Цезарис-CSP-JM.
Руководство программиста Java.
Алгоритмы защиты ключей и
защищенный контейнер ключа

В этом руководстве предоставлены примеры применения крипто-провайдера Java «Цезарис-CSP-JM» для операций, определенных технической Спецификацией «Вимоги до алгоритмів формування ключів шифрування ключів та захисту особистих ключів електронного цифрового підпису та особистих ключів шифрування» (Наказ Міністерства юстиції України, Адміністрації Державної служби спеціального зв'язку та захисту інформації України 27.12.2013 № 2782/5/689), «Вимоги до форматів транспортних контейнерів особистих ключів електронного цифрового підпису та особистих ключів шифрування» (Наказ Міністерства юстиції України, Адміністрації Державної служби спеціального зв'язку та захисту інформації України 27.12.2013 № 2782/5/689) и т.д. этого приказа.

Крипто-провайдер соответствует требованиям Java JCE/JCA, и должен подключаться статически (через конфигурацию в файле java.security), или динамически:

```
Provider provider = new AMBProvider();
Security.addProvider(provider);
```

Путь к библиотекам провайдера, а именно: ambprovider.jar, ambasn.jar – должен быть прописан в системе, или эти библиотеки должны быть в папке Java (…\jre\lib\ext\).

1. PBKDF2-функция

PBKDF2-функция предназначена для формирования симметричного ключа (шифрования контекста) на основе пароля, случайных данных и числа итераций (см. стандарт PKCS#5 и Спецификацию).

```
import com.amb.security.crypto.kdf.PBKDF2;
import com.amb.security.crypto.kdf.PBKDF2ParameterSpec;
import com.amb.security.provider.AMBProvider;
…
String passw = "password"; //пароль
int count = 2048; //число итераций
String sSalt = "salt"; //случайные данные
int dkLen = 32; //длина ключа в байтах
String prfAlg = "HMAC_GOST34311"; // PRF-функция = Keyed-Hash Message
                                  //Authentication Code, HMAC
```

```
//параметры PBKDF2-функции
PBKDF2ParameterSpec paramSpec =
        new PBKDF2ParameterSpec(salt.getBytes(), count, dkLen, prfAlg);

// PBKDF2-функция
PBKDF2 pbkdf2 = new PBKDF2();
pbkdf2.init(passw.getBytes("UTF-8"), paramSpec);
//генерировать ключ
byte[] key = pbkdf2.deriveKey(dkLen);
```

Этом пример применим и для таких PRF-функций (в prfAlg):
"HmacSHA1" (RFC 3370, PKCS#5);
"HmacSHA224" (PKCS #5);
"HmacSHA256" (PKCS #5);
"HmacSHA384" (PKCS #5);
"HmacSHA512" (PKCS #5);
"HMAC_GOSTR3411" (RFC 4357).

2. PBES2-функция и формат защищенного транспортного контейнера ключа

PBES2-функция предназначена для шифрования данных на основе парольной информации с использованием PBKDF-функции и заданного алгоритма шифрования (см. стандарт PKCS#5 и Спецификацию).

Пример 1. Применение ДКЕ №1 (по умолчанию)

```
import com.amb.security.crypto.dstu4145.DSTU4145PrivateKey;
import com.amb.security.crypto.kdf.PBKDF2ParameterSpec;
import com.amb.security.crypto.pkcs.pkcs5.PBEKey;
import com.amb.security.crypto.pkcs.pkcs5.PBES2ParameterSpec;
import com.amb.security.crypto.pkcs.pkcs8.EncryptedPrivateKeyInfo;
import com.amb.security.provider.AMBProvider;

import java.security.AlgorithmParameters;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Provider;
import java.security.Security;

import javax.crypto.Cipher;
…
String passw = "password"; //пароль
```

```
int count = 2048; //число итераций
String sSalt = "salt"; //случайные данные
int dkLen = 32; //длина ключа в байтах
String prf = "HMAC_GOST34311"; // PRF-функция = Keyed-Hash Message
                                        //Authentication Code, HMAC

//случайный вектор инициализации
byte[] iv = new byte[8];
SecureRandom random = SecureRandom.getInstance("DSTU4145", "AMB");
random.nextBytes(iv);
//"DSTU4145" – соотвествует стандарту ДСТУ 4145-2002
// может применяться "Hash_DRBG", соответствует NIST 800-90:2007
// или любой другой Java генератор случайных чисел

//Закрытый ключ можно получить для примера так
int keyLenth = 191;// полиномиальный базис m=191
KeyPairGenerator keyPairGenerator =
        KeyPairGenerator.getInstance("DSTU4145", "AMB");
keyPairGenerator.initialize(keyLength);
KeyPair keyPair = keyPairGenerator.genKeyPair();
DSTU4145PrivateKey privKey = (DSTU4145PrivateKey)keyPair.getPrivate();

//параметры PBKDF2-функции
PBKDF2ParameterSpec pbkdfSpec = new
        PBKDF2ParameterSpec(salt.getBytes(), count, dkLen, prf);
//параметры PBES2-функции
PBES2ParameterSpec pbeSpec = new PBES2ParameterSpec(pbkdfSpec, iv,
"GOST28147CFB");
AlgorithmParameters algParams = AlgorithmParameters.getInstance("PBES2",
"AMB");
algParams.init(pbeSpec);

PBEKey skey = new PBEKey(password.toCharArray());

//функция шифрования
Cipher cipher = Cipher.getInstance("PBES2", "AMB");
cipher.init(Cipher.ENCRYPT_MODE, skey, pbeSpec);

// cleartext - DSTU4145PrivateKey
// Encrypt the cleartext (см. Спецификация 2)
byte[] encryptedKey = cipher.doFinal(privKey.getEncoded());

//создать структуру EncryptedPrivateKeyInfo (см. Спецификация 2)
```

EncryptedPrivateKeyInfo epkinfo = new EncryptedPrivateKeyInfo(algParams, encryptedKey);
byte[] out = epkinfo.getEncoded();

В этом примере также применимы PRF-функции, указанные в предыдущем примере. Кроме того могут применяться алгоритмы шифрования rc2-cbc, des-ede3-cbc, aes128-ecb, aes128-cbc, aes128-cfb, aes128-ofb, aes192-ecb, aes192-cbc, aes192-cfb, aes192-ofb, aes256-ecb, aes256-cbc, aes256-cfb, aes256-ofb, des-ecb, des-ofb, des-cfb, GOST28147-OFB.

Пример 2. Применение заданного ДКЕ

```
import com.amb.security.crypto.dstu4145.DSTU4145PrivateKey;
import com.amb.security.crypto.kdf.PBKDF2;
import com.amb.security.crypto.kdf.PBKDF2ParameterSpec;
import com.amb.security.crypto.pkcs.pkcs5.PBEKey;
import com.amb.security.crypto.pkcs.pkcs5.PBES2ParameterSpec;
import com.amb.security.crypto.pkcs.pkcs8.EncryptedPrivateKeyInfo;
import com.amb.security.provider.AMBProvider;

import java.security.AlgorithmParameters;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.Provider;
import java.security.Security;

import javax.crypto.Cipher;
…
String passw = "password"; //пароль
int count = 2048; //число итераций
String sSalt = "salt"; //случайные данные
int dkLen = 32; //длина ключа в байтах
String prf = "HMAC_GOST34311"; // PRF-функция = Keyed-Hash Message
                                //Authentication Code, HMAC

//случайный вектор инициализации
byte[] iv = new byte[8];
SecureRandom random = SecureRandom.getInstance("DSTU4145", "AMB");
random.nextBytes(iv);
//"DSTU4145" – соотвествует стандарту ДСТУ 4145-2002
// может применяться "Hash_DRBG", соответствует NIST 800-90:2007
// или любой другой Java генератор случайных чисел

//Закрытый ключ можно получить для примера так
```

```
int keyLenth = 191;// полиномиальный базис m=191
KeyPairGenerator keyPairGenerator =
        KeyPairGenerator.getInstance("DSTU4145", "AMB");
keyPairGenerator.initialize(keyLength);
KeyPair keyPair = keyPairGenerator.genKeyPair();
DSTU4145PrivateKey privKey = (DSTU4145PrivateKey)keyPair.getPrivate();

//параметры PBKDF2-функции
PBKDF2ParameterSpec pbkdfSpec = new
        PBKDF2ParameterSpec(salt.getBytes(), count, dkLen, prf);

//параметры PBES2-функции
PBES2ParameterSpec pbeSpec = new PBES2ParameterSpec(pbkdfSpec, iv,
"GOST28147CFB");

//параметры алгоритма (функции) PBKDF2
AlgorithmParameters pbkdfParam =
    AlgorithmParameters.getInstance("PBKDF2", "AMB");
pbkdfParam.init(pbkdfSpec);

//идентификатор алгоритма (функции) PBKDF2/"1.2.840.113549.1.5.12"
AlgorithmIdentifier keyDerivationFunction =
    new AlgorithmIdentifier(PBKDF2.OID, pbkdfParam);

//параметры алгоритма шифрования с ДКЕ №2
GOST28147ParameterSpec gostSpec  =
    new GOST28147ParameterSpec("SBOX-2", iv);
AlgorithmParameters gostParam =
    AlgorithmParameters.getInstance("GOST28147CFB", "AMB");
gostParam.init(gostSpec);

//идентификатор алгоритма шифрования
AlgorithmIdentifier encryptionScheme =
    new AlgorithmIdentifier("GOST28147CFB", gostParam);

//параметры функции PBES2
PBES2ParameterSpec pbeSpec =
    new PBES2ParameterSpec(keyDerivationFunction, encryptionScheme);

AlgorithmParameters algParams = AlgorithmParameters.getInstance("PBES2",
"AMB");
algParams.init(pbeSpec);

PBEKey skey = new PBEKey(password.toCharArray());
```

```
//функция шифрования
Cipher cipher = Cipher.getInstance("PBES2", "AMB");
cipher.init(Cipher.ENCRYPT_MODE, skey, pbeSpec);

// cleartext - DSTU4145PrivateKey
// Encrypt the cleartext (см. Спецификация 2)
byte[] encryptedKey = cipher.doFinal(privKey.getEncoded());

//создать структуру EncryptedPrivateKeyInfo (защищенный транспортный контейнер, см. Спецификацию 2)
EncryptedPrivateKeyInfo epkinfo = new EncryptedPrivateKeyInfo(algParams, encryptedKey);
byte[] out = epkinfo.getEncoded();
```

3. PBMAC-функція

PBMAC-функция предназначена для контроля целостности данных на основе парольной информации (см. Спецификацию).

```
import java.security.Key;
import java.security.spec.AlgorithmParameterSpec;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import com.amb.security.crypto.params.ParametersWithSBox;
…
//генерировать ключ (как в примере пункта 1)
…
byte[] keyValue = pbkdf2.deriveKey(dkLen);

// ДКЕ
byte[] sBox = GOST28147Cipher.getSBox("SBOX-1");

//задать параметры – только ДКЕ, без вектора инициализации, ДКЕ задан
AlgorithmParameterSpec params = new ParametersWithSBox(null, sBox);

Key key = new SecretKeySpec(keyValue, "GOST28147");
Mac mac = Mac.getInstance("GOST28147MAC", "AMB");
mac.init(key, params);
mac.update(plainText);
byte[] macValue = mac.doFinal();
```

Примечание. Параметры алгоритма (AlgorithmParameterSpec) не требуются также для GOST28147MAC, если используется ДКЕ №1 по умолчанию ("SBOX-1"), тогда имеем:

```
import java.security.Key;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

…
byte[] keyValue = pbkdf2.deriveKey(dkLen);

Key key = new SecretKeySpec(keyValue, "GOST28147");
Mac mac = Mac.getInstance("GOST28147MAC", "AMB");
mac.init(key);
mac.update(plainText);
byte[] macValue = mac.doFinal();
```

4. Открыть контейнер ключей

Контейнер хранения личных ключей цифровой подписи/шифрования и сертификатов ключей (см. стандарт PKCS#5 и Спецификацию).

4.1. Международный контейнер (файлы типа *.p12, *.pfx)

Для международных контейнеров применяются такие алгоритмы защиты: HmacSHA1, PBEWithSHA1AndRC2_40 для шифрования сертификатов, PBEWithSHA1AndDESede для шифрования ключей.

```
import java.security.Key;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.cert.X509Certificate;
…

//путь, имя файла *.p12, *.pfx
String pfx = "…";
KeyStore keyStore = KeyStore.getInstance("PKCS12", "AMB");
InputStream istream = new FileInputStream(pfx);
//загрузить
keyStore.load(istream, passw.toCharArray());

//получить алиасы объектов (ключ + сертификат)
Enumeration<String> aliases = keyStore.aliases();

String alias = null;
```

```
while (aliases.hasMoreElements()) {
        alias = aliases.nextElement();
      System.out.println("alias: " + alias);
      //сертификат ключа
      X509Certificate cert = (X509Certificate)keyStore.getCertificate(alias);
      System.out.println("Certificate Subject: " +
                  cert.getSubjectDN().getName());
      System.out.println("Certificate Issuer : " +
                  cert.getIssuerDN().getName());
      System.out.println("Serial number      : " + cert.getSerialNumber());
      System.out.println(cert.getPublicKey().toString());

      //цепочка сертификатов
      Certificate[] certChain = keyStore.getCertificateChain(alias);
      if (certChain.length > 1) {
        System.out.println("\nCertificate chain...");

        X509Certificate cer;
        for (int k = 1; k < certChain.length; k++) {
          cer = (X509Certificate)certChain[k];
          System.out.println("Certificate Subject: " +
                      cer.getSubjectDN().getName());

          System.out.println(cer.getPublicKey().toString());
        }
        //закрытый ключ
        Key key = keyStore.getKey(alias, passw.toCharArray());
        System.out.println(key.toString());
}
```

4.2. Национальный контейнер

Для национальных контейнеров применяются такие алгоритмы защиты: HMAC_GOST34311, ГОСТ 28147:2009 для шифрования сертификатов и ключей.

Пример аналогичный предыдущему, за исключением имени контейнера

```
KeyStore keyStore = KeyStore.getInstance("PKCS12UA", "AMB");
```

5. Создать контейнер ключей

Контейнер хранения личных ключей цифровой подписи/шифрования и сертификатов ключей (стандарт PKCS#12).

5.1. Международный контейнер (файлы типа *.p12, *.pfx)

Можно создать для любого типа ключа. Пример:

```
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.Certificate;

import java.io.FileOutputStream;
import java.util.UUID;
….
//получить закрытый ключ
PrivateKey privKey = …;

//получить цепочку сертификатов: с индексом 0 – юзера, 1 – ЦСК и т.д до корневого
Certificate[] certChain = …;

//создать хранилище
KeyStore keyStore = KeyStore.getInstance("PKCS12", "AMB");
keyStore.load(null, null);

//задать алиас объектов хранилища
String alias = UUID.randomUUID().toString();
keyStore.setKeyEntry(alias, privKey, password.toCharArray(), certChain);

//сохранить в файл
String path = "…";
FileOutputStream fOut = new FileOutputStream(path + "pkcs12store.pfx");
keyStore.store(fOut, password.toCharArray());
```

5.2. Национальный контейнер

Можно создать для любого типа ключа. Пример аналогичный предыдущему, за исключением имени контейнера:

```
KeyStore keyStore = KeyStore.getInstance("PKCS12UA", "AMB");
…
FileOutputStream fOut = new FileOutputStream(path + "pkcs12store.kua");
```

Вопросы просьба направлять по адресу
tech@itsway.kiev.ua.