Query rewriting notes

This note is a brief description of the query rewriting algorithm of CryptDB [1].

A query is first parsed into a lex tree using the MySQL parser, and then it is rewritten. The unit in a lex tree is a SQL item, which can be a field, a constant, an operations, or even an expression. Query rewriting from a lex tree happens in two phases:

1. Gather: gathers information about all the possible ways an item could be rewritten to support the operations in the query.
2. Rewrite: Based on the information gathered, rewrite encrypts constants, replaces fields with the names of their onions, and rewrites some operations into new operations or in UDFs.

We defined a class for each type of field, constant, or operation, that is meant to handle gather (`do_gather_type`) and rewrite (`do_rewrite_type`) for this type of item.

1 Gather

```
virtual EncSet do_gather_type(Item_field *i, reason &tr, Analysis & a) const;
```

Before defining EncSets, let us define *OLK*, which stands for (onion, security level, key). This is all the information one needs to encrypt or rewrite an item. The key is a pointer to the metadata of a field (FieldMeta) because each field has its own key in CryptDB. The key is not stored in FieldMeta directly, but rather it is stored as a list of EncLayers, which implement encryption or decryption with this key given a value.

An *EncSet* is "a set of possible encryption types of an item". That is, a set of OLKs with which an item could be encrypted or rewritten. Instead of listing all the OLKs in an EncSet, the EncSet rather contains a map of onions to the highermost security level and a key, with the understanding that all lower levels on the same onions are also in the EncSet.

`do_gather_type` returns an *outgoing* EncSet, which consists of all the different OLKs with which item i could be encrypted/rewritten.

`reason` indicates the reason why such an outgoing EncSet was returned, and it is useful for printing out detailed error messages when some operation cannot be supported.

We now describe how `do_gather_type` works. It traverses the lex structure bottom up. Consider the case that i is a leaf node. If i is a field, it returns the encset that i can support ("PLAIN" if the field is not encrypted). If i is a constant, it returns "all enc layers with any key and PLAIN"

Let's present a generic gather for an operation.

Algorithm 1 (Gather at item i)**.**

1. Run gather on each child of i. Let EncSet_j be the encset returned from each child.
2. Consider the encryption levels needed to perform the operation of i, *needed EncSet*. Based on the child encsets EncSet_j and the needed EncSet, find solutions: sets of OLKs that could be used for each child and the computation at i to support the operation at i.

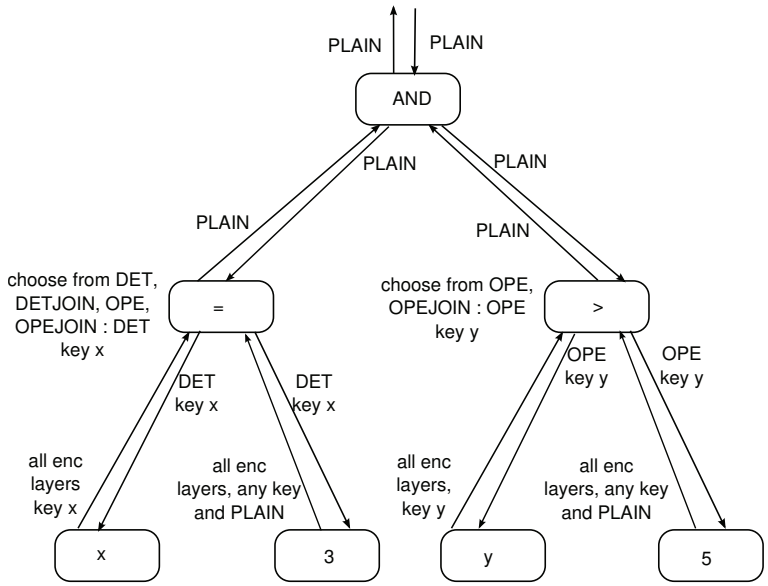

Figure 1: Example gather/rewrite for `x = 3 AND y > 5`. EncSets go up the tree in gather and down the tree in rewrite.

(a) If the set is null, throw unsupported query error and use the information at this node and the `reason`s from the children to explain why.

(b) If there are solutions, store them in a RewritePlan to be used during rewrite. Compute the resulting OLKs for each possible solution, and aggregate them in the outgoing EncSet, to be returned.

Figure 1 provides an example.

2 Rewrite

After we computed gather on the entire lex tree, we run rewrite.

```
virtual Item *
do_rewrite_type(Item_field *i, const OLK & constr, Analysis & a) const
```

`do_rewrite_type` receives as argument an item i and returns its rewritten version.

In gather, the top of the lex tree returned an EncSet, from which we pick the OLK that leaks the least security and provide it as `constr` argument to `do_rewrite_type` at the root. When `do_rewrite_type` writes for a certain item type, the following typically happens. First, we consult the `RewritePlan` for item i (stored in `a.itemRewritePlans`). Using this information and `constr`, we call `do_rewrite_type` on the children of node i. Finally, we rewrite i, again based on `constr` and `do_rewrite_type`.

If a rewrite needs to perform onion adjustment, an exception is thrown, the onions adjusted, and the entire gather/rewrite process restarted.

References

[1] 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*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.