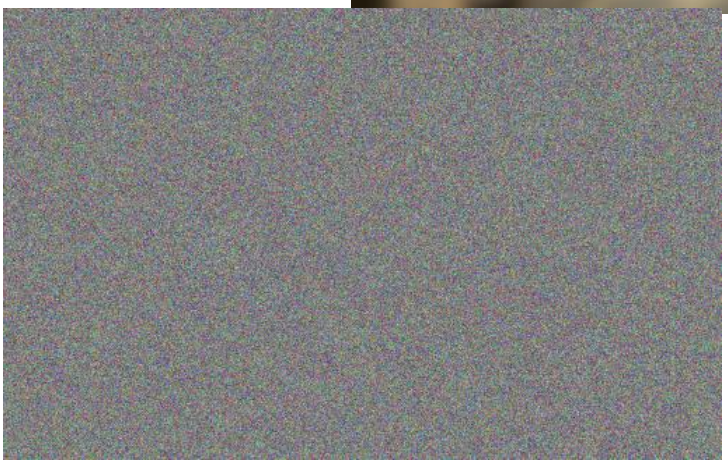
Sprawozdanie Lab3

Szyfrowanie

Grupa: **Środa 8:00**

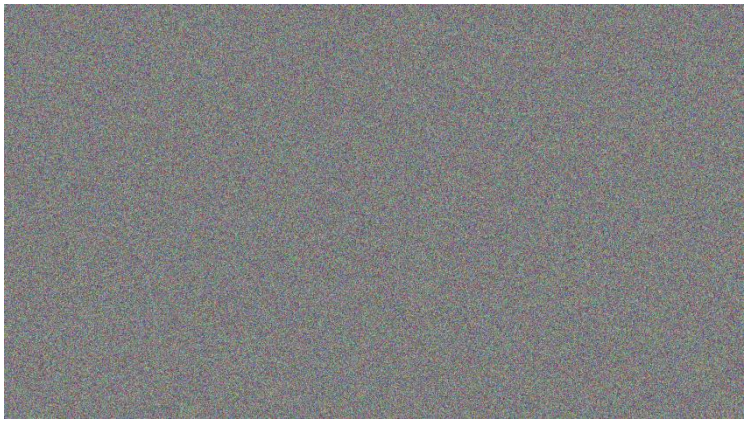
2. Wyświetl zawartość zaszyfrowanych plików. Umieść je w sprawozdaniu. Omów uzyskane wyniki. Uzasadnij zauważone charakterystyczne cechy poszczególnych wyników. Czy we wszystkich szyfrogramach są takie same i dlaczego? Przeanalizuj swoje obserwacje.

[10/28/2015 09:02] student@vhost1:~/a$ *openssl enc -aes-128-cbc -e -in drzewo1_small.bmp -out drzewo_cbc.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:05] student@vhost1:~/a$ *openssl enc -aes-128-ecb -e -in drzewo1_small.bmp -out drzewo_ecb.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:05] student@vhost1:~/a$ *openssl enc -aes-128-ecb -e -in Jednolity_z_textem.bmp -out tekst_ecb.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:07] student@vhost1:~/a$ *openssl enc -aes-128-cbc -e -in Jednolity_z_textem.bmp -out tekst_cbc.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:07] student@vhost1:~/a$ *openssl enc -aes-128-cbc -e -in Chest.bmp -out chest_cbc.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:08] student@vhost1:~/a$ *openssl enc -aes-128-ecb -e -in Chest.bmp -out chest_ecb.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:08] student@vhost1:~/a$ *openssl enc -aes-128-ecb -e -in kitchen.bmp -out kitchen_ecb.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*
[10/28/2015 09:08] student@vhost1:~/a$ *openssl enc -aes-128-cbc -e -in kitchen.bmp -out kitchen_cbc.bmp -k 0000112233445566778899aabbcc -iv 01020304050607*

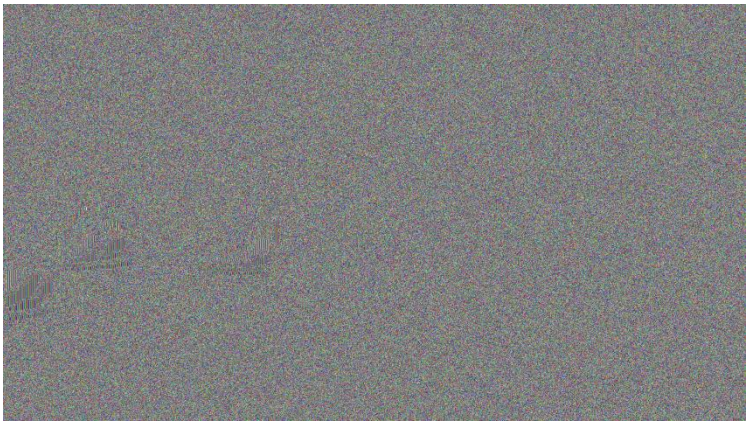


chest_cbc.bmp

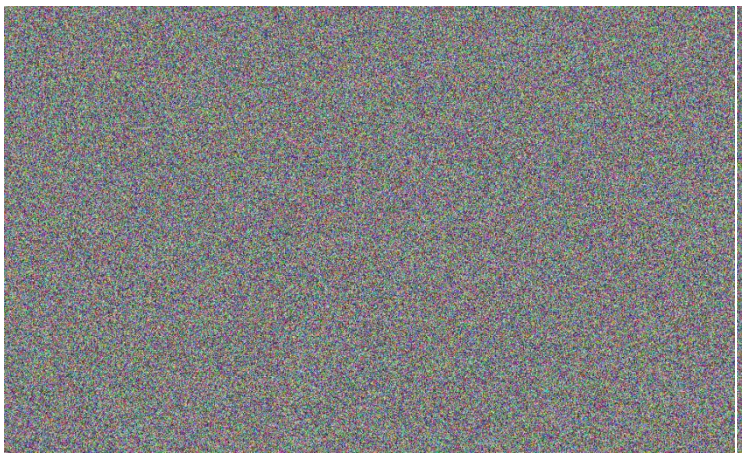
chest_ecb.bmp



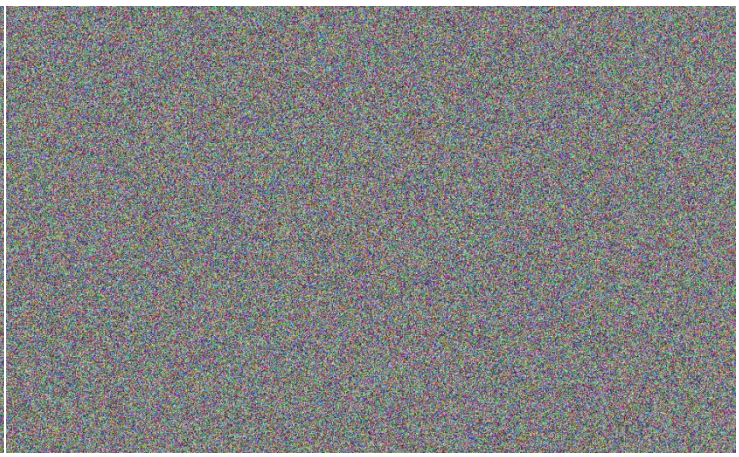
kitchen_cbc.bmp



kitchen_ecb.bmp

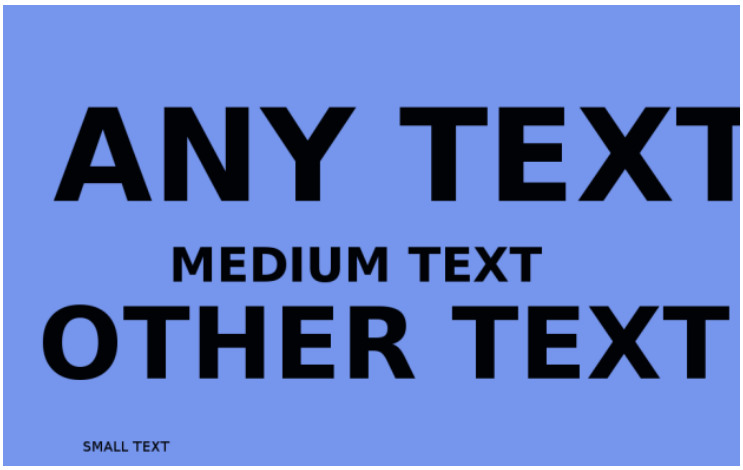


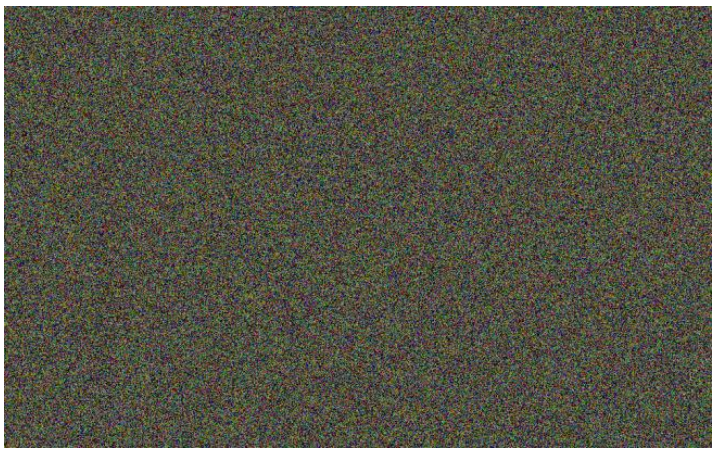
drzewo_cbc.bmp



drzewo_ecb.bmp

W przypadku powyższych przykładów nie widać żadnych znaków szczególnych w przypadku szyfrowania w trybie CBC oraz ECB.





Jtekst_cbc.bmp

Jtekst_ecb.bmp

Natomiast obraz o jednolitym kolorze i jednokolorowym tekscie w trybie ECB da się w miarę odczytać tekst (krórego szczegóły są większe od bloku), natomiast w cbc jest bez zmian.
Jest to spowodowane tym, że ECB uwzględnia tylko jeden blok podczas szyfrowania, a inne bloki nie maja na niego wpływu.

Odpowiedz na pytania: 1.) Ile danych jesteś w stanie odczytać po odszyfrowaniu uszkodzonych plików zaszyfrowanych algorytmem szyfrującym pracującym w trybie ECB, CBC, CFB, oraz OFB? Opisz jakiego algorytmu użyłeś. 2.) Wyjaśnij uzyskane wyniki 3.) Jakie wnioski można wyciągnąć z przeprowadzonego badania?

```
[11/04/2015 09:02] student@vhost1:~/a$ openssl enc -aes-128-ecb -e -in asd.txt -out asd-ecb.bin -K 00000000129978012345678912312 3 -iv 0121201
01222212312
[11/04/2015 09:02] student@vhost1:~/a$ openssl enc -aes-128-cbc -e -in asd.txt -out asd-cbc.bin -K 00000000129978012345678912312 3 -iv 0121201
01222212312
[11/04/2015 09:02] student@vhost1:~/a$ openssl enc -aes-128-cfb -e -in asd.txt -out asd-cfb.bin -K 00000000129978012345678912312 3 -iv 0121201
01222212312
[11/04/2015 09:03] student@vhost1:~/a$ openssl enc -aes-128-ofb -e -in asd.txt -out asd-ofb.bin -K 00000000129978012345678912312 3 -iv 0121201
01222212312
[11/04/2015 09:03] student@vhost1:~/a$ openssl enc -aes-128-ofb -d -in asd-ofb.bin -out asd-ofb.txt -K 00000000129978012345678912312 3 -iv 012
201201222212312
[11/04/2015 09:05] student@vhost1:~/a$ openssl enc -aes-128-cfb -d -in asd-cfb.bin -out asd-cfb.txt -K 00000000129978012345678912312 3 -iv 012
201201222212312
[11/04/2015 09:05] student@vhost1:~/a$ openssl enc -aes-128-ecb -d -in asd-ecb.bin -out asd-ecb.txt -K 00000000129978012345678912312 3 -iv 012
201201222212312
[11/04/2015 09:06] student@vhost1:~/a$ openssl enc -aes-128-cbc -d -in asd-cbc.bin -out asd-cbc.txt -K 00000000129978012345678912312 3 -iv 012
201201222212312
[11/04/2015 09:06] student@vhost1:~/a$
```

ddsdsdsds
test test test test
xxxxxxxxxxx
asdasfaerteasddddddd

Plik oryginalny

Ň˙đ¿?;Ů»SIâŕ€šest$test test
xxxxxxxxxxx
asdasfaerteasddddddd

CBC

pŁźËÓ„©śÖuKyę©öCest test test
xxxxxxxxxxx
asdasfaerteasddddddd

ECB

dds`sdsds
test test test test
xxxxxxxxxxx
asdasfaerteasddddddd

OFB

dds`sdsds
test
tů«@ŹđJ‡…WPśCĐRě——————————————x
xxxxxxxxx
asdasfaerteasddddddd

CFB

Najlepiej poradził sobie tryb OFB w którym tylko jedna litera została podmieniona. W pozostałych testach zamiana jednego bitu spośród 512b spowodowała, że więcej danych jest zapisana niepoprawnie.
W ECB cały pierwszy blok jest uszkodzony.
W CBC pierwszy blok oraz *lekko* następny.
W CFB pierwszy blok jest *lekko* uszkodzony natomiast następny jest kompletnie uszkodzony.

W CBC oraz CFB występuje propagacja błędów, a w OFB oraz ECB nie.

1. Manual `openssl` mówi, że `openssl` używa standardu PKCS5 do uzupełniania bloków. Opracuj eksperyment pozwalający na zweryfikowanie tego stwierdzenia. W szczególności sprawdź działanie algorytmu dla algorytmu AES, szyfrującego dane o rozmiarze 20 oraz 32 bajtów. W sprawozdaniu umieść zrzut ekranu dokumentujący wyniki oraz wyjaśnij dlaczego rozmiary plików po zaszyfrowaniu są takie a nie inne.

2. Użyj trybów ECB, CBC, CFB, oraz OFB do zaszyfrowania plików. w tym przypadku możesz użyć dowolnego algorytmu. Określ które z trybów wymagają uzupełniania bloków. W sprawozdaniu umieść zrzut ekranu prezentujący wyniki badania oraz wyjaśnij dlaczego niektóre z trybów pracy algorytmów szyfrujących nie wymagają uzupełnienia, a także określ co z tego wynika.

```
1/04/2015 09:44] student@vhost1:~/a$ openssl enc -aes-128-ecb -e -in u32.txt -out a32ecb.bin -K 00000000129978012345678912312 -iv 012120120
22212312
1/04/2015 09:45] student@vhost1:~/a$ openssl enc -aes-128-ofb -e -in u32.txt -out a32ofb.bin -K 00000000129978012345678912312 -iv 012120120
22212312
1/04/2015 09:45] student@vhost1:~/a$ openssl enc -aes-128-cfb -e -in u32.txt -out a32cfb.bin -K 00000000129978012345678912312 -iv 012120120
22212312
```

```
-rwxrwx--- 1 student student 32 Nov  4 09:42 a20cbc.bin
-rwxrwx--- 1 student student 20 Nov  4 09:44 a20cfb.bin
-rwxrwx--- 1 student student 32 Nov  4 09:42 a20ecb.bin
-rwxrwx--- 1 student student 20 Nov  4 09:44 a20ofb.bin
-rwxrwx--- 1 student student 48 Nov  4 09:44 a32cbc.bin
-rwxrwx--- 1 student student 32 Nov  4 09:46 a32cfb.bin
-rwxrwx--- 1 student student 48 Nov  4 09:45 a32ecb.bin
-rwxrwx--- 1 student student 32 Nov  4 09:45 a32ofb.bin
-rwxrwx--- 1 student student 20 Nov  4 09:40 u20.txt
-rwxrwx--- 1 student student 32 Nov  4 09:40 u32.txt
```

CBC oraz ECB wymagają uzupełnienia(do pełnej ilości lub o dodatkowy blok) ponieważ są typu blokowego, a nie strumieniowego i wymagają dopełniania do pełnego bloku, lub bloku dodatkowego.

W tryby bez uzupełnienia lepiej nadają się np. w komunikatorach internetowych ponieważ nie wymagają nadmiarowych danych, lub do szyfrowania wielu małych plików.

% cat /proc/sys/kernel/random/entropy_avai

Proszę kilkukrotnie wywołać powyzsze polecenie i zaobserwować uzyskane wartości. Następnie proszę poruszać myszą, po czym proszę wykonać kolejną serię odczytów entropii. Podobne doświadczenie porszę wykonać po użyciu klawiatury do przepisania fragmentu tekstu. Prosze również sprawdzić jak wpływają na entropię operacje dyskowe. Można użyć polecenia *cat /dev/zero >test* , które będzie w sposób ciągły zapisywało zera do pliku generując operacje dyskowe. Uwaga! Prosze usunąć utworzony plik! Wnioski i obserwacje z niniejszego ćwiczenia zamieść w sprawozdaniu.

Ciągłe wywoływanie polecenia powoduje entropię na poziomie ~100-200.

```
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
145
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
163
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
180
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
133
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
150
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
169
[11/04/2015 14:24] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
188
```

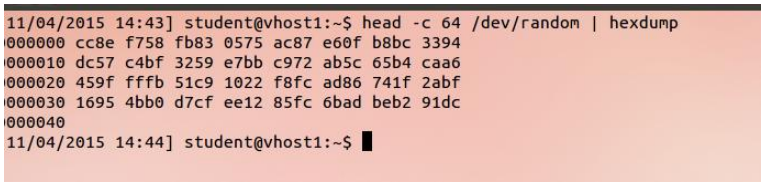
Natomiast interakcja *z systemem powoduje jej zwiększenie, a samo wywołanie polecenia zmniejsza jej ilość.*

```
[11/04/2015 14:27] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
129
[11/04/2015 14:27] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
1702
[11/04/2015 14:27] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
1465
[11/04/2015 14:27] student@vhost1:~$ cat /proc/sys/kernel/random/entropy_avail
```

Dzieje się tak ponieważ, system zbiera dane z losowych czynników tj. zewnętrznych, a zużywa je do procesów jak np. nadanie losowego id procesowi, który sprawdzi wartość entropii.

```
% head -c 16 /dev/random | hexdump
```

Uruchom powyższe polecenie kilka razy. Zaobserwuj, że w pewnym momencie przestanie odpowiadać bezzwłocznie. Konieczne będzie oczekiwanie na wynik polecenia. Zaobserwuj co się stanie jeśli podczas oczekiwania nie będizesz wykonywać żadnych czynności w systemie, co jeśli będziesz poruszać myszą, używać klawiatury, wykonywać operacje dyskowe. Obserwacje i wnioski z przeprowadzonego badania umieść w sprawozdaniu.



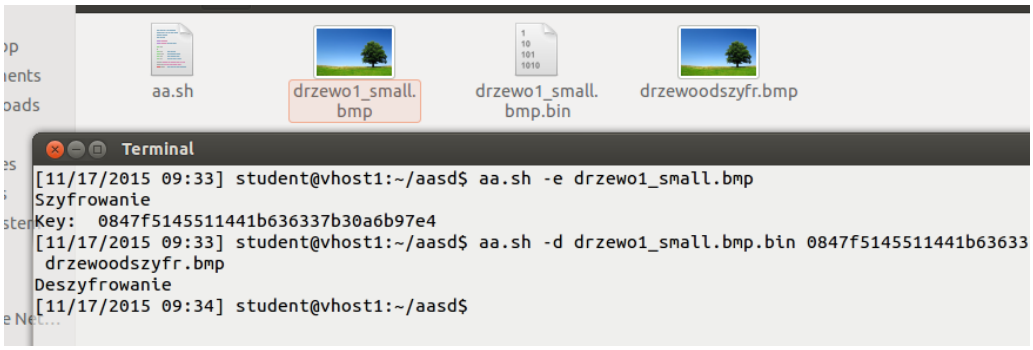
Jeśli nie będziemy wykonywać żadnych interakcji z systemem, to program będzie 'wisiał' i będzie czekać, aż *wygenerują się* *(tj. zostaną wygenerowane na podstawie źródeł losowych)* nowe dane losowe.

Opracuj program szyfrujący pliki wskazane przez użytkownika wybranym algorytmem współczesnym. Użyj entropii systemu jako źródła danych do wygenerowania klucza kryptograficznego. Opisz działanie poszczególnych fragmentów programu, Zaprezentuj efekty jego działania.

Program (˘ɜ˘):

```
1   #!/bin/bash
2   if [ "$1" == "-e" ]; then                                   //Szyfrowanie
3           echo 'Szyfrowanie'
4           if [ "$2" == "" ]; then                             //sprawdzam czy użytkownik wpisał nazwę
5                   echo "wpisz nazwe pliku"
6           else                                                //jeśli ją wpisał
7           a=$(cat /dev/urandom | tr -cd 'a-f0-9' | head -c 32) //używam /dev/random do wygenerowania 32
8           echo 'Key: ' $a                                     znakowego key'a i go podaje użytkownikowi
9           openssl enc -aes-128-cbc -e -in $2 -out $2.bin -k $a //używam openssl z algorytmem AES-128-CBC, jako
10          fi                                                  argumenty podaję nazwe pliku oraz wygenerowany
11  else                                                        key, a jako output ustawim nazwe pliku i dodaje „.bin"
12          if [ "$1" == "-d" ]; then
13                  echo 'Deszyfrowanie'                        //Deszyfrowanie
14                  if [ "$4" == "" ]; then                     //jeśli nie ma wybranego outputu, to ucinam ostatnie
15                          arg2=$(echo $2 | sed "s/....$//g")  4 znaki z nazwy pliku
16                          echo arg2
17                  else
18                          arg2=$4
19                  fi                                          //wypisuje użytkownikowi jak będzie nazywać się plik
20                  echo "Output: " $arg2                       wyjsciowy
21                  openssl enc -aes-128-cbc -d -in $2 -out $arg2 -k //deszyfrowanie analogiczne do szyfrowania
22  $3
23          else                                                //Jeśli „Program" uruchomiono bez argumentów to
24                  echo "Info:"                                wypisuje instrukcję
25                  echo "-e [plik]- szyfowanie"
26                  echo "-d [plik] [key] +[output] - deszyfrowanie"
27          fi
28  fi
```

Przykład działania:



NOTE: Pytania w PDF'ie nie nadają się do bezpośredniego skopiowania, dlatego są w formie zrzutu.