CryptDB

Protecting Confidentiality with Encrypted Query Processing

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan

MIT SAIL

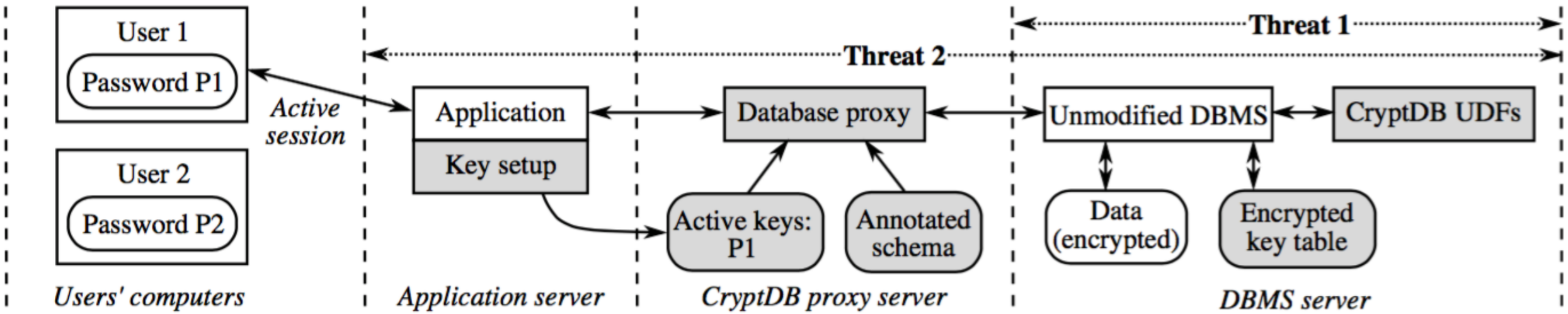
Motivation

- Protect DMBS against confidential data leaks
  - Curious DBAs
  - Adversaries that take over the application and the DBMS server

Contributions

- The first DBMS to perform SQL queries over encrypted data (*SQL-aware* encryption strategy)
- Moderate overhead
- Requires no modifications to applications and DBMS

System Architecture



Techniques

1. ***SQL-aware* encryption**

   - all queries are composed of a set of primitive operations
   - data are encrypted in a way that allow execution on encrypted data

2. ***Adjustable query-based encryption***

   - Dynamically adjust the encryption scheme depending on the types of queries
   - Avoids a priori leak of information

3. ***Chain encryption keys to user passwords***

   - decryption only by using the password of one of the users with access to the data

Threat 1: Curious DBA

- Passive attacker with full access to the DBMS server
- Goal: preserve confidentiality
  - Sensitive data never available on plaintext
  - May reveal some information depending in the classes of computation required by the queries
  - The DBMS server cannot compute the result of queries that involve computation classes that are not requested by the application

Threat 2: Arbitrary Attack

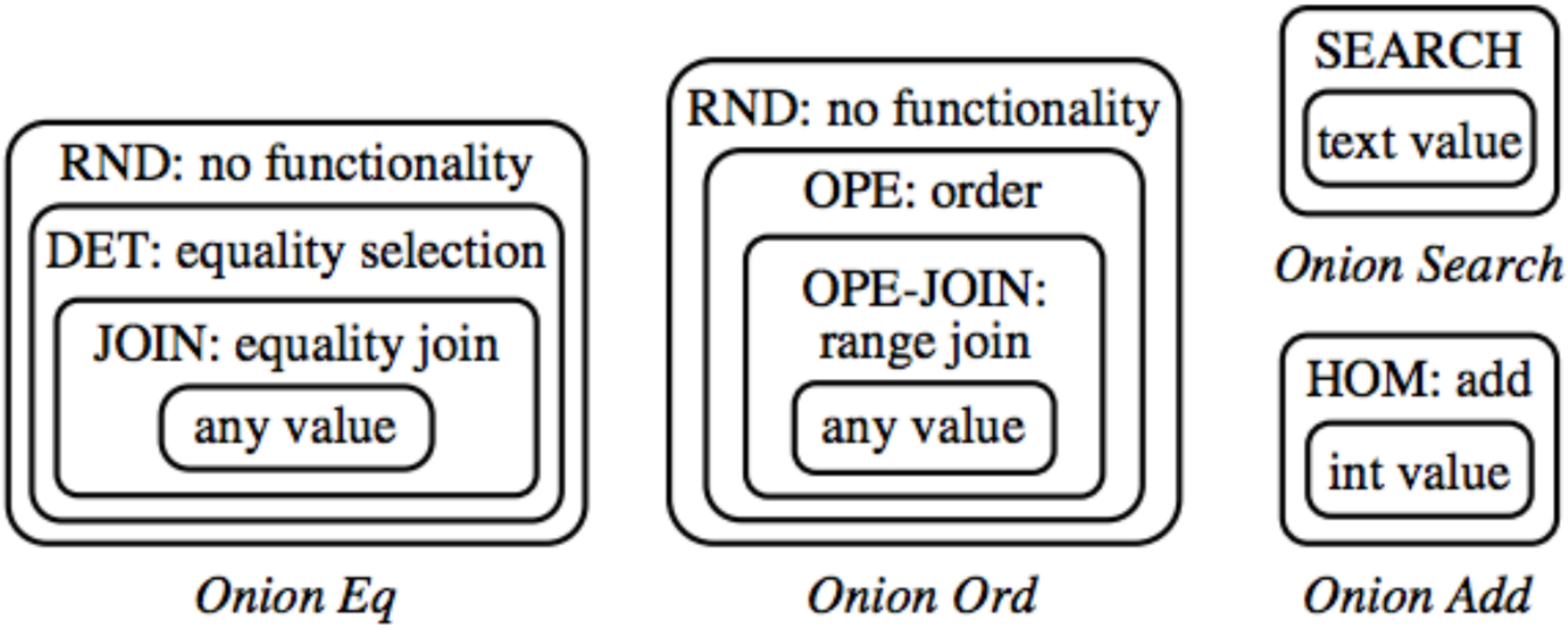
- The attacker can gain full access to the DBMS, the proxy and the application servers
  - Can access the keys!!
  - Solution: Use user passwords to encrypt the different items
  - The attacker can still gain access to the data available to currently logged in users!!!

Executing Queries on Encrypted Data: *SQL-aware encryption*

- Different encryption strategies depending on the type of the computations
  - **RND:** maximum security, does not allow any computation
  - **DET:** reveal which values are equal to each other; allows equality checks (GROUP BY, COUNT, DISTINCT)
  - **OPE:** reveals the order relations on encrypted values; allows queries that involve ordering (ORDER BY, MIN, SORT)
  - **HOM:** allows computations to be performed directly on the ciphertext (e.g. summation); inefficient for some operations
  - **JOIN/OPE-JOIN:** allows equality joins/joins by order relations
  - **SEARCH:** allows searches on encrypted text

Onion encryption

- Goal: **dynamically adjust the layer of encryption**
- Wrap values in layers of increasingly stronger encryption
- Onions layer the classes of computation they allow
- Onion layer decryption depending on the computation required by the query



Query Execution Example

```
SELECT ID FROM Employees WHERE Name = 'Alice',
```

- Requires lowering the encryption layer to **DET**
- The proxy issues the following queries:

```
SELECT C1-Eq, C1-IV FROM Table1 WHERE C2-Eq = x7..d,
```

```
UPDATE Table1 SET
    C2-Eq = DECRYPT_RND(K_{T1,C2,Eq,RND}, C2-Eq, C2-IV),
```

Multiple Principals

- Goal: **Confidentiality when the application and the proxy are untrusted, especially for multi-user apps**

- Schema annotations to specify principals and the data each principal has access to

- 3 steps

    1. Specify *principal types (e.g. users, groups, messages)*

    2. Specify columns with sensitive data and which principals will have access to them

    3. Specify how to delegate a principle's rights to another with a *speaks for* relation

Key Chaining

- Each principal is associated with a randomly chosen key
- Sensitive fields are encrypted with the key of the principal
- Onion keys are derived from a principal's key (instead of a single master key in single-principal mode)
- Only the data of inactive users is protected at the time of the attack!

Security Improvements

- **Minimum onion layers:** specify the lowest possible layer that may be revealed
- **In-proxy processing:** evaluate predicates in the proxy; less information revealed to the server
- **Training mode:** allows the developer to provide a trace of queries and examine the results
- **Onion re-encryption:** re-encrypt onions back to a higher layer after a querie

Performance Optimizations

- **Developer annotations:** indicate sensitive fields, avoid encryption overhead

- **Known query set:** adjust the onion levels beforehand

- **Ciphertext pre-computing and caching:** pre-compute (for HOM) or cache (for OPE) encryptions of frequently used constants

Implementation

- **C++ library**
  - query parser; query encryptor/rewriter; result decryption module
- **Lua module**
  - Passes queries and results from and to the C++ library
- 8000 lines of C++ code; 150 lines of Lua code; 10000 lines of testing code

Evaluation

- Difficulty of modifying and application to run on CryptDB
- Supported queries/applications
- Performance overhead

Application changes

Application	Annotations	Login/logout code	Sensitive fields secured, and examples of such fields
phpBB	31 (11 unique)	7 lines	23: private messages (content, subject), posts, forums
HotCRP	29 (12 unique)	2 lines	22: paper content and paper information, reviews
grad-apply	111 (13 unique)	2 lines	103: student grades (61), scores (17), recommendations, reviews
TPC-C (single princ.)	0	0	92: all the fields in all the tables encrypted

- few changes for multi-principal mode
- no changes for single-principal mode

Functional/Security Evaluation (I)

- Analyzed the queries from 6 web application
- They support most queries (very few columns need to be in plaintext)
- They evaluate the amount of information leaked using the weakest onion encryption scheme than needs to be exposed (*minEnc*)
- They show that most of the sensitive columns are encrypted with the highest security schemes (RND, HOM and DET if no rep.)

Functional/Security Evaluation (II)

Application	Total cols.	Consider for enc.	Needs plaintext	Needs HOM	Needs SEARCH	Non-plaintext cols. with MinEnc: RND	SEARCH	DET	OPE	Most sensitive cols. at HIGH
phpBB	563	23	0	1	0	21	0	1	1	6 / 6
HotCRP	204	22	0	2	1	18	1	1	2	18 / 18
grad-apply	706	103	0	0	2	95	0	6	2	94 / 94
OpenEMR	1,297	566	7	0	3	526	2	12	19	525 / 540
MIT 6.02	15	13	0	0	0	7	0	4	2	1 / 1
PHP-calendar	25	12	2	0	2	3	2	4	1	3 / 4
TPC-C	92	92	0	8	0	65	0	19	8	—
Trace from `sql.mit.edu`	128,840	128,840	1,094	1,019	1,125	80,053	350	34,212	13,131	—
... with in-proxy processing	128,840	128,840	571	1,016	1,135	84,008	398	35,350	8,513	—
... col. name contains *pass*	2,029	2,029	2	0	0	1,936	0	91	0	—
... col. name contains *content*	2,521	2,521	0	0	52	2,215	52	251	3	—
... col. name contains *priv*	173	173	0	4	0	159	0	12	2	—

Performance Evaluation (I)

- Two machines:

1. 2.4 GHz Intel Xeon E5620 4-core processors and 12 GB of RAM to run the MySQL 5.1.54 server

2. 2.4 GHz AMD Opteron 8431 6-core processors and 64 GB of RAM to run the CryptDB proxy and the clients

Performance Evaluation (II): TPC-C experiments

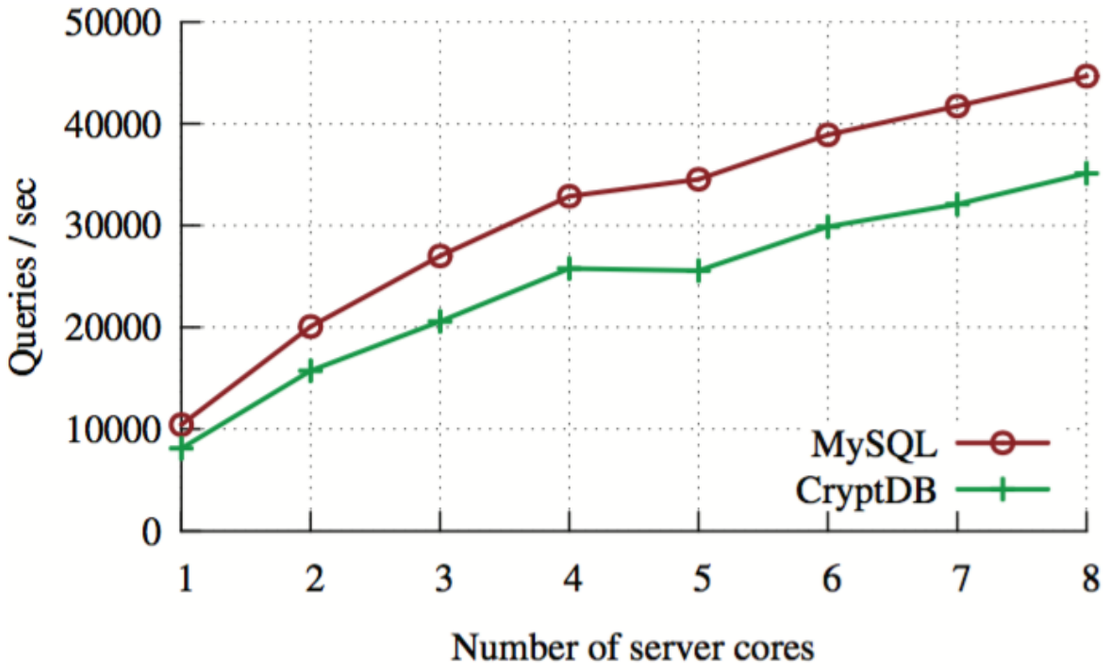


Figure 10: Throughput for TPC-C queries, for a varying number of cores on the underlying MySQL DBMS server.

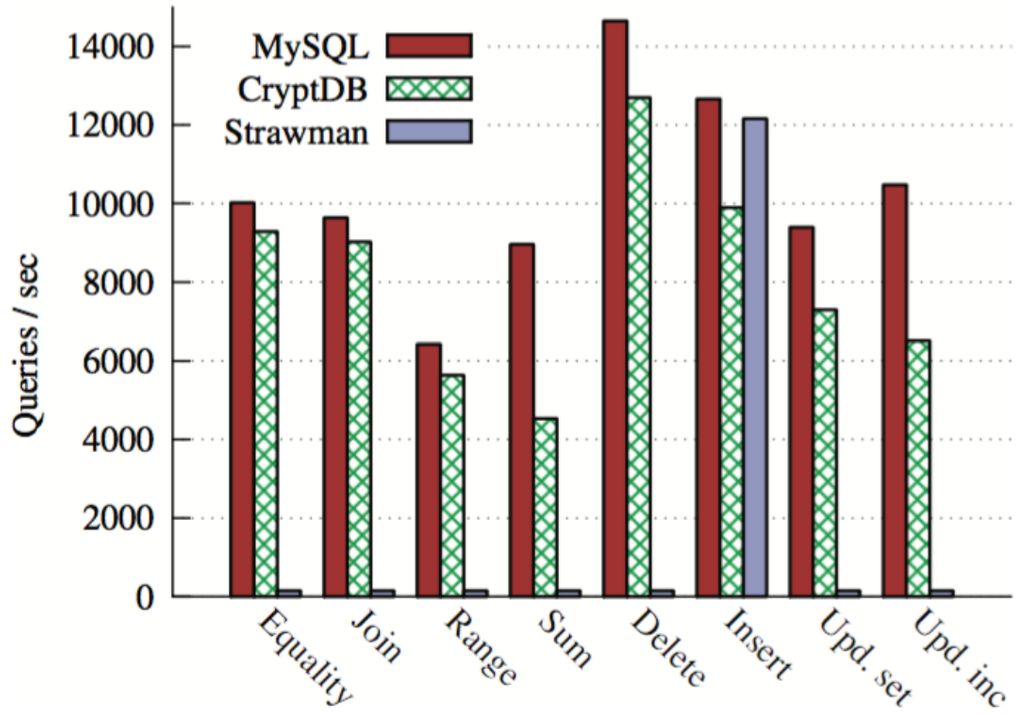


Figure 11: Throughput of different types of SQL queries from the TPC-C query mix running under MySQL, CryptDB, and the strawman design. "Upd. inc" stands for `UPDATE` that increments a column, and "Upd. set" stands for `UPDATE` which sets columns to a constant.

***strawman design*:** performs each query over data encrypted with RND by decrypting the relevant data using a UDF, performing the query over the plaintext, and re-encrypting the result

Performance Evaluation (III): Multi-user web applications

- Throughput of phpBB for workload with 10 parallel clients

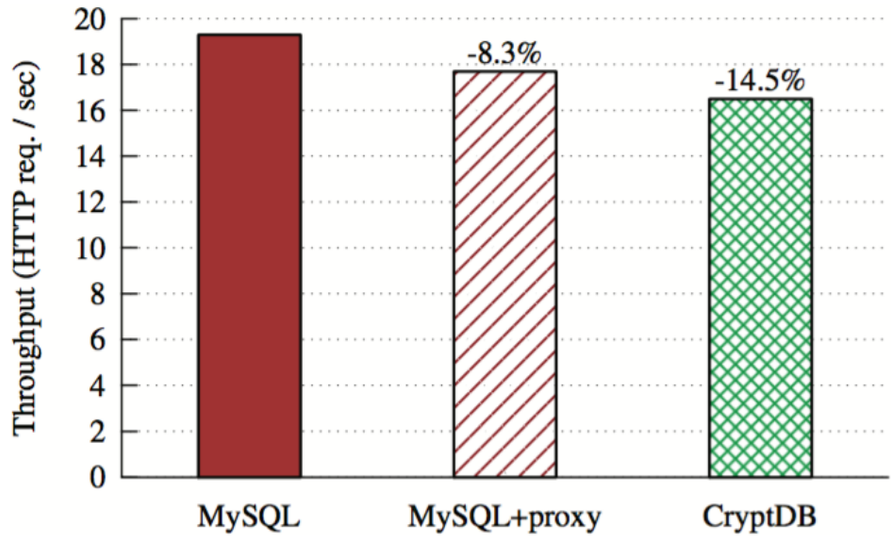


Figure 14: Throughput comparison for phpBB. "MySQL" denotes phpBB running directly on MySQL. "MySQL+proxy" denotes phpBB running on an unencrypted MySQL database but going through MySQL proxy. "CryptDB" denotes phpBB running on CryptDB with notably sensitive fields annotated and the database appropriately encrypted. Most HTTP requests involved tens of SQL queries each. Percentages indicate throughput reduction relative to MySQL.

DB	Login	R post	W post	R msg	W msg
MySQL	60 ms	50 ms	133 ms	61 ms	237 ms
CryptDB	67 ms	60 ms	151 ms	73 ms	251 ms

Figure 15: Latency for HTTP requests that heavily use encrypted fields in phpBB for MySQL and CryptDB. R and W stand for read and write.

Storage Overhead

- Increased the DB size by 3.76x at most
- Cryptographic expansion of integer fields HOM (32 bits to 2048 bits)
- phpBB
  - before: 2.6MB (10 users; 1000 private messages ; 1000 posts)
  - after: 3.3MB

Conclusion

- Practical confidentiality in the face of two different classes of threats
  - Reasonable performance
    - strong security (most of the times)
    - No significant modifications to applications and DBMS
- May limit the possible queries or reduce security
- No guarantees for active users

Controversy

- ***"On the Difficulty of Securing Web Applications using CryptDB".***
  Ihsan Haluk AKIN and Berk Sunar

  - show that cryptDB is ineffective for threat 2
  - demonstrate that an attacker can steal information and even gain administrator privileges

- ***"Inference Attacks on Property-Preserving Encrypted Databases".***
  Muhammad Naveed, Seny Kamara and Charles V. Wright

  - *inference attacks on encrypted database systems like CryptDB*

  - The authors of cryptDB claim that they used cryptDB wrong; Neveed et al. insist that they used it correctly

Questions?