PGDAY FRANCE 2022

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES POUR UN CHIFFREMENT DE DONNÉES TRANSPARENT

DataBene

Assistance Technique – Support – Administration à Distance – Conseil

Nous recrutons des profils ventes et techniques !

LES ORIGINES DU PROJET

LE CHIFFREMENT TRANSPARENT DE DONNÉES

(Au Niveau du Bloc de Données)

LUKS

Chiffrement des blocs des disques durs
Niveau en dessous du système de fichiers

1 Master Key
8 ou 32 mots de passe pour déverrouiller la Master Key

Rotation de la clé
- déchiffrer tout le disque
- Changer la Master Key
- Rechiffrer tout le disque
- Le tout en une seule fois => service arrêté longtemps…

- Ou avoir deux serveurs à disposition (pg_basebackup + catch up)
    - Base de données uniquement (si fichiers applicatifs en sus… KO)

- Base de données de 60 To

LE CHIFFREMENT DE DONNÉES

(Au Niveau de l'Attribut d'une Table)

PGCRYPTO OU LIBSODIUM

Chiffrement d’un attribut

- Appel pg_crypt()
- Impact fort sur le code de l’application

LE CHIFFREMENT TRANSPARENT DE DONNÉES

(Au Niveau de l’Attribut d’une Table)

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE TYPE name (
 INPUT = input_function,
 OUTPUT = output_function
 [ , RECEIVE = receive_function ]
 [ , SEND = send_function ]
 [ , TYPMOD_IN = type_modifier_input_function ]
 [ , TYPMOD_OUT = type_modifier_output_function ]
 [ , ANALYZE = analyze_function ]
 [ , SUBSCRIPT = subscript_function ]
 [ , INTERNALLENGTH = { internallength | VARIABLE } ]
 [ , PASSEDBYVALUE ]
 [ , ALIGNMENT = alignment ]
 [ , STORAGE = storage ]
 [ , LIKE = like_type ]
 [ , CATEGORY = category ]
 [ , PREFERRED = preferred ]
 [ , DEFAULT = default ]
 [ , ELEMENT = element ]
 [ , DELIMITER = delimiter ]
 [ , COLLATABLE = collatable ]
)
```

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE TYPE name (
 INPUT = input_function,
 OUTPUT = output_function
 [ , RECEIVE = receive_function ]
 [ , SEND = send_function ]
 [ , TYPMOD_IN = type_modifier_input_function ]
 [ , TYPMOD_OUT = type_modifier_output_function ]
 [ , ANALYZE = analyze_function ]
 [ , SUBSCRIPT = subscript_function ]
 [ , INTERNALLENGTH = { internallength | VARIABLE } ]
 [ , PASSEDBYVALUE ]
 [ , ALIGNMENT = alignment ]
 [ , STORAGE = storage ]
 [ , LIKE = like_type ]
 [ , CATEGORY = category ]
 [ , PREFERRED = preferred ]
 [ , DEFAULT = default ]
 [ , ELEMENT = element ]
 [ , DELIMITER = delimiter ]
 [ , COLLATABLE = collatable ]
)
```

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE TYPE public.tde_text;
```

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE OR REPLACE FUNCTION public.tde_textin(pg_catalog.cstring)
RETURNS public.tde_text
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;


CREATE OR REPLACE FUNCTION public.tde_textout(public.tde_text)
RETURNS pg_catalog.cstring
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;
```

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE OR REPLACE FUNCTION public.tde_textrecv(pg_catalog.internal)
RETURNS public.tde_text
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;


CREATE OR REPLACE FUNCTION public.tde_textsend(public.tde_text)
RETURNS pg_catalog.bytea
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;
```

CRÉATION D’UN NOUVEAU TYPE DE DONNÉES

```
CREATE TYPE public.tde_text (
    LIKE          = bytea
  , INPUT         = public.tde_textin
  , OUTPUT        = public.tde_textout
  , RECEIVE       = public.tde_textrecv
  , SEND          = public.tde_textsend
);
```

CODE SOURCE DE tde_textin()

```
PG_FUNCTION_INFO_V1(tde_textin);

Datum
tde_textin(PG_FUNCTION_ARGS)
{

   char *input = PG_GETARG_CSTRING(0);
   int32 len = strlen( input );

   // Encrypt Now
   bytea *result = (bytea*)tde_encrypt( input, len );

   PG_RETURN_BYTEA_P( result );

}
```

CODE SOURCE DE tde_textout()

```
PG_FUNCTION_INFO_V1( tde_textout );

Datum
tde_textout( PG_FUNCTION_ARGS )
{

   bytea *encrypted = PG_GETARG_BYTEA_PP(0);

   // Decrypt
   text *decrypted = (text*)tde_decrypt( encrypted );

   PG_RETURN_CSTRING( TextDatumGetCString( decrypted ) );

}
```

CODE SOURCE DE tde_encrypt()

```
Datum tde_encrypt( char *decrypted, int32 len ) {

   // Stupid Encryption for Testing Purpose

   bytea *result = NULL;
   unsigned char *buffer = NULL;
   int32 i = 0;

   result = (bytea*) palloc0( VARHDRSZ + len );
   SET_VARSIZE( result, VARHDRSZ + len );

   buffer = (unsigned char*) VARDATA( result );

   for ( i = 0; i < len; i++ ) {
      buffer[i] = decrypted[i] ^ 0x55;
   }

   PG_RETURN_BYTEA_P( result );

}
```

CODE SOURCE DE tde_decrypt()

```
Datum tde_decrypt( bytea* encrypted ) {
   // stupid Reverse Function
   text *decrypted = NULL;

   char *encrypted_data = NULL, *decrypted_data = NULL;
   int32 len = 0, i = 0;

   len = VARSIZE_ANY_EXHDR( encrypted );

   decrypted = (text*) palloc0( VARHDRSZ + len );
   SET_VARSIZE( decrypted, VARHDRSZ + len );

   encrypted_data = (char*)VARDATA_ANY( encrypted );
   decrypted_data = (char*)VARDATA( decrypted );

   for ( i = 0; i < len; i++ ) {
      decrypted_data[i] = encrypted_data[i] ^ 0x55;
   }

   PG_RETURN_TEXT_P( decrypted );
}
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
CREATE SCHEMA app;

CREATE TABLE app.test ( plain text, encrypted public.tde_text );

INSERT INTO app.test VALUES ( 'plain content', 'encrypted content’ );

SELECT plain, encrypted FROM app.test;
     plain      |      encrypted
----------------+-------------------
 plain content | encrypted content
(1 row)
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
CREATE EXTENSION pageinspect;
SET bytea_output = escape;

SELECT unnest(t_attrs)
FROM heap_page_item_attrs(
     get_raw_page('app.test', 0), 'app.test'::regclass
);
       unnest
--------------------
 \035plain content      <-- ‘plain content’
 %0;6',%!01u6:;!0;!     <-- ‘encrypted content’
(2 rows)
```

DataBene

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
\COPY app.test TO STDOUT WITH CSV HEADER DELIMITER E'\t'
plain encrypted
plain content     encrypted content
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
psql -c "\COPY app.test TO STDOUT WITH CSV DELIMITER E'\t'" | \
tr 'ae' 'ea' | \
psql -c "\COPY app.test FROM STDIN WITH CSV DELIMITER E'\t'"
COPY 1
postgres=# table app.test ;
     plain     |     encrypted
---------------+-------------------
 plain content | encrypted content
 plein contant | ancryptad contant
(2 rows)
```

OPÉRATEUR D’ÉQUALITÉ ET SON COMPLÉMENT

```
CREATE FUNCTION public.tde_byteaeq(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;

CREATE FUNCTION public.tde_byteane(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;
```

OPÉRATEUR D’ÉQUALITÉ ET SON COMPLÉMENT

```
CREATE OPERATOR public.= (
    LEFTARG = public.tde_text
  , RIGHTARG = public.tde_text
  , COMMUTATOR = OPERATOR(public.=)
  , NEGATOR = OPERATOR(public.<>)
  , PROCEDURE = public.tde_byteaeq
);
```

OPÉRATEUR D'ÉQUALITÉ ET SON COMPLÉMENT

```
CREATE OPERATOR public.<> (
    LEFTARG = public.tde_text
  , RIGHTARG = public.tde_text
  , COMMUTATOR = OPERATOR(public.<>)
  , NEGATOR = OPERATOR(public.=)
  , PROCEDURE = public.tde_byteane
);
```

CODE SOURCE DE tde_byteaeq ET tde_byteane

```
PG_FUNCTION_INFO_V1( tde_byteaeq );
Datum tde_byteaeq( PG_FUNCTION_ARGS ) {
   bytea *left  = PG_GETARG_BYTEA_PP(0);
   bytea *right = PG_GETARG_BYTEA_PP(1);
   return DirectFunctionCall2(byteaeq, PointerGetDatum(left), PointerGetDatum(right));
}

PG_FUNCTION_INFO_V1( tde_byteane );
Datum tde_byteane( PG_FUNCTION_ARGS) {
   bytea *left  = PG_GETARG_BYTEA_PP(0);
   bytea *right = PG_GETARG_BYTEA_PP(1);
   return DirectFunctionCall2(byteane, PointerGetDatum(left), PointerGetDatum(right));
}
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
SELECT a.encrypted,   b.encrypted,
       a.encrypted =  b.encrypted  AS eq,
       a.encrypted <> b.encrypted AS ne
FROM app.test a CROSS JOIN app.test b;

    a.encrypted     |    b.encrypted     | eq | ne
-------------------+-------------------+----+----
 encrypted content | encrypted content | t  | f
 encrypted content | ancryptad contant | f  | t
 ancryptad contant | encrypted content | f  | t
 ancryptad contant | ancryptad contant | t  | f
(4 rows)
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
SELECT * FROM app.test WHERE encrypted = 'ancryptad contant';
     plain      |     encrypted
----------------+--------------------
 plein contant | ancryptad contant
(1 row)
```

```
SELECT * FROM app.test WHERE encrypted = 'ancryptad contant'::tde_text;
```

CAST AUTOMATIQUE DE text VERS tde_text

```
CREATE FUNCTION public.tde_cast_from_text(pg_catalog.text)
RETURNS public.tde_text
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;

CREATE CAST (text AS tde_text)
  WITH FUNCTION public.tde_cast_from_text(pg_catalog.text)
  AS IMPLICIT;
```

CODE SOURCE DE tde_cast_from_text

```
PG_FUNCTION_INFO_V1( tde_cast_from_text );

Datum tde_cast_from_text( PG_FUNCTION_ARGS ) {
    text  *decrypted = PG_GETARG_TEXT_PP(0);
    bytea *encrypted = (bytea*) tde_encrypt( VARDATA_ANY(decrypted),
                                             VARSIZE_ANY_EXHDR(decrypted));
    PG_RETURN_BYTEA_P(encrypted);
}
```

DataBene

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
SELECT * FROM app.test WHERE encrypted = 'ancryptad contant'::text;
     plain      |     encrypted
----------------+--------------------
 plein contant | ancryptad contant
(1 row)
```

DataBene

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
INSERT INTO app.test VALUES ('identique', 'identique');
SELECT a.plain, b.encrypted,
       a.plain = b.encrypted AS eq,
      a.plain <> b.encrypted AS ne
FROM app.test a CROSS JOIN app.test b;
     plain     |      encrypted      | eq | ne
---------------+---------------------+----+----
 plain content | encrypted content   | f  | t
 plain content | ancryptad contant   | f  | t
 plain content | identique           | f  | t
…
 identique     | identique           | t  | f
```

OPÉRATEURS LIKE (HACK => PAS DE %_)

```
CREATE OPERATOR public.~~ (
    LEFTARG = public.tde_text
  , RIGHTARG = public.tde_text
  , COMMUTATOR = OPERATOR(public.~~)
  , NEGATOR = OPERATOR(public.!~~)
  , PROCEDURE = public.tde_byteaeq
);
```

OPÉRATEURS NOT LIKE (HACK => PAS DE %_)

```
CREATE OPERATOR public.!~~ (
    LEFTARG = public.tde_text
  , RIGHTARG = public.tde_text
  , COMMUTATOR = OPERATOR(public.!~~)
  , NEGATOR = OPERATOR(public.~~)
  , PROCEDURE = public.tde_byteane
);
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
SELECT * FROM app.test
WHERE encrypted LIKE 'ancryptad contant'::tde_text;

     plain      |     encrypted
----------------+--------------------
 plein contant | ancryptad contant

SELECT * FROM app.test
WHERE encrypted LIKE 'ancryptad contant'::text;

     plain      |     encrypted
----------------+--------------------
 plein contant | ancryptad contant
```

OPERATOR CLASS POUR BTree ET tde_text

```
CREATE FUNCTION public.tde_bytealt(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;

CREATE FUNCTION public.tde_byteale(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;
```

Operator class

```
CREATE FUNCTION public.tde_byteage(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;

CREATE FUNCTION public.tde_byteagt(public.tde_text, public.tde_text)
RETURNS boolean
AS '$libdir/pg_encrypted_types'
LANGUAGE C IMMUTABLE STRICT SECURITY INVOKER COST 1;
```

Operator class

```
CREATE OPERATOR <# (
  FUNCTION=tde_bytealt,
  LEFTARG=tde_text,
  RIGHTARG=tde_text
);

CREATE OPERATOR <=# (
  FUNCTION=tde_byteale,
  LEFTARG=tde_text,
  RIGHTARG=tde_text
);
```

Operator class

```
CREATE OPERATOR >=# (
  FUNCTION=tde_byteage,
  LEFTARG=tde_text,
  RIGHTARG=tde_text
);
CREATE OPERATOR ># (
  FUNCTION=tde_byteagt,
  LEFTARG=tde_text,
  RIGHTARG=tde_text
);
```

Operator class

```
CREATE OPERATOR CLASS public.btree_tde_text_ops
    DEFAULT FOR TYPE public.tde_text USING btree AS
    OPERATOR        1     <#,
    OPERATOR        2     <=#,
    OPERATOR        3     =,
    OPERATOR        4     >=#,
    OPERATOR        5     >#,
    FUNCTION        1     public.bttde_textcmp( tde_text, tde_text);
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

PostgreSQL 11

```
EXPLAIN (COSTS off) SELECT * FROM app.test WHERE tde_text = ‘150';
                    QUERY PLAN
--------------------------------------------------
 Index Only Scan using test_encrypted_idx on test
   Index Cond: (encrypted = '150'::tde_text)
(2 rows)
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
CREATE INDEX ON app.test USING BTREE(encrypted btree_tde_text_ops);
EXPLAIN ANALYZE SELECT * FROM app.test WHERE encrypted = '150';
```

```
PostgreSQL 14                  QUERY PLAN
---------------------------------------------------------------------
 Seq Scan on test  (cost=0.00..1790.04 rows=50002 width=64)
                   (actual time=0.077..39.207 rows=1 loops=1)
   Filter: (encrypted = '150'::tde_text)
   Rows Removed by Filter: 100002
 Planning Time: 0.187 ms
 Execution Time: 39.245 ms
(5 rows)
```

RÉSULTAT – QUE POUVONS-NOUS FAIRE ?

```
SET enable_seqscan = off
EXPLAIN ANALYZE SELECT * FROM app.test WHERE encrypted = '150’;
PostgreSQL 14                       QUERY PLAN
---------------------------------------------------------------------
 Bitmap Heap Scan on test  (cost=943.81..2108.83 rows=50002 width=10)
                           (actual time=0.195..0.199 rows=1 loops=1)
   Recheck Cond: (encrypted = '150'::tde_text) Heap Blocks: exact=1
   ->  Bitmap Index Scan on test_encrypted_idx
       (rows=50002, actual time=0.184..0.185 rows=1 loops=1)
         Index Cond: (encrypted = '150'::tde_text)
 Planning Time: 0.329 ms
 Execution Time: 0.246 ms
```

SYNTHÈSE DES RÉSULTATS

Création d’un nouveau type de données

- **la representation interne est chiffrée de manière transparente,**
- **Support des opérateurs =, <>, ~~ et !~~**
- **Support du CAST automatique depuis le type “text”**
- **Support de l’indexage (PostgreSQL 11…)**

VERS UN CHIFFREMENT UTILE

STRUCTURE DE LA REPRESENTATION INTERNE

```
struct encrypted_value {
   int key_id;                 // CRC32
   int data_size;              //
   byte data_content[0];       //
};
```

STRUCTURE DE LA REPRESENTATION INTERNE

```
struct key {                      // Provient d’un vault ou équivalent
   int key_id;                    // CRC32
   xxx key_type;
   xxx encryption_method;
   byte salt[32];                 // Salage statique par clé
   byte encryption_key;           // La clé…
}
```

Gestion des clés

La collection des clés de chiffrement est en ajout seul.

Ajout d’une clé si aucune collision CRC32.

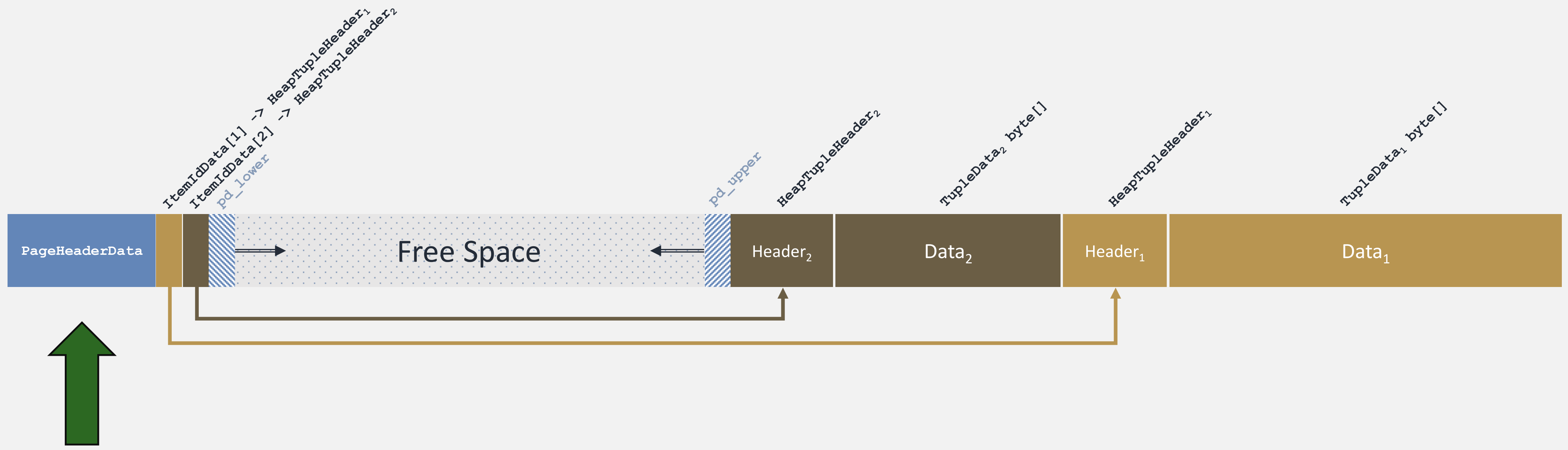
Une nouvelle clé tous les 3 mois => 100 clés sur 25 ans.

La manipulation d’une donnée chiffrée avec une clé ancienne :

- **Lorsque random() < 0.05**
- **Déchiffrement de la donnée**
- **Chiffrement avec la dernière clé**

=> Rotation des clés à la volée

Appliquée au niveau page de données…



DES QUESTIONS ?

https://data-bene.io

frederic.delacourt@data-bene.io