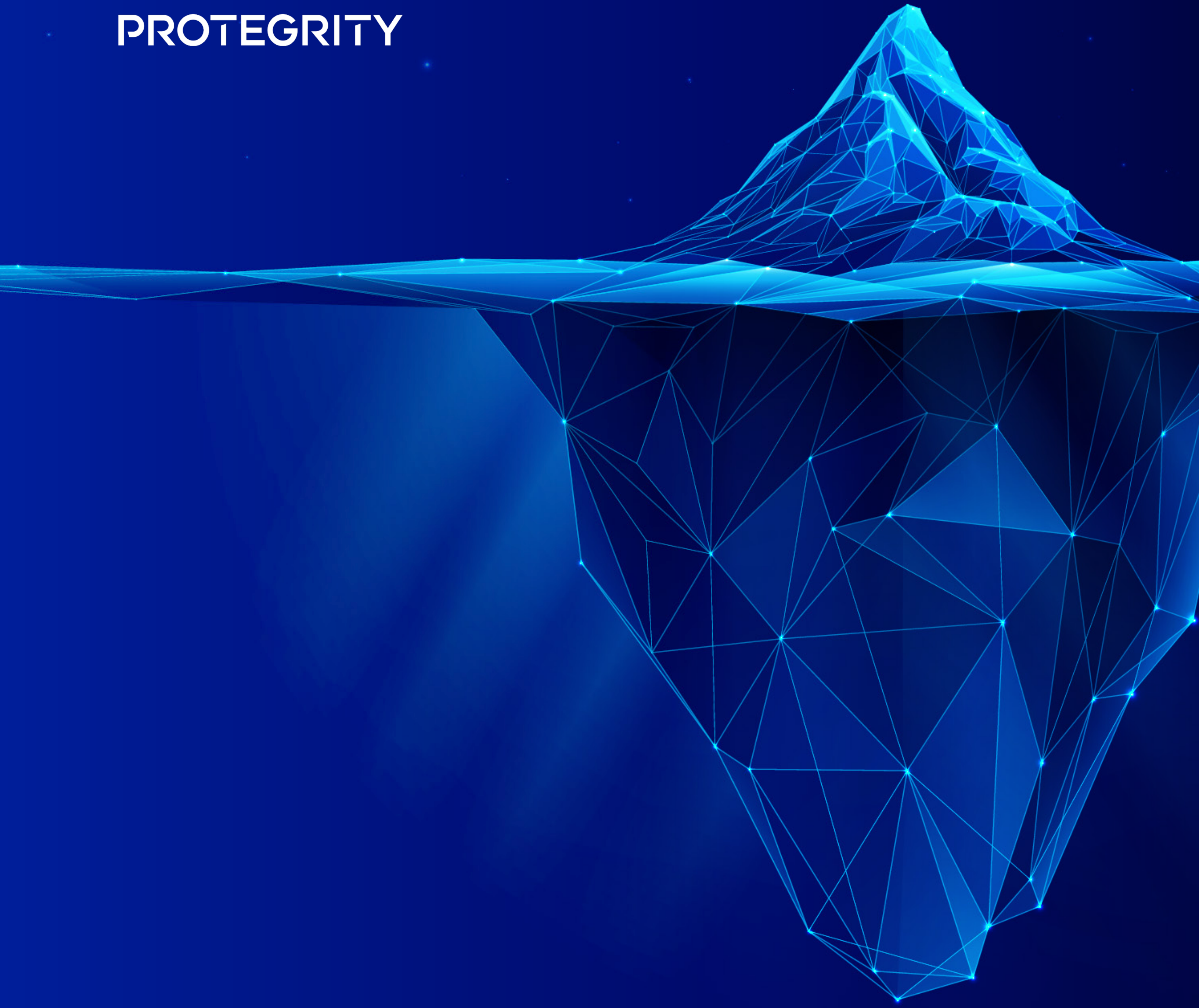
PROTEGRITY



DATA SECURITY IN APACHE ICEBERG

1. EXECUTIVE SUMMARY

Apache Iceberg is an open-source table format increasingly adopted for large-scale data analytics. However, it currently lacks comprehensive data security features for enterprise environments where sensitive data protection is imperative. This document proposes an enterprise-ready solution for implementing granular data security in Apache Iceberg by extending the Parquet Modular Encryption (PME) framework. We are calling this the Iceberg Data Security framework.

The Iceberg Data Security framework addresses critical security challenges in distributed data environments by enabling:

A. **Pervasive Data Protection:** Security policies follow data across the enterprise regardless of storage location or processing system, ensuring consistent protection throughout the data lifecycle.

B. **Granular Access Controls:** Fine-grained security at the column, row, and cell levels based on user context and data classification attributes.

C. **Optimized Performance:** Maintains Iceberg's performance advantages through selective reads, predicate push-down, and columnar projection while applying security controls.

D. **Segregation of Concerns:** Separates the Iceberg system from the data protection mechanisms, allowing specialized security services to manage protection policies.

E. **Enterprise Integration:** Enables integration with existing enterprise security infrastructure including Key Management Systems (KMS) and Hardware Security Modules (HSM).

F. **External Security Service Support:** Allows different third-party security solutions to interface with Iceberg through standardized APIs.

G. **Transparent Implementation:** Applications interact with protected data without needing to implement encryption/decryption logic.

The proposed architecture builds upon established open-source standards, specifically extending Parquet Modular Encryption while enabling integration with external data protection services. This approach positions Apache Iceberg as an enterprise-ready table format that can meet stringent regulatory compliance requirements while maintaining the performance characteristics that make Iceberg valuable for analytics workloads.

This solution brings significant business value by enabling organizations to implement consistent data security across their data landscape, particularly for sensitive data subject to regulatory requirements such as PII (Personally Identifiable Information). It also extends Iceberg's utility to security-sensitive industries including finance, healthcare, and government sectors.

2. INTRODUCTION

Apache Iceberg currently does not provide granular level data security capabilities. In distributed heterogeneous environments, data flows between systems and is frequently replicated across tables and views. To maintain consistent security controls on sensitive data, organizations need mechanisms to trace each instance of protected data and apply appropriate controls regardless of location.

In an optimal enterprise environment, data should be protected from the moment of acquisition, with security properties persisting across the organization regardless of the data stores and processing systems involved. Every system in the data flow should recognize protected data and enforce the appropriate security controls during processing.

A. High level requirements for data security

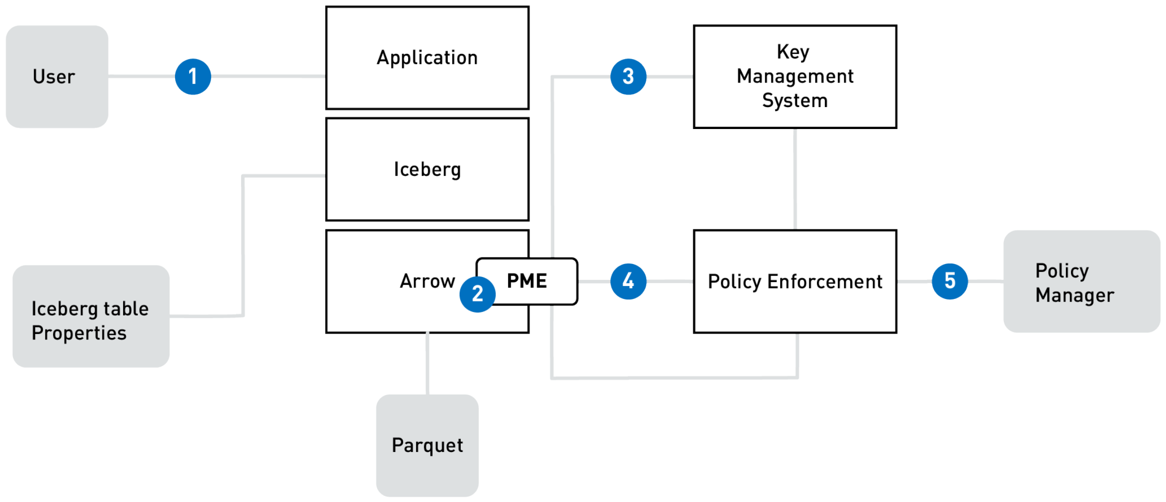
The proposed solution aims to satisfy the following key requirements:

1. Enable data identification and classification as integral components of the data protection mechanism.
2. Support ingestion of protected data into Iceberg tables.
3. Allow attachment of files containing protected data to existing Apache Iceberg tables.
4. Optimize data retrieval based on cleartext criteria, eliminating the need for full table scans.
5. Enforce data access controls based on user context and data attributes.
6. Maintain separation between the Iceberg system and data protection mechanisms.
7. Enable centralized management of attributes assigned to protected data.
8. Provide centralized management of access controls for protected data.
9. Allow systems without access to data protection mechanisms to handle protected data in a pass-through mode.

3. PROPOSED SOLUTION: ICEBERG DATA SECURITY FRAMEWORK

A. Architecture Overview

The solution architecture enables Apache Iceberg tables to be serialized in Parquet format with the Iceberg Data Security framework.



The above network diagram illustrates the high-level architecture of the solution. Iceberg tables serialized in Parquet format with the Iceberg Data Security framework.

The set up of this solution requires encryption configuration parameters to be stored in Iceberg Table properties. Once this is done, encrypted column data is fetched from a Parquet file and column chunks are sent over to a Data Protection API with the user context for decipherment.

1. User authenticates with the application.
2. PME API is invoked by Arrow to decipher column chunks and metadata (statistics and indexes).
3. PME unwraps the metadata encryption key and deciphers metadata for query optimization (skip chunks or files).
4. PME transmits selected encrypted column chunks to Data Protection for decipherment, with Key Id assigned to a column and User context.
5. The Policy Enforcement uses the Data Encryption Key from the KMS and applies policies fetched from the Policy Manager for data encryption and decryption.

B. PME Integration

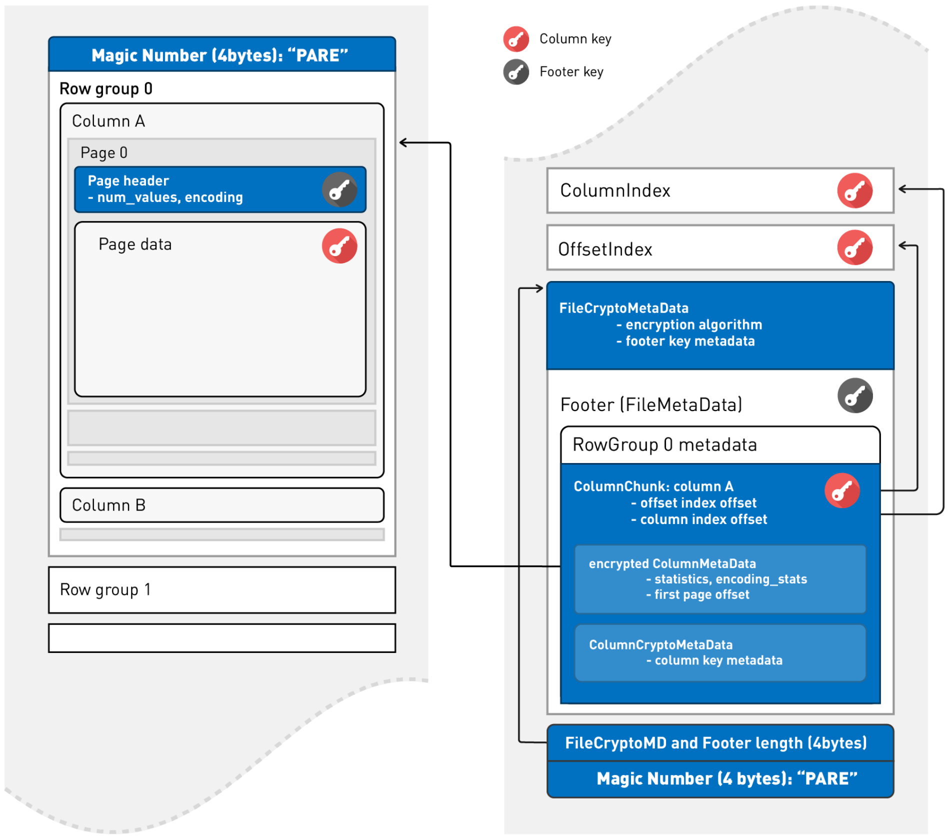
Integration and extension of PME goal is to enable and define a standard protocol and mechanism to delegate the column chunk encryption to an external service.

Currently PME implementation relies on internal encryption implementation with the ability to wrap data encryption keys (DEK) with an external key encryption key (KEK).

In a similar manner, the Iceberg Data Security framework would require an application to provide details about an external data protection service and the required security attributes such as credentials and TLS.

PME enables protection of 2 parts in Parquet independently:

1. **Metadata:** footer data.
2. **Column data:** column data chunks.



The main change is to instead of specifying footer and column keys, the configuration would define an optional attribute to indicate if the encryption of that part is external. If external, then details on connection must be provided.

There should be 2 types of providers:

1. **Dynamic Loaded Library (DLL)** – the protection module is dynamically loaded into the application, and direct API calls be made to encrypt and decrypt.
2. **External** – the protection module is deployed outside of the application process, and all calls to encrypt and decrypt be made over the network using REST API.

PME Reference: https://parquet.apache.org/docs/file-format/data-pages/encryption/

C. PME Configuration

Parquet PME requires encryption configuration parameters to be set for the read/write parquet APIs and stored with the Iceberg Table properties.

Column Encryption:

```
parameter name: "encryption.column.keys"
parameter value: "<key ID>:<column>,<column>;<key ID>:<column>,.."
```

Footer Encryption:

```
parameter name: "encryption.footer.key"
parameter value: "<key ID>"
```

Alternatively, a footer can be left in clear for compatibility with clients that do not support PME, but the encryption footer key must be specified for integrity validation where the key is used for signature.

```
parameter name: "encryption.plaintext.footer"
parameter value: "true"
```

Encryption algorithm:

By default, Parquet modular encryption uses the AES-GCM algorithm that provides full protection against tampering with data and metadata in Parquet files. However, as Spark 2.3.0 runs on Java 8, which doesn't support AES acceleration in CPU hardware (this was only added in Java 9), the overhead of data integrity verification can affect workload throughput in certain situations.

To compensate this, you can switch off the data integrity verification support and write the encrypted files with the alternative algorithm AES-GCM-CTR. AES-GCM Counter mode allows integrity verification of the metadata parts only and standard encryption of the data parts. AES-GCM_CTR has a lower throughput overhead compared to the AES-GCM algorithm.

```
parameter name: "encryption.algorithm"
parameter value: "AES_GCM_CTR_V1"
```

Key Management System/Hardware Security Module Integration:

PME can accept data encryption keys in the clear from the application, however in real-world scenario the keys are wrapped with a Key Encryption Key stored in KMS or HSM. The application has to provide with KMS client class that supports 2 main interfces for wrapping and un-wrapping keys.

```
parameter name: "parquet.ecnryption.kms.client.class"
parameter value: "<class pat to KMS client>"
```

The KMS client might require specific initialization paraments which can include the URL and access tokens.

```
// Wraps a key - encrypts it with the master key, encodes the result and
// potentially adds KMS-specific metadata.
public String wrapKey(byte[] keyBytes, String masterKeyIdentifier)
// Decrypts (unwraps) a key with the master key.
public byte[] unwrapKey(String wrappedKey, String masterKeyIdentifier)
// Initialize KMS client as an example:
public void initialize(Configuration configuration, String kmsInstanceID,
String kmsInstanceURL, String accessToken)
```

Sample KMS client code: https://github.com/apache/parquet-java/blob/master/parquet-hadoop/src/test/java/org/apache/parquet/crypto/keytools/samples/VaultClient.java

5. ADVANTAGES AND BENEFITS

The proposed solution leverages open-source capabilities while enabling third-party data protection services to enforce access controls, dynamic data masking, encryption, and tokenization. Key advantages include:

A. Optimized Performance:

- Column statistics and footer indexes operate on cleartext.
- Search and sort operations can skip irrelevant files and column chunks.
- Parallel access to column chunks improves performance.
- Bulk processing of column chunks reduces API call overhead.
- Compression before encryption minimizes network latency.

B. End-to-End Protection:

- Data protection from acquisition through distribution and use.
- Consistent protection during ingestion, ETL processes, and analytics.

C. Application Transparency:

- Applications remain unaware of underlying encryption.
- Queries operate on cleartext representations without special handling.

D. Enhanced Security:

- Applications and Iceberg systems avoid exposure to Data Encryption Keys.
- Centralized management of data security policies .
- Compatible with fully managed deployments (SaaS).

Applications requiring access to encrypted Parquet files in Iceberg only need to provide user credentials or access tokens, without additional configuration related to encrypted columns.

6. SUMMARY

The proposed solution is based on open-source capabilities that enable 3rd party data protection services to enforce access controls, dynamic data masking, encryption, and tokenization.

The advantages of extending PME with granular data protection are:

1. **Optimized access to the encrypted data:**
    a. Column statistics and footer indexes are based on clear text.
    b. Search and sort can skip files and column chunks if they are not matching conditions, eliminating full table scan.
    c. Access to column chunk is in parallel.
    d. Column chunk retrieval allows bulk data processing instead of calling an API for each data item (bulk vs. scalar).
    e. Column data chunks are compressed by Arrow before PME encryption, further minimizing the network overhead latency.

2. **Carry protection properties across systems in the data flow:**
    a. Protect column data in Hadoop during ingestion (acquisition).
    b. Invoke an ETL data flow to copy data into Iceberg, or simply "attach" the underlying parquet table files to an Iceberg table (distribution).
    c. Retrieve column data from an Iceberg table for analytics and reporting (use).

3. **Transparent to the application:**
    The application does not have to be aware that the column data is encrypted, and the queries are based on clear text. There is no need for the application to encrypt the clear text to compare with the stored cipher text.

4. **Enhanced Security:**
    a. Application and Apache Iceberg are not exposed to Data Encryption Keys. Which makes the solution suited for fully managed deployments (SaaS).
    b. Data security policy managed centrally and enforced by the Data Protection service.

Applications that require access to encrypted Parquet files in Iceberg would need to provide user account credentials or access tokens, without any additional properties related to encrypted columns.