Project Proposal: CryptDB-style Encryption for Apache Spark

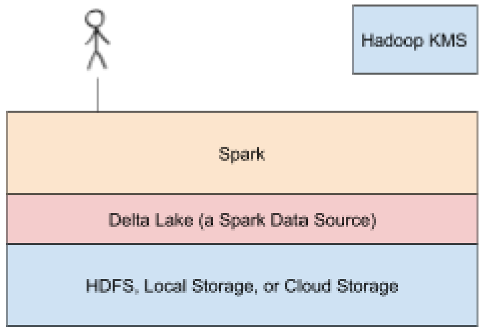
1. Introduction

I plan on creating a fork of Delta Lake, an open-source storage layer for Apache Spark, and attempting to add CryptDB-style column-level encryption. If this proves to be impossible or I have extra time, I plan on working on data lineage features to simplify GDPR compliance.

I believe that by doing this, column-encryption could become standard in the analytics world in the future. Delta Lake, open-sourced in April 2019, has a good chance at becoming the de-facto "data format" for Spark in the future because of how it simplifies data lake management. If column encryption is supported when more users start adopting Delta Lake, users may be compelled to encrypt semi-sensitive data like name and email rather than exposing it to all data analysts.

2. Implementation Overview

Column-level encryption was added to Apache Hive for the ORC file format in version 1.6 (released September 2019). Since ORC is similar to Parquet and Hive is similar to Spark, this is a good sign that it is feasible to add column-level encryption to Delta Lake.



Hadoop KMS
Spark
Delta Lake (a Spark Data Source)
HDFS, Local Storage, or Cloud Storage

The above diagram shows how Delta Lake fits into the Spark ecosystem. The Delta Lake and Spark codebases will need to be changed.

Below are the primary steps in my implementation plan:

1. Review the changes made to the ORC, Hive, and Hadoop projects for the implementation described above.
2. Identify parts of Delta Lake, Spark, and Hadoop KMS that would need to be changed to support CryptDB-like encryption. In particular, I need to look for places to define encryption UDFs and per-query column decryption.
3. Implement and add OPE-JOIN encryption. Incrementally add support for other encryption types.
4. Add support for joins and other advanced features.

As I will be working with larger codebases, I anticipate that I will need to work a bit slower to cope with slower compile times. Later on, the strong testing suites in the above codebases may make development easier.

3. Techniques Applied

The primary techniques from class that I will apply are in Section 3 of the CryptDB paper. Given that the authors wrote 18k+ lines of C++ code, I expect to spend a decent amount of time getting this to work.

I could take advantage of Delta Lake's Z-Ordering (multi-dimensional clustering) to decrypt fewer files. I could also take advantage of the fact that network I/O is often the bottleneck in queries. One anticipated downside is that column encryption will reduce the effectiveness of column encoding, resulting in larger files.

I will need to familiarize myself with the Delta Lake source code, and Hadoop KMS API. I have used Spark and Delta Lake before and am familiar with the architecture of both tools.

4. Problem Overview

This project aims to make it easy for users to mask columns in Spark-based data lakes without affecting future queries. Because this functionality does not exist, many data lakes expose fields like 'email' or 'company' to those querying the data.

The goal here is to make it easy enough to turn encryption on for semi-sensitive columns so that Delta Lake users do it by default.