Guidelines for Using the CryptDB System Securely

Raluca Ada Popa
UC Berkeley

Nickolai Zeldovich
MIT CSAIL

Hari Balakrishnan
MIT CSAIL

1 Introduction

This report has two goals. First, we review guidelines for using the CryptDB system [PRZB11, Pop14] securely by the administrators of database applications. These guidelines were already described in [PRZB11] and elaborated on in [Pop14], but in light of some recent work [NKW15] that applied these guidelines incorrectly, a short document devoted to summarizing these guidelines may be useful.

Second, we explain that the recent study of Naveed, Kamara, and Wright [NKW15] represents an unsafe usage of CryptDB, in which the authors violate CryptDB's security guidelines. Hence, the conclusions drawn in that paper regarding CryptDB are both unfounded and incorrect: had the guidelines been followed, none of the claimed attacks would have been possible.

2 Encryption schemes in CryptDB

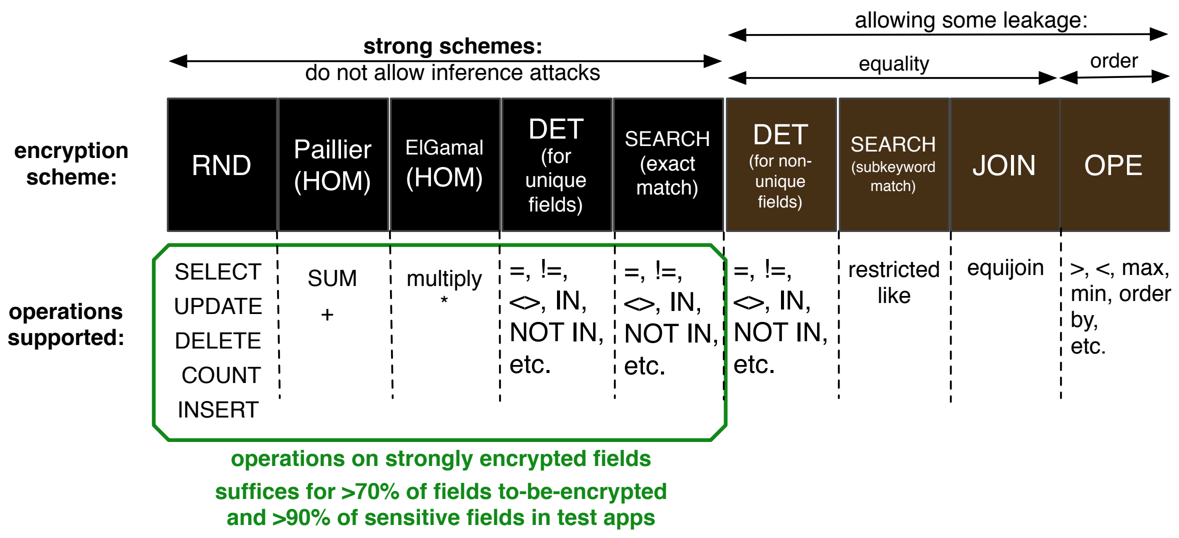
allowing some leakage:
strong schemes:
do not allow inference attacks
equality
order
encryption scheme:
RND
Paillier (HOM)
ElGamal (HOM)
DET (for unique fields)
SEARCH (exact match)
DET (for non-unique fields)
SEARCH (subkeyword match)
JOIN
OPE
operations supported:
SELECT UPDATE DELETE COUNT INSERT
SUM +
multiply *
=, !=, <>, IN, NOT IN, etc.
=, !=, <>, IN, NOT IN, etc.
=, !=, <>, IN, NOT IN, etc.
restricted like
equijoin
>, <, max, min, order by, etc.
operations on strongly encrypted fields
suffices for >70% of fields to-be-encrypted and >90% of sensitive fields in test apps

Figure 1: Encryption schemes, security levels, and operations supported. The encryption schemes without El Gamal are described in the CryptDB paper [PRZB11], and El Gamal was contributed later to the implementation by an outside team.

We assume the reader is familiar with the CryptDB system [PRZB11, Pop14]. As a brief reminder, Fig. 1 shows the encryption schemes used in CryptDB. The figure indicates which schemes are strong, providing semantic security or similar, and which are weaker, allowing some leakage. The strong encryption schemes protect against inference attacks even when an attacker has side information.

The HOM, DET, and SEARCH encryption schemes are non-standard schemes, while RND is standard. DET is safe to use for sensitive fields if those fields have a UNIQUE constraint, meaning there are no repetitions. In this case, DET achieves a similar security guarantee to semantic security because the ciphertexts in the column can be simulated without knowledge of their underlying values. SEARCH for exact match is used to encrypt the entire data item and can support equality match. In the absence of a search token, SEARCH also provides a strong security guarantee, IND-CPA.

This figure also shows the operations that can be computed with each encryption scheme, and presents statistics from test applications, showing that most of the data fields can be supported solely with the strong encryption schemes.

Hence, from Fig. 1, it is important to note that, unlike the claim made in a recent blog posting [Kam15], a wide range of queries can be performed on strongly encrypted data fields in a way that leaks no additional information, as indicated by the green rectangle in the figure. We elaborate on this point in Sec. 5, which presents additional queries that CryptDB can support on strongly encrypted data items.

Recall CryptDB's system setup: on the client side, there is an application and the CryptDB proxy, both of which are trusted; on the server side, there is the encrypted database. The server might have been compromised by a honest-but-curious attacker (a passive attacker). The application issues SQL queries passing through the proxy and going to the database server.

3 Guarantees

The CryptDB system provides guarantees only for columns marked "sensitive". That is, the administrator of a database application must specify which fields are sensitive and CryptDB will keep them encrypted with strong encryption schemes. In this case, the encryption leaks no information about the data items in these columns, other than their lengths. These encryption schemes do not permit inference attacks even when the attacker has side information. As discussed in Sec. 5, a wealth of queries can still be performed on such sensitive fields; the "sensitive" annotation maintains a significant amount of CryptDB's functionality.

The following theorem [Pop14] expresses this guarantee. The relevant security definitions and proofs are in [Pop14].

Theorem 3.1 ([Pop14]) *CryptDB provides (distinct-) semantic security for every column marked "sensitive", under standard cryptographic assumptions.*

Using the "sensitive" annotation, CryptDB ensures that, even if an attacker steals an encrypted database, the database does not leak the values of sensitive fields, even if the attacker has side information. We note that, if an attacker watches queries, the access patterns due to queries may still reveal some information (e.g., based on how often a data item is accessed). Such access-pattern leakage is well-known in the literature and arises in many types of encrypted systems, such as encrypted file systems, searchable encrypted databases, key-value stores, and so on—it is not new to CryptDB. Oblivious RAM (ORAM) [SS13] is one of the best tools we have today for hiding such access patterns. Although ORAM is currently slow, if new and faster schemes are developed, they can be added on top of CryptDB to hide the access patterns in CryptDB.

4 Guidelines

We now explain the guidelines an administrator must follow to obtain the security guarantees of CryptDB.

Step 1: Mark as "sensitive" any column whose content you want to protect.

The next two steps are optional, but recommended for performance. For these steps, the administrator should supply the set of query types ran in the application. The constants don't matter here, only the operations. For example, web applications typically have a set of queries hardcoded in them, which are run with different constants, based on user data. Applications often have a relatively small set of query types; for instance, a TPC-C benchmark issues about 30 types of queries.

Optional:
Step 2: Provide the query set to CryptDB. CryptDB checks if all queries are supported, and if there are unsupported queries, it outputs the reason: which operations cannot be supported, and why. If there are no unsupported queries, we are done.
Step 3: Address each unsupported query/operation with one or both of these options:
Option 1: (No change to applications) Run a part of the query in the proxy, using Monomi [TKMZ13].
Option 2: (Modify the application) Determine if the unsupported operations can be removed from the application or can be handled within the application.

Unsupported queries are of two kinds: queries that cannot be supported with any of the encryption schemes in CryptDB, and queries that cannot be supported because some column was marked "sensitive". We have found the second category to be rare. The administrator is expected to handle both types of queries in the same way.

Option 1: Run the part of the query that cannot be supported on the client-side, in the proxy, as recommended in Sec. 3.5.1 of [PRZB11]. Monomi [TKMZ13] is a system designed to automate this process, without requiring any change to the application. Monomi can split a query into server- and client-side execution subject to some constraints. Monomi can operate with different constraints, and it receives the list of constraints as input. The constraints specify the operations that can happen on the field at the server. These operations correspond to the entire encryption set for a non-sensitive field, or to the strong encryption schemes for a sensitive field. Monomi finds an optimal splitting of these queries to maximize performance.

Option 2: The administrator can use the reason reported by CryptDB to understand which operations are not supported; the admin then decides if those operations are needed, if the application can run those queries differently, or if the computation can happen in the application.

5 Functionality for "sensitive" fields

Although fields marked "sensitive" are restricted only to strong encryption schemes, CryptDB can still perform a wide range of queries on these fields. Fig. 1 shows the various operations that the server can compute directly on such fields in the database. The server uses a set of non-standard encryption schemes in addition to the standard scheme RND. Moreover, the CryptDB paper [PRZB11] discusses that with a small amount of in-proxy computation, even more queries can run efficiently on such fields.

Hence, the queries supported on sensitive-marked fields are:

1. Fetch, modify, insert, or delete data due to RND. SQL examples include: SELECT, UPDATE, DELETE, COUNT, and INSERT.

2. Sum due to HOM (Paillier). SQL examples include: sum, +.

3. Product due to HOM (ElGamal). SQL examples include *.

4. Equality operations due to DET and SEARCH. SQL examples include: =, !=, <>, IN, NOT IN, etc. The proxy chooses between DET and SEARCH depending on whether the values in the column of interest are unique (that is, they do not repeat within the column).

   (a) If the column is declared unique by the administrator, the proxy allows the server to use DET. Indexes on this column work as expected.

   (b) If the values in the column are not unique, the server uses SEARCH. If there is no index built on this column, the performance of using SEARCH is comparable to using DET because the server scans the column of interest in both cases, and SEARCH is based on fast hardware-based AES instructions. In typical applications, by far, most of the columns do not have an index on them. The columns that have an index are typically either not sensitive or are unique (e.g., primary keys). If there is a sensitive column that is not unique and has an index on it, the server can still perform equality operations using SEARCH on this column; this operation will be slower because the index is not used.

5. Certain order operations, such as ORDER BY with no limit. For example, the CryptDB paper [PRZB11] discusses the example of a query of the form: `SELECT` [..] `WHERE` [...] `ORDER BY sensitive field`. CryptDB can support this query efficiently by executing `SELECT` [..] `WHERE` [...] at the server, and performing the order on the result set at the proxy. This strategy is efficient because the proxy anyways needs to receive and process the result set, and sorting the result set is a proportional amount of work. The expensive operation (which involves more data), the filtering due to `WHERE`, happens at the server.

6. Arithmetic functions or other functions that are computed on the result set and do not perform filtering. These can be executed at the proxy with low performance overhead, again, because the proxy needs to do work only in the result set. SQL examples include FORMAT, SUBSTRING, sin, etc. Sec. 8.2 of the CryptDB paper [PRZB11] elaborates on these schemes.

A recent blog posting [Kam15] claims that CryptDB restricts sensitive fields to RND only, and no queries can run on such data—this statement is false, as shown by the list above!

In fact, the operations supported on fields marked "sensitive" are qualitatively the same as the queries for a field to be encrypted that is not marked sensitive: the difference is only in which cases an equality or an order can run efficiently. For equality, almost all sensitive fields will be supported with a comparable efficiency to non-sensitive fields. The reason is that in a typical application most fields do not have indexes built on them, and out of those that have indexes, a significant fraction are either not sensitive or are unique (e.g., primary keys). For example, for the OpenEMR medical application, more than 95% of fields do not have indexes built on them. Another example from OpenEMR is the field "SSN", which is unique per patient and can be encrypted and queried by equality using DET.

Regarding order operations, a non-trivial number of fields in real-world applications process ORDER BY with no limit. For example, Fig. 9 in [PRZB11] shows that, for a large database server (sql.mit.edu) with >1000 databases and >128,000 data fields, out of the original 13,131 data fields doing some order operation, more than >4600 data fields use ORDER BY with no limit. Out of the presumed sensitive fields in this trace, only a fraction of 0.001 had some order computation performed on them, suggesting that

CryptDB will perform in-proxy processing for very few fields. For fields not marked sensitive, the server can additionally run range queries and ORDER BY LIMIT at the server. If an ORDER BY limit is needed for a sensitive field, Monomi optimizes the way in which this query gets split between proxy and server computation: in some cases, these queries can remain approximately as fast as before, in other cases, these can be slower.

As an example, Fig. 9 in [PRZB11] shows that the strong encryption schemes already provide the functionality for most of the fields in common applications: they support >65%–70% of the fields considered for encryption, and >90% of the fields considered sensitive with *no* in-proxy processing. A lot of the remaining sensitive fields can also be supported efficiently based on categories 4, 5, and 6 above.

This means that if an administrator wants to protect a sensitive field by marking it "sensitive", there is a high chance the queries on this field will be supported with the expected performance. Only for a small number of fields does the administrator need to follow through Step 3. Of course, there exist applications that have queries CryptDB cannot support even for non-sensitive fields, or with queries CryptDB cannot support efficiently on sensitive fields – CryptDB does not claim to support every possible application and should not be used for these applications.

Thus, we can see that (1) a significant class of computations can be performed on strongly encrypted fields, and (2) keeping sensitive fields encrypted with strong encryption schemes is likely to result in few unsupported fields, for which we proposed measures.

6 Fields not marked "sensitive"

If an administrator does not mark a field as sensitive, CryptDB provides no security guarantees. In this case, CryptDB employs a "best-effort" approach [Pop14] using the most secure encryption scheme that supports queries on these fields at the database server. Hence, such fields could end up being encrypted with weak encryption schemes such as OPE and DET for non-unique values. It is well-known that such schemes can leak data [BCO11, KS12] when coupled with other information an attacker might have. The CryptDB publications [PRZB11, Pop14] acknowledge this fact, and as a result require the use of the "sensitive" annotation to avoid such leakage.

We suggest using these encryption schemes only for less-sensitive or non-sensitive fields, a common example being timestamps [PRZB11]. Timestamps do not repeat, and are high entropy values from a sparse domain, requiring an attacker to have a significant amount of side information to decrypt them. In particular, the attacks of Naveed et al. do not work for timestamps. We advise using OPE only for less-sensitive fields with a similar distribution.

7 Unsafe usage in Naveed et al. [NKW15]

Naveed, Kamara, and Wright [NKW15] investigate the information an attacker can obtain from a medical database that is encrypted *only* with OPE and DET (and no strong encryption), as long as the attacker has certain auxiliary information about the database. The authors extrapolate incorrectly that their scenario arises when running the medical application OpenEMR on CryptDB, leading to the unfounded conclusion that CryptDB would leak this information and be unsafe for OpenEMR.

Their conclusion is unfounded because:

- The attacks presented represent an unsafe usage of CryptDB because the authors do not mark any fields "sensitive". The first step in a security-conscious use of CryptDB for OpenEMR is to mark fields that should be held confidential as "sensitive". CryptDB then protects those fields with the strong encryption schemes, which prevent any inference attacks; in particular, none of the attacks in [NKW15] would be possible.

- The authors *do not* actually evaluate CryptDB on OpenEMR: instead, they imagine an application with certain sensitive (but not marked as sensitive) fields, of which roughly half do not even exist in OpenEMR. For the other half of the fields that exist in OpenEMR (or have a related field in OpenEMR), we show that by using the "sensitive" annotation, CryptDB secures them with strong encryption schemes with no change to the OpenEMR application. As a result, the attacks of [NKW15] are not possible. Moreover, we show that numerous other fields in OpenEMR, that are arguably even more sensitive, are also supported with strong security by CryptDB. These fields are not mentioned in [NKW15].

Thus, the study in [NKW15] is limited to conclusions *only* about OPE and DET and not about CryptDB. The fact that OPE and DET leak is in fact not novel to [NKW15]: it has already been widely known in the literature [BCO11, KS12], and also explained in the CryptDB papers [PRZB11, Pop14]. This is why CryptDB handles these encryption schemes carefully, as was discussed in this document.

Regarding the first point above, the fact that the authors used CryptDB against its security guidelines invalidates their conclusions with respect to CryptDB.

Regarding the second point, the authors did not actually evaluate the CryptDB system on OpenEMR. Naveed et al. [NKW15] imagine a scenario in which twelve hand-picked confidential data fields are encrypted only with DET or OPE, a scenario that does not occur when CryptDB is used correctly.

Some of the fields examined in [NKW15] do not exist in OpenEMR. Their paper confirms only six of the twelve fields to exist in OpenEMR: primary payer, age, sex, race, admission month, patient died; it does not confirm the existence of the other fields in OpenEMR, which are arguably more sensitive, such as major diagnostic category, disease severity, or mortality risk. Indeed, we could not find these three fields (or fields with the same distribution as these fields as explained in [NKW15]) in OpenEMR. We examined OpenEMR 4.2.0, the latest stable version of OpenEMR. Despite these fields not existing in OpenEMR, the authors still conclude that CryptDB leaks these fields, a conclusion that does not make sense.

Some of the fields in [NKW15] exist in OpenEMR[1]. The authors claim that six such fields exist in OpenEMR and we confirm the presence of a subset of these. For these fields, we analyzed OpenEMR and found that by using the "sensitive" annotation, CryptDB can secure these fields with strong encryption schemes. For all the queries on these fields that CryptDB supports without the "sensitive" annotation, adding the "sensitive" annotation on these fields preserves the support of these queries with no change to the OpenEMR application, and with a low or acceptable performance overhead, as we explain below. For a few of these fields, there exist queries that CryptDB cannot support even when these fields don't have the "sensitive" annotation because these queries perform complicated operations that the union of all the encryption schemes (strong or weak) in CryptDB cannot support (Fig. 9 in [PRZB11] reports a total of 7 such sensitive fields across OpenEMR).[2]

First, CryptDB will not reveal the DET encryption at the server to perform equality on these fields. None of the fields have an index built on them, so the server would use SEARCH to perform the equality check, as described in category 4b of Sec. 5. The proxy would change "=" for these fields into the special CryptDB "like" operation, and as a result, there would be no change to the application. Morever, the performance

[1]We note that some of the fields Naveed et al. claim to exist in OpenEMR occur there with a significantly different distribution than the distribution assumed in their paper: for example, there is no age in the patient data table, but there is date of birth, which has a significantly different cumulative distribution than the one the authors assumed.

[2]Such queries won't run at the server and preserve security trivially.

overhead is modest because, in the absence of an index, the database server scans the relevant column in both the encrypted and unencrypted databases. This scan constitutes a significant fraction of this query's cost; the per-row operation due to SEARCH is fast because it is implemented based on hardware AES. Hence, none of the DET-based attacks to these fields claimed in [NKW15] are possible.

Second, most of the queries on these fields that perform order do "ORDER BY" without a LIMIT. This falls in category 5 of Sec. 5 allowing CryptDB to support them with no application change and low proxy overhead. Regarding other types of order queries, for each field considered, we either did not find any or we found very few such queries. Monomi can handle these queries with no application change, by sending the relevant data to the proxy. Moreover, we confirmed with the lead developer of OpenEMR that sending the relevant data to the proxy for these queries is not problematic because the events when these queries are called are rare (e.g., doing order by limit on sex does not make sense), and, if these queries are invoked at all, they are outside of the critical path for performance when a slower response is acceptable (e.g., occasional patient reports could issue some queries like this, but the critical operation of looking up a patient by equality on id or name during a visit will not).

Finally, besides the data fields considered in [NKW15], CryptDB can secure numerous other sensitive fields with semantic security with minimal performance overhead, such as vitals (respiration, temperature, pulse), lab report documents, medical report document, messages from the doctor, or doctor notes. These fields are not mentioned in [NKW15].

To sum up, the study in [NKW15] is limited to conclusions only about OPE and DET, and their conclusion on CryptDB is unfounded.

References

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *C,RYPTO*, 2011.

[Kam15] Seny Kamara. Attacking encrypted databases. 2015.

[KS12] Vladimir Kolesnikov and Abdullatif Shikfa. On the limits of privacy provided by order-preserving encryption. *Bell Labs Technical Journal*, 17(3):135–146, 2012.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. 2015.

[Pop14] Raluca Ada Popa. *Building practical systems that compute on encrypted data*. PhD thesis, M.I.T., 9 2014.

[PRZB11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[SS13] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious cloud storage. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pages 253–267, 2013.

[TKMZ13] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. pages 289–300, 2013.