Integrovaný informační systém Státní pokladny (IISSP)

Utilita pro účetní jednotky

Dokumentace API - integrační dokumentace

Obsah

1. Úvod

Následující dokument prezentuje popis API šifrovací utility, která umožňuje realizaci procesů:

- dekódování osobních přístupových kódů,
- přípravu dat k odeslání
- zpracování přijatých dat.

V dokumentu je popsán technický popis funkčních procesů v jazyce Java. Pro generování a verifikaci XML elektronického podpisu je využito knihovny Apache XML Security ve verzi 1.4.3. Pro transformaci XML zpráv a jako SAX Parser je využito Apache Xalan (ve verzi 2.7.1) a Apache Xerces-J (ve verzi 2.9.0). Vzhledem k tomu, že je použito silné kryptografie (AES s klíčem délky 256 bitů), je nutné použít Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files (ke stažení na stránkách výrobce JRE). Šifrovací utilita je distribuována formou Java Webstart aplikace, pro tvorbu grafického rozhraní bylo použito frameworku JGoodies (forms a looks).

Součástí dokumentu je také popis východisek pro tvorbu osobních přístupových kódů, přípravu a zpracování dat (dle vyhlášky č. 383/2009 Sb.), který je součástí kapitoly 2.

2. Východiska a postupy

2.1 Způsob dešifrování a ověření sady přístupových údajů

V následujícím popisu jsou používány tyto operace:

- || - zřetězení dvou polí znaků
- **C_DecryptECB** – provedení dešifrování algoritmem AES v ECB módu
- **Base64** – kódování binárních dat v textové formě
- **HexString** – hexadecimální řetězec
- **Decode_Base64** – převede data z kódování Base64 do binárního tvaru
- **StrBytes** – textový řetězec v kódování ASCII převede do binárního tvaru
- **DigestSHA256** – kontrolní součet vytvořený pomocí algoritmu SHA-256
- **padding** – doplněk zprávy, tak aby délka zprávy byla celistvým násobkem 512 bitů
- **removePadding** – odstraní zarovnání na šířku šifrovacího bloku
- **decode** – osobní dekódovací kód v binárním tvaru

Algoritmy a výstupy (všechny výstupy mají formu ASCII řetězce):

1. Dešifrování přístupových údajů
   - `[out] passComm = removePadding(C_DecryptECB(Decode_Base64(paccCommEnc), decode))`
   - `[out] aesKey = C_DecryptECB(Decode_Base64(aesKeyEnc), decode)`
   - `[out] loginComm = removePadding(C_DecryptECB(Decode_Base64(loginCommEnc), decode))`
2. Ověření integrity sady přístupových údajů
   - `[out] confirmHash = HexString(DigestSHA256(StrBytes (ujeid) || StrBytes(datnar) || StrBytes(jmezo) || StrBytes(prizo) || Base64(aesKey) || StrBytes (loginCommOpen) || StrBytes (passCommOpen) || padding()))`

2.2 Způsob přípravy zprávy k odeslání

Postup přípravy zprávy je následující:

1. ÚJ nebo CSUIS připraví datový obsah zprávy ve formě XML dat. Tato data obsahují logické části: hlavička, tělo, patička realizované konkrétními elementy (viz XSD schéma pro přenos dat). V takto připravených datech nejsou vloženy žádné bezpečnostní atributy související s přenosem dat.
2. ÚJ nebo CSUIS vytvoří identifikátor celistvosti, který bude vložen do patičky XML dokumentu vytvořeného v bodě 1. Tvorba identifikátoru celistvosti je popsána v kapitole 2.2.1
3. ÚJ nebo CSUIS provedou zašifrování dat pro přenos postupem uvedeným v kapitole 2.2.2

2.2.1 Tvorba identifikátoru celistvosti

Vzhledem k tomu, že identifikátor celistvosti má prokazovat, že data nebyla přenosem změněna či poškozena, je potřeba přesně definovat, z jakých dat a jakým způsobem se identifikátor vypočte. Vzhledem k povaze dat (XML) bude identifikátor celistvosti vytvořen pomocí postupů definovaných ve standardu [XMLSig11]. Identifikátor celistvosti bude vytvořen ve formě elementu Signature s využitím

mechanizmu HMAC-SHA256. Pro výpočet HMAC nebude použit utajený klíč a identifikátor celistvosti tak nebude sloužit k autentizaci. Jako hodnota klíče pro HMAC bude použito 32 nulových bajtů.

Vložená XML signatura může vypadat například takto (použité transformace a algoritmy jsou závazné):

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="identifikator-celistvosti">
<ds:SignedInfo>
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#WithComments" />
  <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#hmac-sha256">
     <ds:HMACOutputLength>256</ds:HMACOutputLength>
     </ds:SignatureMethod>
   <ds:Reference URI="">
   <ds:Transforms>
     <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#WithComments" />
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
     <ds:DigestValue>iq9zxO+eiTXm5MYHiCi3MCLWPbTL6SvD+3zvakh3FaM=</ds:DigestValue>
    </ds:Reference>
   </ds:SignedInfo>
  <ds:SignatureValue>St8d0mDDnWvLgT7jBG0xdM4gP2rR0TAi5TnH/dJ0GHo=</ds:SignatureValue>
  <ds:KeyInfo>
     <ds:KeyName>KVS HMAC</ds:KeyName>
  </ds:KeyInfo>
</ds:Signature>
```

Pro tvorbu XML podpisu jsou závazné následující požadavky:

Element signatury je identifikován pomoci atributu **Id="identifikator-celistvosti"**
Element signatury je vložen jako poslední element do elementu <msg:EnvelopeFooter>
Pro zpracování se uvažuje s postupem, kdy je kořenovým elementem <msg:Envelope> a signatura je vypočtena před jakýmkoli zanořením elementu <msg:Envelope> do dalších XML struktur (např. SOAP)
Metoda kanonikalizace v rámci signatury: **`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`**
Algoritmus podpisu: **`http://www.w3.org/2001/04/xmldsig-more#hmac-sha256`**
Reference

- **`URI=""`** – celý aktuální XML dokument
- Transformace 1 - **`http://www.w3.org/2000/09/xmldsig#enveloped-signature`**
- Transformace 2 - **`http://www.w3.org/2001/10/xml-exc-c14n#WithComments`**
- DigestMethod – **`http://www.w3.org/2001/04/xmlenc#sha256`**

2.2.2 Šifrování dat pro přenos

Pro šifrování se XML dokument serializuje a vytvoří se jeho binární reprezentace. Tato binární reprezentace se šifruje dle postupu popsaného v příloze č. 6 vyhlášky.

Operace použité v popisu postupu:

- *serializeXML* – operace, která z abstraktních XML dat produkuje reálná binární data (nutno volit kódování, typ oddělení řádků, pořadí atributů apod.)
- *{}* – konstantní data vyjádřená po bajtech hexadecimální hodnotou
- *length()* – operace, jejímž výsledkem je délka binárních dat, která jsou parametrem
- *%* - zbytek po dělení (mod)
- *AES_CBC(iv)* – mechanizmus šifrování AES v modu CBC s definovanou hodnotou iniciálního vektoru IV
- *Encode_Base64()* – převede binární data do kódování Base64
- *C_Encrypt (mechanismus, klíč, data)* – provede zašifrování definovaným mechanismem a využití daného klíče

Postup je následující:

```
1. xmlData=serializeXML()
2. toEncryptnNoPad = RND(16) || xmlData || {0x0D,0x0A}
3. toEncrypt = toEncryptNoPad || RND( 16 - lenght(toEncryptNoPad)%16}
4. aesIV={0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00}
5. encrypted = encode_Base64 (C_Encrypt(AES_CBC(aesIV), aesKey, toEncrypt))
```

Výsledkem procesu šifrování jsou binární data zakódovaná v base64.

2.3 Způsob zpracování přijaté zprávy

2.3.1 Dešifrování zprávy

Zpráva je přijata ve formě Base64 kódovaných binárních dat. Data reprezentují zašifrovanou XML zprávu. Dešifrování probíhá podle následujícího postupu, kde jsou využity následující funkce:

- Decode_Base64() – převede data z kódování Base64 do binárního tvaru
- *AES_CBC(iv)* – mechanizmus šifrování AES v modu CBC s definovanou hodnotou iniciálního vektoru IV
- *C_Dencrypt (mechanismus, klíč, data)* – provede dešifrování definovaným mechanismem a využití daného klíče
- *{}* – konstantní data vyjádřená po bajtech hexadecimální hodnotou
- *stripIV()* – provede odříznutí prvních 16 bajtů dat
- *stripPadding()* – provede odříznutí bajtů doplněných na konec XML za poslední odřádkování (0x0D,0x0A)

Postup je následující

```
1. encryptedData=decode_Base(encrypted_encoded)
2. aesIV={0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00}
3. plainWithPaddingAndIV=C_Decrypt(AES_CBC(aesIV), aesKey, encryptedData)
4. plainWithPadding= stripIV(plainWithPaddingAndIV)
5. plain=stripPadding(plainWithPadding)
```

2.3.2 Ověření identifikátoru celistvosti

Data získaná dešifrováním jsou parsována jako stream obsahující XML. Na získaných XML datech je provedeno ověření XML-Dsig podpisu ve formě HMAC. Ověření probíhá takto:

- binární dešifrovaná data jsou parsována jako XML
- je vyhledán element <ds:Signature> s Id="identifikator-celistvosti"
- je proveden výpočet ověření HMAC podpisu nad obsahem vybraného elementu <ds:Signature>

3. Technický popis funkčních procesů

3.1 Dekódování osobních přístupových údajů

Vstup:

- Adresář s rozbalenými daty
  - AESKEY_C.TXT
  - LOGIN_C.TXT
  - PASSWD_C.TXT
  - DTRANS_PWD.TXT
  - PAC_PWD.TXT
  - UJEID.TXT
  - DATNAR.TXT
  - JMEZO.TXT
  - PRIZO.TXT
- Osobní dekódovací kód
  - 64-místní číslo
- Heslo k novému ZIP archivu (bude obsahovat dešifrované data)
  - Řetězec splňující podmínky pro heslo

Postup (viz zdrojový kód třída com.logica.statetreasury.kvs.clienttool.util.CryptoUtils metoda decodeIdData()):

Validace vstupních dat

- zkontrolujeme existenci potřebních souborů
- zvalidujeme nově zadané heslo k ZIP archivu (viz třída com.logica.statetreasury.kvs.clienttool.util.CryptoUtils metoda validatePasswords())

Dešifrování zašifrovaných souborů

- načteme soubory AESKEY_C.TXT, LOGIN_C.TXT a PASSWD_C.TXT
- pro dešifrování použijeme AES algoritmus (jako dešifrovací klíč použijeme osobní dekódovací kód, který ale musí být převeden do hexadecimálního tvaru – třída com.logica.statetreasury.kvs.common.crypto.Lutils metoda parseHex) viz zdrojový kód com.logica.statetreasury.kvs.client.crypto.KVSProviderClientSW metoda decryptIdentificationData (parametr textMode určuje, zda se jedná o textové nebo binární data – v našem případě jsou LOGIN_C.TXT a PASSWD_C.TXT textové zatím co data v AESKEY_C.TXT jsou binární)

Zabalení rozšifrovaných dat do ZIP archivu

- do zipu přibalíme k rozšifrovaným souborům (teď už s názvy AESKEY_C.DEC, LOGIN_C.DEC a PASSWD_C.DEC) i soubory DTRANS_PWD.TXT a PAC_PWD.TXT (jsou v otevřené podobě a teda není potřeba měnit jejich formát)
- nastavíme k ZIP archivu heslo zadané uživatelem

Vygenerování kontrolního součtu

- načteme soubory UJEID.TXT,DATNAR.TXT, JMEZO.TXT a PRIZO.TXT
- kontrolní součet vznikne výpočtem hash kódu zřetězených už dešifrovaných dat (AESKEY_C.DEC, LOGIN_C.DEC a PASSWD_C.DEC) a souborů načtených v předcházejícím kroku (viz třída com.logica.statetreasury.kvs.common.crypto.KVSCryptoUtils metoda computeConfirmationHash)

Výstup:

- zobrazení dešifrovaného loginu a hesla spolu s vypočteným kontrolním součtem uživateli
- ZIP archiv

3.2 Příprava dat v XML formátu před odesláním do CSÚIS

Vstup:

- XML soubor
- ZIP archiv obsahující AES klíč
- Heslo k ZIP archivu

Postup (viz zdrojový kód třída com.logica.statetreasury.kvs.clienttool.util.CryptoUtils metoda encodeXMLFile()):

Validace vstupních dat

- zkontrolujeme existenci obou zadaných souborů

Zašifrování XML souboru

- načteme XML soubor
- načteme AES klíč ze ZIP archivu (viz zdrojový kód třída com.logica.statetreasury.kvs.common.crypto.Lutils metoda loadFromZip)
- podepíšeme (použitý typ podpisu je HMAC signatura) a zašifrujeme XML soubor (viz zdrojový kód třída com.logica.statetreasury.kvs.common.KVSServiceShared metoda signAndEncrypt())

Výstup:

- zašifrovaný XML soubor

3.3 Dešifrování dat přijatých z CSÚIS

Vstup:

- Zašifrovaný soubor, který uživatel obdržel z CSÚIS
- ZIP archiv obsahující AES klíč
- Heslo k ZIP archivu

Postup (viz zdrojový kód třída com.logica.statetreasury.kvs.clienttool.util.CryptoUtils metoda decodeFile()):

Validace vstupních dat

- zkontrolujeme existenci obou zadaných souborů

Dešifrování souboru

- načteme zašifrovaný soubor
- načteme AES klíč ze ZIP archivu (viz zdrojový kód třída com.logica.statetreasury.kvs.common.crypto.Lutils metoda loadFromZip)
- dešifrujeme soubor a ověříme HMAC signaturu zajišťující integritu šifrovaného souboru (viz zdrojový kód třída com.logica.statetreasury.kvs.common.KVSServiceShared metoda decryptAndVerify ())

Výstup:

- zobrazení informace o průběhu dešifrování a výsledek ověření integrity uživateli
- dešifrovaný soubor