

BaroCRYPT Guide(PostgreSQL)

Index

Index	0
1. PostgreSQL encryption/decryption module.....	1
1.1 pgcrypto module	1
1.2 Pre-confirmation items.....	1
1.3 Encryption/Decryption function.....	2
1.4 Create encryption/decryption function.....	3
1.5 Confirm creation of encryption/decryption functions.....	4
1.6 Encryption/decryption function test.....	5
2. About BaroCRYPT.....	7

1. PostgreSQL encryption/decryption module

1.1 pgcrypto module

The pgcrypto module provides cryptographic functions for PostgreSQL.

This module is considered a trusted module. That is, a user other than the super user who has the CREATE privilege on the current database can install it.

The raw encryption function of the pgcrypto module is as follows.

- convert_to/convert_from: convert/restore string
- encode/decode: hexadecimal encoding/decoding
- encrypt/decrypt: encrypt/decrypt

For encryption, convert to utf8, encrypt using the 'aes' algorithm as an encryption key, and then encode the value in hexadecimal (hex).

Ex) `encode(encrypt(convert_to('name005', 'utf8'), 'ENC_KEY', 'aes'), 'hex')`

ENC_KEY should be replaced with the actual encryption key to be used, and utf8, aes, and hex are fixed values.

Decryption is the reverse of encryption. After decoding the hexadecimal value, the encryption is released, and the value is converted back to utf8.

Ex) `convert_from(decrypt(decode(mem_name, 'hex'), 'ENC_KEY', 'aes'), 'utf8')`

ENC_KEY should be replaced with the actual decryption key to be used, and utf8, aes, and hex are fixed values

1.2 Pre-confirmation items

1) Check currently installable modules

```
[root@baropam ~]# su - postgres
Verification code:

[postgres@baropam ~]$ psql barocryptdb
psql (10.17)
Type "help" for help.

postgres=# select * from pg_catalog.pg_available_extensions;
```

If not, proceed as follows.

2) Find the settings file

[Module name].control is the configuration file for installing the module.

```
$ find / -name pgcrypto.control -print
```

If you cannot find it, you must install postgresql-contrib using the following command.

```
$ yum install postgresql-contrib -y
```

3) Compiling and installing extension modules

```
$ cd /root/postgresql-1x.x/contrib/pgcrypto
$ ./configure
$ make -f makefile
$ make install
```

If you go through the above process, the following files will be copied.

```
/usr/local/pgsql/share/contrib/pgcrypto.sql
/usr/local/pgsql/share/contrib/uninstall_pgcrypto.sql
/usr/local/pgsql/lib/pgcrypto.so
```

4) Install expansion module

[Important] Connect to the relevant DB administrator to install.
Check who the owner of the DB is using the WI command.
Basically, if no option values are specified, public schema functions are created.

```
postgres=# create extension pgcrypto;
```

5) Check currently installed modules

```
postgres=# select * from pg_catalog.pg_extension;
```

6) Function test

```
postgres=# select encode(encrypt(convert_to('barocrypt', 'utf-8'), 'utf-8', 'AES'), 'hex');
```

1.3 Encryption/Decryption function

1) TO_ENCRYPTS function

- NAME
TO_ENCRYPTS
- SYNOPSIS
char TO_ENCRYPTS(data char)
varchar TO_ENCRYPTS(data varchar)
txt TO_ENCRYPTS(data text)
- DESCRIPTION

A function that encrypts data.

data: data to encrypt

- RETURN VALUES
- return encrypted data

2) TO_DECRYPTS function

- NAME
- TO_DECRYPTS

- SYNOPSIS
- char TO_DECRYPTS(data char)
- varchar TO_DECRYPTS(data varchar)
- txt TO_DECRYPTS(data text)

- DESCRIPTION
- A function to decrypt data.
- data: data to decrypt

- RETURN VALUES
- Return the decrypted data

1.4 Create encryption/decryption function

1) Install extension modules

```
postgres=# CREATE EXTENSION pgcrypto;
CREATE EXTENSION
postgres=#
```

2) Create encryption/decryption functions

```
postgres=# CREATE OR REPLACE FUNCTION TO_ENCRYPTS(data char)
postgres=# RETURNS char AS
postgres=# $$ BEGIN
postgres$$ RETURN encode(encrypt(convert_t_to(data, 'utf8'), 'Encryption KEY', 'aes'), 'hex');
postgres$$ END; $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE OR REPLACE FUNCTION TO_DECRYPTS(data char)
postgres=# RETURNS char AS
postgres=# $$ BEGIN
postgres$$ RETURN convert_t_from(decrypt(decode(data, 'hex'), 'Decryption KEY', 'aes'), 'utf8');
postgres$$ END; $$
postgres=# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE OR REPLACE FUNCTION TO_ENCRYPTS(data varchar)
postgres=# RETURNS varchar AS
postgres=# $$ BEGIN
```

```

postgres## RETURN encode(encrypt(convert_to(data, 'utf8'), 'Encryption KEY', 'aes'), 'hex');
postgres## END; $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres-# CREATE OR REPLACE FUNCTION TO_DECRYPTS(data varchar)
postgres-# RETURNS varchar AS
postgres-# $$ BEGIN
postgres## RETURN convert_from(decrypt(decode(data, 'hex'), 'Decryption KEY', 'aes'), 'utf8');
postgres## END; $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres-# CREATE OR REPLACE FUNCTION TO_ENCRYPTS(data text)
postgres-# RETURNS text AS
postgres-# $$ BEGIN
postgres## RETURN encode(encrypt(convert_to(data, 'utf8'), 'Encryption KEY', 'aes'), 'hex');
postgres## END; $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres-# CREATE OR REPLACE FUNCTION TO_DECRYPTS(data text)
postgres-# RETURNS text AS
postgres-# $$ BEGIN
postgres## RETURN convert_from(decrypt(decode(data, 'hex'), 'Decryption KEY', 'aes'), 'utf8');
postgres## END; $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres-#
    
```

1.5 Confirm creation of encryption/decryption functions

```

postgres-# \df
    
```

List of functions				
Schema	Name	Result data type	Argument data types	Type
public	armor	text	bytea	normal
public	armor	text	bytea, text[], text[]	normal
public	crypt	text	text, text	normal
public	dearmor	bytea	text	normal
public	decrypt	bytea	bytea, bytea, text	normal
public	decrypt_iv	bytea	bytea, bytea, bytea, text	normal
public	digest	bytea	bytea, text	normal
public	digest	bytea	text, text	normal
public	encrypt	bytea	bytea, bytea, text	normal
public	encrypt_iv	bytea	bytea, bytea, bytea, text	normal
public	gen_random_bytes	bytea	integer	normal
public	gen_random_uuid	uuid		normal
public	gen_salt	text	text	normal
public	gen_salt	text	text, integer	normal
public	hmac	bytea	bytea, bytea, text	normal
public	hmac	bytea	text, text, text	normal

public	pgp_armor_headers	SETOF record	text, OUT key text, OUT value text	normal
public	pgp_key_id	text	bytea	normal
public	pgp_pub_decrypt	text	bytea, bytea	normal
public	pgp_pub_decrypt	text	bytea, bytea, text	normal
public	pgp_pub_decrypt	text	bytea, bytea, text, text	normal
public	pgp_pub_decrypt_bytea	bytea	bytea, bytea	normal
public	pgp_pub_decrypt_bytea	bytea	bytea, bytea, text	normal
public	pgp_pub_decrypt_bytea	bytea	bytea, bytea, text, text	normal
public	pgp_pub_encrypt	bytea	text, bytea	normal
public	pgp_pub_encrypt	bytea	text, bytea, text	normal
public	pgp_pub_encrypt_bytea	bytea	bytea, bytea	normal
public	pgp_pub_encrypt_bytea	bytea	bytea, bytea, text	normal
public	pgp_sym_decrypt	text	bytea, text	normal
public	pgp_sym_decrypt	text	bytea, text, text	normal
public	pgp_sym_decrypt_bytea	bytea	bytea, text	normal
public	pgp_sym_decrypt_bytea	bytea	bytea, text, text	normal
public	pgp_sym_encrypt	bytea	text, text	normal
public	pgp_sym_encrypt	bytea	text, text, text	normal
public	pgp_sym_encrypt_bytea	bytea	bytea, text	normal
public	pgp_sym_encrypt_bytea	bytea	bytea, text, text	normal
public	to_decrypts	character	data character	normal
public	to_decrypts	character varying	data character varying	normal
public	to_decrypts	text	data text	normal
public	to_encrypts	character	data character	normal
public	to_encrypts	character varying	data character varying	normal
public	to_encrypts	text	data text	normal

(42 rows)
postgres=#

1.6 Encryption/decryption function test

1) Encryption function (TO_ENCRYPTS)

```
postgres=# SELECT TO_ENCRYPTS('qwer1234');
 to_encrypts
-----
2b9a5a43f720621f1c791f8882a75195
(1 row)
postgres=#
```

2) Decryption function (TO_DECRYPTS)

```
postgres=# SELECT TO_DECRYPTS('2b9a5a43f720621f1c791f8882a75195');
 to_decrypts
-----
qwer1234
(1 row)
postgres=#
```


2. About BaroCRYPT



Version 1.0 – Official Release – 2016.12.1
Copyright © Nurit corp. All rights reserved.
<http://www.nurit.co.kr>

Company: Nurit Co., Ltd.
Registration Number: 258-87-00901
CEO: Jongil Lee
Tel: +82-2-2665-0119(Technical support, sales inquiry)
email: mc529@nurit.co.kr
Address: #913, 15, Magokjungang 2-ro, Gangseo-gu, Seoul (Magok-dong, Magok Techno Tower 2)