Encryption/Decryption Service

Summary

Encryption is the strongest means for security. At this time, original message is called Plan Text (Clear Text) while encrypted message is called Cipher Text (Cryptogram). Encryption (Ciphering) creates the encoded sentence by reconstructing the clear text to make the message vague. The reconstruction method of message used this time is Encryption Algorithm. Encryption algorithm uses key to improve confidentiality of encryption in. Decryption (deciphering) is the adversarial process of encryption and the process to return original message from unclear message. In general, same algorithm is used for decryption same as the one used for encryption. The system that consists of encryption and decryption that is applied with encryption technique is called Crypto System. Crypto system includes key or algorithm and is classified into Conventional Crypto System that uses a private key (Secret Key) for both encryption and decryption and the Public Key System that can use the private and public keys.

Description

JASYPT (Java simplified encryption)

Jasypt is the open source Java library and support the developer to develop encryption/decryption program without deep knowledge related to encryption. We'll explain focusing on API using as encryption and decryption module.

Encryption binaries

Jasypt provides **org.jasypt.encryption.pbe.StandardPBEByteEncryptor** implementing **org.jasypt.encryption.pbe.PBEByteEncryption** interface for binaries (byte[] objects) encryption.

Configuration

PBE (Password Based Encryption) operations,
org.jasypt.encryption.pbe.StandardPBEByteEncryptor requires password and key. How to set this is as shown below

* Use default setting value provided by package(except password)
* Set org.jasypt.encryption.pbe.config.PBEConfig
* setAlgorithm(...), setProvider(...), setProviderName(...), setPassword(...),
  Set setKeyObtentionIterations(...) or setSaltGenerator(...) methods

Initialization

Before encryption, encrptor should be initialized. Initialization is as follows:

* Call initialize() method.
* encrypt(...) or decrypt(...) method is called first.(initialized even if initialize() method is not called)
* After encryptor is initialized, setting is changed and if initialize() method is called, AlreadyInitializedException occurs.

Usage

If encrypting message, encrypt(..) method call
If decrypting message, decrypt(..) method call

Encrypting texts

Jasypt provides **org.jasypt.encryption.pbe.StandardPBEStringEncryptor implementing org.jasypt.encryption.pbe.PBEStringEncryptor** interface for texts encryption.

Basics

Jasypt uses byte (binary) encryption method to text encryption.

* All result (encryption) character string is encoded to BASE64 by default and is safely saved as US-ASCII character set.
Encoding method can be selected using setStringOutputType method.

Configuration

PBE (Password Based Encryption) operations,
org.jasypt.encryption.pbe.StandardPBEStringEncryptor is the algorithm requiring encryption and key, and can be set as follows:

* Use the default setting value provided by package(except password)
* Set org.jasypt.encryption.pbe.config.PBEConfig
* setAlgorithm(...), setProvider(...), setProviderName(...), setPassword(...), setKeyObtentionIterations(...) or setSaltGenerator(...) methods

Initialization

Before encryption, encrptor should be initialized. Initialization is as follows.

* Call initialize() method
* encrypt(...) or decrypt(...) method is called first.(initialized even if initialize() method is not called)
* After encryptor is initialized, the setting changes and if initialize() method is called again, AlreadyInitializedException occurs.

Usage

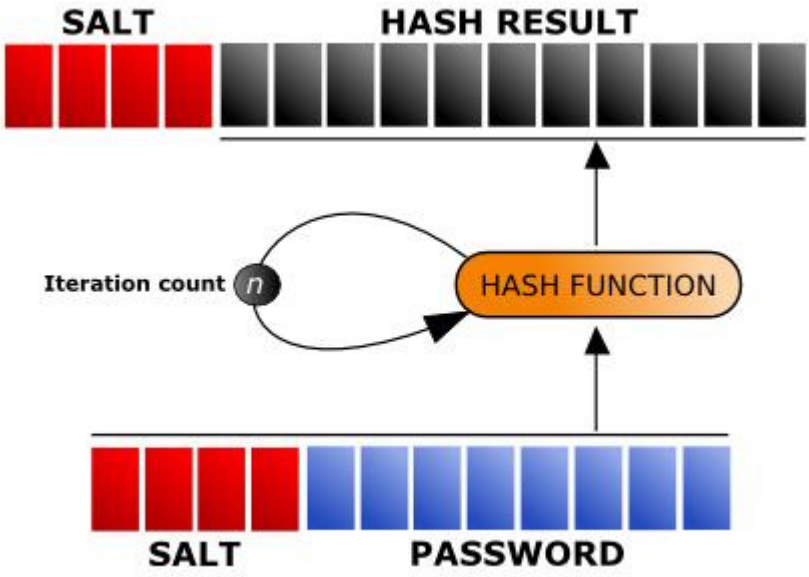
If encrypting the message, call encrypt(..) method
If decrypting the message, call decrypt(..) method

Encrypting passwords

Jasypt supports the encryption of system password or database password that can be attached from outside.

Process of Creating Digest using Jasypt's Standard[Byte|String]Digester

1. The indicated salt size is crated. (see org.jasypt.salt.SaltGenerator). If the size is 0, salt is not used. For higher security, create salt by random (like org.jasypt.salt.RandomSaltGenerator) by default.
2. salt bytes is added to the starting part of message.
3. The Hash function is applied to salt and message, and function results are repeated in the number of iterations specified.
4. If the salt is created randomly, the undigested salt is added to the starting part of hash result.



Encrypting numbers

Jasypt provides **org.jasypt.encryption.pbe.PBEBingIntegerEncryptor** for encryption of numbers, **org.jasypt.encryption.pbe.StandardPBEBigIntegerEncryptor** implementing **org.jasypt.encryption.pbe.PBEBingDecimalEncryptor** interface and **org.jasypt.encryption.pbe.StandardPBEBigDecimalEncryptor**.

Basics

Jasypt uses the byte (binary) encryption for number encryption. (same as the Text encryption method.)

Configuration

PBE (Password Based Encryption) operations, **org.jasypt.encryption.pbe.StandardPBEByteEncryptor** is the algorithm requiring password and key and can be set as follows:

* Use the default setting value provided by package(except password)
* Set org.jasypt.encryption.pbe.config.PBEConfig
* setAlgorithm(...), setProvider(...), setProviderName(...), setPassword(...), Set setKeyObtentionIterations(...) or setSaltGenerator(...) methods

Initialization

Before encryption, encrptor should be initialized. Initialization is as follows:

* Call initialize() method.
* encrypt(...) or decrypt(...) method is called first.(initialized when initialize() method is not called)
* Setting changes after encryptor is initialized. When calling initialize() method again, AlreadyInitializedException occurs.

Usage

If encrypting the message, call encrypt(..) method
If decrypting the message, call decrypt(..) method

Using Jasypt with JCE providers

Jasypt can be used if any algorithm in addition to encryption-based algorithm uses JCE provider implementing security.provider of java.

How can you use your own providers in jasypt?

```
...
Security.addProvider(new BouncyCastleProvider());
...
StandardPBEStringEncryptor mySecondEncryptor = new StandardPBEStringEncryptor();
mySecondEncryptor.setProviderName("BC"); // jce provider name
mySecondEncryptor.setAlgorithm("PBEWITHSHA256AND128BITAES-CBC-BC");
mySecondEncryptor.setPassword(myPassword);

String mySecondEncryptedText = mySecondEncryptor.encrypt(myText);
...
...
StandardStringDigester digester = new StandardStringDigester();
digester.setProvider(new BouncyCastleProvider()); // create jce provider instance
digester.setAlgorithm("WHIRLPOOL");

String digest = digester.digest(message);
...
```

ARIA Block Password Algorithm

ARIA is the general-purpose block password algorithm that has Involutional SPN structure and is optimized for lightweight environment and hardware implementation.

- Block size: 128 bit
- Key size: 128/192/256 bit (same specification as AES)
- Overall structure: Involutional Substitution-Permutation Network
- Number of round: 12/14/16 (determined depending on key size)

ARIA is developed to improve efficiency at light environment and hardware. Most operations used by ARIA consists of simple byte unit operation like XOR. The name called ARIA is the abbreviation of Academy, Research Institute, Agency, and expresses the joint cooperation of academy, research institute and agency participating at ARIA development.

Standardization Trend

ARIA was designated as Korea Standard (K) by the Ministry of Knowledge and Economy in accordance with National Standard Basic Act in last 2004.

- Standard No.: KSX1213:2004
- Division: X-Information Industry < Information Technology(IT) Application
- Standard Name: 128 bit block encryption algorithm ARIA
- History: enacted on Dec. 30, 2004
- Scope of Application: this specification specifies the block password algorithm of performing encryption and decryption of the data in the unit of 128 bite using the encryption key of variable size

Safety and Efficiency

ARIA was designed to have tolerance against all known attacks of block password. It was evaluated by the supervision authority of NESSIE (New European Schemes for Signatures,Integrity, and Encryption) for objective evaluation of safety and effectiveness after internal safety analysis by designers primarily. ARIA has excellent effectiveness in 8 bit environment and hardware implementation so that it can be applied to various environments including IC card, VPN equipment. In addition, it has the performance approaching AES and faster than Camellia at the effectiveness evaluation of Leuven University, Belgium at software implementation.

ARIA Efficiency Comparison (Unit: cycle/byte)

CPU	ARIA	AES	Camellia	SEED

Pentium III	**37.3**	**23.3**	**33.4**	**42.4**
Pentium IV	**49.0**	**30.5**	**83.9**	**81.3**

This comparison table of effectiveness was created based on the ARIA analysis report of Leuven university and effectiveness analysis report of NESSIE. ARIA calculated the number of cycle to 120% of evaluation data since the number of round increased from 10 to 12 compared to ver. 8.0, the target of evaluation of Leuven University in case of 128 bit key length. Since two data were not performed at accurately same environment while these two reports were performed by the same institution(COSIC group of Leuven), the performance ratio of both platform should be calculated from the data on Camellia and AES of 2 reports to estimate the speed of SEED.

Guide Program

Configuration

```
<!--bean id="cryptotool" class="egovframework.rte.fdl.crypto.EgovCryptoset">
            <property name="cryptoBin" ref="cryptolerBin" />
            <property name="cryptoBin1" ref="cryptolerBin1" />
            <property name="cryptoTxt" ref="cryptolerTxt" />
            <property name="cryptoTxt1" ref="cryptolerTxt1" />
            <property name="cryptoPwd" ref="cryptolerPwd" />
            <property name="cryptoNum" ref="cryptolerNum" />
            <property name="cryptoNum1" ref="cryptolerNum1" />
           <property name="cryptoAriaTxt" ref="cryptolerAriaTxt" />
           <property name="cryptoAriaTxt1" ref="cryptolerAriaTxt1" />
           <property name="cryptoAriaBin" ref="cryptolerAriaBin" />
           <property name="cryptoAriaBin1" ref="cryptolerAriaBin1" />
           <property name="cryptoAriaNum" ref="cryptolerAriaNum" />
           <property name="cryptoAriaNum1" ref="cryptolerAriaNum1" />
</bean-->
<bean id="cryptolerBin" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionBinServlceImpl" />
<bean id="cryptolerBin1" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionBinServlceImpl" />
<bean id="cryptolerTxt" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionTxtServiceImpl" />
<bean id="cryptolerTxt1" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionTxtServiceImpl" />
<bean id="cryptolerPwd"
class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionPasswdServlceImpl" />
<bean id="cryptolerNum" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionNumServiceImpl"
/>
<bean id="cryptolerNum1" class="egovframework.rte.fdl.crypto.impl.EgovEDcryptionNumServiceImpl"
/>
<bean id="cryptolerAriaTxt"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionTxtServiceImpl" />
<bean id="cryptolerAriaTxt1"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionTxtServiceImpl" />
<bean id="cryptolerAriaBin"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionBinServiceImpl" />
<bean id="cryptolerAriaBin1"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionBinServiceImpl" />
<bean id="cryptolerAriaNum"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionNumServiceImpl" />
<bean id="cryptolerAriaNum1"
class="egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionNumServiceImpl" />
```

PARAMETER	Description
cryptolerBin	egovframework.rte.fdl.crypto.impl.EgovEDcryptionBinServlceImpl(Binary Encryption Implement)
cryptolerBin1	egovframework.rte.fdl.crypto.impl.EgovEDcryptionBinServlceImpl(Binary Encryption Implement)
cryptolerTxt	egovframework.rte.fdl.crypto.impl.EgovEDcryptionTxtServiceImpl(Text Encryption Implement)
cryptolerTxt1	egovframework.rte.fdl.crypto.impl.EgovEDcryptionTxtServiceImpl(Text Encryption

	Implement)
cryptolerPwd	egovframework.rte.fdl.crypto.impl.EgovEDcryptionPasswdServlceImpl(Password Encryption Implement)
cryptolerNum	egovframework.rte.fdl.crypto.impl.EgovEDcryptionNumServiceImpl(Number Encryption Implement)
cryptolerNum1	egovframework.rte.fdl.crypto.impl.EgovEDcryptionNumServiceImpl(Number Encryption Implement)
cryptolerAriaBin	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionBinServiceImpl(ARIA Binary Encryption Implement)
cryptolerAriaBin1	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionBinServiceImpl(ARIA Binary Encryption Implement)
cryptolerAriaTxt	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionTxtServiceImpl(ARIA Text Encryption Implement)
cryptolerAriaTxt1	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionTxtServiceImpl(ARIA Text Encryption Implement)
cryptolerAriaNum	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionNumServiceImpl(ARIA Number Encryption Implement)
cryptolerAriaNum1	egovframework.rte.fdl.crypto.impl.EgovARIAEDcryptionNumServiceImpl(ARIA Number Encryption Implement)

* Set Binary,Text,Number,Password Test Module

```
crypto.password=N5h+Lz1rd/24YvL1KhV1pO0OSIc=
cyrpto.algorithm=PBEWithSHA1AndDESede
crypto_password.algorithm=
```

* Encryption private key, algorithm, password encryption algorithm setting property

```
<context:property-placeholder location="classpath*:spring/crypto_config.properties" />
<bean id="config" class="egovframework.rte.fdl.crypto.CryptoConfig">
        <property name="password" value="${crypto.password}" />
        <property name="algorithm" value="${cyrpto.algorithm}" />
        <property name="passwordAlgorithm" value="${crypto_password.algorithm}" />
</bean>
```

PARAMETER	Description
password	Encryption private key
algorithm	Encryption algorithm
passwordAlgorithm	Password encryption algorithm

Sample Source

```
@Resource(name = "cryptolerBin")
private EgovEDcryptionService cryptoBin;
@Resource(name = "cryptolerBin1")
private EgovEDcryptionService cryptoBin1;
@Resource(name = "cryptolerTxt")
private EgovEDcryptionService cryptoTxt;
@Resource(name = "cryptolerTxt1")
private EgovEDcryptionService cryptoTxt1;
@Resource(name = "cryptolerPwd")
private EgovEDcryptionService cryptoPwd;
@Resource(name = "cryptolerNum")
private EgovEDcryptionService cryptoNum;
@Resource(name = "cryptolerNum1")
private EgovEDcryptionService cryptoNum1;
/** ARIA Text Encryption Class */
@Resource(name = "cryptolerAriaTxt")
private EgovEDcryptionService cryptoAriaTxt;
```

```
/** ARIA Text Encryption Class */
@Resource(name = "cryptolerAriaTxt1")
private EgovEDcryptionService cryptoAriaTxt1;
/** ARIA Bin Encryption Class */
@Resource(name = "cryptolerAriaBin")
private EgovEDcryptionService cryptoAriaBin;
/** ARIA Bin Encryption Class */
@Resource(name = "cryptolerAriaBin1")
private EgovEDcryptionService cryptoAriaBin1;
/** ARIA Num Encryption Class */
@Resource(name = "cryptolerAriaNum")
private EgovEDcryptionService cryptoAriaNum;
/** ARIA Num Encryption Class */
@Resource(name = "cryptolerAriaNum1")
private EgovEDcryptionService cryptoAriaNum1;
```

* Encryption Test Module Annotation

```
@Test
public void testCryptoTxt()throws UnsupportedException
{
          cryptoTxt.setComformStr("Egov");
          cryptoTxt.setConfig(-1, "Text Encryption Test");

          byte ret_encryt[] = cryptoTxt.encrypt();
          String str = new String(ret_encryt);
         assertNull(str);

          cryptoTxt1.getComformStr("Egov");
          cryptoTxt1.setConfig(-1, new String(ret_encryt));
          byte ret_decrypt[] = cryptoTxt1.decrypt();
          String str1 = new String(ret_decrypt);
         assertNull(str1);

}
```

* Text Encryption Test Module

```
@Test
public void testCryptoBin()throws UnsupportedException, IOException
{
          cryptoBin.setComformStr("Egov");
          cryptoBin.setConfig(0, "C:\\temp\\Update Control Activity(for person in charge).xls");
          cryptoBin.encrypt();


          cryptoBin1.getComformStr("Egov");
          cryptoBin1.setConfig(0, "C:\\temp\\Update Control Activity(for person in
charge)_egov_encrypt_.xls");
          cryptoBin1.decrypt();
}
```

* Binary Encryption/Decryption Test Module

```
@Test
public void testCryptoPwd()throws UnsupportedException
{
          cryptoPwd.setPlainDigest(true);
          cryptoPwd.setConfig(-1, "Password Test");
          cryptoPwd.setAlgorithm("");

          byte ret_encryt[] = cryptoPwd.encrypt();
```

```
        String str = new String(ret_encryt);
        assertNull(str);
        logger.debug("After Encryption: " + str);

        cryptoService1 = cryptoPwd;
        if(cryptoService1.checkPassword("Password Test", ret_encryt))
                logger.debug("Password matches.");
        else
                logger.debug("Password is not matched.");
}
```

* Password Encryption Test Module

```
@Test
public void testCryptoNum()throws UnsupportedException
{
        BigDecimal tmp_ = null;
        BigDecimal bigdecimal = new BigDecimal(10000);

       cryptoNum.setComformStr("Egov");
        tmp_ = cryptoNum.encrypt(bigdecimal);
        assertNull(tmp_.toString());
        logger.debug("After Encryption:"+tmp_.toString());

        cryptoNum1.getComformStr("Egov");
        tmp_ = cryptoNum1.decrypt(tmp_);
        assertNull(tmp_.toString());
        logger.debug("After Decryption:"+tmp_.toString());
}
```

* Number Encryption/Decryption Test Module

```
@Test
public void testAriaCryptoTxt()throws UnsupportedException
{
            // TODO Auto-generated method stub
            cryptoAriaTxt.setComformStr("Egov");
            cryptoAriaTxt.setConfig(-1, "701122-1167411");

            byte ret_encryt[] = cryptoAriaTxt.encrypt();
            String str = new String(ret_encryt);
            logger.debug("After Encryption : " + str);

            cryptoAriaTxt1.getComformStr("Egov");
            cryptoAriaTxt1.setARIAConfig(-1, ret_encryt);
            byte ret_decrypt[] = cryptoAriaTxt1.decrypt();
            String str1 = new String(ret_decrypt);
            logger.debug("After Decryption: " + str1);
}
```

* ARIA Text Encryption/Decryption Test Module

```
@Test
public void testCryptoAriaBin()throws UnsupportedException, IOException
{
        // TODO Auto-generated method stub
        cryptoAriaBin.setComformStr("Egov");
        cryptoAriaBin.setConfig(0, "C:\\temp\\Korean.hwp");
        cryptoAriaBin.encrypt();


        cryptoAriaBin1.getComformStr("Egov");
```

```
        cryptoAriaBin1.setConfig(0, "C:\\temp\\Korean_egov_encrypt_.hwp");
        cryptoAriaBin1.decrypt();
}
```

* ARIA Binary Encryption/Decryption Test Module

```
@Test
public void testCryptoAriaNum()throws UnsupportedException, IOException
{
        // TODO Auto-generated method stub
        byte[] tmp_ = null;
        BigDecimal tmp_bigdec = null;
        BigDecimal bigdecimal = new BigDecimal(-365290002);

        cryptoAriaNum.setComformStr("Egov");
        tmp_ = cryptoAriaNum.Aria_encrypt(bigdecimal);

        cryptoAriaNum1.getComformStr("Egov");
        tmp_bigdec = cryptoAriaNum1.Aria_decrypt(tmp_);
}
```

* ARIA Number Encryption/Decryption Test Module

Reference

ARIA Block Password Algorithm