Encryption Codec

Documentation

OVERVIEW

The Lucene encryption codec enables the encryption of different part of the Lucene index, such as the Term Dictionary, the Stored Fields, and the Term Vectors.

In comparison with approaches where all data is encrypted (e.g., file system encryption, index output / directory encryption), encryption at a codec level enables a more fine-grained control on which blocks of data are encrypted. This is more efficient since less data has to be encrypted. This also gives more flexibility such as the ability to select which field to encrypt.

Some of the requirements for this project were:

- The performance impact of the encryption should be reasonable.
- The user can choose which field to encrypt.
- The user can control how a data block is encrypted, e.g., the encryption function to use, the generation of the initialisation vectors, etc.
- Key management: During the life cycle of the index, the user can provide a new version of the encryption key. Multiple key versions can co-exist in one index.

What is currently supported ?

- Block tree terms index and dictionary
- Compressed stored fields format
- Compressed term vectors format
- Index upgrader: command to upgrade all the index segments with the latest key version available.

What is currently NOT supported ?

- Lucene's doc values format, which encodes/decodes per-document values. A prototype format that encrypts doc values has been implemented, but it is not production-ready.
- Lucene's point format, which encodes dimensional values in a block KD-tree structure for fast shape intersection filtering.

Warning: The EncryptedLucene50PostingsFormat currently encrypts only the term dictionary but does not encrypt the postings list. The postings list does not contain the textual data of the term but instead information about where a particular term appears, its frequency and positions. An attacker could potentially use the frequency information of a term to derive a term frequency distribution and maps it to the term frequency distribution of a given language, e.g., english, in order to guess the deciphered textual value of a term. This could potentially work for large textual fields, e.g., the content of an article, but will likely generate scrambled text. It will however not work for fields with a single or few values, e.g., numerical fields or the name of the author of the article.

CODEC API

Cipher Factory

The CipherFactory provides an API to specify how a data block is encrypted. Implementations of the CipherFactory will be responsible in:

- Providing instances of javax.crypto.Cipher for a given crypto algorithm and initialised with the secret key specifications;
- Generating a secure initialisation vector for each data block.

The class DummyCipherFactory is an example of such an implementation. This class is based on a fixed secret key and is not meant to be used in production.

In a real world scenario, the user can for example provide a CipherFactory implementation that will fetch the secret key for a given index from a secure location (e.g., from a local file or from a rest endpoint), initialise the cipher instance with the secret key, and finally clean any trace of the secret key from the java heap memory.

Cipher Version and Key Management

One requirement was the ability to be able to change the secret key at runtime for a given index. This means that an index can be encrypted using multiple keys. To support such a requirement, a cipher instance is associated with a unique version number, and the CipherFactory implementation is responsible in providing the cipher instance for a given version number.

At an index level, one index segment is encrypted with a single cipher version. An index can have multiple segments, each one encrypted using a different cipher version. The cipher

version for a segment is stored in the segment info at indexing time, and read at query time to instantiate the correct cipher instance for a given segment.

Encrypted Lucene Codec

The EncryptedLucene60Codec is an abstract class that defines a default encryption codec based on the Lucene60 codec. When implementing this class, users can specify which CipherFactory to use, which part of the index is encrypted, and which field is encrypted. The class DummyEncryptedLucene60Codec is an example of such an implementation.

The CipherFactory instance to use when encrypting the index must be specified with the method 'getCipherFactory()'.

> **Note:** It is recommended to enable the term dictionary, the stored fields and term vectors when encrypting a field. If the field is both indexed and stored, and only the term dictionary is encrypted, then the original content of the field is stored on disk in an unencrypted form. The attacker could then simply read the stored fields to retrieve the content of the field.

How to encrypt the Term Dictionary ?

If the term dictionary must be encrypted, the user has to specify with the method 'getPostingsFormatForField(String field)' an instance of the EncryptedLucene50PostingsFormat for each field that must be encrypted. If a field should not be encrypted, an instance of the original Lucene50PostingFormat should be provided.

How to encrypt the Stored Fields ?

If the stored fields must be encrypted, the user has to specify with the method 'storedFieldFormat()' an instance of the EncryptedLucene50StoredFieldsFormat. This class is an abstract class itself, and user can specify which field to encrypt by overriding the method 'isFieldEncrypted(String field)'.

How to encrypt the Term Vectors ?

If the term vectors must be encrypted, the user has to specify with the method 'termVectorsFormat()' an instance of the EncryptedLucene50TermVectorsFormat. This class is an abstract class itself, and user can specify which field to encrypt by overriding the method 'isFieldEncrypted(String field)'.

Index Upgrader

During the life cycle of the index, we have seen that it is possible to change the secret key at runtime. This means that during at a given point in time, an index can be composed of index segments that are encrypted with more or less old secret keys.

The codec provides a tool to help upgrading an index with the latest available secret key. It is a fork of the org.apache.lucene.index.IndexUpgrader tool that upgrades all segments of an index that are encrypted with an old cipher version by re-encrypting them with the latest cipher version.

This tool can be used from command line:

```
$ java -cp "lucene-core.jar:lucene-codecs.jar"
org.apache.lucene.codecs.encrypted.IndexUpgrader [-delete-prior-commits]
[-verbose] indexDir
```

Alternatively this class can be instantiated and the method ‘upgrade()’ invoked. It uses a CipherUpgradeIndexMergePolicy and triggers the upgrade via a forceMerge request to IndexWriter.

ROADMAP

- Finalising the existing doc values format prototype, measuring its performance through benchmarks, optimising code.
- Adding an encrypted point format.
- Solr integration