التشفير باستخدام مكتبة OpenSSL
-مثال عملي-

-عصام علي
issamsoft.com

مقدمة

أصبح التشفير(التعمية)أحد المتطلبات الشائعة جداً في صناعة البرمجيات، ومن الطبيعي أن تجد نفسك بحاجة لاستخدام شيء من التشفير حتى عند تطويرك لبرامج بسيطة. هناك خوارزميات شائعة للتشفير ويمكنك العثور على الكثير من الموارد التي تشرحها بالتفصيل على الانترنت. وبالنسبة لتطبيق هذه الخوارزميات فأنت أيضاً أمام خيارات كثيرة حسب لغة البرمجة أو أداة التطوير التي تستخدمها. طبعاً يمكنك أنت أيضاً أن تقوم بتطوير أو كتابة تطبيق implementation لهذه الخوازميات بنفسك واستخدامه من ثم في برامجك، إلا أن هذا الأمر غير محبّذ ولا ينصح به، كما أنه من غير المحبّذ الاعتماد على خوارزميات تشفير خاصة تم تطويرها من قبلك، إذ إنه من الأفضل في مجال التشفير بالتحديد اللجوء لاستخدام نصوص مصدرية أو تطبيقات implementations تم ويتم اختبارها ومراجعتها باستمرار من قبل جهات ومطورين ومختصين في مجال البرمجة والتشفير، بهذا أنت تضمن أن تطبيق خوارزمية التشفير الذي ستستخدمه هو خالي من العيوب والثغرات. وفي هذا السياق تأتي OpenSSL كخيار ممتاز حيث تعتبر من أكثر المكاتب استخداماً بالاضافة إلى أنها مفتوحة المصدر وتحظى بكثير من المتابعة والاهتمام من قبل مبرمجين ومختصين من مختلف أنحاء العالم.

في هذا المقال سأعرض مثال لكيفية التشفير وفك التشفير باستخدام إجرائيات التشفير ضمن OpenSSL.خوارزمية التشفير التي سأستخدمها هي AES ولغة البرمجة التي سأستخدمها في هذا المثال هي C.

واحدة من العادات الجيدة والتي ينصح المبرمج باكتسابها هي محاولة أخذ فكرة ولو عامة عن أي تقنية أو موضوع يواجهه ضمن مشروعه البرمجي، ولا ينصح المبرمج بالاكتفاء بالبحث عن حلول على الانترنت ومن ثم استخدامها بدون الغوص في أي تفاصيل.إن أخذ فكرة ولو عامة عن الموضوع أو التقنية التي يستخدمها المبرمج هو أمر جيد، حيث بإمكانه لاحقاً أن يعمق فكرته هذه أكثر فأكثر كلما زاد عمله أو تخصصه بهذه التقنية أو ذاك المجال، وفي هذا الاطار ربما لابأس بشيء من العرض التاريخي لخوارزمية التشفير التي سأستخدمها هنا وهي AES فقد يعطينا هذا العرض التاريخي شيئاً من الحماس أو الدافع للخوض أكثر في غمار هذا المجال الممتع. والشيء الجميل في مجال التشفير أننا لن ننضطر للذهاب بعيداً في التاريخ، حيث أن معظم خوارزميات التشفير الشائعة حالياً هي حديثة العهد نسبياً إذ يمكن إعتبار تسعينات القرن الماضي هي عملياً الفترة التي شهدت بداية التطور في مجال التشفير الى الشكل الذي نعرفه اليوم، والكثير من علماء وأساطير التشفير ومبتكري هذه الخوارزميات هم مازالو على قيد الحياة حتى كتابة هذه السطور.

كيف تم ابتكار معيار/خوارزمية التشفير AES؟

تتمتع AES بمزايا كثيرة جعلت الكثير من المختصين يعتبروها أهم خوارزمية تشفير في العالم حالياً. طبعا لايتوفر إحصائيات دقيقة في هذا المجال ولكن من الواضح أن هناك عدد كبير جداً من البرمجيات حالياً تعتمد على AES في التشفير.

قبل خوارزمية AES كان هناك معيار أو خوارزمية تشفير شائعة في مجال تشفير البيانات الرقمية وهي DES والتي جاء اسمها اختصاراً لـ (Data Encryption Standard). خوارزمية DES تم تطويرها داخل شركة IBM الأمريكية في بداية السبعينات. لاحقاً وفي عام 1997 تم إعلان خوارزمية DES أنها غير آمنة وبدأ البحث عن بدائل. وفي نفس العام قام المعهد الوطني للمعايير والتكنولوجيا الأمريكي NIST بإعلان مسابقة لاختيار ما أطلق عليه اسم معيار التشفير المتقدم AES أختصاراً ل (Advanced Encryption Standard) وطلب من العلماء والمختصين حول العالم تقديم أبحاث وخوارزميات تشفير ليقوم المعهد باختيار الخوارزمية الأنسب واعتمادها كمعيار. بعد حوالي عام ونصف تقريباً أي في عام 1998كان المعهد قد تلقى 15خوارزمية تشفير فقط من علماء ومختصين حول العالم وهو رقم ربما كان أقل من التوقعات. ثم بدأ التدقيق في هذه الخوارزميات ومحاولة كسرها وإيجاد عيوب فيها، أيضاً بمشاركة مختصين من كل أنحاء العالم. وفي عام 1999 كان قد تم استبعاد 10 خوارزميات من الخوارزميات المتنافسة، واقتصرت المنافسة النهائية على 5خوارزميات، هذه الخوارزميات التي أصبحت معروفة فيما بعد هي كلها خوارزميات تشفير آمنة ولكن كان على المعهد اختيار واحدة منها فقط لتصبح معياراً. واستغرق الأمر عامين تقريباً حيث أعلن في شهر تشرين الأول عام 2000عن فوز خوارزمية رايندال Rijndael -قد يلفظها البعض رايندول- بالمسابقة وتم اعتمادها رسمياً كمعيار تشفير متقدم للبيانات الرقمية AESعام 2001. الخوارزمية الفائزة هي من ابتكار أثنين من المختصين في مجال التشفير من بلجيكا وتسميتها جاءت كاختصار لاسم هذين الشخصين وهما Vincent Rijmen و Joan Daemen غير أن هذا الاسم (Rijndael) لم يستخدم كثيراً بعد أن فازت هذه الخوارزمية وشاع بدلاً عنه اسم AES.

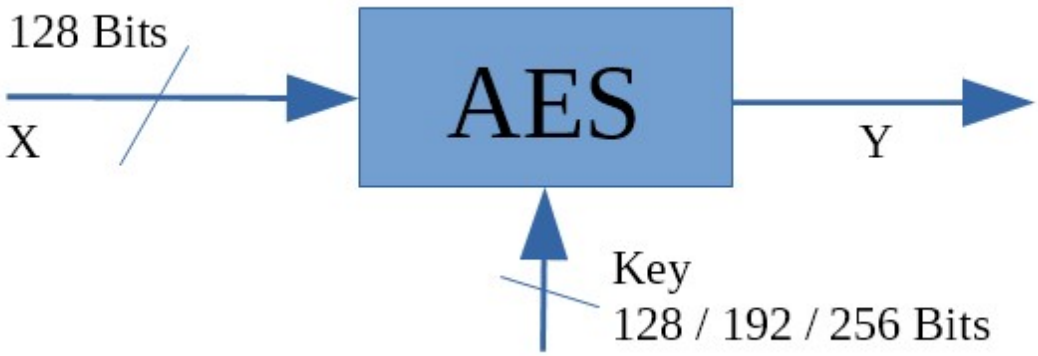
يتضح من هذا العرض التاريخي أن معهد NISTحاول التغلب على أهم نقطة ضعف في DES وهي أنها قد تم ابتكارها خلف الأبواب المغلقة لـ IBM وفي جو من السرية والتكتم، بينما AES تم اختيارها من بين 15خوارزمية تشفير نشرت على الملأ وشارك المئات من المختصين حول العالم في محاولة كسرها على مدى أعوام قبل أن يتم اختيار الخوارزمية الأقوى واعتمادها.

كيف تعمل خوارزمية التشفير Rijndael (AES)؟

سأحاول أن لا أخوض في عمق آلية عمل هذه الخوارزمية واكتفي فقط بإعطاء معلومات عامة ستكون مفيدة في حال رغبت باستخدام هذه الخوارزمية. أما إذا كان مشروعك أو برنامجك أكثر تخصصاً بالتشفير فيلزمك الحصول على معلومات أكثر ولهذا سأترك في نهاية المقال بعض الروابط لمن يرغب في التوسع والحصول على معلومات أكثر.

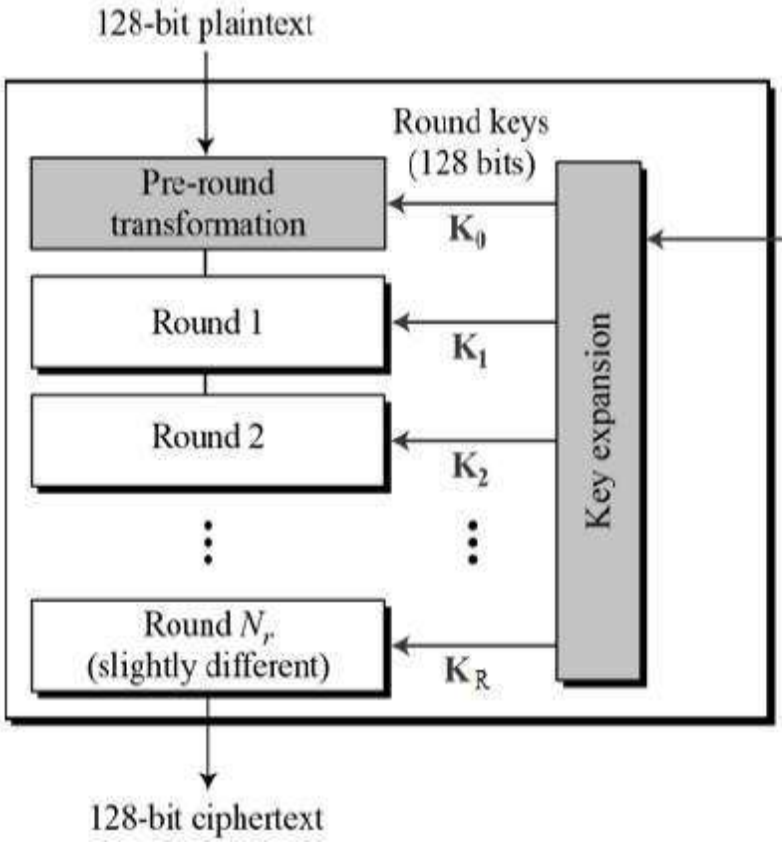
إن خوارزمية رايندال أو كما سنسميها من الآن وصاعداً AES هي خوارزمية تشفير مقطعية (كتلية)، أي تعتمد على تشفير مقاطع/كتل البيانات Data Block، طول المقطع -الافتراضي- في AES هو 128بت (16بايت)، بينما كان طول المقطع في DES

هو 64 بت فقط. يتم تشفير مقطع البيانات في AES باستخدام مفتاح تشفير، خوارزمية AES تدعم ثلاثة أطوال لمفاتيح التشفير (128, 192, 256)، وبالمناسبة فقد كان هذا مطلباً أساسيا من قبل NIST عندما طرح مسابقته لاختيار خوارزمية التشفير، إذ اشترط أن تدعم الخوارزميات المقدمة للمسابقة ثلاثة مستويات من الأمان، قياسي ومتوسط وعالي.
المخطط التالي يبسط آلية العمل هذه، حيث X هنا هي البيانات قبل التشفير و Y هي البيانات المشفرة.



ما يميز هذه الخوارزمية هو عدد جولات التشفير Roundsحيث يتغير عدد جولات التشفير فيها حسب طول مفتاح التشفير.
128بت = 10جولات
192بت = 12جولة
256بت = 14جولة

بنية AES
بنية AES تميزها أيضاً عن بقية خوازميات التشفير، حيث أن AES تقوم بتشفير كامل ال 128بت خلال كل جولة من جولات التشفير، الصورة التالية من ويكيبيديا توضح هذا الأمر



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

كل جولة تشفير تتألف من أربع طبقات (مراحل):

1- Byte Substitution
2- Shift Row
3- Mix Columns
4- Key Addition

لن نغوص أكثر في تفاصيل هذه المراحل، ونترك هذا لمن يرغب، لكن أشير أن هذه المراحل تشتمل على الكثير من العلاقات والحسابات الرياضية، كضرب وعكس المصفوفات وغيرها. رغم أنه في بعض التطبيقات implementations لهذه الخوارزمية قد يعتمد المبرمجين على جدوال استبدال S-box ولكن حقيقة الأمر أن محتويات هذا الجدول تم حسابها مسبقاً وفق العلاقات الرياضيه لهذه الخوارزمية. وذلك لتوفير طاقة المعالج، حيث يقوم التطبيق باستبدال كل بايت بما يقابلها في جدول الاستبدال عوضاً عن إجراء الحسابات المعقدة في كل مرة وعند كل بايت، هذا الأسلوب سيسرع العمل كما سيؤدي إلى توفير طاقة المعالج وهو أمر حيوي جداً بالنسبةللكثير من منصات التشغيل.

أنظمة العمل عند التشفير المقطعي

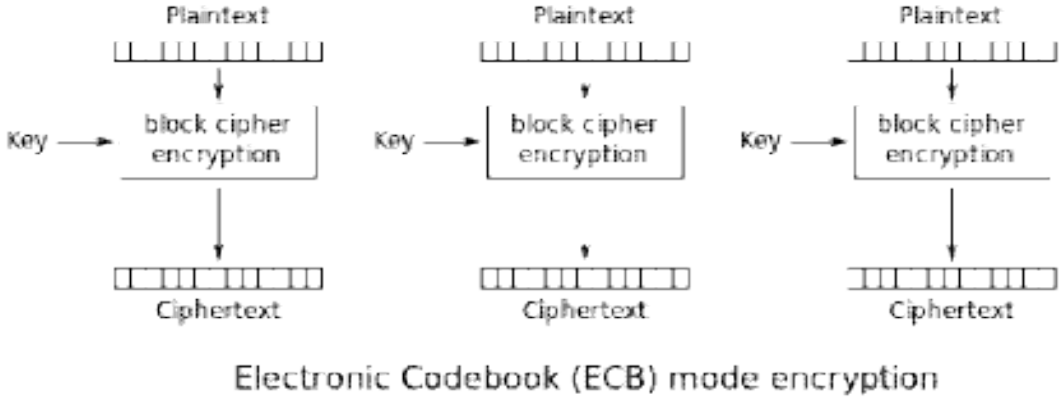
بما أن خوارزميات تشفير المقاطع Data Blocks هي لتشفير مقطع أو مجموعة محددة من البايتات لذلك يلزمنا آلية أو نظام mode لكيفية تطبيق هذه الخوارزمية على مقطع يحتوي على عدد بايتات أكبر. أي بمعنى آخر إن خوارزمية التشفير تقوم بتشفير كل مقطع على حدة لذلك يلزمنا آلية أو نظام أو خوارزمية لكيفية تكرار تشفيرووصل هذه المقاطع المشفرة مع بعضها بحيث نحافظ على أمن البيانات المشفرة. وهنا لدينا أنظمة Modes مختلفة ومتعددة منها (ECB, CBC, OFB, CFB, CTR). هذه الأنظمة يشار إليها بأنظمة العمل، أو بأنظمة التشفير.

معظم هذه الأنظمة تطلب مصفوفة من البايتات تسمى مصفوفة (أو موجه/شعاع)تهيئة initialization vector أو اختصاراً IV هذه المصفوفة من البايتات يجب أن تكون عشوائية. هذا الـ IV له أيضاً خصائص ومتطلبات تختلف عن مفتاح التشفير الـ Key مثلاً الـ IV ليس بالضرورة أن يكون سرياً.

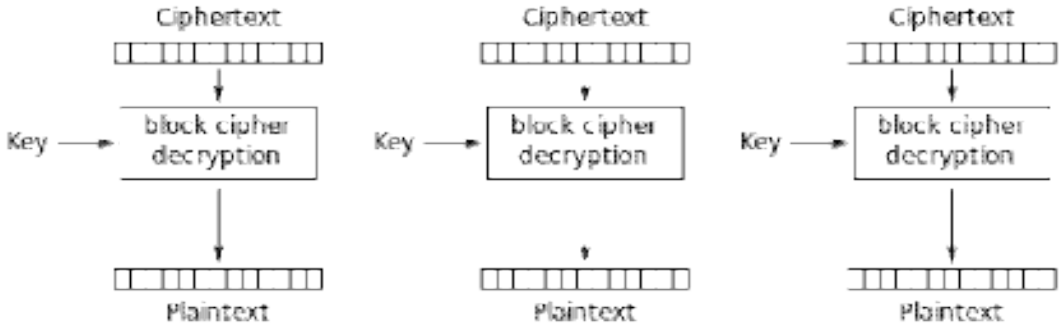
بعض الأنظمة (تحديداً ECB و CBC) تتطلب إضافة حشوة padding الى المقطع الأخيرقبل القيام بالتشفير ليصبح طول النص عبارة عن جداء عدد صحيح في طول المقطع Block Size. عادة الحشو يكون باضافة بايتات صفرية Null Bytes .

نظام التشفير ECB

رغم أننا لن نعتمد هذا النظام في هذا المثال لكن من الجيد إعطاء لمحة عن هذا النظام. ECB هو أبسط نظام عمل (نظام تشفير) وجاء اختصاراً من Electronic Codebook . هذا النظام يقوم ببساطة على تشفير كل مقطع على حدة ومن ثم جمع المقاطع بعد التشفير، وفي فك التشفير تحدث العملية المعاكسة. هذه الصور التالية توضح آلية العمل بشكل جيد.



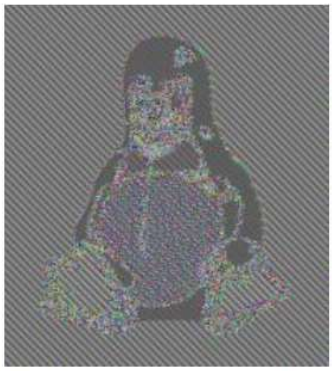
Electronic Codebook (ECB) mode encryption



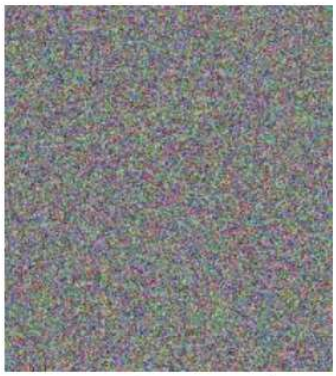
Electronic Codebook (ECB) mode decryption

نظام التشفير ECB هذا يعتبر ضعيفاً خصوصاً عند تشفير بيانات معينة كبيانات الصورغير المضغوطة مثلاً، حيث يمكن أن تبقى بعض ملامح أو حدود الصورة واضحة حتى بعد التشفير، هذا مثال من الويكيبيديا.

الصورة الأصلية



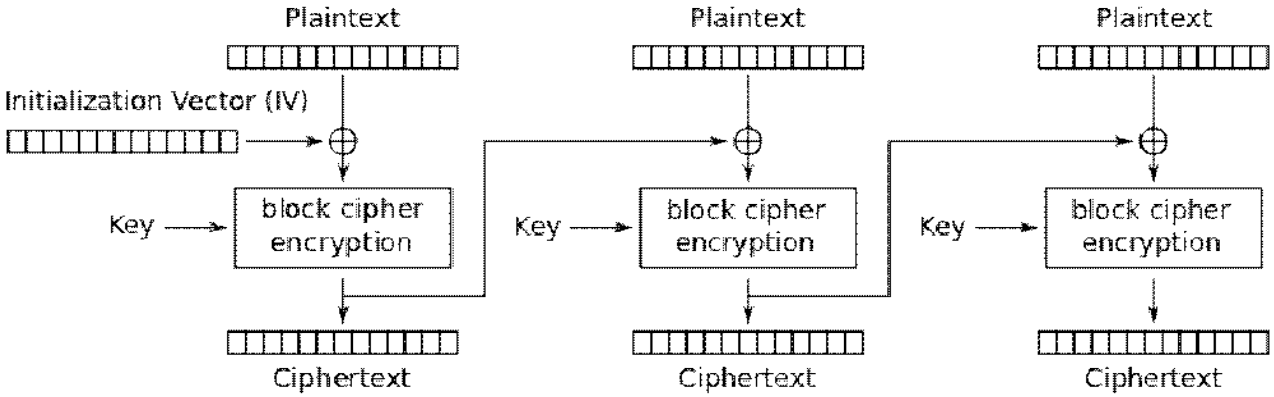
الصورة مشفرة بنظام ECB



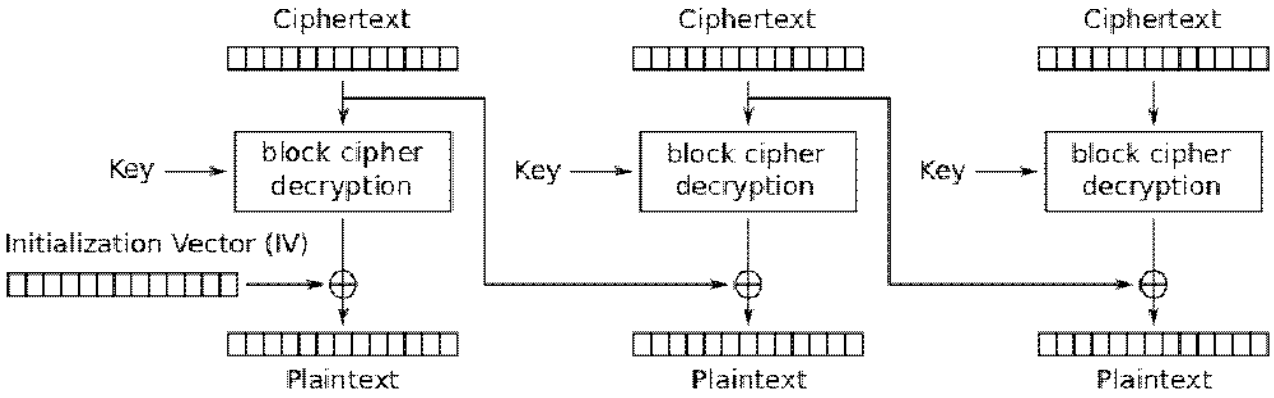
الصورة مشفرة بنظام CBC

نظام التشفيرCBC

الاسم هو اختصار ل Cipher Block Chaining وقد تم ابتكاره في عام 1976 . في هذا النظام يتم دمج/ربط كل مقطع بيانات بالمقطع السابق له بعلاقة XOR . وبهذا كل مقطع مشفر سيكون معتمداً على المقاطع المشفرة السابقة له.
الصور التالية توضح عملية التشفير وفك التشفير بهذا النظام.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

مكتبة OpenSSL

بعد هذه المقدمة عن التشفير سنعرض للمكتبة التي سنستخدمها. وهي مكتبة ال OpenSSL. هذه المكتبة تستخدم لتأمين الاتصالات الآمنة عبر الشبكة ولها استخدامات واسعة في مخدمات الويب وغيرها من البرامج. تحوي مكتبة OpenSSLعلى تطبيق implementation لبروتكولات SSL و TLS. والتي لن نخوض في تفاصيلها ضمن هذا المقال. تأتي OpenSSL على شكل مكتبتين أساسيتين libssl و libcrypto ، وهذه المكتبة الأخيرة هي التي تحوي إجرائيات وتوابع التشفير.

مثال/ تشفير نص باستخدام AESونظام التشفير CBC:

لمزيد من السهولة والتنظيم سأقوم بكتابة وجمع توابع التشفير وفك التشفير ضمن ملف رأسي واحدHeader File.

```
#ifndef _CRPT_H
#define _CRPT_H

#include <openssl/evp.h>
#include <openssl/err.h>
//#include <openssl/applink.c> // you may need this with some compilers due a bug

char*crpt_lastError()
{
long error = ERR_get_error();
return ERR_error_string(error,NULL);
}


int encrypt(unsignedchar*plaintext,int plaintext_len,constunsignedchar*
key,constunsignedchar* iv,unsignedchar* ciphertext)
{
EVP_CIPHER_CTX *ctx;
int len;
int ciphertxt_len;
if(!(ctx = EVP_CIPHER_CTX_new()))return-1;

//initialize the encryption operation
if(1!= EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(),NULL, key, iv))return-1;
//provide the text to be encrypted
if(1!= EVP_EncryptUpdate(ctx, ciphertext,&len, plaintext, plaintext_len))return-1;
ciphertxt_len = len;

//finalize the encryption, may add extra bytes
if(1!= EVP_EncryptFinal_ex(ctx, ciphertext + len,&len))return-1;
ciphertxt_len += len;
//clean up
EVP_CIPHER_CTX_free(ctx);
return ciphertxt_len;
}

int decrypt(unsignedchar*ciphertext,int
ciphertext_len,constunsignedchar*key,constunsignedchar*iv,unsignedchar*plaintext)
{
EVP_CIPHER_CTX *ctx;
int len;
int plaintext_len;
if(!(ctx = EVP_CIPHER_CTX_new()))return-1;
//initialize the encryption operation
if(1!= EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(),NULL, key, iv))return-1;
//provide the encrypted text to be decrypted
if(1!= EVP_DecryptUpdate(ctx, plaintext,&len, ciphertext, ciphertext_len))return-1;
```

```
plaintext_len = len;
//finalize the encryption
if(1!= EVP_DecryptFinal_ex(ctx, plaintext + len,&len))return-1;
plaintext_len += len;
//finalize the decryption
EVP_CIPHER_CTX_free(ctx);
return plaintext_len;
}


#endif
```

ويمكن اعطاء شرح بسيط عن التوابع السابقة:
EVP_CIPHER_CTX_new إجراء يقوم بخلق سياق تشفير ضمن الذاكرة وهو الخطوة الأولى سواء عند التشفير أو عند فك التشفير.
EVP_CIPHER_CTX_free إجراء يقوم بحذف/تفريغ سياق التشفير وتحرير كل الذاكرة المرتبطة به، وهو الخطوة الأخيرة سواء عند التشفير أو عند فك التشفير.
EVP_EncryptInit_ex وهو إجراء يقوم بإعداد سياق التشفير للبدء بعملية التشفير (التشفير تحديداً وليس فك التشفير)، وكما هو واضح فنحن نقوم بتمرير التوابع اللازمة لهذا الأمر وهي سياق التشفير الذي تم انشاءه سابقاً مع مفتاح التشفير بالإضافة إلى إجراء التشفير الذي يتضمن نظام العمل (نظام التشفير وآلية وصل المقاطع) وهنا كما هو واضح أننا نستخدم AESبنظام CBC 128 bitواسم الاجراء المستخدم يدل على ذلك (EVP_aes_128_cbc).
EVP_EncryptUpdate وتعريفه الكامل هو كما يلي:

```
int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx,unsignedchar*out,
int*outl,constunsignedchar*in,int inl);
```

هذا الاجراء يقوم بتشفير عدد محددinlمن البايتات الموجودة في الذاكرة المشار إليها بـ in ويضع النتيجة في الذاكرة المشار إليها بـ out. إن طول النتيجة يحدد بطول النص الأصلي زائد طول المقطع ناقص واحد1 (inl + cipher_block_size - 1) ويجب أخذ هذا بعين الاعتبار عند حجز الذاكرة للنص المشفر outبحيث يكون هناك مساحة كافية للنص المشفر. يمكن استدعاء هذا الاجراء بشكل متتالي أكثر من مرة إذا كنا بصدد تشفير مقاطع بيانات متواصلة ومتتالية.
EVP_EncryptFinal_ex هذا الاجراء يستدعى بعد الانتهاء من تشفير البيانات بواسطة الاجراء السابق، ويضمن هذا الاجراء تشفير أي بيانات متبقية من مقاطع غير مكتملة، فكما أشرنا سابقاً طول المقطع في AESهو 16بايت، ومن الممكن أن يكون طول النص الذي نريد تشفيره 20بايت مثلاً، حينها سيضمن استدعاء الاجراء EVP_EncryptFinal_exتشفير هذه الأربع بايتات الزائدة. ومن المهم الانتباه إلى أنه بعد استدعاء هذا الاجراء يجب عدم استدعاء اجراء التشفير الاساسي أي EVP_EncryptUpdate.

هذا بالنسبة لإجرائيات التشفير الثلاثة، ويقابلها ثلاثة إجرائيات لفك التشفير هي: EVP_DecryptInit_ex و EVP_DecryptUpdate و EVP_DecryptFinal_ex وينطبق عليها نفس الشرح السابق إنما من جهة فك التشفير.
طبعا هناك المزيد من الاجرائيات الخاصة بالتشفير ضمن هذه المكتبة لكن سنكتفي بعرض التوابع التي تم استخدامها في هذا المثال.
بعد أن أعددنا توابع التشفير و فك التشفير في الملف الرأسي crpt.h لنعرض مثالاً ليكيفية الاستخدام:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../crpt.h"


#define STR_OUT_OF_MEMORY "Out of memory!\n"
#define STR_ERROR_UD "Unknown error occurred!"
```

```
intmain(int argc,char**argv)
{
unsignedchar
key[]={0xFC,0xCB,0x44,0xEF,0xAC,0x12,0x87,0x71,0x55,0x21,0xCC,0x03,0x11,0x23,0x33,0
x41,0x57,0x91,0xee,0xbb,0x66,0x37,0xe3,0xa1,0x17,0xc1,0xd5,0xf7,0x81,0xAE,0x65,0x89
};
unsignedchar
iv[]={0x27,0x11,0xAD,0x17,0xF9,0x1D,0x77,0x5A,0x93,0x7E,0xF3,0x71,0x3C,0x4F,0x34,0x
CF};


if(argc <3||(strcmp(argv[1],"-d")!=0&&strcmp(argv[1],"-e")!=0)){
printf("USAGE: %s -e/d <input>\n", argv[0]);
exit(EXIT_FAILURE);
}

if(strcmp(argv[1],"-e")==0){
//encrypt ...
int text_len =strlen(argv[2]);
int cipher_len = text_len + EVP_MAX_BLOCK_LENGTH;//Add extra space
EVP_MAX_BLOCK_LENGTH = 32byte = 256bit
unsignedchar*cipher_text =(unsignedchar*)malloc(cipher_len *sizeof(unsignedchar));
if(!cipher_text){
printf(STR_OUT_OF_MEMORY);
exit(EXIT_FAILURE);
}
int out_len = encrypt((unsignedchar*)argv[2], text_len, key, iv, cipher_text);
if(out_len <0){
//encryption failed, try to get the error
char*err = crpt_lastError();
if(err)
puts(err);
else
printf(STR_ERROR_UD);
exit(EXIT_FAILURE);
}
//print the result
int i;
for(i =0; i < out_len; i++){
printf("%02X", cipher_text[i]);
}
printf("\n");
}
else{
//decrypt ...
//we expect the input to be a hexadecimal string
int text_len =strlen(argv[2]);
if(text_len %2!=0){
printf("Input error!\n");
exit(EXIT_FAILURE);
}
int cipher_len = text_len/2;
unsignedchar*cipher_text =(unsignedchar*)malloc(cipher_len *sizeof(unsignedchar));
if(!cipher_text){
printf(STR_OUT_OF_MEMORY);
```

```
exit(EXIT_FAILURE);
}
//convert the hexadecimal string into byte array
char*pos = argv[2];
size_t cnt;
for(cnt =0; cnt < cipher_len; cnt++){
sscanf(pos,"%02X",&cipher_text[cnt]);
pos +=2;
}
unsignedchar*plain_text =(unsignedchar*)malloc(cipher_len *sizeof(unsignedchar));
if(!plain_text){
puts(STR_OUT_OF_MEMORY);
exit(EXIT_FAILURE);
}

int out_len = decrypt(cipher_text, cipher_len, key, iv, plain_text);
if(out_len <0){
char*err = crpt_lastError();
if(err)
printf("%s\n", err);
else
printf(STR_ERROR_UD);
}
plain_text[out_len]='\0';
printf("%s\n",(char*)plain_text);
}



return0;
}
```

طبعا هذا النص المصدري بسيط والملاحظات المكتوبة داخله كافية لتوضيحه ولا يحتاج إلى مزيد من الشرح، لكن أورد هنا فقط بعض الملاحظات الإضافية التي قد تكون مفيدة:

- لغرض التبسيط قمت بكتابة مفتاح التشفير مع مصفوفة/موجه التهيئة IV ضمن النص المصدري وهو أمر غير محبّذ في النصوص المصدرية الانتاجية (Production Source Code) أي بمعنى آخر النصوص المصدرية للمشاريع الحقيقية، حيث يفضل في مثل هذه المشاريع تمرير مفتاح التشفير للمترجم Compiler أثناء ترجمة المشروع. والسبب أن النصوص المصدرية قد تكون عرضة لأن تكشف على أشخاص كثر وهكذا يكون مفتاح التشفير متاحاً أكثر من اللازم.
- في هذا المثال نقوم بالتشفير ومن ثم نعرض النص المشفر بصيغته الست عشرية hexadecimal فقط لكي نتمكن من مشاهدته على الشاشة، في المشاريع الحقيقية لن تكون مضطراً لهذا على الأغلب، إذ يمكنك تخزين نتيجة التشفير في ملف أو في قاعدة البيانات مباشرةً كما هي.
- بما أننا في هذا المثال نعرض نتيجة التشفير بصيغة ست عشرية، لذلك فأن آلية فك التشفير في هذا المثال تتوقع أن يكون دخلها هو بصيغة ست عشرية أيضاً، حيث تقوم بتحويله إلى بايتات ومن ثم فك تشفيره، لكن في المشاريع الحقيقية دخل فك التشفير سيكون بايتات تم احضارها من ملف أو من قاعدة بيانات أو من الأنترنت ...الخ.

أما بالنسبة لكيفية تجربة المثال السابق، فالصورة التالية توضح كيفية تشغيل التطبيق في وضع التشفير وفك التشفير مع النتيجة.

النص المصدري:

يمكن تحميل النص المصدري من ال github من الرابط التالي:

https://github.com/issamalidev/opensslexample

روابط إضافية:

في مايلي بعض الروابط لمن يحب التوسع والقراءة أكثر حول هذا الموضوع:

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

https://wiki.openssl.org/index.php/Main_Page

https://www.openssl.org/docs/man1.1.0/crypto/EVP_CIPHER_CTX_new.html