Secure Spark Shuffle:
A fast and convenient approach using Apache Commons Crypto

Cheng Xu @Intel
xuf@apache.org
About me

Cheng Xu
Software Engineer@Intel
Committer of Apache Commons Crypto, Apache Hive
Contributor of Apache Spark and Apache Parquet
Now focus on Big data security related work
What’s Spark shuffle?
Why needs encryption in shuffle?
Why uses Apache Commons Crypto?
How to configure Apache Commons Crypto in Spark?
What’s Apache Commons Crypto?
How to use Commons Crypto?
What's Shuffle
Shuffle in MapReduce
Shuffle in Spark

“Mapper”
- Emit the data

“Reducer”
- Consume the data

“Shuffle”
- What happens between “Mapper” and “Reducer”
Why needs encryption in Spark shuffle?
Why encryption is needed for shuffle?

Network
- Sensitive data transferred via network

Disk
- Sensitive data stored at disk
Where needs encryption in Spark Shuffle?

**RPC**
- Block Transfer Service
- External Shuffle Service

**Data-at-Rest**
- Spilled files
- Shuffle output files
Spark shuffle file encryption implementation

Spark Client -> SparkHadoopUtils

Generate Key

Credentials

Get Key

Shuffle Reader

Plain text

Compression

Crypto InputStream

Initial Vector

Key

Encrypted local shuffle files

16 BITS

Initial Vector

Encrypted text

Shuffle Writer

Plain text

Decompression

Crypto OutputStream

Initial Vector

Key

Crypto InputStream
Why uses Apache Commons Crypto?
Why Apache Commons Crypto?

Performance Matters!
Apache Commons Crypto

History
- Origin from the Apache Hadoop
- Was Chimera contributed by Intel
- Become parts of Apache Commons

Features
- Cryptographic library optimized with AES-NI
- Provide Java API for both cipher level and Java stream level
- Provide Java API for security random
Performance Matters

AES-NI

- Intel Advanced Encryption Standard New Instructions
- An extension to the x86 instruction set architecture for microprocessors from Intel and AMD proposed by Intel in March 2008
- New instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AESENC</td>
<td>Perform one round of an AES encryption flow</td>
</tr>
<tr>
<td>AESENCLAST</td>
<td>Perform the last round of an AES encryption flow</td>
</tr>
<tr>
<td>AESDEC</td>
<td>Perform one round of an AES decryption flow</td>
</tr>
<tr>
<td>AESDECLAST</td>
<td>Perform the last round of an AES decryption flow</td>
</tr>
<tr>
<td>AESKEYGENASSIST</td>
<td>Assist in AES round key generation</td>
</tr>
<tr>
<td>AESIMC</td>
<td>Assist in AES Inverse Mix Columns</td>
</tr>
<tr>
<td>PCLMULQDQ</td>
<td>Carryless multiply</td>
</tr>
</tbody>
</table>
Performance Matters

AES-NI support

• OpenSSL 1.0.1c or above
• JDK 1.7.045 or JDK 8 (not optimized)

JDK optimized:

• CTR mode optimization: 5~7X gain
• CBC mode optimization: ~50% gain for 256 bit key and ~15% gain with 192 bit key on 64 bit system
• Will ship in JDK9
Secure Random

OpenSSL random

RDRAND instruction
Commons Crypto Performance

Micro Benchmark
TeraSort
TPCx-BigBench
Measure throughput of decryption and encryption

Comparisons in CTR and CBC mode

- AES-NI VS. Non AES-NI (OpenSSL VS. JDK 1.6.037)
- AES-NI VS. AES-NI without optimized (OpenSSL VS. JDK 1.7.060)
AES-NI VS. Non AES-NI in CTR mode (JDK 1.6.0_37)
AES-NI VS. Non AES-NI in CBC mode (JDK 1.6.0_37)

Throughput in CBC mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>AES-NI</th>
<th>Non AES-NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>548.47</td>
<td>98.56</td>
</tr>
<tr>
<td>Decryption</td>
<td>1774.69</td>
<td>106.84</td>
</tr>
</tbody>
</table>

- AES-NI: 16X faster
- Non AES-NI: 5X faster
AES-NI vs. AES-NI without optimized in CTR mode (JDK 1.7.060)

Throughput in CTR mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>AES-NI</th>
<th>AES-NI without optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>1845.04</td>
<td>188.99</td>
</tr>
<tr>
<td>Decryption</td>
<td>1878.89</td>
<td>220.21</td>
</tr>
</tbody>
</table>

9.7X
8.5X
AES-NI VS. AES-NI without optimized in CBC mode (JDK 1.7.060)

Throughput in CBC mode

**Encryption**
- AES-NI: 542.08
- AES-NI without optimized: 432.79

**Decryption**
- AES-NI: 1903.345
- AES-NI without optimized: 254.53

1.25X
5X
TeraSort benchmark

**TeraSort**
- SortByKey

**Execution Environment**
- CDH cluster with three node
- Spark on Yarn

**Encryption performance**
- RPC
- File
Performance in TeraSort: file encryption

Execution Time for file encryption

- **CUMULATIVE (IN SEC)**: 36 - 78
- **SORTBYKEY (IN SEC)**: 31 - 42
- **TOTAL (IN SEC)**: 28 - 138

- **encryption using Openssl cipher**
- **encryption using JCE cipher**
- **unencryption**
Performance in TeraSort: RPC encryption

Execution Time in PRC encryption

- CUMULATIVE (IN SEC): 234
- SORTBYKEY (IN SEC): 216
- TOTAL (IN SEC): 264

The chart shows the execution time for different operations using encryption methods: AES-NI, 3DES, and unencryption. The y-axis represents the time in seconds, and the x-axis shows the cumulative time for encryption and decryption tasks.
TPCx-BigBench

TPCx-BB
• Measure the performance of Hadoop based Big Data systems
• SQL for structured data
• Machine learning algorithms for semi-structured and unstructured data

Workload
• Hive On Spark
• Hive On MR
• Hive On Tez
Performance in PCx-BigBench: PRC encryption

EXECUTION TIME

- A-Default(s)
- B-Encryption with 3DES(s)
- C-Encryption with AES-NI(s)
How to configure Spark shuffle file encryption

Spark.shuffle.encryption.enabled

Spark.commons.crypto.cipher.classes
  • org.apache.commons.crypto.cipher.OpensslCipher

Spark.shuffle.crypto.cipher.transformation
  • AES/CTR/NoPadding
How to configure Spark shuffle RPC encryption

External shuffle service on Yarn

Yarn side (Sasl sever)
- spark.authenticate = true
- spark.authenticate.enableSaslEncryption = true
- spark.authenticate.sasl.encryption.aes.enabled = true

Spark side (Sasl client)
- spark.authenticate = true
- spark.authenticate.enableSaslEncryption = true
- spark.authenticate.sasl.encryption.aes.enabled = true
How to configure Spark shuffle RPC encryption

Spark

• `spark.authenticate`
• `spark.authenticate.enableSaslEncryption`
• `spark.authenticate.sasl.encryption.aes.enabled`
Future work

Apache Commons Crypto
• Prepare release 1.0.0

Spark
• Merge existing PR (https://github.com/apache/spark/pull/8880) into upstream
• Open PR for RPC optimization

Apache Commons Crypto enabling
• Optimize other Hadoop core components (e.g. Hadoop, Hbase etc.)
Any Questions?
thank you
Reference

https://bugs.openjdk.java.net/browse/JDK-7184394
https://bugs.openjdk.java.net/browse/JDK-8143925
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
http://www.tpc.org/tpcx-bb/
https://github.com/apache/commons-crypto
http://issues.apache.org/jira/browse/Spark-5682
Mode of operation

- An algorithm that uses a block cipher to encrypt messages of arbitrary length in a way that provides confidentiality or authenticity (National Institute of Standards and Technology)

Common modes

- Counter mode
- Cipher Block Chaining mode
Cipher Block Chaining mode

CBC mode

XOR

Use case

General file or packet encryption
Counter mode

CTR mode
• Stream cipher
• Allow random access when decryption

Use case
• Very high-speed data
Usage of Commons Crypto

Simple
• Add Commons Crypto Jar to Classpath

Cipher API
• Low level cryptographic operations

Java stream API
• High level stream encryption/decryption

Security Random API
• High speed random data generator
Properties properties = new Properties();

// Creates a CryptoCipher instance with the transformation and properties.

CryptoCipher cipher = Util.getCipherInstance(CipherTransformation.AES_CTR_NOPADDING, properties);

String input = "hello world!";
int inputOffset = 0;
int inputLen = input.length();
byte[] output = new byte[1024];
int outputOffset = 0;

// Initializes the cipher with ENCRYPT_MODE, key and iv.
cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(key,"AES"), new IvParameterSpec(iv));

//Continues a multiple-part encryption/decryption operation for byte array.
cipher.update(input.getBytes("UTF-8"), inputOffset, inputLen, output, outputOffset);

// We should call doFinal at the end of encryption/decryption.
cipher.doFinal(inBuffer, outBuffer);

// Closes the cipher.
cipher.close();
int bufferSize = 4096;
String input = "hello world!";
byte[] decryptedData = new byte[1024];
//Encryption with CryptoOutputStream.
//Constructs the original OutputStream.
OutputStream outputStream = new ByteArrayOutputStream();
//Creates a CryptoCipher instance with the transformation and properties.
CryptoCipher cipher = CryptoStream.getCipherInstance(CipherTransformation.AES_CTR_NOPADDING, properties);
//Constructs the instance of CryptoOutputStream.
CryptoOutputStream cos = new CryptoOutputStream(outputStream, cipher, bufferSize,
        new SecretKeySpec(key, "AES"), new IvParameterSpec(iv));
cos.write(input.getBytes("UTF-8"));
cos.flush();
cos.close();
//Constructs a byte array to store random data.
byte[] key = new byte[16];
byte[] iv = new byte[16];
Properties properties = new Properties();
//Gets the 'CryptoRandom' instance.
CryptoRandom CryptoRandom = CryptoRandomFactory.getCryptoRandom(properties);
//Generates random bytes and places them into the byte array.
CryptoRandom.nextBytes(key);
CryptoRandom.nextBytes(iv);
//Closes the CryptoRandom.
CryptoRandom.close();
Commons Crypto Configurations

commons.crypto.crypto.cipher.classes
• implementation class for crypto cipher
• "org.apache.commons.crypto.cipher.JceCipher" and "org.apache.commons.crypto.cipher.OpensslCipher"
• the provider class for JCE cipher

commons.crypto.crypto.secure.random.device.file.path
• the file path for secure random device
• Linux: /dev/urandom

commons.crypto.crypto.secure.random.java.algorithm
• the algorithm of secure random
• SHA1PRNG

commons.crypto.crypto.secure.random.classes
• the implementation class for secure random
• "org.apache.commons.crypto.random.JavaCryptoRandom" and "org.apache.commons.crypto.random.OpensslCryptoRandom"

commons.crypto.stream.buffer.size
• Default value: 8192
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL’S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel, Quark, VTune, Xeon, Cilk, Atom, Look Inside and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright ©2016 Intel Corporation.
The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations.

Demand could be different from Intel’s expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel’s products; actions taken by Intel’s competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Intel’s results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel’s products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel’s results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel’s SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.