

OPENSSL Project

OS used: Ubuntu 16.04

How OPENSSL is called: Only used terminal commands. Created a script "BATCH_SCRIPT" to execute all commands.

Objectives: To encrypt using symmetric and public key encryptions the file lena512color.tiff. Also, to hash lena512color.tiff.



Figure 1. lena512color.tiff

1) Symmetric Encryption

For symmetric encryption, I used the following commands to produce AES on CBC, CFB 128 bits, CFB 1 bit, CFB 8 bits, OFB, and ECB modes. I also produced each key size possible: 128, 192, and 256. Take note that I used a bmp image rather than the tiff image since it is easier to show the encrypted image on bmp.

```
openssl enc -aes-128-cbc -e -in lena.bmp -out CBC-128.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb -e -in lena.bmp -out CFB-128.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb1 -e -in lena.bmp -out CFB1-128.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb8 -e -in lena.bmp -out CFB8-128.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-ofb -e -in lena.bmp -out OFB-128.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-ecb -e -in lena.bmp -out ECB-128.bmp -K 1010101
openssl enc -aes-192-cbc -e -in lena.bmp -out CBC-192.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb -e -in lena.bmp -out CFB-192.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb1 -e -in lena.bmp -out CFB1-192.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb8 -e -in lena.bmp -out CFB8-192.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-ofb -e -in lena.bmp -out OFB-192.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-ecb -e -in lena.bmp -out ECB-192.bmp -K 1010101
openssl enc -aes-256-cbc -e -in lena.bmp -out CBC-256.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-cfb -e -in lena.bmp -out CFB-256.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-cfb1 -e -in lena.bmp -out CFB1-256.bmp -K 1010101 -iv 0101010
```

```
openssl enc -aes-256-cfb8 -e -in lena.bmp -out CFB8-256.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-ofb -e -in lena.bmp -out OFB-256.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-ecb -e -in lena.bmp -out ECB-256.bmp -K 1010101
```

To see each of the pictures when they are encrypted, you need to set the headers of the encrypted file to their original (decrypted) form since they were also encrypted. The header files of bmp images consist of the first 54 bytes of the file. Thus, the following commands were used to put the correct headers to the encrypted files.

```
dd if=lena.bmp of=CBC-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB1-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB8-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=OFB-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=ECB-128.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CBC-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB1-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB8-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=OFB-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=ECB-192.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CBC-256.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB-256.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB1-256.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=CFB8-256.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=OFB-256.bmp bs=1 count=54 conv=notrunc
dd if=lena.bmp of=ECB-256.bmp bs=1 count=54 conv=notrunc
```

I expected that the outcome of the AES encryption of the image would be unreadable except for the ECB mode. However, the ECB mode outputs were also unreadable. This may be due to the fixed block size of AES of 128 bits (16 bytes) compared to the 24-bit size of the image. The block size may not be enough to encrypt the whole color byte. Thus, a color byte may have been decoded into separate blocks, leading to an unreadable encrypted AES-ECB image.

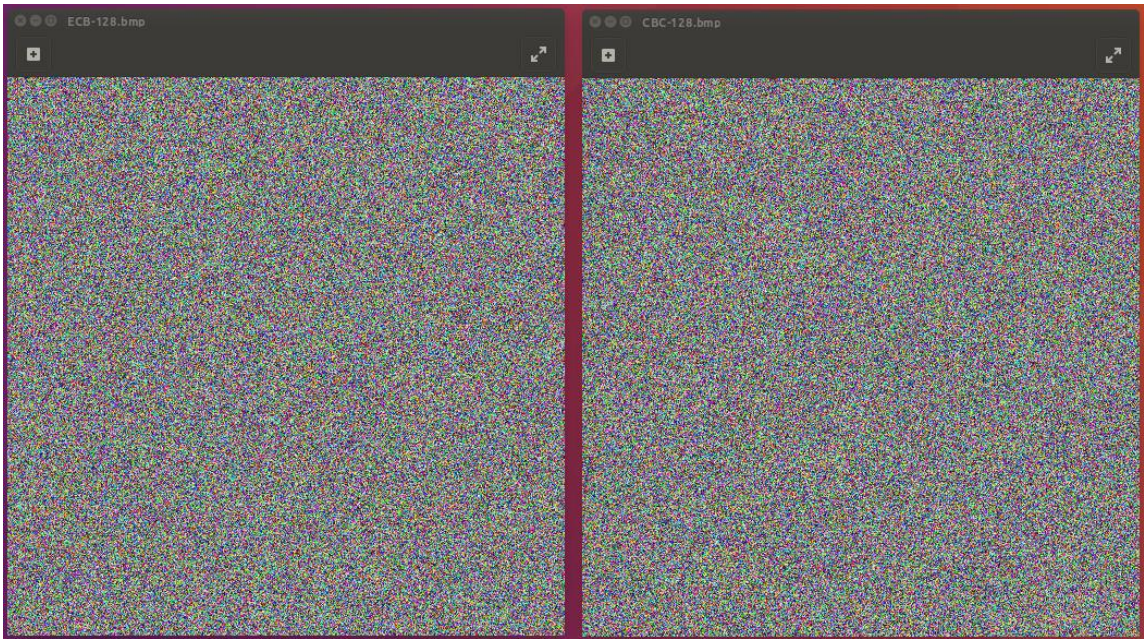


Figure 2. AES-ECB-128 encrypted and AES-CBC-128 encrypted of lena512color.tiff

To decrypt the encrypted images, I ran the following commands. Take note that you need to return the encrypted files with their original encrypted headers.

```
openssl enc -aes-128-cbc -d -in CBC-128.bmp -out CBC-128-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb -d -in CFB-128.bmp -out CFB-128-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb1 -d -in CFB1-128.bmp -out CFB1-128-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-cfb8 -d -in CFB8-128.bmp -out CFB8-128-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-ofb -d -in OFB-128.bmp -out OFB-128-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-128-ecb -d -in ECB-128.bmp -out ECB-128-decrypted.bmp -K 1010101
openssl enc -aes-192-cbc -d -in CBC-192.bmp -out CBC-192-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb -d -in CFB-192.bmp -out CFB-192-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb1 -d -in CFB1-192.bmp -out CFB1-192-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-cfb8 -d -in CFB8-192.bmp -out CFB8-192-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-ofb -d -in OFB-192.bmp -out OFB-192-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-192-ecb -d -in ECB-192.bmp -out ECB-192-decrypted.bmp -K 1010101
openssl enc -aes-256-cbc -d -in CBC-256.bmp -out CBC-256-decrypted.bmp -K 1010101 -iv 0101010
```

```
openssl enc -aes-256-cfb -d -in CFB-256.bmp -out CFB-256-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-cfb1 -d -in CFB1-256.bmp -out CFB1-256-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-cfb8 -d -in CFB8-256.bmp -out CFB8-256-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-ofb -d -in OFB-256.bmp -out OFB-256-decrypted.bmp -K 1010101 -iv 0101010
openssl enc -aes-256-ecb -d -in ECB-256.bmp -out ECB-256-decrypted.bmp -K 1010101
```

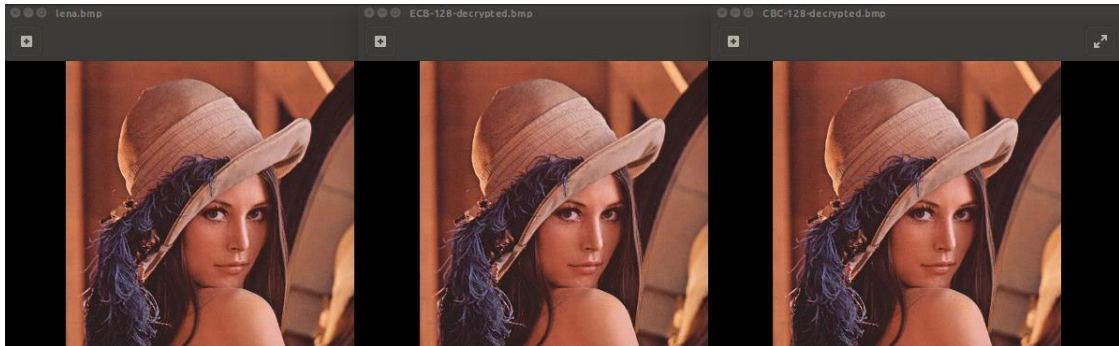


Figure 3. lena.bmp with the decrypted AES-ECB-128 and AES-CBC-128 images

2) Hashing

To produce the SHA1, SHA256, and SHA512 hashes of lena512color.tiff, I used the following commands.

```
openssl dgst -sha1 lena512color.tiff
openssl dgst -sha256 lena512color.tiff
openssl dgst -sha512 lena512color.tiff
```

The output of the hashes is showed on figure 4.

```
psalpano@ubuntu:~/Desktop/PROJECT$ openssl dgst -sha1 lena512color.tiff
SHA1(lena512color.tiff)= e647d0f6736f82e498de8398eccc48cf0a7d53b9
psalpano@ubuntu:~/Desktop/PROJECT$ openssl dgst -sha256 lena512color.tiff
SHA256(lena512color.tiff)= c056da23302d2fb0d946e7ffa11e0d94618224193ff6e2f78ef8097bb8a3569b
psalpano@ubuntu:~/Desktop/PROJECT$ openssl dgst -sha512 lena512color.tiff
SHA512(lena512color.tiff)= 2cb9d7df53eb8640dc48d736974f472a98d9c7186de7a972490455f5f3ed29dfc5b75c95c
cb3ed4596bc2bfc4b1e52cf4d76bcee27d334dd155bb426617392dc
psalpano@ubuntu:~/Desktop/PROJECT$
```

Figure 4. Hash of lena512color.tiff with SHA1, SHA256, and SHA512

3) Public Key Encryption

For public key encryption, we have to produce a private key and a public key. I generated the private key of RSA-2048 using the following command.

```
openssl genrsa -out RSA-private.pem 2048
```


To produce its public key, I used the following command.

```
openssl rsa -in RSA-private.pem -outform PEM -pubout -out RSA-public.pem
```

However, the file is too large to be encrypted by the RSA key. Thus, we would generate a key to encrypt lena512color.tiff by symmetric encryption then we encrypt the generated key using the public key of RSA. I did produce and encrypt the new symmetric key by invoking these commands.

```
openssl rand -base64 128 > SYM-key.bin  
openssl rsautl -encrypt -inkey RSA-public.pem -pubin -in SYM-key.bin -out SYM-key.bin.enc
```

To perform the encryption of lena512color.tiff using the generated key for symmetric encryption, I used the following commands. Take note that I used AES-CBC-256 for the symmetric encryption.

```
openssl enc -e -aes-256-cbc -in lena512color.tiff -out lena512color.enc -pass file:./SYM-key.bin
```

To decrypt the image, we should decrypt the symmetric key first using the private RSA key, since it was encrypted with the public RSA key.

```
openssl rsautl -decrypt -inkey RSA-private.pem -in SYM-key.bin.enc -out SYM-key-received.bin
```

After decrypting the symmetric key, we use this key to decrypt the encrypted image.

```
openssl enc -d -aes-256-cbc -in lena512color.enc -out lena512color-RSA-decoded.tiff -pass file:./SYM-key-received.bin
```

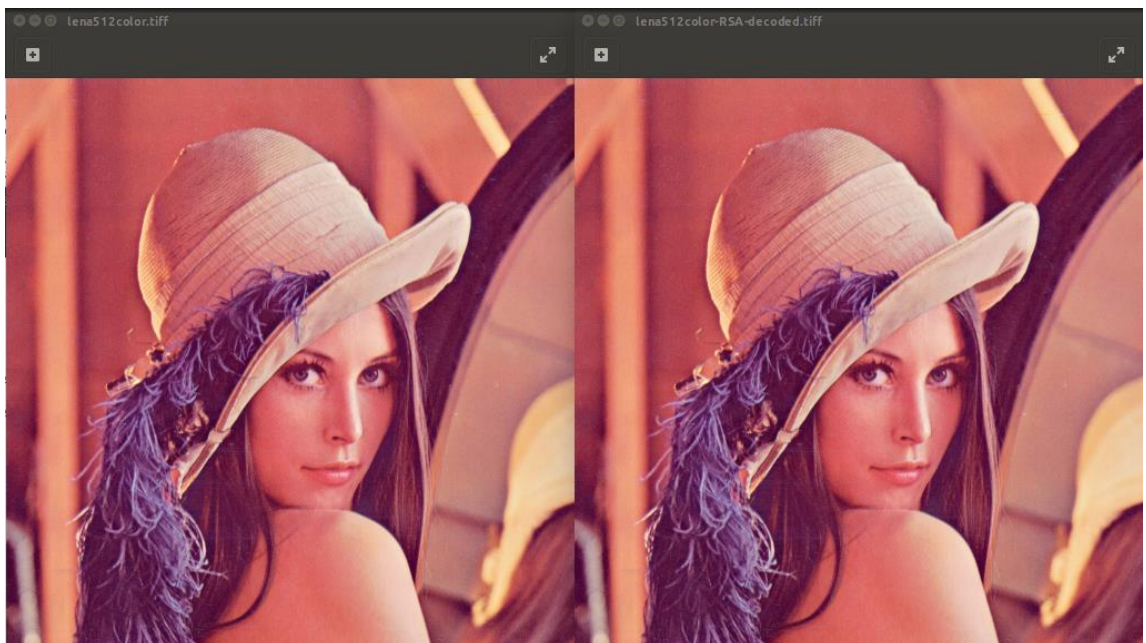


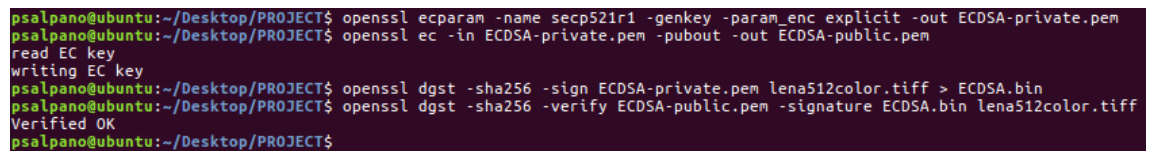
Figure 5. lena512color.tiff with the decoded RSA image.

For the elliptic curve encryption, we have to choose which curve to use and produce a private and public ECDSA key from it. I used the curve *secp521r1* and produced the private and public ECDSA key using the following commands.

```
openssl ecparam -name secp521r1 -genkey -param_enc explicit -out ECDSA-private.pem
openssl ec -in ECDSA-private.pem -pubout -out ECDSA-public.pem
```

To sign the image *lena512color.tiff*, we need to use a hash function and the public ECDSA key generated. To verify, we need to use the private ECDSA generated. I used SHA-256 for the hash and the following commands to sign and verify by ECDSA.

```
openssl dgst -sha256 -sign ECDSA-private.pem lena512color.tiff > ECDSA.bin
openssl dgst -sha256 -verify ECDSA-public.pem -signature ECDSA.bin lena512color.tiff
```



```
psalpano@ubuntu:~/Desktop/PROJECT$ openssl ecparam -name secp521r1 -genkey -param_enc explicit -out ECDSA-private.pem
psalpano@ubuntu:~/Desktop/PROJECT$ openssl ec -in ECDSA-private.pem -pubout -out ECDSA-public.pem
read EC key
writing EC key
psalpano@ubuntu:~/Desktop/PROJECT$ openssl dgst -sha256 -sign ECDSA-private.pem lena512color.tiff > ECDSA.bin
psalpano@ubuntu:~/Desktop/PROJECT$ openssl dgst -sha256 -verify ECDSA-public.pem -signature ECDSA.bin lena512color.tiff
Verified OK
psalpano@ubuntu:~/Desktop/PROJECT$
```

Figure 6. ECDSA signature and verification of *lena512color.tiff*