



APACHECON

---

---

# From Column-Level to Cell-Level: Towards Finer-grained Encryption in Apache Parquet

— Xinli Shang, Mohammad Islam —

---

---

# Speaker Intro

- Xinli Shang
  - Apache Parquet PMC Chair
  - Manager at Uber Data
- Mahammad Islam
  - Distinguished Engineer at Uber
  - PMC of Apache Oozie and Tez

# Agenda

- Apache Parquet introduction
- Modular/column-level encryption
- Cell-level encryption
  - Use cases
  - Challenges
  - Solutions
  - Benchmarking

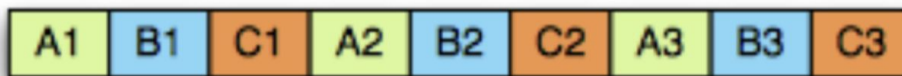
# Big Data Storage File Format

- Columnar storage file format
  - **Apache Parquet**
    - **Widely used Big Data File Format**
    - **Designed for efficiency, security & interoperability**
  - Apache ORC
- Row storage file format
  - Apache Avro, JSON, CSV

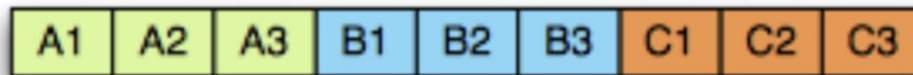
# Row-oriented v.s. column-oriented storage

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

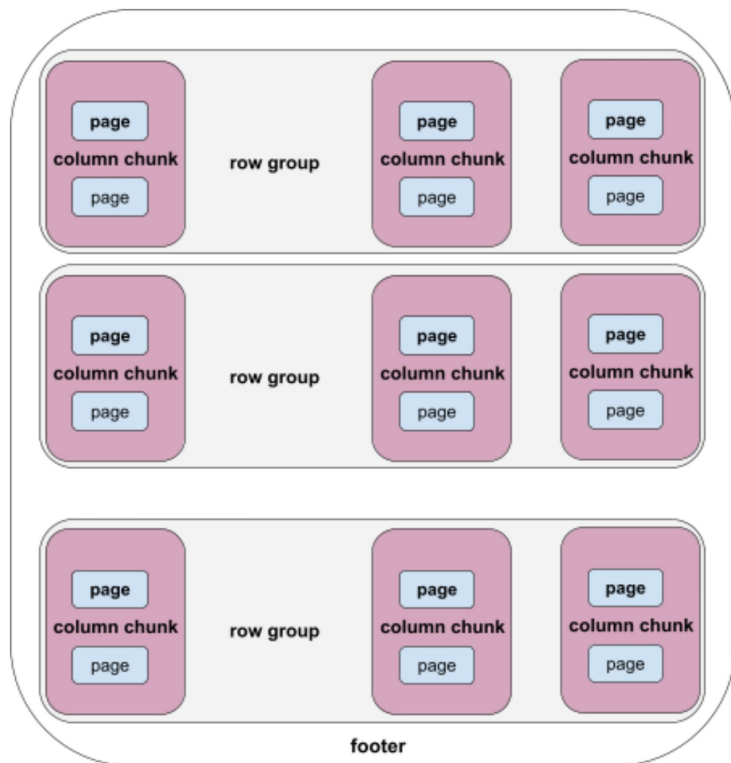
In row-oriented storage, data is one row at a time



In column-oriented storage, data is one column at a time



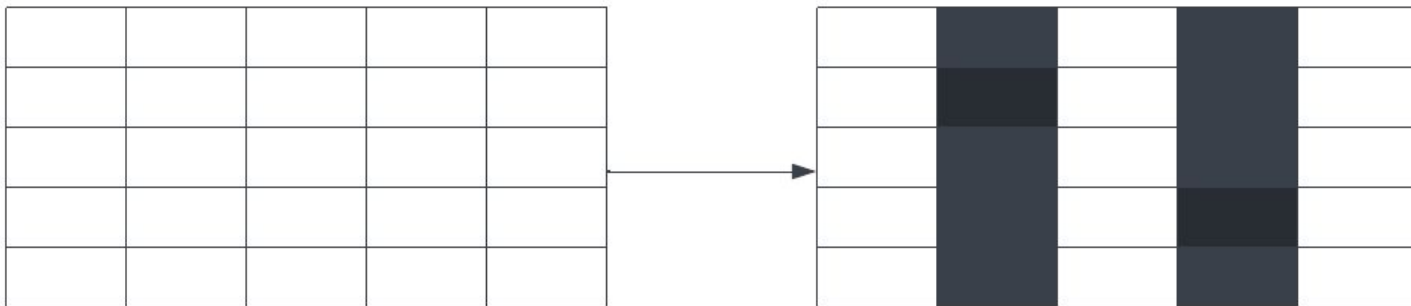
# Apache Parquet structure in a high level



- Each file has a footer - metadata & schema of the file
- Data is divided into 'row group'
- Each 'row group' has all column data called 'column chunk'
- Each 'column chunk' is further divided into 'page'
- Page is the unit for encoding, compression and encryption

# Column(modular) encryption in Apache Parquet

- Released in parquet-mr 1.12.0
- Column (also called module) is encrypted independently with it's own key
- Already adopted by the industry
  - e.g. data retention, encryption-on-write-then-delete after x days
  - [One Stone, Three Birds: EngBlog](#)



# Finer grain encryption than column level is needed

- A table have mixed data from different country and pii/non-pii
- Different country has different requirements for access and data retention
  - Encrypt data firstly then delete the encryption key after m days
- Different PII data has different sensitivity and requires different protection

country	non-pii	latitue	longitude	non-pii	email	non-pii	non-pii
country_a							
country_b							
country_c							
country_d							
country_e							



# Technical challenges

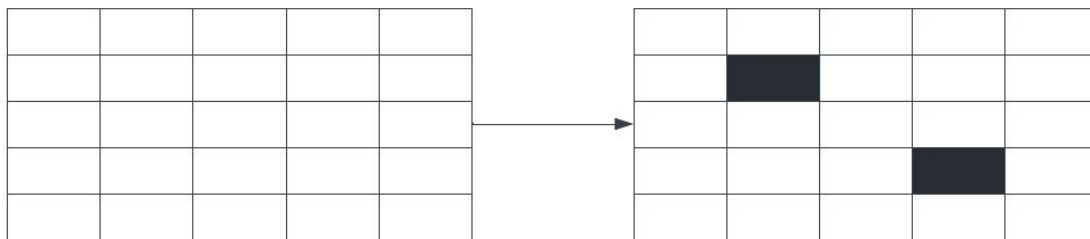
- Parquet is columnar storage
  - Field/record level encryption go against the design
  - Key metadata, algorithm info need to pass to be stored in place
- Encryption is generally a block operation
  - Doesn't apply to some data types like integer, float, boolean ...

# Technical approaches

- FPE(Format Preserving Encryption) in-place encryption
- Column-splitting then column encryption
- Adding string column then record level encryption

# Solution 1: FPE in-place encryption

- FPE is used to encrypt cell data while preserving the original data type.
  - double->double
  - string->string
  - ...
- The encryption can be done in-place
  - Plaintext cell data is encrypted in existing cell



# Pros & cons of FPE solution

- Pros
  - In-place encryption, no need extra place to hold encrypted data
- Cons
  - Need to record which cell is encrypted and store it somewhere
    - Specification change could impact multiple version of Parquet implementation
  - There are ongoing concerns about FPE
    - FF2 & FF3 are not considered to be cryptographically secure

# Solution 2: Column-splitting then column encryption

- Clone columns with the same data type as the original column
  - Adding, write-splitting, read-merging are done inside Parquet
- Apply modular(column) encryption to hidden columns

<u>Original Values</u>		<u>Values with Cell-Level Encryption</u>		
column_name		column_name	_9b06dfc1_column_name_1	_9b06dfc1_column_name_2
3		3	NULL	NULL
5		NULL/MASKED	encr_key1(5)	NULL
100		100	NULL	NULL
2		NULL/MASKED	encr_key1(2)	NULL
8		NULL/MASKED	NULL	encr_key2(8)

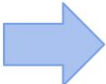
# Pros & cons of column splitting & then module encryption

- Pros
  - AES is more secure and mature than FPE
  - Column encryption is already adopted in industry, stable and mature now
  - No need specification change of Parquet
- Cons
  - Add overhead for splitting columns and merging
  - Synchronization is needed cross columns when applying filter

# Solution 3: Add string column & record encryption

- Similar as column-splitting, but only add one single column
- Cell data is encrypted individually and is stored in the string column
  - The encrypted string contains key metadata, algorithm info...
  - Merge the two columns when reading

<u>Original Values</u>		<u>Values with Cell-Level Encryption</u>
column_name		column_name    _9b06dfc1_column_name_1
3		3    NULL
5		NULL/MASKED    'key1_aes_xxx'
100		100    NULL
2		NULL/MASKED    'key1_aes_yyy'
8		NULL/MASKED    'key2_aes_zzz'



# Pros & cons of adding string column & record encryption

- Pros
  - Add less columns than approach #2
- Cons
  - String column has more overhead
  - Each record need to carry the key metadata, encr algorithm. Add more space overhead



# Current status

- Approach #2 (column-splitting) is recommended in community
- Requesting for more comments ([parquet-2116](#), [design doc](#))
- Internal implementation is rolled out to production. Will open PR shortly.

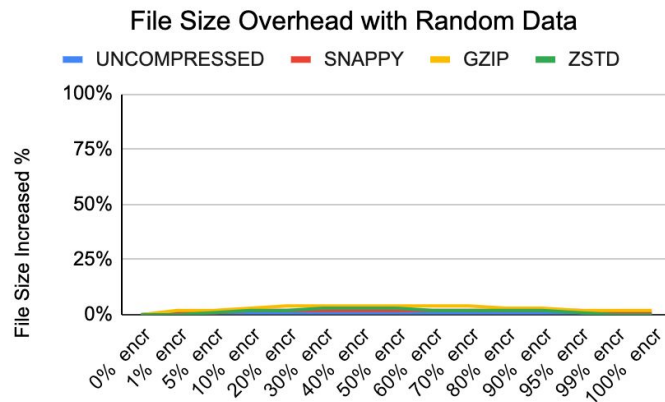
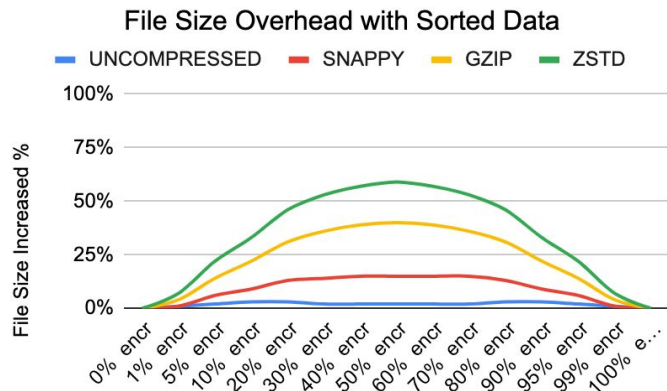
# Overhead benchmark of approach #2

- Space overhead
  - Hidden columns add more size
  - Change of data order in original column can result in size increase
- Time overhead
  - More time needed for splitting in write and merging in read
  - Need to deal with more data as discussed in space overhead

# Space overhead

- 3.1 GHz Quad-Core Intel Core i7,
- 16G 2133 MHz Memory
- macOS Monterey Version 12.2.1(21D62)

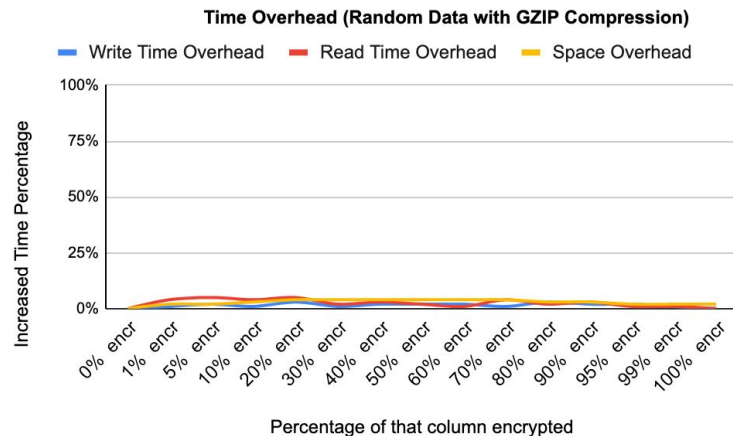
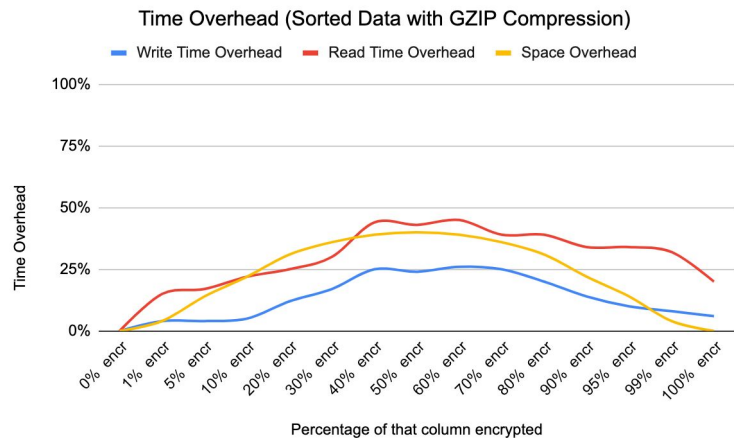
- 5 columns: 1 long and 4 string
- Uncompressed file size is 152MB



# Time overhead

- 3.1 GHz Quad-Core Intel Core i7,
- 16G 2133 MHz Memory
- macOS Monterey Version 12.2.1(21D62)

- 5 columns: 1 long and 4 string
- Uncompressed file size is 152MB



# Compatibility of Approach #2

- Backward compatibility
  - Parquet with this feature read data written by older version Parquet
  - No changed behavior is expected
- Forward compatibility
  - Older version Parquet read cell-level encrypted data
  - No specification change but it adds hidden column
  - Next slide →

## Forward compatibility of Approach #2

Requested Schema	Behavior
No cell-encrypted data is requested	No change
Request original column but no hidden column	return data with null or masked value
Request original column with hidden column like 'select *...'	Either throw exception or user see extra columns

# Summary

- Introduce cell-level encryption
- Several approaches available
  - FPE & in-place
  - **Column-splitting then modular encryption (recommended)**
  - Adding string column then record encryption
- Benchmarking, compatibility, current status

# Q & A

**We are hiring!**

email: [shangxinli@apache.org](mailto:shangxinli@apache.org)