Apache Commons Crypto: Another wheel of Apache Commons

Dapeng Sun/ Xianda Ke
About us

Dapeng Sun @Intel
• Apache Commons Committer
• Apache Sentry PMC

Xianda Ke @Intel
• Apache Commons Crypto
• Apache Pig(Pig on Spark)
• OpenJDK
Agenda

• Brief introduction to cryptography
• What’s Commons Crypto?
• Why did we create another wheel?
• Performance secrets
• Securing big data & API samples
A glance at cryptographic primitives

- Symmetric encryption (Secret Key): **Confidentiality**
  - DES/3DES, RC4, AES, ...
- Asymmetric encryption (Public Key)
  - **Authentication / Key exchange**: RSA, Diffie-Hellman, ECC, ...
- Hash algorithms: **Integrity**
  - MD5, SHA-1, SHA-2, GMAC (a variant of the GCM), ...
- Authenticated Encryption (AE): **Confidentiality + Integrity**
  - RC4 + HmacMD5, RC4 + HMAC-SHA-1, AES-GCM, ...
- Random Number Generator
  - PRNG (Pseudo) / TRNG (True)
- mode of operation: blurs the cipher output
What is *Commons Crypto?*(1)

- High-performance Java cryptography library optimized with AES-NI
- It wraps OpenSSL as the engine(via JNI)
History of Commons Crypto(1)

• HDFS Crypto codec
  • Intel AES-NI
  • Native library(OpenSSL) support
  • 17x performance improvement
History of Commons Crypto(2)

• Promote the advancement to other projects
• Intel’s big data team created **Chimera** library
  • [https://github.com/intel-hadoop/Chimera](https://github.com/intel-hadoop/Chimera)
  • Easily contribute towards the advancement of AES-NI
History of Commons Crypto(3)

• Worked with Commons community contribute to Apache Community
  • https://commons.apache.org/proper/commons-crypto/
What is *Commons Crypto*? (2)

- **Low level Cipher API**
  Only support AES algorithm now

- **Stream API**
  To encrypt Stream/Channel

- **Random API**
  True Random Number Generator
JCE (Java Cryptography Extension) in hand, why did we create another wheel?

*background & motivation*
Don’t end up in a bottleneck…

Some Java cryptographic protocols (e.g. SASL) are using inefficient or weak low-level algorithms: 3DES, RC4, …

Hadoop/Spark users own very ‘big’ data…

• User’s data is vulnerable to attack
• Performance of JCE is the bottleneck!

Take security & performance into account, JCE is impractical
Choosing algorithm for data encryption

DES is broken

3DES is too slow

RC4 is ‘fast’, but weak.

prohibited by RFC 7465

AES is secure
However, the problem is that
AES’s performance in JDK is NOT good enough...
THROUGHPUT OF SYMMETRIC ALGORITHMS (MB/S)

- 3DES(JDK8): 14.59 MB/S
- AES-CTR-128(JDK8): 228.98 MB/S
- RC4(JDK8): 297.54 MB/S
- AES-CTR-128(Crypto): 1196.01 MB/S

* CPU: Xeon(R) CPU E5-2690 v2 @ 3.00GHz. Memory: DDR3-1600, 64 G.
THROUGHPUT OF AES ALGORITHMS (MB/S)

* CPU: Xeon(R) CPU E5-2690 v2 @ 3.00GHz. Memory: DDR3-1600, 64 G.
THROUGHPUT OF RANDOM NUMBER GENERATOR (MB/S)

SecureRandom(JDK8)  CryptoRandom

~13X

*Java 8. CPU: Xeon(R) CPU E5-2690 v2 @ 3.00GHz. Memory: DDR3, 64 G.
Commons Crypto is a better alternative!

SECURE

FAST
Using Commons Crypto to secure your big data, and no performance bottleneck

You might ask what makes crypto so fast?
Performance Secret (1): AES in hardware (Intel®AES-NI)

AES algorithm: each round consists of several processing steps:
MixColumn, ShiftRow, Substitute bytes
All these steps in a round can be done by a **SINGLE** instruction

```
AESENC xmm1, xmm2
```
Performance Secret (2): Pipelining (instruction-level parallelism)

AESENC instruction takes 7 CPU clock cycles. Data dependence exists, instructions have to be executed one by one.

Pipelining instructions, 6 instructions take only 12 clock cycles.
Other performance Secrets:

- Well-optimized assembly code in OpenSSL
  - Better SSE/AVX registers usage (x86 platform)
  - Data locality, etc...

- Hardware acceleration in OpenSSL
  - Intel® Carry-Less Multiplication Instruction (PCLMULQDQ for GCM)
  - Intel® Secure Key (RDRAND, Random Number Generator)
  - in other architectures

- HotSpot Intrinsics for AES are NOT well-optimized (JDK 8)
JDK 9 is becoming fast...

- **JDK-8073108** enables PCLMULQDQ for AES-GCM, ~60X gain, still falls far behind Commons Crypto

We’ve contributed two patches to OpenJDK (HotSpot):
- **JDK-8143925** x86 AES-CTR, 5~8X gain
- **JDK-8152354** x86 AES-CBC Decryption, 15%~50% gain

But we have to wait for a few years to adopt Java 9 in our software production... 😞
Big Data components are using Commons Crypto/AES-NI

- AES support for over-the-wire encryption (SPARK-13331)
- Add encrypted shuffle in spark (SPARK-5682)
  - brings 12.5% overall performance gain
- Optimize HDFS Encrypted Transport performance (HDFS-6606)
  - brings 17x overall performance gain
- Improve transparent table/CF encryption with Commons Crypto (HBASE-16463)
  - showed 4% to 42% gain for HFile encryption using HfilePerformanceEvaluation tool
- Improve performance for RPC encryption (HBASE-16414)
  - improves performance about 9x when enable delegation token

(in development):
- Optimize Hadoop RPC encryption performance (HADOOP-10768)
- Replace Hadoop Common crypto code with Apache Commons Crypto (HADOOP-13635)
API & Samples

```java
// random number generator (hardware)
byte[] keyBytes = new byte[16];
byte[] ivBytes = new byte[16];
CryptoRandom rand = CryptoRandomFactory.getCryptoRandom();
rnd.nextBytes(keyBytes);
rnd.nextBytes(ivBytes);
```

```java
// Encrypt byte array
SecretKeySpec key = new SecretKeySpec(keyBytes, "AES");
IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

Cipher cipher = CryptoCipherFactory.getCryptoCipher("AES/CTR/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
cipher.doFinal(plainText, 0, plainText.length, cipherText, 0);
cipher.close();
```
API & Samples

```java
// Decrypt ByteBuffer
ByteBuffer inBuffer = ByteBuffer.allocateDirect(1000);
ByteBuffer outBuffer = ByteBuffer.allocateDirect(1000);
inBuffer.put(cipherText);
inBuffer.flip();

cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
cipher.doFinal(inBuffer, outBuffer);
cipher.close();
```
String input = "hello world!";

Properties properties = new Properties();
final String transform = "AES/CBC/PKCS5Padding";

ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

try (CryptoOutputStream cos = 
     new CryptoOutputStream(transform, properties, outputStream, key, iv)) {
    cos.write(getUTF8Bytes(input));
    cos.flush();
}
Status & Future work:

Status:
• v1.0.0 is released. AES-CBC, AES-CTR are supported
• AES-GCM (will be released in next release)

Future work: to cover more facets:
• Support Asymmetric key algorithms: RSA, DSA
• ...

How to contribute:
• SCM: https://github.com/apache/commons-crypto
• Apache Bugtracker (JIRA): https://issues.apache.org/jira/
Thank You

Q & A