AS Lab Assignment: SQL∗

08 Feb. 2017

Ali Abdulmadzhidov

1. Standard database encryption.

Firstly we install our DBMS. I chose PostgreSQL. Installation is quite easy. After it we login into psql command line.

```
~ sudo apt install postgresql-9.5
~ sudo -u psql
```

Now in the command line we are going to create our test database.

```
CREATE DATABASE test1;
```

Then we can go into psql command line one more time to create table and populate test data into it.

```
~ sudo -u psql test1;
```

Here we define new function for generating random sequences and insert 10 million records to our test table.

```
CREATE OR REPLACE FUNCTION random_string(length     integer)RETURNS varchar AS $$

SELECT array_to_string(ARRAY(SELECT     substr('abcdefghijklmnopqrstuv',
trunc(random()*21+1)::int,1) FROM generate_series(1,$1)),'')
$$ LANGUAGE sql VOLATILE;

CREATE table test(a serial primary key, b varchar);

INSERT into test(b) random_string(8) from generate(1,10000000);

SELECT a, b from test LIMIT 10;
```

```
   a    |     b
--------+-----------
 106561 | kkmmpdpp
 106562 | lakdfnpl
 106563 | moflbbtt
 106564 | rtlalfna
 106565 | pfmkiheu
 106566 | ipcohreu
 106567 | ljkmljki
 106568 | ibitgbub
 106569 | gmlptqoh
 106570 | krasoere
(10 rows)
```

```sql
CREATE EXTENSION pgcrypt;

INSERT into test(b) select pgp_sym_encrypt(random_string(8), 'password') from
generate_series(1,10000000);

SELECT a, pgp_sym_decrypt(b::bytea, 'password') from test LIMIT 10;
```

```
  a   | pgp_sym_decrypt
------+-----------------
 8945 | erolqass
 8946 | esitajtb
 8947 | ursulhuk
 8948 | ocifjinn
 8949 | fhdbhbtm
 8950 | tgltpjap
 8951 | gsprfoht
 8952 | rqdfertj
 8953 | dbklcidb
 8954 | poakdafj
(10 rows)
```

```
  a   |                                                                   b
------+------------------------------------------------------------------------------------------------------------------------------------------------------------------
 8945 | \xc30d040703026482c5003c22a1ea77d23901050dce1fca567d85ae9366311fc528de38f5fa586050589d26ad57a499750420fa80a1099cd414167b7733a27daecb0dd17ca145dc5ba12d
 8946 | \xc30d040703022d9f67330efcbad073d23901be4722dd51d020faaf5872b2ecd430c1c20707648b88a5b7bf02dc14324002dbb8ca0d5b6767d80d5c63cadb26a318cd8824c708b380ce4b
 8947 | \xc30d04070302f3da099a9cf0734572d23901b2b5ab1e6a804ee3288a371bb100aeada082403d52d72287657ee901a2acf4d4ddfcb1a53dd3be15f0edc99bdde020138052da239e5e3bee
 8948 | \xc30d0407030285d6ab4be0d51bfa6dd23901f17b961ae9c22d9709fb1a972de0f437ceb156b7a1533bcbe8d0dbccc90c5777fcec62e2ed2399024d666d735c3930a70509b1493e0d6a9e
 8949 | \xc30d040703029ba9432cb802616968d23901581059f651e0f832e89f7480be89e85000001be39c005911b77417ab24e3f94009a48fd4694f91c8b81da40bbcf261fb898a4a66e3d2700f
 8950 | \xc30d0407030269924041 8b8151db62d23901524312fc85526a32c281de3d59d425b00b3e1420c7e7466a40d8dc7d38c4b6b8d89b045867e9da907c841044135f7c417fcb48db3f78a71d
 8951 | \xc30d040703023ad9df25b2deae3973d23901aa4f64ab7b4b0b4f123916662ff7c3afc0678dd867f0cd08688998ea948bc1a04dd1414e90ff5eacaba8430dfe8b376830262f62ff4dbea8
 8952 | \xc30d0407030271510bf166984af766d239013146d456c2fe9db14f81ae4ed3c02aad9ba14e5a2451a12857ec8ef8f248090138e66eb46609220f14c43881f0fbde33068c5cef4ab85655
 8953 | \xc30d040703022d9104c4096946e378d239015ca26baf587e8d7ab87e1cc9311b83307d05fee4ff718bc332585b44d5777cdbed481c5401d90e3b204dd44a7b33758582a7782f702da6f6
 8954 | \xc30d040703021519b8a81e6c4e026fd2390167d9990225ef8e9918a8fab54ddcb9bb03e0a78588ffaa66b08a553a4085c7e1d379b6fd721eb57131056bb3c3809db8eaf8b51bbe41a7a0
```

By default PgSQL supports: bf, aes128, aes192, aes256. With OpenSSL also supports 3des and cast5.
Default is aes128.

For full database encryption i used full disk encrypted installation of xubuntu.

Testing:

```sql
SELECT a,b from test LIMIT 10000;
SELECT a, pgp_sym_decrypt(b::bytea, 'password') from test LIMIT 10000;
```

	Average time(ms)
Full disk encryption	13.9
Column encryption	3903.27

- What are the attack vectors with this approach?

  There is problem with key management, so attacker can try to steal it.

  Also we can see big performance problem with column encryption, and attacker can to try DDoS that server.

  And at the end, there maybe some vulnerabilities in encryption method, that helps attacker to decode tables without key.
- Can you get the plain text from encrypted data in a way?

  It depends of encryption algorithm and key that was used
- What difference do you see between one algorithm (e.g. AES) and another algorithm of your choice?

	Blowfish	**AES**

Developed	1993	2000
Key Length	32 - 448 bits	128, 192, or 256 bits
Block Size	64 bits	128 bits

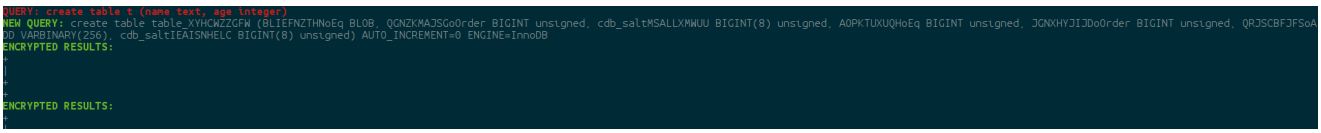
2. CryptDB.

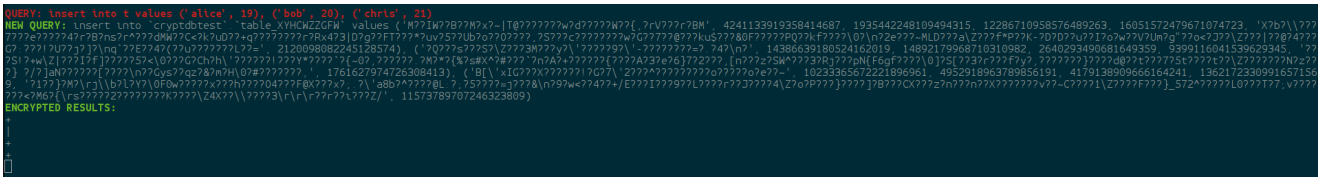
It is easy to install and set up cryptdb + mysql in docker.

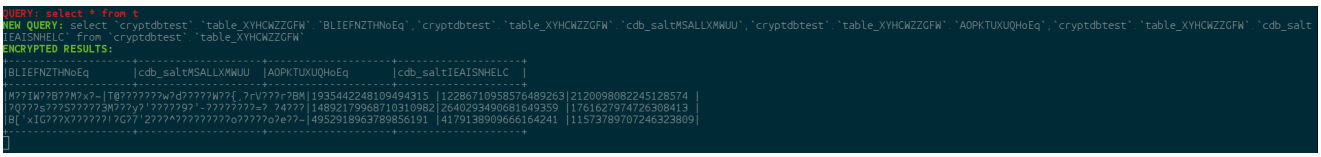
```
~ docker run -d -P --name cdb mycrypt/cryptdb
~ ssh root@192.168.59.103 -p49153
~ service mysql start
~ /opt/cryptdb/bins/proxy-bin/bin/mysql-proxy --plugins=proxy --event-threads=4 --max-open-files=1024 --proxy-lua-script=$EDBDIR/mysqlproxy/wrapper.lua --proxy-address=0.0.0.0:3307 --proxy-b-addresses=127.0.0.1:3306
```

Trying demoqueries

```
-> create database cryptdbtest;
Query OK, 1 row affected (0,26 sec)

-> create database cryptdbtest_control;
Query OK, 1 row affected (0,33 sec)

-> create table t (name text, age integer)
Query OK, 0 rows affected (0,41 sec);

-> insert into t values ('alice', 19), ('bob', 20), ('chris', 21);
Query OK, 3 rows affected (0,19 sec)

-> select * from t;
+-------+------+
| name  | age  |
+-------+------+
| alice | 19   |
| bob   | 20   |
| chris | 21   |
+-------+------+
3 rows in set (0,10 sec)

-> select * from t where name = 19;
Empty set (0,60 sec)

-> select sum(greatest(age,20)) from t;
+-----------------------+
| sum(greatest(age,20)) |
+-----------------------+
| 61                    |
+-----------------------+
1 row in set (0,67 sec)
```







- Would you use this in a production environment?

  Depends on data that i need to store. If it is not sensitive information, that is needed often and fast - i wouldn't use it, but if i need save sensitive data with rare access i tried column encryption and cryptdb.

- What are the concerns you wouldencryption have if you do so?

  Performance problems. It is 100 times more slow than regular unencrypted DB.

3. Performance.

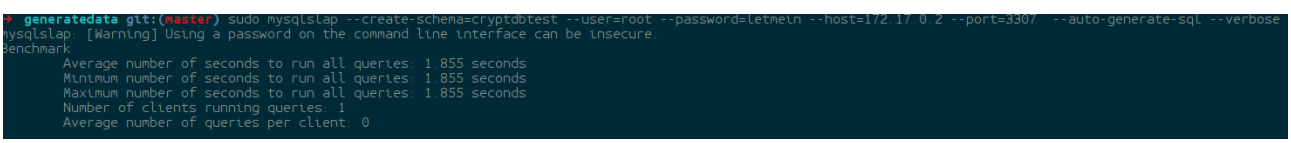
Using select queries:

```
1  SELECT * FROM cryptdbtest LIMIT 10000;
```

	Average time(ms)
Full disk encryption	13.9
Column encryption	3903.27
Cryptdb	2522

Performance with mysqlslap benchmark:

- 1 user 1 iteration



- 50 users 10 iterations