_RANDOM para gerar chaves de criptografia

PONTOS DE ATENÇÃO:

- Existem funções e procedures para a criptografia

```
DBMS_CRYPTO.ENCRYPT(  
  src IN RAW,  
  typ IN PLS_INTEGER,  
  key IN RAW,  
  iv  IN RAW  
  DEFAULT NULL)  
RETURN RAW;
```

```
DBMS_CRYPTO.DECRYPT(  
  src IN RAW,  
  typ IN PLS_INTEGER,  
  key IN RAW,  
  iv  IN RAW DEFAULT NULL)  
RETURN RAW;
```

```
DBMS_CRYPTO.DECRYPT(  
  dst IN OUT NOCOPY BLOB,  
  src IN BLOB,  
  typ IN PLS_INTEGER,  
  key IN RAW,  
  iv  IN RAW  
  DEFAULT NULL);
```

```
DBMS_CRYPTO.ENCRYPT(  
  dst IN OUT NOCOPY BLOB,  
  src IN BLOB,  
  typ IN PLS_INTEGER,  
  key IN RAW,  
  iv  IN RAW  
  DEFAULT NULL);
```

Usar as funções para
criptografar RAW

Usar as procedures para tipos
LOB (CLOB e BLOB)

PROBLEMAS DE SEGURANÇA QUE A CRIPTOGRAFIA NÃO RESOLVE

- Falhas no controle de acesso e permissões.
 - O controle dos acessos deve ser bem planejado independentemente do uso de criptografia
- Mau uso do poder dos DBAs.
 - Controlar os poderes dos administradores pode ser feito pelo Oracle Database Vault.
- Criptografar tudo não deixa os dados seguros.
 - Essa abordagem impactaria a performance do sistema
 - Uma troca de chave obrigaria a descriptografar e recriptografar todo o banco de dados.
 - A chave precisaria ser compartilhada com todos os usuários.

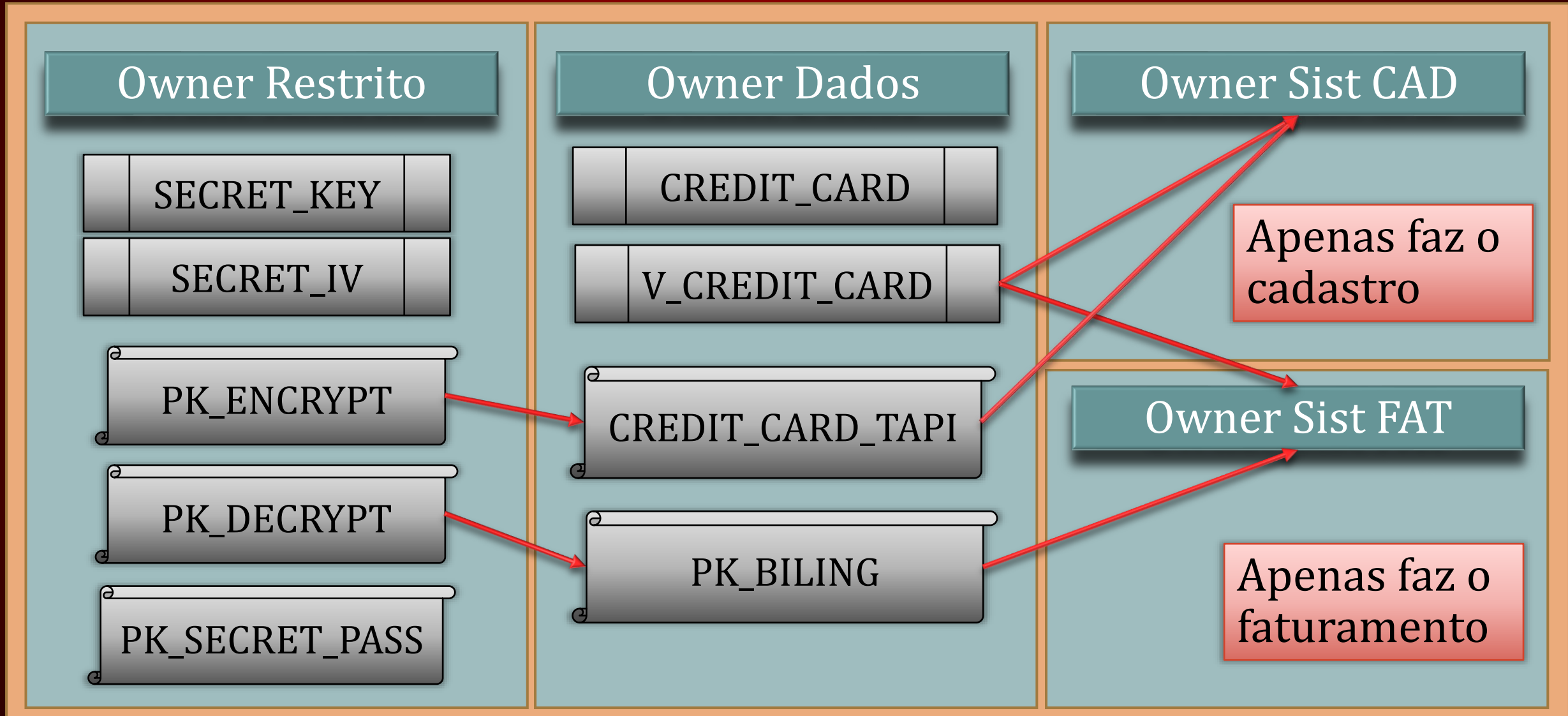
DESCRIÇÃO DO EXEMPLO CRIADO PARA O MEETUP

- Simula um cadastro de Cartões de Crédito.
 - Número (Varchar2) e Assinatura (BLOB) serão armazenados criptografados
- Foram criados vários Owners para possibilitar o controle de acesso e aumentar a segurança no processo.
 - Owner para controlar as Chaves
 - Owner par armazenar os dados dos cartões
 - Owner para a conexão com o serviço de cadastro
 - Owner para a conexão com o serviço de faturamento
- Exemplo bem simples, apenas para exemplificar um processo de criptografia de Texto e Binário.

DESAFIOS DO EXEMPLO

- Permitir alteração da chave sem que todos os cartões precisem ser recriptografados.
- Garantir que a chave não será alterada enquanto está sendo usada.
- Permitir o uso de vetor de inicialização (IV) para evitar que textos iguais gerem criptografias iguais.
- Garantir que os Owners usados pelos serviços não conseguirão acessar diretamente os mecanismos de descriptografia e tão pouco os dados criptografados.

REPRESENTAÇÃO GRÁFICA DO EXEMPLO



OWNER COM INFORMAÇÕES RESTRITAS

Owner Restrito

SECRET_KEY

SECRET_IV

PK_ENCRYPT

PK_DECRYPT

PK_SECRET_PASS

Armazena as chaves geradas com o intervalo (data_inic e data_fim) de validade

Armazena o vetor de inicialização que é gerado para cada chamada a PK_ENCRYPT

Criptografa um VARCHAR2 ou um BLOB

Descriptografa um VARCHAR2 ou um BLOB

Controla o retorno/geração das chaves e o armazenamento dos vetores de inicialização

GERAÇÃO DE CHAVES

```
SQL> @/home/oracle/Documents/Meetup/Test/test_key_generation
```

```
INICIO
```

```
-----  
25/08/2018 21:33:49
```

```
Key: 49s =BA70E30D23EEF4327A98800760BB28F9DD8  
New 01-53s = B875BA2A878D369D66DC67FD1CA336B7450  
New 02-55s = 2F8A37ECEE5CF353C3F44BFFE1AB350266E  
New 03-01s = 98A6A1719A00A52D8D2D694F78C69544C08  
New 04-04s = 73743E7AF871F81550B2AE1BAF4CA52270F  
New 05-07s = FDD15C6DEAE889B6A5ED3608448F5A6A0D2
```

```
PL/SQL procedure successfully completed.
```

```
FIM
```

```
-----  
25/08/2018 21:34:06
```

```
SQL> █
```

Script de teste: test_key_generation.sql

Características do controle das chaves do exemplo:

- Regras controladas por:
 - pk_secret_pass.generate_new_key;
- Uma nova chave só pode ser criada se ninguém estiver usando a chave atual.
- Uma chave deve ficar ativa por ao menos 1s.

	START_DATE	END_DATE	TIME_ACTIVE	RAW_KEY
1	25/08/2018 21:34:07	(null)	324s	...48F5A6A0D2
2	25/08/2018 21:34:04	25/08/2018 21:34:06	2s	...F4CA52270F
3	25/08/2018 21:34:01	25/08/2018 21:34:03	2s	...8C69544C08
4	25/08/2018 21:33:55	25/08/2018 21:34:00	5s	...1AB350266E
5	25/08/2018 21:33:53	25/08/2018 21:33:54	1s	...CA336B7450
6	25/08/2018 21:24:06	25/08/2018 21:33:52	586s	...0BB28F9DD8

Algoritmo usado:

```
dbms_crypto.encrypt_aes256 +  
dbms_crypto.chain_cbc +  
dbms_crypto.pad_pkcs5;
```

Object de retorno :

```
1 create or replace TYPE to_encrypted_blob_data  
2 AUTHID definer AS OBJECT (  
3     encrypted_data    BLOB,  
4     reference_date    DATE,  
5     iv_id            NUMBER,  
6     /**      * Creates an object of to_encrypted_da  
7     ↓ CONSTRUCTOR FUNCTION to_encrypted_blob_data (  
13         reference_date IN DATE DEFAULT SYSDATE  
14     ) RETURN SELF AS RESULT,  
15  
16     /*Just to make tests easier*/  
17     ↓ MEMBER FUNCTION to_string RETURN VARCHAR2,  
18  
19     /*Just to make tests easier*/  
20     member           procedure print_line  
21 );|
```

PK_ENCRYPT

Características :

- Pode criptografar VARCHAR2 e BLOB
- Usa a chave ativa no SYSDATE
- Sempre gera e armazena um IV aleatório
 - autonomous_transaction
- Retorna a informação criptografada com o ID do IV e a data de referência da chave em um Objeto
 - to_encrypted_raw_data
 - to_encrypted_blob_data

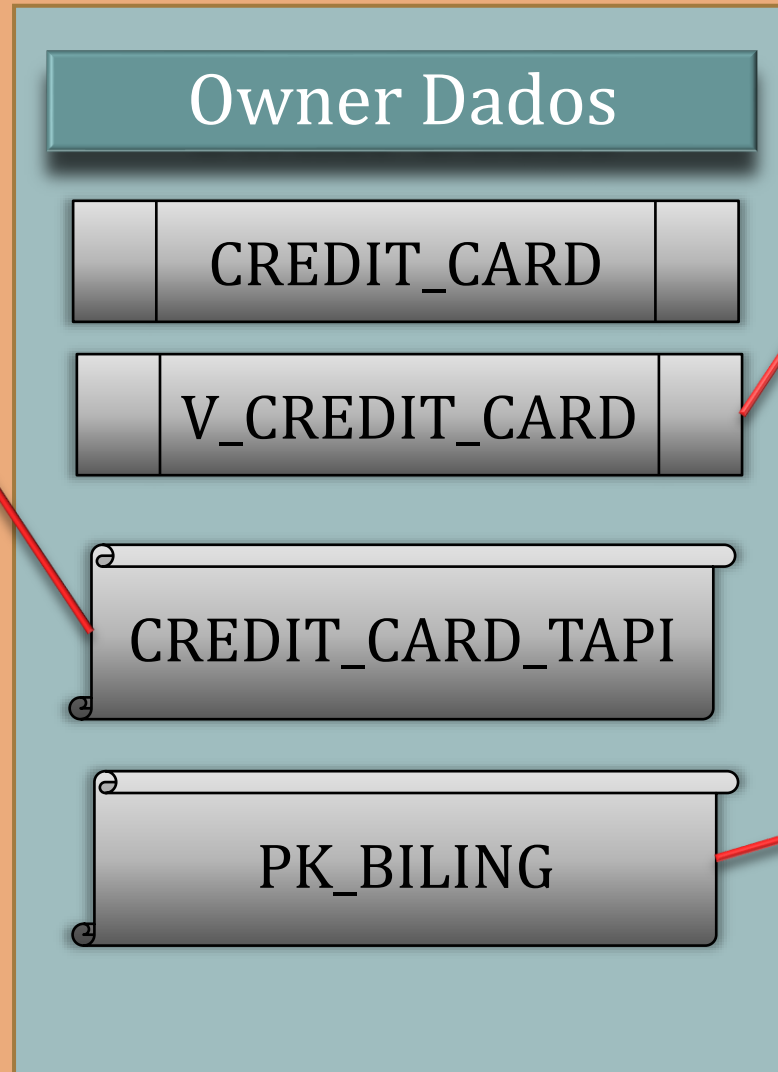
PK_DECRYPT

Características :

- Descriptografa os Objects retornados pela PK_ENCRYPT
- Usa a chave ativa na data de referência do Object
- Usa o ID do IV do Object
- Retorna a informação no formato VARCHAR2 ou BLOB

OWNER COM OS DADOS DOS CARTÕES

API para controlar
Insert, Update e
Delete da tabela
CREDIT_CARD



View com os dados
não sigilosos do
Cartão de Crédito

Simula uma package
que faria uso de um
cartão de crédito
para um faturamento

```
SQL> @/home/oracle/Documents/Meetup/Test/test_insert_card
```

```
INICIO
```

```
-----  
25/08/2018 23:14:28
```


```
PL/SQL procedure successfully completed.
```

```
FIM
```

```
-----  
25/08/2018 23:14:49
```

```
SQL> █
```

```
/* insert*/  
FUNCTION ins (  
    p_card_number    IN VARCHAR2,  
    p_signature       IN BLOB  
) RETURN credit_card.credit_card_id%TYPE;
```



```
/*encrypt credit card*/  
v_encrypted_credit_card := pk_encrypt.encrypt(p_original_text    => p_card_number);  
v_encrypted_signature   := pk_encrypt.encrypt(p_original_blob    => p_signature);
```

	CREDIT_CARD_ID	FINAL_CARD_NUMBER	ENCRYPTED_CARD_NUMBER	DECRYPTED_CARD_NUMBER
1	2597	7450	...01ABA83C6C646D5	0236 4854 9447 7450
2	2598	3833	...59256C841062F7D	7587 8654 5251 3833
3	2599	2952	...469767FE7F185DE	0906 4733 6611 2952
4	2600	2259	...09A0419DAA0938C	4116 3271 7806 2259
5	2601	9949	...4DDC44EC5C8C405	2638 4577 1844 9949
6	2602	7487	...4B51217716CF15D	4551 0348 0010 7487
7	2603	1328	...A98D25322DBD7F1	3743 2913 0435 1328

CREDIT_CARD_TAPI

Script de teste: test_insert_card.sql

Característica:

- Encapsula a regra para inserir, alterar ou excluir dados na tabela CREDIT_CARD
- Usa a package PK_ENCRYPT para criptografar a assinatura e o número do cartão de crédito

PK_BILLING

Script de teste: test_billing.sql

Característica:

- Simula um processo de faturamento que faz uso do cartão criptografado.
- Apenas faz o OUTPUT dos dados do cartão para exemplificar a descriptografia.

```
1 SET SERVEROUTPUT ON
2
3 DECLARE
4   P_CREDIT_CARD_ID NUMBER;
5 BEGIN
6   P_CREDIT_CARD_ID := 58;
7
8   sis_dados.PK_BILLING.INVOICE(
9     P_CREDIT_CARD_ID => P_CREDIT_CARD_ID
10  );
11 END;
12 /
```

```
14 dbms_output.put_line(' DECRYPTED: ' ||
15   sis_restrito.pk_decrypt.decrypt(v_card.card_number) );
```

```
SQL> @test_billing.sql
===== I N V O I C E =====
CARD NUMBER
ENCRYPTED: 8739B2989FD273920649F0BBBD9C4B77DF9D4492C8C528E27F6DDC5C67E9D466
DECRYPTED: 9136 3041 2528 8566
```

PL/SQL procedure successfully completed.

MATERIAL DE APOIO

- Manually Encrypting Data – https://docs.oracle.com/database/121/DBSEG/data_encryption.htm
- DBMS_CRYPTO - https://docs.oracle.com/database/121/ARPLS/d_crypto.htm
- UTL_I18N - https://docs.oracle.com/database/121/ARPLS/u_i18n.htm
- DBMS_LOCK - https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_lock.htm
- AES - https://pt.wikipedia.org/wiki/Advanced_Encryption_Standard
- Fontes dos exemplos disponíveis em:
<https://github.com/plsqlcamp/Meetup/tree/master/003%20180903/Source>

OBRIGADO!!!

RODRIGO EDSON FERNANDES

- E-mail: rodrigoedson@gmail.com
- LinkedIn: <https://www.linkedin.com/in/rodrigo-edson-fernandes/>