Efficient Spark Analytics on Encrypted Data

Gidon Gershinsky, IBM Research - Haifa
gidon@il.ibm.com

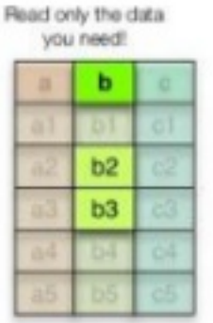
#SAISDev14

Overview

- What problem we are solving?
- Parquet modular encryption
- Spark encryption with Parquet
  - connected car scenario: demo screenshots
  - performance implications of encryption
  - getting from prototype to real thing

What Problem Are We Solving?

- Protect sensitive data-at-rest
  - data confidentiality: encryption
  - data integrity
  - in any storage - untrusted, cloud or private, file system, object store, archives
- Preserve performance of Spark analytics
  - advanced data filtering (projection, predicate) with encrypted data
- Leverage encryption for fine-grained access control
  - per-column encryption keys
  - key-based access in any storage: private -> cloud -> archive

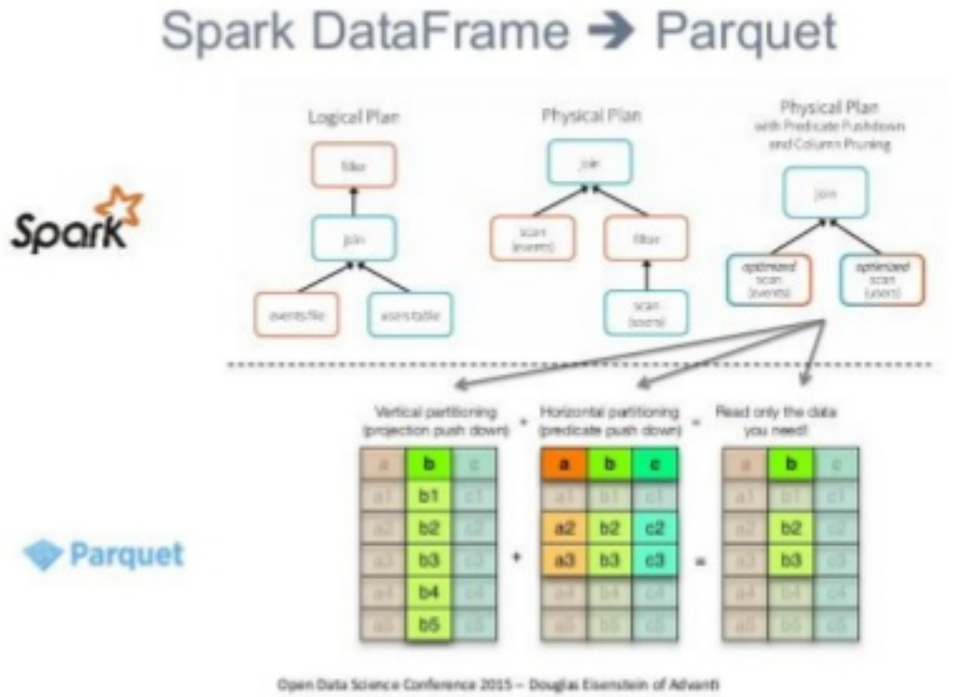


Background

- EU Horizon2020 research project
- EU partners: Adaptant, IT Innovation, OCC, Thales, UDE
  - usage-based car insurance, social services
- Collaboration with UC Berkeley RISELab
  - Opaque technology
- Secure cloud analytics (Spark and H/W Enclaves)
  - how to protect “data-in-use”?
  - how to protect “data-at-rest” – **without losing analytics efficiency?**

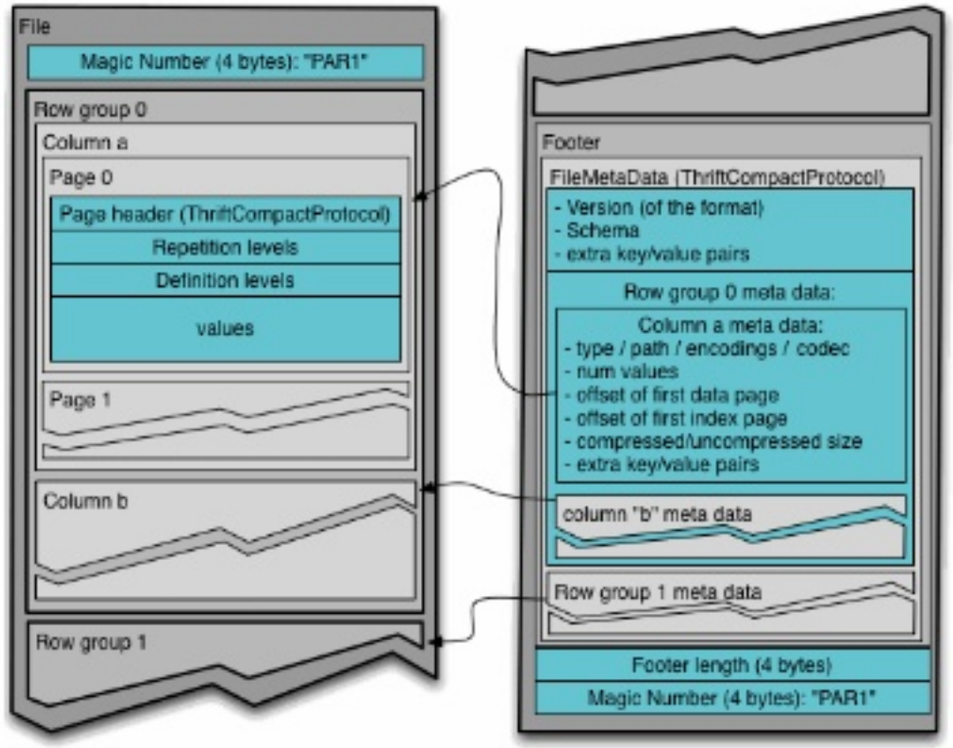
Apache Parquet

- Integral part of Apache Spark
- Encoding, compression
- Advanced data filtering
  - *columnar projection*: skip columns
  - *predicate push down*: skip files, or row groups, or data pages
- Performance benefits
  - less data to fetch from storage: I/O, latency
  - less data to process: CPU, latency
- How to protect sensitive Parquet data?
  - in any storage - keeping performance, supporting column access control, data tamper-proofing etc.



Parquet Modular Encryption

- Apache Parquet community work
- Full encryption: all data and metadata modules
  - metadata: indexes (*min/max*), ***schema***, *size and number of secret values*, *encryption key metadata, etc ..*
  - encrypted and **tamper-proof**
- Enables columnar projection and predicate pushdown
- Storage server / admin never sees encryption keys or cleartext data
  - "client-side" encryption (e.g., Spark-side)



Parquet Modular Encryption

- Works in any storage
  - supporting range-read
- Multiple encryption algorithms
  - different security and performance requirements
- Data integrity verification
  - AES GCM: “authenticated encryption”
  - data not tampered with
  - file not replaced with wrong version
- Column access control
  - encryption with column-specific keys

```
union EncryptionAlgorithm {
  1: AesGcmV1 AES_GCM_V1
  2: AesGcmCtrV1 AES_GCM_CTR_V1
}

struct FileCryptoMetaData {
  1: required EncryptionAlgorithm encryption_algorithm

  /** Parquet footer can be encrypted, or left as plaintext **/
  2: required bool encrypted_footer

  /** Retrieval metadata of key used for encryption of footer,
   *  and (possibly) columns **/
  3: optional binary footer_key_metadata

  /** Offset of Parquet footer (encrypted, or plaintext) **/
  4: required i64 footer_offset
}
```

AES Modes: GCM and CTR

- AES: symmetric encryption standard supported in
  - Java and other languages
  - CPUs (x86 and others): hardware acceleration of AES
- GCM (Galois Counter Mode)
  - encryption
  - integrity verification
    - basic mode: make sure data not modified
    - AAD mode: “additional authentication data”, e.g. file name with a version/timestamp
      - prevent replacing with wrong (old) version
- CTR (Counter Mode)
  - encryption only, no integrity verification
  - useful when AES hardware acceleration is not available (e.g. Java 8)

Status & Roadmap

- **PARQUET-1178**
  - full encryption functionality layer, with API access
  - format definition PR merged in Apache Parquet feature branch
  - Java and C++ implementation
    - one PR merged, others being reviewed

- **Next: tools on top**
  - PARQUET-1373: **key management tools**
    - number of different mechanisms
    - options for storing key material
  - PARQUET-1376: **data obfuscation layer**
    - per-cell masking and aggregated anonymization
    - reader alerts and choice of obfuscated data
  - PARQUET-1396: **schema service interface**
    - "no-API" interface to encryption
    - all parameters fetched via loaded class
  - Demo feature: **Property interface**
    - "no-API" interface to encryption
    - most parameters passed via (Hadoop) config properties
    - key server access via loaded class

Spark Integration Prototype

- Standard Spark 2.3.0 is built with Parquet 1.8.2
- Encryption prototype: standard Spark 2.3.0 with Parquet 1.8.2-**E**
  - Spark-side encryption and decryption
    - storage (admin) never sees data key or plain data
  - Column access control (key-per-column)
  - Cryptographic verification of data integrity
    - replaced with wrong version, tampered with
  - Filtering of encrypted data
    - column projection, predicate push-down

Connected Car Usecase

Personal Data of Drivers
- driver ID (license #)
- name
- home address
- email address
- credit card

who is speeding at night? (20% and more above limit)

Usage-based Insurance Service

Raise premium, Notify by email

Gateway

Spark

Parquet

Car Monitoring Events
- car ID (plate #)
- driver ID (license #)
- speed
- location

Car Event Schema

```
scala> val events_01_09_2018 = spark.read.schema(
     |           StructType(Seq(
     |             StructField("CarID", IntegerType),
     |             StructField("DriverID", IntegerType),
     |             StructField("Timestamp", TimestampType),
     |             StructField("CarIDHash", StringType),
     |             StructField("DriverIDHash", StringType),
     |             StructField("Latitude", DoubleType),
     |             StructField("Longitude", DoubleType),
     |             StructField("Speed", IntegerType),
     |             StructField("SpeedLimit", IntegerType),
     |             StructField("TransmissionGearPosition", IntegerType),
     |             StructField("AcceleratorPedalPosition", IntegerType),
     |             StructField("BrakePedalStatus", BooleanType),
     |             StructField("Odometer", LongType),
     |             StructField("FuelLevel", IntegerType),
     |             StructField("EngineSpeed", IntegerType),
     |             StructField("HeadlampStatus", BooleanType),
     |             StructField("HighBeamStatus", BooleanType),
     |             StructField("DoorStatus", IntegerType),
     |             StructField("WindshieldWiperStatus", BooleanType),
     |             StructField("CarServiceMessage", StringType)
     |  ))).csv("/home/gidon/Summit/Demo/CarEvents_01_09_2018.csv")
```

Secret data, to be encrypted

Unencrypted columns

Driver Table Schema

```
scala> val peopleTable = spark.read.schema(
     |           StructType(Seq(
     |             StructField("PersonID", IntegerType),
     |             StructField("PersonIDHash", StringType),
     |             StructField("FullName", StringType),
     |             StructField("Gender", StringType),
     |             StructField("DateOfBirth", DateType),
     |             StructField("PhoneNumber", StringType),
     |             StructField("Email", StringType),
     |             StructField("StreetAddress", StringType),
     |             StructField("City", StringType),
     |             StructField("State", StringType),
     |             StructField("CreditCard", StringType),
     |             StructField("MaskedCreditCard", StringType)
     |  ))).csv("/home/gidon/Summit/Demo/People_09_2018.csv")
peopleTable: org.apache.spark.sql.DataFrame = [PersonID: int,
more fields]
```

Secret data, to be encrypted

Masked data

Writing Encrypted Parquet Files

```
scala> sc.hadoopConfiguration.set("encryption.key.list" ,
     |     "k0:iKwfmI5rDf7HwVBcqeNE6w==," +
     |     "k1:LjxH/aXxMduX6IQcwQgOlw==," +
     |     "k2:rnZHCxhUHr79Y6zvQnxSEQ==," +
     |     "k3:6b2G9UsRmCAsoGsd3IMQrA==," +
     |     "k4:mORupskWLHAfuvDJbXrCEw==," +
     |     "k5:NG6Hi85MW04sqMlXJHt5lg==," +
     |     "k6:AAECAwQFBgcICQoLDA0ODw==")

scala>

scala> events_01_09_2018.write.
     |     option("encryption.footer.key" , "k0").
     |     option("encryption.column.keys" ,
     |                "CarID:k1, DriverID:k2, Longitude:k3, Latitude:k3, Speed:k4").
     |     option("encryption.writer.aad" , "E_01_09_18.parquet.encrypted").
     |   parquet("/home/gidon/Summit/Demo/CarEvents_Sept2018/E_01_09_18.parquet.encrypted")
```

Data Keys (base64)

Writing Encrypted Parquet Files

```
scala> peopleTable.write.
     |     option("encryption.footer.key" , "k5").
     |     option("encryption.column.keys" ,
     |       "PersonID:k2, FullName:k2, DateOfBirth:k2, PhoneNumber:k2, Email:k2, StreetAddress:k2, " +
     |       "CreditCard:k6").
     |     option("encryption.writer.aad" , "People_09_2018.parquet.encrypted").
     |  parquet("/home/gidon/Summit/Demo/People_09_2018.parquet.encrypted")
```

Reading Encrypted Parquet Files

```
scala> val carEventsEncrypted = spark.read.parquet("/home/gidon/Summit/Demo/CarEvents_Sept2018/*")
18/09/06 14:10:51 ERROR Executor: Exception in task 0.0 in stage 0.0 (TID 0)
Caused by: java.lang.RuntimeException: Trying to read file with encrypted footer. No keys available
  at org.apache.parquet.hadoop.ParquetFileReader.readFooter(ParquetFileReader.java:638)
  at org.apache.parquet.hadoop.ParquetFileReader.readFooter(ParquetFileReader.java:582)
  at org.apache.parquet.hadoop.ParquetFileReader.readFooter(ParquetFileReader.java:553)
  at org.apache.spark.sql.execution.datasources.parquet.ParquetFileFormat$$anonfun$readParquetFoo
```

table schema and other metadata are protected

```
scala> sc.hadoopConfiguration.set("encryption.key.list" ,
     |    "k0:iKwfmI5rDf7HwVBcqeNE6w==," +
     |    "k1:LjxH/aXxMduX6IQcwQgOlw==," +
     |    "k2:rnZHCxhUHr79Y6zvQnxSEQ==," +
     |    "k4:mORupskWLHAfuvDJbXrCEw==," +
     |    "k5:NG6Hi85MW04sqMlXJHt5lg==")

scala>

scala> val carEventsEncrypted = spark.read.parquet("/home/gidon/Summit/Demo/CarEvents_Sept2018/*")
carEventsEncrypted: org.apache.spark.sql.DataFrame = [CarID: int, DriverID: int ... 18 more fields]
```

location key: not eligible

credit card key: not eligible

Hidden Columns

```
scala> carEventsEncrypted.createOrReplaceTempView("encryptedEvents")

scala>

scala> spark.sql("SELECT DriverID, Speed FROM encryptedEvents").show(2)
+---------+-----+
| DriverID|Speed|
+---------+-----+
|237857628|   90|
|237462937|  107|
+---------+-----+
only showing top 2 rows


scala>

scala> spark.sql("SELECT DriverID, Speed, Latitude, Longitude FROM encryptedEvents
18/09/06 09:02:59 ERROR Executor: Exception in task 0.0 in stage 4.0 (TID 4)
org.apache.parquet.crypto.HiddenColumnException: [Latitude]
```

Query Execution

```
scala> val speedingAtNight = spark.sql(
     |   "SELECT DriverID, MAX(ROUND(Speed/SpeedLimit - 1.0, 2)) AS AboveLimit, MAX(Speed) AS MaxSpeed " +
     |   "FROM encryptedEvents WHERE Speed > 1.2 * SpeedLimit AND HOUR(Timestamp) < 5 " +
     |   "GROUP BY DriverID ORDER BY MAX(Speed/SpeedLimit) DESC"
     | )
speedingAtNight: org.apache.spark.sql.DataFrame = [DriverID: int, AboveLimit: double ... 1 more field]

scala> speedingAtNight.createOrReplaceTempView("speedingAtNight")

scala> spark.sql(
     |   "SELECT FullName, AboveLimit, MaxSpeed, Email, MaskedCreditCard FROM (speedingAtNight " +
     |   "INNER JOIN encryptedPeopleTable ON speedingAtNight.DriverID=encryptedPeopleTable.PersonID)"
     | ).show
+--------------+----------+--------+--------------------+----------------+
|      FullName|AboveLimit|MaxSpeed|               Email|MaskedCreditCard|
+--------------+----------+--------+--------------------+----------------+
|   Jack Wilson|       0.8|     162|   jackw@example.com|         XX-8727|
|Anthony Grosse|      0.39|     125| antonyg@example.com|         XX-5613|
|Kayla Woodcock|      0.39|     125|  kaylaw@example.com|         XX-2338|
|     Amy Trefl|      0.37|     123|    amyt@example.com|         XX-7263|
| Jonathan Rupp|      0.34|     121|jonathanr@example...|         XX-9283|
|   Eva Muirden|      0.28|     115|    evam@example.com|         XX-8127|
+--------------+----------+--------+--------------------+----------------+
```

who is speeding at night?

Tampering with Data

```
$ $ $ $ cp People_08_2018.csv People_09_2018.csv
```

```
scala> peopleTableCSVSept18.createOrReplaceTempView("unprotectedPeopleTableSept18")

scala> spark.sql(
     |   "SELECT FullName, AboveLimit, MaxSpeed, Email, MaskedCreditCard FROM (speedingAtNight " +
     |   "INNER JOIN unprotectedPeopleTableSept18 ON speedingAtNight.DriverID=unprotectedPeopleTableSept18.PersonID)"
     | ).show
+---------------+----------+--------+--------------------+----------------+
|       FullName|AboveLimit|MaxSpeed|               Email|MaskedCreditCard|
+---------------+----------+--------+--------------------+----------------+
| Anthony Grosse|      0.39|     125| antonyg@example.com|         XX-5613|
| Kayla Woodcock|      0.39|     125|  kaylaw@example.com|         XX-2338|
|      Amy Trefl|      0.37|     123|    amyt@example.com|         XX-7263|
|  Jonathan Rupp|      0.34|     121|jonathanr@example...|         XX-9283|
|    Eva Muirden|      0.28|     115|    evam@example.com|         XX-8127|
+---------------+----------+--------+--------------------+----------------+
```

missed Jack Wilson, 80% above speed limit !!

Parquet Data Integrity Protection

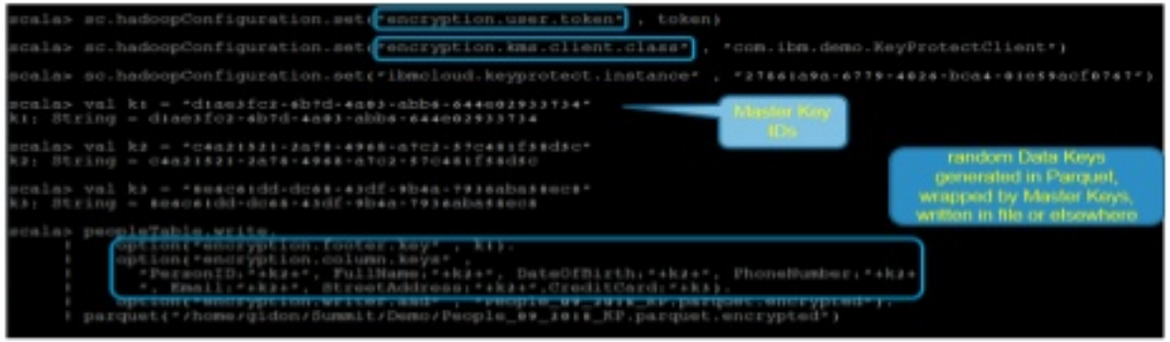
```
$
$ rm -rf People_09_2018.parquet.encrypted/
$ cp -r People_08_2018.parquet.encrypted/ People_09_2018.parquet.encrypted/
$
```

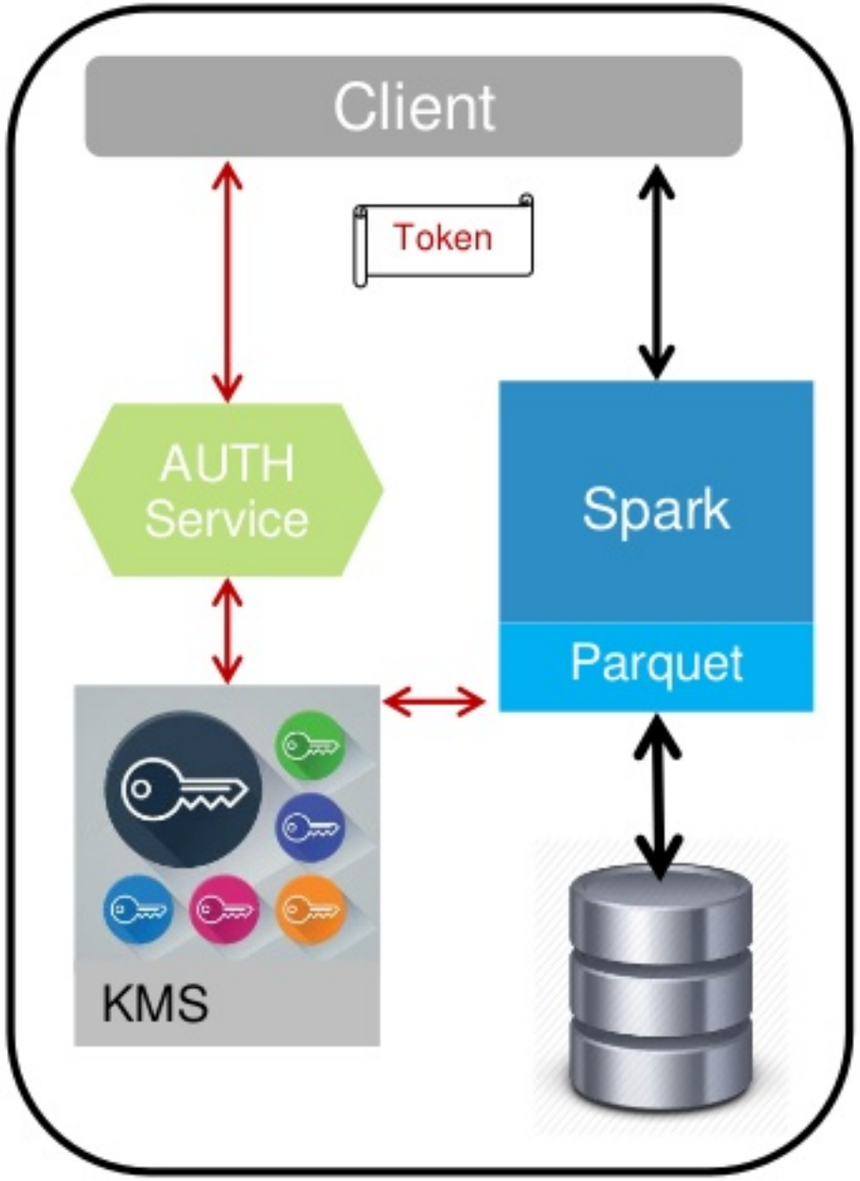
```
scala> val peopleTableEncrypted =
     |   spark.read.parquet("/home/gidon/Summit/Demo/People_09_2018.parquet.encrypted")
18/09/06 10:36:59 WARN TaskSetManager: Lost task 0.0 in stage 112.0 (TID 3963, localhost, executor dri
ver): java.io.IOException: Failed to decrypt
        at org.apache.parquet.crypto.AesDecryptor.decrypt(AesDecryptor.java:134)
```

```
Caused by: javax.crypto.AEADBadTagException: Tag mismatch!
  at com.sun.crypto.provider.GaloisCounterMode.decryptFinal(GaloisCounterMode
```

Key Management

- PARQUET-1373
  - randomly generated Data keys
  - wrapped in KMS with Master keys
  - storage options for wrapped keys
    - in file itself: easier to find
    - in separate location: easier to re-key
      - compromised Master key
  - client: specify Master key **ID** per column





IBM Spark-based Services

- IBM Analytics Engine
  - on-demand Spark (and Hadoop) clusters in IBM cloud
- Watson Studio Spark Environments
  - cloud tools for data scientists and application developers
  - dedicated Spark cluster per Notebook
- DB2 Event Store
  - rapidly ingest, store and analyze streamed data in event-driven applications
  - analyze with a combination of Spark and C++ engines, store with C++ Parquet
- SQL Query Service
  - serverless scale-out SQL execution on TB of data
  - Spark SQL engine, running in IBM Cloud

Column Access Control with Encryption @Uber

- Access control to sensitive columns only
- PARQUET-1178 provides fundamentals to encrypt columns
- PARQUET-1396 provides schema and key retrieval interface
- Currently under development - stay tuned!

Encryption Performance

- Ubuntu 16.04 box
  - 8-core Intel Core i7 6820HQ CPU / 2.70GHz
  - 32GB memory
- Raw Java AES-GCM speed
  - no Spark or Parquet, just encrypting buffers in loop
    - 8 threads, 1MB buffers
  - **Java 8: ~ 250 MB/sec**
  - **Java 9 and later: ~ 2-5 GB/sec**
    - AES-NI in Hotspot (hardware acceleration)
    - Slow warmup in decryption: Bug report JDK-8201633
      - can be bypassed with a workaround
      - *if you know someone who knows someone in Java Hotspot team…*

Spark Encryption Performance

- Local mode, 1 executor: 8 cores, 32GB memory
- Storage: Local RamDisk
- Write / encryption test
  - load carEvents.csv files (10GB)
  - cache in Spark memory!
  - write as parquet{.encrypted}: measure time (of second write)
- Query / decryption test
  - ‘read’ carEvents.parquet{.encrypted} files (2.7GB)
  - run “speeding at night” query: measure time

Spark Encryption Performance

Test	Write (sec)	Query (sec)	Notes
no encryption	26	2	query on 4 columns: input ~12% of data
encryption (GCM)	28.5	2.5	GCM on data and metadata
encryption (GCM_CTR)	26.8	2.2	CTR on data, GCM on metadata

- Typical usecases - only 5-10% of columns are sensitive (encrypted)
- Java 8… If needed, use GCM_CTR instead of GCM
- **Next Java version in Spark will support AES hardware acceleration**
- Encryption is not a bottleneck
  - app workload, data I/O, encoding, compression

To Be Benchmarked!

From Prototype to Real Thing

- Spark encryption prototype
  - standard Spark code, built with modified Parquet
  - encryption is triggered via Hadoop config
    - transparent to Spark
- That's it? Just upgrade to new Parquet version? *Not really ..*
  - partitions!
    - exception on encrypted columns? obfuscate or encrypt sensitive partition values?
  - Spark support for other things above file format
- TBD!

Parquet encryption: Community work. On track. Feedback welcome!

Spark encryption: Prototype, with gaps. Open challenge.

Backup

Existing Solutions

	Flat file encryption	Storage server encryption with range read	Storage client encryption with range read
Preserves analytics performance	X	V	V
Works in any storage system	V	X	X
Hides data and keys from storage system	V	X	V
Supports data integrity verification	V	V	X
Enables column access control (column keys)	X	X	X

- Bottom line: Need data protection mechanism built in the file format

Obfuscated Data

Transportation Safety Agency

no keys for personal data

anti-speeding ad campaign: budget per State?

```
scala> val speedingAtNight = spark.sql(
     |   "SELECT DriverIDHash, MAX(ROUND(Speed/SpeedLimit, 2)) AS AboveLimit " +
     |   "FROM encryptedEvents WHERE Speed > 1.2 * SpeedLimit AND HOUR(Timestamp) < 5 " +
     |   "GROUP BY DriverIDHash ORDER BY MAX(Speed/SpeedLimit) DESC"
     | )
speedingAtNight: org.apache.spark.sql.DataFrame = [DriverIDHash: string, AboveLimit: double]

scala> speedingAtNight.createOrReplaceTempView("speedingAtNight")

scala> spark.sql(
     |   "SELECT State, AVG(AboveLimit) FROM (speedingAtNight " +
     |   "INNER JOIN encryptedPeopleTable ON speedingAtNight.DriverIDHash=encryptedPeopleTable.PersonIDHash " +
     |   "GROUP BY State ORDER BY AVG(AboveLimit) DESC"
     | ).show
+-----+----------------+
|State|avg(AboveLimit)|
+-----+----------------+
|   NJ|           1.595|
|   PA|           1.365|
|   NY|            1.28|
+-----+----------------+
```

where the worst drivers live?