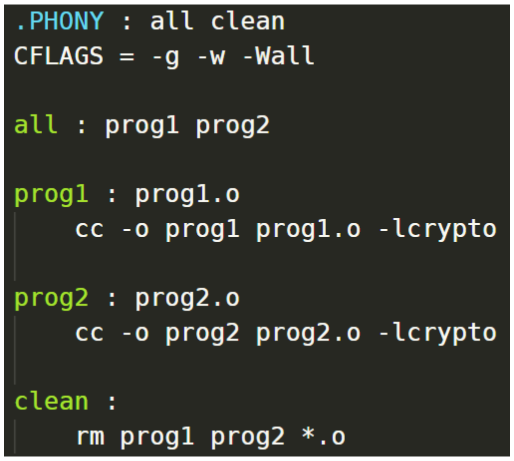
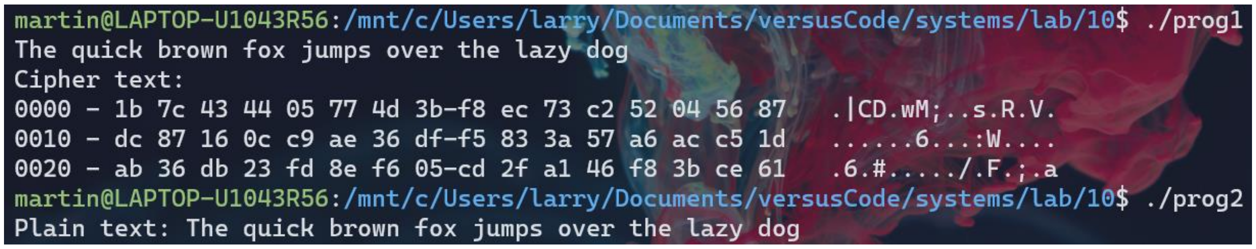
Lab 10 — OpenSSL Cryptography

Makefile



```
.PHONY : all clean
CFLAGS = -g -w -Wall

all : prog1 prog2

prog1 : prog1.o
	cc -o prog1 prog1.o -lcrypto

prog2 : prog2.o
	cc -o prog2 prog2.o -lcrypto

clean :
	rm prog1 prog2 *.o
```

Output



```
martin@LAPTOP-U1043R56:/mnt/c/Users/larry/Documents/versusCode/systems/lab/10$ ./prog1
The quick brown fox jumps over the lazy dog
Cipher text:
0000 - 1b 7c 43 44 05 77 4d 3b-f8 ec 73 c2 52 04 56 87   .|CD.wM;..s.R.V.
0010 - dc 87 16 0c c9 ae 36 df-f5 83 3a 57 a6 ac c5 1d   ......6...:W....
0020 - ab 36 db 23 fd 8e f6 05-cd 2f a1 46 f8 3b ce 61   .6.#...../.F.;.a
martin@LAPTOP-U1043R56:/mnt/c/Users/larry/Documents/versusCode/systems/lab/10$ ./prog2
Plain text: The quick brown fox jumps over the lazy dog
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/random.h>
#include <stdio.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <string.h>
#include <unistd.h>

#define BLOCK 32

struct secretStruct {
	unsigned char key[BLOCK];
	unsigned char iv[BLOCK/2];
} secret;

void handleErrors() {
	ERR_print_errors_fp(stderr);
	abort();
}

void make_key() {
	int fout;
	ssize_t ret;

	fout = open("secret", O_WRONLY | O_CREAT | O_TRUNC, 0600);

	ret = getrandom(&secret.key, BLOCK, 0);
	if(ret != BLOCK) {
		printf("random key generation failed\n");
		abort();
	}

	ret = getrandom(&secret.iv, BLOCK/2, 0);
	if(ret != BLOCK/2) {
		printf("intialization vector generation failed\n");
		abort();
	}

	write(fout, &secret, sizeof(secret));
	close(fout);
}

int encrypt(unsigned char *plaintext, int length, unsigned char *ciphertext) {
	int len;
	int ciphertext_len;
	EVP_CIPHER_CTX *ctx;

	if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();
	if(1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, secret.key, 
secret.iv)) handleErrors();

```

```c
        if(1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, length))
handleErrors();
        ciphertext_len = len;

        if(1 != EVP_EncryptFinal_ex(ctx, ciphertext+len, &len)) handleErrors();
        ciphertext_len += len;

        EVP_CIPHER_CTX_free(ctx);
        return(ciphertext_len);
}

int main(int argc, char **argv) {
        unsigned char plain[BUFSIZ];
        unsigned char *cipher;

        int len;
        int n;
        int fout;

        make_key();

        bzero(&plain, BUFSIZ);
        fgets((char*) plain, BUFSIZ, stdin);

        n = strlen((char*) plain);
        plain[n - 1] = 0;

        n = (n/BLOCK + 1)*BLOCK;
        cipher = (unsigned char *) malloc(n);
        len = encrypt((unsigned char*) &plain, strlen((char*) plain)+1, cipher);

        printf("Cipher text: \n");
        BIO_dump_fp(stdout, (const char*) cipher, len);
        fout = open("message", O_WRONLY | O_CREAT | O_TRUNC, 0644);
        n = write(fout, cipher, len);

        close(fout);
}
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/random.h>
#include <stdio.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <string.h>
#include <unistd.h>

#define BLOCK 32

 struct secretStruct {
       unsigned char key[BLOCK];
       unsigned char iv[BLOCK/2];
} secret;

void handleErrors() {
       ERR_print_errors_fp(stderr);
       abort();
}

void read_key() {
       int fin;

       fin = open("secret", O_RDONLY, 0600);
       read(fin, &secret, sizeof(secret));
       close(fin);

}

int decode(unsigned char *cipher, int length, unsigned char *plain) {
       EVP_CIPHER_CTX *ctx;
       int len;
       int plaintext_len;

       if(!(ctx = EVP_CIPHER_CTX_new())) handleErrors();
       if(1 != EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, secret.key,
secret.iv)) handleErrors();

       if(1 != EVP_DecryptUpdate(ctx, plain, &len, cipher, length)) handleErrors();
       plaintext_len = len;

       if(1 != EVP_DecryptFinal_ex(ctx, plain+len, &len)) handleErrors();
       plaintext_len += len;

       EVP_CIPHER_CTX_free(ctx);
       return(plaintext_len);
}

int main(int argc, char ** argv) {
       unsigned char buffer[BUFSIZ];
       unsigned char *plain;

```

```c
        int len;
        int n;
        int fin;

        read_key();

        fin = open("message", O_RDONLY, 0644);
        n = read(fin, buffer, BUFSIZ);
        close(fin);

        plain = (unsigned char *) malloc(n);
        len = decode(buffer, n, plain);

        printf("Plain text: %s\n", plain);
}
```