Final Project

CryptDB

Eric Marriott

csci e63 Big Data Analytics, 2014

Harvard Extension School

Professor: Zoran B. Djordjević

Background

- Most online and otherwise networked applications use a SQL-based database-management system (DBMS), such as MySQL or PostgreSQL.

- These systems are tried and tested, well supported and consistently updated FOSS (free and open-source software)

- However, these systems are vulnerable to different attacks and have several inherent security flaws and concerns

Background (DBMS security)

- An application server sends SQL queries to the DBMS server and receives results. An attack on the network, application server, or DBMS server can leak information about the queries being transmitted. An attacker may also acquire the ability to make queries themselves, allowing them to make off with and/or modify the database.
- Anyone with direct access to the DBMS server can read the plaintext database. Even if the database is encrypted server-side, it must be unencrypted to be used with the DBMS, so a hacker that has gained access to the server, or even a nosy administrator, can passively read the data and even view queries being made by the application server live.

Background (DBMS security)

- Ideally, data should only be accessible by certain people. This is especially true for personal data, such as social security numbers, bank info, credit card information, etc. This is (arguably) becoming more important in the post-Snowden world, as people are becoming more security-conscious with the personal data. Why should an online merchant have permanent access to your credit card information, even when you are not logged in? What if their servers are compromised? The news is always reporting on major data thefts. For example, the Target credit breach in December 2013 in which information on tens of millions of credit cards were stolen.

Background (Solutions)

There are several possible solutions to these problems

- Columns can be fully encrypted, and decrypted by the client, who possesses the decryption key for certain entries. This has several major disadvantages:

    - The client has to perform any of the computation required, and send data back to the application server. If the clients are end-users, there can be massive hardware variation, slowing down their systems. However, the main problem here is security (and general feasibility). For security reasons, clients cannot perform the computation for many types of operations, such as initiating bank transactions.

- SQL queries can be done on encrypted data by using fully homomorphic encryption. Homomorphic encryption is a type of encryption that allows operations to be carried out on encrypted data, resulting in an encrypted result, without every un-encrypting the data. A fully homomorphic encryption system allows for any operation to be performed on the encrypted data without ever decrypting it. Until 2009, this was only theoretically possible, but a working implementation was created by an IBM researcher. However, this technique is computationally VERY expensive, making it useless in most applications for DBMSes, where any performance at all is desired.

http://www-03.ibm.com/press/us/en/pressrelease/27840.wss

Background (CryptDB)

CryptDB is a piece of software developed by researchers at MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL), to solve some of these issues.

- CryptDB is actually a misnomer, as it is not a DBMS, it is a sort of proxy layer between the user/application server and an *unmodified* DBMS.
- CryptDB sits between an application and the DBMS server. The application connects to CryptDB and authenticates with a password-protected keychain stored on the proxy server (in theory it doesn't matter too much where the keychain is stored, as long as the password is sufficiently strong).
- CryptDB processes the SQL queries sent by the application server on encrypted data stored on the DBMS server - the DBMS server never sees unencrypted data, so anyone that has compromised the server cannot understand any of the data in the database.
- If the proxy server is compromised, the only query information that that attacker can possibly get is from users that are logged in, as those keychains are unencrypted.

How it Works

- CryptDB uses an ‘onion’ style encryption scheme, where entries in the database are stored under layers of different encryption schemes. CryptDB has 6 layer types - 1 ‘random’ layer with strong encryption, and 5 layers of different encryption schemes which support various operations on encrypted data. These layers make up different ‘onions’, which each get their own column.

- The ‘random’ layer RND, which is the outer layer of encryption. This layer is very secure, using industry-standard AES and Blowfish (depending on data type). This layer is not homomorphic, and thus is ‘peeled’ off when any operation is done on the column. Leaks no information about the data.

- The homomorphic layer HOM which uses the homomorphic Paillier algorithm, which is used to add INT values. This layer is also very secure.

- The SEARCH layer, which is used for searching for a text value using the SQL LIKE operation. It is secure, but can only work on text values.

How it Works

- The deterministic layer DET, which is used solely for equality determination (for instance the SQL query “SELECT * FROM table WHERE number=’one’;” (however SEARCH may be better for this particualar operation). Secure but weaker than HOM or SEARCH, starts off with a layer of RND on top until needed.
- The ordering layer OPE (Order-Preserving-Encryption), which reveals the order of column entries (does not give away values). This is kept under a layer of RND, as it leaks order (as is its purpose).
- The equality and range-join layer OPE-JOIN, is a new encryption scheme developed by the developers of CryptDB. This layer allows for equality JOIN operations, as well as range join. This leaks both order and equality, and is kept under a layer of OPE (if range is all that is needed, there is no reason to remove OPE) and a layer of RND. Note: JOIN is the same as OPE-JOIN, just not under the OPE layer, so does not reveal order (only used for equality join).

How it Works

- Each column is actually secured with 3 onion columns in the database. These columns are the onion equality column (layers RND,DET,JOIN) the onion order column (RND,OPE,OPE-JOIN), and (if a text column) onion search column (SEARCH), or (if an INT column) onion add column (HOM).

- Currently, once an onion layer is peeled off, it is not put back on again. This is to save computation time. As you will see, the initially decryption is what takes up most of the query time.

Installation and Requirements

- Install dependencies for building CryptDB using ‘official’ install script on an updated Ubuntu 12.04 or 13.04 system

$ sudo apt-get install ruby git

- Get CryptDB source, compile using official install script

$ git clone -b public git://g.csail.mit.edu/cryptdb

$ echo "export EDBDIR=$HOME/cryptdb" >> ~/.bashrc && source ~/.bashrc

$ cd cryptdb && ./scripts/install.rb

That’s it! Ready to start using CryptDB...

Using CryptDB

- Start CryptDB proxy

$ $EDBDIR/bins/proxy-bin/bin/mysql-proxy --plugins=proxy --event-threads=4 \ --max-open-files=1024 \ --proxy-lua-script=$EDBDIR/mysqlproxy/wrapper.lua

--proxy-address=127.0.0.1:3307 \ --proxy-backend-addresses=localhost:3306

- Connect to the DBMS through CryptDB proxy

$ mysql -u root -pletmein -h 127.0.0.1 -P 3307

- That’s it, you can now perform queries.

Demonstration

Table viewed through CryptDB proxy

Tatyana	Mosley	4034	6814B148-7DEE-03D6-1441-70E9061B814A	143	95246
Aurora	Hensley	5064	B48C6854-3B67-78C3-9295-8F52085B6CE9	153	62370
Briar	Summers	6996	BC8860C8-55DD-9911-2224-FA850DEE4E55	139	42147
Conan	Rivas	3928	FA0F544B-F471-B5E8-5E58-1580FFA21D9E	183	29387
Abigail	Kent	1246	B6A782F8-47ED-DC92-FDD6-C690D759056B	158	54145
Olivia	Goff	4694	03C6D5C0-28D1-F0A0-EC79-A91709466D4E	201	15432
Jenette	Rodriguez	503	887FCD90-15DD-90D7-3F25-79234C832BD7	173	19281
Theodore	Spencer	5064	22355BD3-0EAD-E011-1076-6999D820E3CE	156	50003
September	Johns	8642	F21B4919-4BF7-C481-46E6-3FC884A37502	1	79801
Faith	Talley	4399	F7C3F0A9-7732-8C95-354A-D74CE9E6967C	157	48443
Clare	Franco	5874	241F03A2-5792-4858-E7E7-833F67A1AAB0	133	53431
Hyacinth	Cummings	228	12AB731B-91E9-F885-38E3-066257324BFB	16	90889
William	Walters	8483	25565133-0772-7FD1-ECE1-6C8F81549995	197	17534
Harriet	Stewart	7235	8DBBD7A9-6FDC-DEA1-E0A0-9EA28A701AFC	142	45752
Fay	Charles	9946	B488913F-A4C3-2139-CB69-8CB7DE9DA59D	148	49133
Nola	Schwartz	66	CAF81DE3-EBAD-ACA0-254E-57B88766E367	166	52985
Tatiana	Terrell	7168	2F65C9C6-10E8-B2B6-45C5-4DA6A6ECC8D5	175	78572
Rinah	Marsh	5528	DA22B812-DA4C-9668-F61A-40F7253D5261	16	92195
Aileen	Holman	6616	CED91DB6-FD5B-34A0-D956-04774DBF099E	152	50281
Deborah	Watkins	6489	9A00080C-016C-142A-3EDC-620366277085	161	18130
Zane	Duran	5884	CFB5FDB4-6A98-97FB-1703-1BC84DF9790D	105	84359
Audrey	Gardner	7401	72B87966-CC08-BB3E-BE8A-1F37730364AE	164	83806
Nathaniel	Macdonald	2945	CB1474AF-691C-9CA3-8380-9ADC2929ABBD	137	74533
Rose	Woodard	3834	D277F050-C0DF-5E2D-72D4-25BD4209894F	183	73513
Quon	Spencer	4701	740A24BC-6B59-24A3-617A-DB1B4140CC4E	167	51958
Echo	Henson	2952	D6B43975-C3D9-6C62-B8C5-60865AE92A4B	136	78432
Karly	Oconnor	4627	0EB82B16-EF89-061B-90AE-C2500B9ACF30	119	96637
Scott	Dodson	6864	D15358EC-DAD2-3275-E349-ED55879CD82E	121	32436
Aspen	Levine	175	33FD8445-B198-13E9-87BD-B63A5C2451E2	203	48575
Mariam	Ashley	514	3FA5D13C-6218-E9F1-161B-0A6ECCA7A7C7	19	68504
Clinton	Brooks	2165	E6BBC7FB-5481-6777-A6DE-139237F86628	155	67847
Ignacia	Whitehead	652	4798F829-2438-583D-65AC-0764BC40240A	151	73557
Zeus	Jensen	8886	5FDA1370-93BB-6753-32AA-8468EB1C44A2	153	27880
Kennan	Lane	3076	A920AD51-9978-7F8D-C260-75DCCEE477F2	174	16339
Keiko	Byrd	5849	8456AF85-065B-7082-BDA9-F492C9BA296D	136	95755
Wayne	Hamilton	4028	551D4669-ECA3-A442-CC3F-5341CC5CB1C2	122	45792
Hedley	Norman	6732	A548FBF6-C1BD-46BB-4F39-8AB2396926D8	124	44594
Sebastian	Villarreal	7283	67652AE7-92B7-0F71-9EFB-77E2B8DBDC54	18	94229
Lenore	Sloan	5132	3D09004F-3EDC-B15D-F838-115C5009B653	159	94438
Rinah	Chandler	2154	7DA137CE-842F-CECD-3986-CC7D3285381F	173	79739
Piper	Saunders	9633	33ACBF95-920F-0B37-A41A-B00280681973	203	27759
Aurora	Fuentes	6199	48591FDD-615F-E020-EEEA-C24070F4DDDF	121	83413
Kennan	Gibbs	9946	D15C0244-F4C5-1E83-85E4-9C7125A8790D	183	61495

100 rows in set (0.22 sec)

The same table viewed through the DBMS server

Demonstration

Initial search query on sent to CryptDB proxy...takes a while

```
mysql> select * from USERDATA where first='Aurora' and height=153 and last='Hensley';
+--------+---------+---------+--------------------------------------+--------+---------+
| first  | last    | balance | userid                               | height | zipcode |
+--------+---------+---------+--------------------------------------+--------+---------+
| Aurora | Hensley | 5064    | B48C6854-3B67-78C3-9295-8F52085B6CE9 | 153    | 62370   |
+--------+---------+---------+--------------------------------------+--------+---------+
1 row in set (4.77 sec)
```

Same query sent again...Takes no time at all! The onions have been peeled...

```
mysql> select * from USERDATA where first='Aurora' and height=153 and last='Hensley';
+--------+---------+---------+--------------------------------------+--------+---------+
| first  | last    | balance | userid                               | height | zipcode |
+--------+---------+---------+--------------------------------------+--------+---------+
| Aurora | Hensley | 5064    | B48C6854-3B67-78C3-9295-8F52085B6CE9 | 153    | 62370   |
+--------+---------+---------+--------------------------------------+--------+---------+
1 row in set (0.18 sec)
```

Demonstration

CryptDB also supports homomorphic addition through the Pallier algorithm!

```
mysql> select sum(balance) from USERDATA;
+--------------+
| sum(balance) |
+--------------+
| 449827       |
+--------------+
1 row in set (0.20 sec)
```

This is how the query looks on the proxy server...CryptDB transforms the user generated query into a query on encrypted data using the user's keychain!

```
================================================
QUERY: INSERT INTO USERDATA VALUES ('Kennan','Gibbs','9946','D15C0244-F4C5-1E83-85E4-9C7125A8790D','183
','61495')
NEW QUERY: insert into `cdb`.`table_NOVTGMZWHV` values ('???z(?>Wo??"???Qp??a?&??o??_????', 11622796338
936162038, 1208654678745155468, '??feW??2??m?\r?M?#?)?????kf?-???%', 1449323329125081078, 4526308223942
085881, 4370367086260022128, 6677476022945139378, '?d?"?"?h??b-9c?]????(\0?sd[???g>RGph???"???2+X??m??i
1??AX???\0~??\'??5???{?????>?? s?@?????u??W??????~Z??(??Y?@???6\n?$\Z???[??knv????f??????bL????????/???
l???!?I?`{????/??????Qf9??????????u?3???4?????????Cy?-?m2???l???7???R`G??R??~;?"L?????????^ "C$B?L?8q?\'
??o7', 7495023673318390424, '????????sW*)n?\'u?K?????Z9??}DO\Zc??f???0???u?????7X?<??o?dA??????#?d????
??y???i?', 13931663824988484985, 7566546906616965385, 1191350495946536742, 18125144066602334919, '?w???
*S??5?%lY??L???2??>\\?>???M?1??????????s?q?DBH???.???;k?????D?5?\nr?????8%r????????f4???O?Z????N?#?y?7??
h???x"?"?G?B^???v?qZu??????S??u?????-??e???|?O??~m????????6%:??3Fo?|?X?76\n??9-?????{?\n ???Y??z_P4?G\'
v???????B?????3Ja?dW???????X????.7?]???nE????J??|?W', 1567538931620004929, 3087223949153179040, 46761264
25384304138, '?V?n???fY????|????H.??]????`???MD?J?? 8??-]S\Z?`2???9h?{<?????O??<@@rd8d?????q??4eP???F2?
??b?1???{????????J??????\rY????l?$&??????`??8?H?=??????b??$&?@?|?4?r?*?C??}??l????\O???????S??=?,8"???g
O????x??[??????????~????i<????-^?K?c???\Zx?V?LF?????,?C?d_?eFO?V({?5', 11647786979448835360)
ENCRYPTED RESULTS:
+
|
+
+
```

Demonstration

Of course, the same queries are substantially faster on an unencrypted DB...

```
mysql> select * from USERDATA where first='Aurora' and height=153 and last='Hensley';
+--------+---------+---------+--------------------------------------+--------+---------+
| first  | last    | balance | userid                               | height | zipcode |
+--------+---------+---------+--------------------------------------+--------+---------+
| Aurora | Hensley |    5064 | B48C6854-3B67-78C3-9295-8F52085B6CE9 |    153 |   62370 |
+--------+---------+---------+--------------------------------------+--------+---------+
1 row in set (0.00 sec)
```

```
mysql> select sum(balance) from USERDATA;
+--------------+
| sum(balance) |
+--------------+
|       449827 |
+--------------+
1 row in set (0.00 sec)
```

Demonstration

Here is the output of a quick Java program I wrote to connect to CryptDB, and read the CSV file into the database. It is easy to connect to the proxy using the MySQL Connector/J JDBC driver, which is FOSS created by MySQL.

```
Connection to CryptDB successful!
Creating database cdbexample...
CREATE DATABASE cdbexample

success
Selecting databasecdbexample...
USE cdbexample

success
Creating table with name USERDATA an
d columns [first, last, balance, use
rid, height, zipcode] of types [VARC
HAR(255), VARCHAR(255), INT, VARCHAR
(255), INT, INT]...
CREATE TABLE USERDATA (
first VARCHAR(255),
last VARCHAR(255),
balance INT,
userid VARCHAR(255),
height INT,
zipcode INT
)
```

Using this method it would be very easy to connect CryptDB to Hadoop. Whether it would be a good idea is another question...

To connect using the driver, on the normal JDBC connection creation string consists of the IP and port of the CryptDB proxy server, as well as the password. Easy!

```
Connection cdbConnection = connectToDb("jdbc:mysql://127.0.0.1:3307","root","letmein");
```

Conclusion

- CryptDB is an excellent proof of concept, and is the first piece of software that efficiently solves most of the security issues of DBMSes.

- CryptDB does not (yet) support mathematical operations beyond addition.

- Complex math would be hard to add, and would require at least one more ‘onion’.

- Queries seem quite fast. The developers report CryptDB to be an average of 26% slower than vanilla MySQL for most cases. I have not noticed it being that fast, but it certainly is quite fast, and I have not done any sort of optimization.

- It is easy to implement CryptDB with Java. Hadoop anyone? Would it be worth it?

CryptDB is a wonderful piece of FOSS that brings a level of security and performance previously unmatched to the DBMS world. Much more development is needed to perfect it, however companies like Google have already forked it and are contributing to development. Onions are the future!

References and Links

- MIT CryptDB website http://css.csail.mit.edu/cryptdb/

- - CryptDB whitepaper http://people.csail.mit.edu/nickolai/papers/raluca-cryptdb.pdf

- - Developer's JOIN encryption scheme whitepaper

http://people.csail.mit.edu/nickolai/papers/popa-join-tr.pdf

- - Developer's OPE encryption scheme whitepaper

http://web.mit.edu/ralucap/www/mope.pdf

- - CryptDB SigOPS SOSP talk

https://www.youtube.com/watch?v=YNmuYh7FbQY

- IBM researcher develops fully homomorphic encryption scheme

http://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=2661

- Links to my presentation on YouTube:

Part 1: https://www.youtube.com/watch?v=l40xRtBe87g

Part 2: https://www.youtube.com/watch?v=K4AauL8xx0A