IMPLEMENTAREA ALGORITMULUI AES ÎN LIMBAJE DE PROGRAMARE

Luminița DEFTA, *Student Doctorand, Universitatea din Piteşti*

Abstract: *Criptarea informației constă în folosirea unui algoritm pentru a transforma un mesaj cunoscut într-unul criptat. Se foloseşte pentru protejarea datelor împortiva accesului neautorizat. Datele protejate pot fi stocate pe un dispozitiv de suport sau pot fi transmise prin rețea. În această lucrare vom descrie o implementare concretă a algoritmului AES în limbajul de programare Java (folosind librăriile disponibile în Java Development Kit 6) şi în C (folosind librăria OpenSSL). AES (Advanced Encryption Standard) este un algoritm de criptare asimetric inițial adoptat de guvernul SUA şi a fost ales în urma unui lung proces de standardizare.*

***Cuvinte cheie**:* criptare, algoritmi, cheie asimetrică, AES, OpenSSL.

1. Introducere

Criptarea a devenit cea mai populară metodă de protecție, atât pentru comunicații, cât şi pentru datele cu caracter secret. Pe măsura conştientizării beneficiilor aduse de utilizarea criptării, a dezavantajelor lipsei de protecție a informațiilor şi a faptului că tehnologia de criptare a devenit mai simplă, mai accesibilă, criptarea devine o metodă atractivă de protejare a datelor, indiferent dacă este vorba de date secrete transmise prin rețea sau date obişnuite stocate în sistemul de calcul.

Algoritmii DES (Data Encryption Standard) şi AES (Advanced Encryption Standard) sunt cei mai cunoscuți algoritmi criptografici cu cheie secretă. Criptarea cu cheie secretă (SKC) foloseşte o singură cheie atât pentru criptare cât şi pentru decriptare. Expeditorul utilizează cheia pentru criptarea textului clar şi trimite textul criptat destinatarului. Destinatarul aplică aceeaşi cheie în scopul decriptării mesajului şi recuperează textul clar. Pentru că procedeul

AES ALGORITHM IMPLEMENTATION IN PROGRAMMING LANGUAGES

Luminița DEFTA, *Ph.D Student, University of Pitesti*

Abstract: *Information encryption represents the usage of an algorithm to convert an unknown message into an encrypted one. It is used to protect the data against unauthorized access. Protected data can be stored on a media device or can be transmitted through the network. In this paper we describe a concrete implementation of the AES algorithm in the Java programming language (available from Java Development Kit 6 libraries) and C (using the OpenSSL library). AES (Advanced Encryption Standard) is an asymmetric key encryption algorithm formally adopted by the U.S. government and was elected after a long process of standardization.*

Key-words: encryption, algorithms, asymmetric key, AES, OpenSSL.

1. Introduction

The encryption has become the most popular method of protection, both for communications and for the secret data. Encryption technology has become simpler, more accessible and is an attractive method for protecting data, regardless if it's secret data transmitted over the network or stored on a conventional computer system.

The DES (Data Encryption Standard) and AES (Advanced Encryption Standard) are the best known secret key cryptographic algorithms. Secret Key Cryptography (SKC) uses a single key for both encryption and decryption. The sender uses the key for plaintext encryption and sends the encrypted text to the recipient. The recipient applies the same key with the purpose of message decryption and recovers the plaintext. Because the process makes use of a single key, this procedure is also called symmetric encryption. The major difficulty of this encryption type is the key distribution between the communication

face uz de o singură cheie, acest procedeu mai este numit şi *criptarea simetrică*. Dificultatea majoră a acestui tip de criptare este distribuirea cheii între cei doi parteneri de comunicaţii.

2. Scurtă descriere a standardului AES

AES a devenit succesorul oficial al standardului DES începând cu sfârşitul anului 2001. AES este bazat pe o metodă, de criptare cu cheie secretă, denumită Rijndael. AES foloseşte chei de 128, 192 si 256 de biţi, fiind eficient la atacurile criptanalitice prelungite.

Algoritmul Rijndael are performanţe remarcabile. Dacă sistemul DES ar putea fi spart cu un calculator capabil să genereze 2^{56} chei într-o secundă, atunci acelaşi calculator ar trebui să calculeze $149 * 10^{13}$ ani pentru ca să poată sparge codul noului sistem.

Specificaţia AES standardizează dimensiunile de 128, 192 si 256 de biţi pentru lungimea cheii, dar restricţionează lungimea blocului la 128 de biţi. Astfel, intrarea şi ieşirea algoritmilor de criptare şi decriptare este un bloc de 128 de biţi. În publicaţia FIPS numărul 197, operaţiile AES sunt definite sub formă de operaţii pe matrice, unde atât cheia, cât şi blocul sunt scrise sub formă de matrice. La începutul rulării cifrului, blocul este copiat într-un tablou denumit stare (state), primii patru octeţi pe prima coloană, apoi următorii patru pe a doua coloană, şi tot aşa până la completarea tabloului. Algoritmul modifică la fiecare pas acest tablou de numere denumit state, şi îl furnizează apoi ca ieşire.

3. Utilizarea algoritmului AES în Java

Standardul AES a fost adoptat de numeroase tehnologii Java. Începând cu Java 2 SDK, versiunea 1.4, extensia criptografică Java (JCE) a fost integrată atât în SDK cât şi în JRE. De atunci, nu a mai fost necesară instalarea unui pachet JCE suplimentar, partners.

2. A brief AES description

AES became the official successor of the DES standard in December 2001. AES is based on a secret key encryption method, called Rijndael.

The AES algorithm uses one of three cipher key strengths: 128, 192, or 256-bit encryption key, being efficient to the extended cryptanalytic attacks.

The Rijndael algorithm has remarkable performances. If the DES system could be broken by a computer capable to generate a number of 2^{56} keys per second, then the same computer would have to calculate $149 * 10^{13}$ years to crack the code of the new system.

The AES specification standardizes the size of 128, 192 and 256-bit key length, but restricts the block length to 128 bits. Thus, the input and the output of the encryption and decryption algorithms is a 128-bit block. In the FIPS 197 publication, the AES operations are defined as matrix operations, where both the key and the block are written as the form of matrix.
At the beginning of the cipher run, the block is copied into an array named state, the first four bytes on the first column, then the next four on the second column, and so on until the array completion.
The algorithm modifies at every step this array named state and it provides it as the output.

3. Using the AES algorithm in Java

The AES standard has been incorporated into several Java technology offerings. Beginning with the Java 2 SDK, Standard Edition version 1.4, the Java Cryptography Extension (JCE) was integrated with the SDK and the JRE. Since then, it has no longer been necessary to install the JCE optional package, since support for strong cryptography is now

deoarece suportul complet pentru criptografie este acum parte din J2SE (Java 2 Standard Edition). JCE are o arhitectură de tip furnizor care permite conectarea mai multor furnizori de securitate sub acelaşi cadru de dezvoltare.

Suportul pentru AES în versiunea 6 a platformei Java Standard Edition este asigurat de furnizorul Sun, cunoscut ca SunJCE.

3.1 Generarea cheii

Clasa SecretKeySpec construieşte cheia secretă plecând de la un vector de biţi de lungime 16 (pentru algoritmul AES 128):

```
byte[] raw = new byte[] {'a', 'c', 'x', 'b', 'l', 'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
SecretKeySpec = new SecretKeySpec(raw, "AES");
```

O altă opţiune este folosirea clasei KeyGenerator prin apelarea metodei statice getInstance. Trebuie să specificăm algoritmul folosit, în cazul nostru AES:

```
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(128);
SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
```

3.2 Generarea cifrului

Generarea cifrului este asemănătoare cu generarea cheii. Pentru iniţializarea obiectului folosim metoda statică getInstance cu parametrii AES/ECB/PKCS5Padding şi SunJCE.

AES/ECB/PKCS5Padding furnizează un algoritm AES în modul Cod Electronic de Carte (ECB) şi stilul de umplere PKCS#5. SunJCE reprezintă furnizorul de securitate folosit (un furnizor implementează una sau mai multe aspecte de securitate):

```
cipher = Cipher.getInstance("AES/ECB/PKCS5Padding", "SunJCE");
```

3.3 Criptarea şi decriptarea

available as part of J2SE. JCE has a provider architecture that enables different providers to be plugged in under a common framework.

Several providers have supported AES in their own clean-room implementations of JCE, or under the existing framework. In Java Platform, Standard Edition version 6, the Sun Provider, referred to as Sun JCE, supports AES.

3.1 The key generation

The SecretKeySpec class builds the secret key starting from a bit array which must be of 16-bit length for the AES 128 algorithm:

```
byte[] raw = new byte[] {'a', 'c', 'x', 'b', 'l', 'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
SecretKeySpec = new SecretKeySpec(raw, "AES");
```

Another option is to use the KeyGenerator class by calling the static method getInstance. We must specify the algorithm to use, in our case AES:

```
KeyGenerator kgen = KeyGenerator.getInstance("AES");
kgen.init(128); // 192 and 256 bits may not be available
SecretKey skey = kgen.generateKey();
byte[] raw = skey.getEncoded();
```

3.2 The cipher generation

The cipher generation is similar to the key generation. For the object initialization we use the static method getInstance with the AES/ECB/PKCS5Padding and SunJCE parameters.

AES/ECB/PKCS5Padding provides an AES algorithm, with the Electronic Codebook (ECB) mode and PKCS#5 style padding. SunJCE represents the security provider used (a provider implements some or all parts of Java Security):

```
cipher = Cipher.getInstance("AES/ECB/PKCS5Padding", "SunJCE");
```

3.3 The encryption and decryption

De îndată ce avem o cheie şi un cifru putem începe procesul de criptare / decriptare. Criptarea lucrează la nivel de octet, aşadar se poate cripta aproape orice. Pentru cheie şi cifru trebuie folosit acelaşi algoritm. De exemplu nu putem avea o cheie iniţializată cu DES şi un cifru cu AES. Obiectul de tip Cipher foloseşte aceleaşi metode pentru a cripta şi decripta date, deci trebuie să-l iniţializăm mai intâi pentru a-i seta modul de lucru:

```
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Acest apel pregăteşte obiectul pentru criptare. Apoi vom apela metoda doFinal pe obiectul Cipher:

```
byte[] cipherText = cipher.doFinal(plainText.getBytes());
```

Rezultatul va conţine acum reprezentarea criptată a datelor transmise.

Pentru decriptarea aceleiaşi informaţii trebuie să reiniţializăm obiectul Cipher:

```
cipher.init(Cipher.DECRYPT_MODE, key);
```

Apoi se poate realiza decriptarea:

```
byte[] decrypted = cipher.doFinal(encypted);
```

Prezentăm mai jos funcţiile folosite pentru criptare şi decriptare, folosind o cheie pe 128 de biţi.

```
public static String encrypt(String plainText) throws Exception {
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] cipherText = cipher.doFinal(plainText.getBytes());
        return new String(new BASE64Encoder().encode(cipherText));
}

public static String decrypt(String codedText) throws Exception {
        byte[] encypted = new BASE64Decoder().decodeBuffer(codedText);
        cipher.init(Cipher.DECRYPT_MODE, key);
```

Once we have a key and a cipher we are ready for encryption / decryption. Encryption works at the byte level, so almost anything can be encrypted. The same algorithm must be used for the key and the cipher. For example we cannot have a key initialized with DES and a cipher initialized with AES.

The Cipher object uses the same methods to encrypt and decrypt data, so we must initialize it first to let it know what we want to do with the data:

```
cipher.init(Cipher.ENCRYPT_MODE, key);
```

This call initializes the Cipher object and gets it ready to encrypt data. Then, we will call the doFinal method on the Cipher object:

```
byte[] cipherText = cipher.doFinal(plainText.getBytes());
```

The result will now contain the encrypted representation of the passed-in data.

To decrypt the same data we must reinitialize the Cipher object:

```
cipher.init(Cipher.DECRYPT_MODE, key);
```

Then we are ready for decryption:

```
byte[] decrypted = cipher.doFinal(encypted);
```

We present below the functions used to encryption and decryption.

It uses a 128-bit key:

```
public static String encrypt(String plainText) throws Exception {
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] cipherText = cipher.doFinal(plainText.getBytes());
        return new String(new BASE64Encoder().encode(cipherText));
}

public static String decrypt(String codedText) throws Exception {
        byte[] encypted = new BASE64Decoder().decodeBuffer(codedText);
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decrypted = cipher.doFinal(encypted);
```

```
        byte[] decrypted = cipher.doFinal(encypted);
        return new String(decrypted);
}
```

Pentru caracterele 'password test', rezultatul va fi: e2aF1oK0MpchRIKHTmEgAA==

În mod implicit, datorită unor restricţii de securitate, Java nu permite criptarea folosind chei de 192 sau 256 de biţi. Pentru a înlătura această restricţie trebuie să suprascriem fişierele local_policy.jar şi US_export_policy.jar din directorul jre\lib\security. Acestea pot fi descărcate de la adresa: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jce_policy-6-oth-JPR@CDS-CDS_Developer

4. Utilizarea algoritmului AES în C

Java are suport încorporat pentru diferite elemente de securitate precum manipularea certificatelor, semnături electronice şi criptare. În limbajul C însă, pentru a putea beneficia de aceste funcţionalităţi, trebuie să folosim o librărie separată. În acest scop putem folosi OpenSSL disponibil la adresa http://www.openssl.org/.

Pentru a putea dezvolta aplicaţii în OpenSSL, trebuie să avem atât fişierele de dezvoltare cât şi de instalare ale acestuia.

Fişierele sursă pot fi descărcate de la adresa http://www.openssl.org/source/ iar binarele pentru sistemul de operare windows se găsesc la adresa: http://www.slproweb.com/products/Win32OpenSSL.html

4.1 Configurarea proiectului AES în Visual Studio 2008

Pentru a putea dezvolta o aplicaţie în Windows care foloseşte OpenSSL trebuie să dispunem de următoarele fişiere şi directoare:

```
        return new String(decrypted);
}
```

For the 'password test' string, the result will be: e2aF1oK0MpchRIKHTmEgAA==

By default, due to some security restrictions, Java does not allow using encryption keys of 192 and 256 bits. To remove this restriction, we must override the local_policy.jar and US_export_policy.jar files located in the jre\lib\security directory. These files can be downloaded from the following location: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jce_policy-6-oth-JPR@CDS-CDS_Developer

4.Using the AES algorithm in C

JDK has a built in support for security features such as certificate handling, encryption and signing as you see in the previous statement. However, to get those functionalities in C/C++ a separate library needs to be installed. For this we can use OpenSSL. (http://www.openssl.org/).
To develop applications it is required to have both the development and installation files for OpenSSL.
The sources can be downloaded from http://www.openssl.org/source/ and the binaries for windows from the following address: http://www.slproweb.com/products/Win32OpenSSL.html

4.1 Setup the AES encryption project in Visual Studio 2008

To start developing an encryption application on Windows which uses the OpenSSL library, we need the following files:

- libeay32.lib
- include directory
- libeay32.dll

- libeay32.lib
- include
- libeay32.dll

Pentru a seta proiectul folosind Visual Studio 2008, trebuie să parcurgem următorii paşi:

1.Selectare 'File' → 'New' → 'Project' → 'Visual C++' → 'General' → 'Empty project'.

Deselectarea opțiunii 'Create directory for solution'.

Selectarea unui folder pentru proiect, de exemplu C:\Encryption.

2.Copierea fişierului libeay32.lib şi a directorului include în folderul C:\Encryption.

Copierea librăriei libeay32.dll în directorul C:\Encryption\Release.

3.Din fereastra 'Solution Explorer' selectăm proiectul Encryption, click dreapta apoi selectăm 'Properties' şi apoi 'Configuration Properties'.

a. Selectăm 'General' → 'Configuration type' → 'Application (.exe)'
b. Selectăm 'General' → 'Use of MFC' → 'Use MFC in a Static Library'
c. Selectăm 'General' → 'Character set' → 'Use Multi-Byte Character Set'
d. Selectăm 'C/C++' → 'Additional Include Directories' → include
e. Selectăm 'Linker' → 'Input' → 'Additional Dependencies' → libeay32.lib

4.Din meniul principal selectăm 'Build' → 'Configuration Manager'.

Pentru opțiunea 'Active Solution Configuration' alegem 'Release'.

Pentru opțiunea 'Active Solution Platform' alegem 'Win32'.

5.Pentru compilarea proiectului, îl selectăm din fereastra 'Solution Explorer', apoi alegem 'Build' din meniul principal şi 'Build Solution'.

6.Pentru a rula aplicația, alegem 'Debug' din meniul principal şi apoi alegem 'Start Without Debugging'.

To setup the project using Visual Studio 2008 we must follow the following steps:

7. Click 'File' → 'New' → 'Project' → 'Visual C++' → 'General' → 'Empty project'.
   Uncheck the 'Create directory for solution' option.
   Choose a folder for the project, for example C:\Encryption.
8. Copy libeay32.lib and include directory to C:\Encryption.
   Copy libeay32.dll to the directory C:\Encryption\Release.
9. From the 'Solution Explorer' window select the Encryption project, right click then select 'Properties' and then 'Configuration Properties'.
   a. Select 'General' → 'Configuration type' → 'Application (.exe)'
   b. Select 'General' → 'Use of MFC' → 'Use MFC in a Static Library'
   c. Select 'General' → 'Character set' → 'Use Multi-Byte Character Set'
   d. Select 'C/C++' → 'Additional Include Directories' → include
   e. Select 'Linker' → 'Input' → 'Additional Dependencies' → libeay32.lib
10. From the top menu select 'Build' → 'Configuration Manager'.
    At 'Active Solution Configuration' option choose 'Release'.
    At 'Active Solution Platform' option select 'Win32'.
11. To compile the project, select it from the 'Solution Explorer', choose 'Build' from the top menu and then 'Build Solution'.
12. To run the application, choose 'Debug' from the top menu and then click 'Start Without Debugging'.

4.2 AES encryption functions

4.2 Funcţiile de criptare

Pentru setarea cheii şi a cifrului la criptare se foloseşte funcţia EVP_EncryptInit_ex iar la decriptare vom folosi EVP_DecryptInit_ex. Criptarea efectivă se realizează prin apeluri ale funcţiei EVP_EncryptUpdate. Aceasta funcţie poate fi apelată de mai multe ori pentru a cripta blocuri de date succesive. Dacă opţiunea de padding (umplere) este activă, atunci funcţia EVP_EncryptFinal_ex criptează informaţiile 'finale', însemnând orice informaţie care a rămas într-un bloc parţial. Se foloseşte padding-ul de bloc standard PKCS5.

Asemănător, pentru decriptare se vor folosi funcţiile EVP_DecryptUpdate şi EVP_DecryptFinal_ex.

Următoarele funcţii permit criptarea şi decriptarea de text folosind algoritmul AES cu o cheie pe 128 de biţi:

```
//criptare folosind AES 128
char* cryptAES128(char* stringToEncrypt)
{
        int outlen, i, inlen =
strlen(stringToEncrypt);
        int maxlen = 0;

        //Set AES 128 Key
        unsigned char key[] = {'a', 'c', 'x', 'b',
'l', 'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
        unsigned char iv[] = {0, 0, 0, 0, 0, 0,
0, 0};
        unsigned char *outbuf;

        EVP_CIPHER_CTX ctx;
        EVP_CIPHER_CTX_init(&ctx);
        EVP_EncryptInit_ex(&ctx,
EVP_aes_128_ecb(), 0, key, iv);

        outbuf = (unsigned char
*)malloc(inlen +
EVP_CIPHER_CTX_block_size(&ctx));
        memset(outbuf, 0, inlen +
EVP_CIPHER_CTX_block_size(&ctx));
        EVP_EncryptUpdate(&ctx, outbuf,
&outlen, (const unsigned char
```

To set the key and the cipher for encryption, we can use the EVP_EncryptInit_ex function, and for decryption we will use the EVP_DecryptionInit_ex function. The actual encryption is performed by calling the EVP_EncryptUpdate function. This function can be called multiple times to encrypt successive blocks of data. If padding is enabled (default) then the EVP_EncryptFinal_ex function encrypts the "final" data that is any data that remains in a partial block. It uses standard block padding (PKCS padding). Similarly, for decryption we'll use the EVP_DecryptUpdate and EVP_DecryptFinal_ex function.

The following functions encrypt and decrypt plaintext using the AES algorithm with a 128-bit key length:

```
//AES 128 encryption
char* cryptAES128(char* stringToEncrypt) {
        int outlen, i, inlen =
strlen(stringToEncrypt);
        int maxlen = 0;

        //Set AES 128 Key
        unsigned char key[] = {'a', 'c', 'x', 'b', 'l',
'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
        unsigned char iv[] = {0, 0, 0, 0, 0, 0, 0,
0};
        unsigned char *outbuf;

        EVP_CIPHER_CTX ctx;
        EVP_CIPHER_CTX_init(&ctx);
        EVP_EncryptInit_ex(&ctx,
EVP_aes_128_ecb(), 0, key, iv);

        outbuf = (unsigned char *)malloc(inlen +
EVP_CIPHER_CTX_block_size(&ctx));
        memset(outbuf, 0, inlen +
EVP_CIPHER_CTX_block_size(&ctx));
        EVP_EncryptUpdate(&ctx, outbuf,
&outlen, (const unsigned char
*)stringToEncrypt,
                                inlen);
        EVP_EncryptFinal_ex(&ctx, outbuf +
outlen, &outlen);
        EVP_CIPHER_CTX_cleanup(&ctx);

        if (outlen > strlen((char*)outbuf))
```

```
*)stringToEncrypt,
                                inlen);
        EVP_EncryptFinal_ex(&ctx, outbuf +
outlen, &outlen);
        EVP_CIPHER_CTX_cleanup(&ctx);

        if (outlen > strlen((char*)outbuf))
maxlen = outlen;
        else maxlen = strlen((char*)outbuf);

        return cryptBase64((unsigned
char*)outbuf, maxlen);
}

//decriptare folosind AES 128
char*                   decryptAES128(char*
stringEncrypted) {
        int length = 0;

        //Set AES 128 Key
        unsigned char key[] = {'a', 'c', 'x', 'b',
'l', 'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
        unsigned char iv[] = {0, 0, 0, 0, 0, 0,
0, 0};
        unsigned char *outbuf;
        EVP_CIPHER_CTX ctx;
        EVP_CIPHER_CTX_init(&ctx);
        EVP_DecryptInit_ex(&ctx,
EVP_aes_128_ecb(), 0, key, iv);

        stringEncrypted                 =
decryptBase64(stringEncrypted, &length);
        outbuf = (unsigned char
*)malloc(length +
EVP_CIPHER_CTX_block_size(&ctx));
        memset(outbuf, 0, length +
EVP_CIPHER_CTX_block_size(&ctx));
        EVP_DecryptUpdate(&ctx, outbuf,
&length, (const unsigned char
*)stringEncrypted,
                                length);
        EVP_DecryptFinal_ex(&ctx, outbuf
+ length, &length);
        EVP_CIPHER_CTX_cleanup(&ctx);

        return (char*)outbuf;
}
```

Rezultatul criptării textului 'password text' este identic cu cel al alogritmului Java

```
maxlen = outlen;
        else maxlen = strlen((char*)outbuf);

        return cryptBase64((unsigned
char*)outbuf, maxlen);
}

//AES 128 decryption
char* decryptAES128(char* stringEncrypted) {
        int length = 0;

        //Set AES 128 Key
        unsigned char key[] = {'a', 'c', 'x', 'b', 'l',
'v', 'm', 'k', 'a', 'v', 't', 'i', 'n', 'b', 't', 'o' };
        unsigned char iv[] = {0, 0, 0, 0, 0, 0, 0,
0};
        unsigned char *outbuf;
        EVP_CIPHER_CTX ctx;
        EVP_CIPHER_CTX_init(&ctx);
        EVP_DecryptInit_ex(&ctx,
EVP_aes_128_ecb(), 0, key, iv);

        stringEncrypted                 =
decryptBase64(stringEncrypted, &length);
        outbuf = (unsigned char *)malloc(length
+ EVP_CIPHER_CTX_block_size(&ctx));
        memset(outbuf, 0, length +
EVP_CIPHER_CTX_block_size(&ctx));

        EVP_DecryptUpdate(&ctx, outbuf,
&length, (const unsigned char
*)stringEncrypted,
                                length);
        EVP_DecryptFinal_ex(&ctx, outbuf +
length, &length);
        EVP_CIPHER_CTX_cleanup(&ctx);

        return (char*)outbuf;
}
```

For the 'password text' example, the output is similar to the Java algorithm:
e2aF1oK0MpchRIKHTmEgAA==

Conclusions

AES may, as all algorithms, be used in different ways to perform encryption. It is easy

descris anterior:
e2aF1oK0MpchRIKHTmEgAA==

Concluzii

AES poate fi folosit (ca şi alţi algoritmi) în diferite moduri pentru realizarea criptării. Este facilă implementarea unui sistem care să folosească AES ca algoritm de criptare, dar mai multă pricepere şi experienţă este necesară pentru implementarea lui într-un mod corect pentru o situaţie dată. Începând cu versiunea 6, Java are suport încorporat pentru algoritmul AES prin intermediul furnizorului SunJCE care – cum s-a şi arătat în acest articol – este uşor de folosit şi există de asemenea multe documentaţii şi exemple disponibile.

Implemetarea în limbajul C este mult mai dificilă deoarece, deşi putem găsi câteva documentaţii bune, exemplele concrete de folosire lipsesc.

Bibliografie

1. Tanenbaum A., *Reţele de Calculatoare*, Ed. Byblos S.R.L., 2003.
2. Knudsen J., *Java Cryptography*, Ed. O'Reilly, 1998.
3. Viega J., Messier M., Pravir C., *Network Security with OpenSSL*, Ed. O'Reilly, 2002.
4. http://download.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html
5. http://ro.wikipedia.org/wiki/AES

www.bitzipper.com/aes-encryption.html

to implement a system using AES as its encryption algorithm, but much more skill and experience is required to do it in the right way for a given situation.

Starting with the version 6, Java has built-in support for the AES algorithm through the SunJCE provider which – as shown in this article – it is easy to use and there are many documentations and examples available.

The implementation in the C language is much more difficult because, even if we can find some good documentations, the concrete examples are missing.

Bibliography

1. Tanenbaum A., *Retele de Calculatoare*, Ed. Byblos S.R.L., 2003.
2. Knudsen J., *Java Cryptography*, Ed. O'Reilly, 1998.
3. Viega J., Messier M., Pravir C., *Network Security with OpenSSL*, Ed. O'Reilly, 2002.
4. http://download.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html
5. http://ro.wikipedia.org/wiki/AES
6. http://www.bitzipper.com/aes-encryption.html