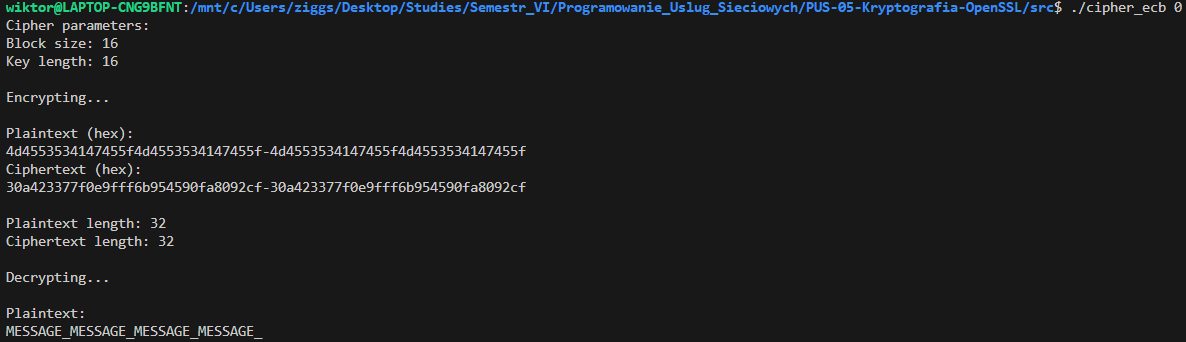
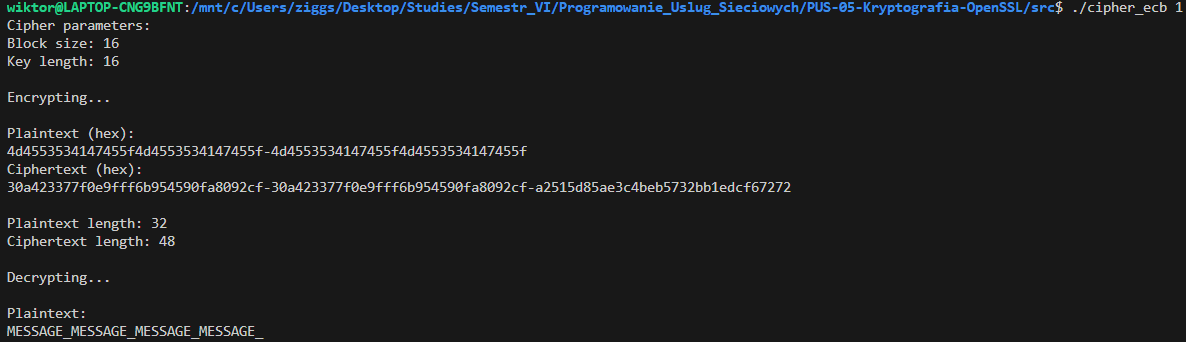
**Kryptografia w OpenSSL  
Laboratorium 5  
Wiktor Zmiendak**

**ZADANIE 1**

Program wypisuje w formie heksadecymalnej tekst jawny i szyfrogram, oddzielając wizualnie bloki o rozmiarze zgodnym z rozmiarem bloku szyfru. Wypisuje również obliczone długości obu ciągów.

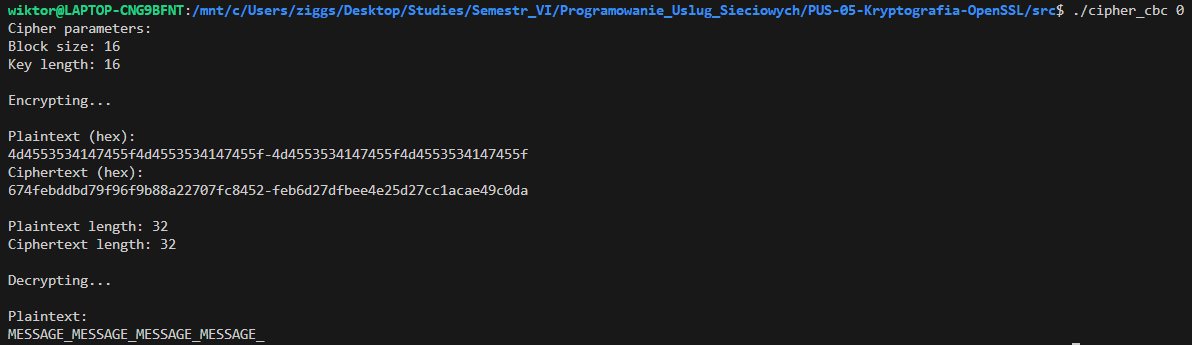




Gdy padding = 0 (brak dopełnienia), długość tekstu jawnego *musi* być wielokrotnością rozmiaru bloku szyfru (16 bajtów dla AES). W tym programie tekst jawny ma 32 bajty, co jest wielokrotnością 16. Dlatego szyfrogram będzie miał dokładnie taką samą długość jak tekst jawny, czyli 32 bajty. Gdy padding = 1 (dopełnienie PKCS włączone), dane są uzupełniane tak, aby ich długość była wielokrotnością rozmiaru bloku. Jeśli długość tekstu jawnego jest już wielokrotnością rozmiaru bloku (jak tutaj 32 bajty), dodawany jest cały dodatkowy blok dopełnienia (16 bajtów). Dlatego szyfrogram będzie miał długość 32 + 16 = 48 bajtów.

W trybie ECB każdy blok tekstu jawnego jest szyfrowany niezależnie od innych bloków, przy użyciu tego samego klucza. Oznacza to, że jeśli w tekście jawnym występują dwa identyczne bloki, to po zaszyfrowaniu odpowiadające im bloki szyfrogramu również będą identyczne.

**ZADANIE 2**

****

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

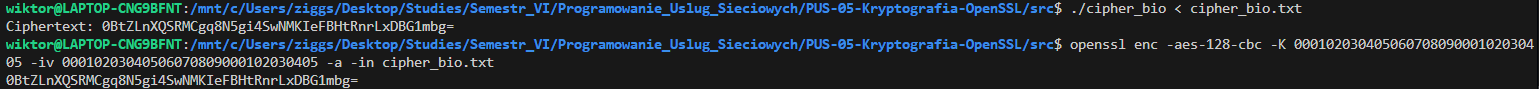
Tak samo jak w trybie ECB. Mechanizm dopełniania PKCS i jego wpływ na końcowy rozmiar szyfrogramu zależy wyłącznie od długości tekstu jawnego i rozmiaru bloku szyfru (16 bajtów), a nie zależy od trybu operacyjnego (ECB czy CBC). Dla tekstu jawnego o długości 32 bajtów:

* Przy padding = 0 (brak dopełnienia): rozmiar szyfrogramu = 32 bajty.
* Przy padding = 1 (dopełnienie PKCS): rozmiar szyfrogramu = 32 + 16 (pełny blok dopełnienia) = 48 bajtów.

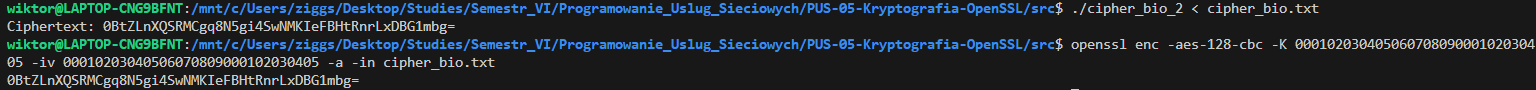
Podczas szyfrowania, każdy blok tekstu jawnego jest najpierw XORowany z poprzednim blokiem szyfrogramu, a dopiero potem szyfrowany kluczem. To powoduje, że każdy blok szyfrogramu zależy od odpowiadającego mu bloku tekstu jawnego oraz wszystkich poprzednich bloków tekstu jawnego. Dzięki temu identyczne bloki tekstu jawnego dadzą w wyniku różne bloki szyfrogramu. Podczas deszyfrowania, każdy blok szyfrogramu jest najpierw deszyfrowany kluczem, a następnie wynik jest XORowany z poprzednim blokiem szyfrogramu, aby odzyskać tekst jawny.

**ZADANIE 3**

Program wykorzystuje abstrakcyjny interfejs BIO (Basic Input/Output) biblioteki OpenSSL do przetwarzania danych w potoku.



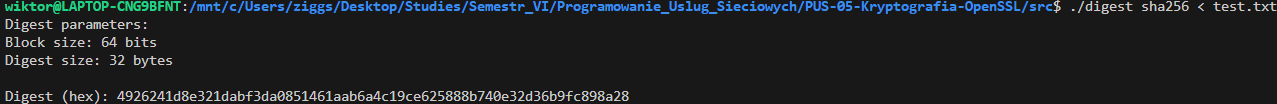
Jak widać wynik działania programu jest identyczny z wynikiem polecenia openssl enc -aes-128-cbc -K <key\_hex> -iv <iv\_hex> -a -in plaintext



Po modyfikacji kodu dokładając pętlę for przechodzącą po każdym znaku w pliku widzimy że wynik końcowy (zaszyfrowany i zakodowany Base64 ciąg na standardowym wyjściu) jest identyczny co w poprzednim przypadku.

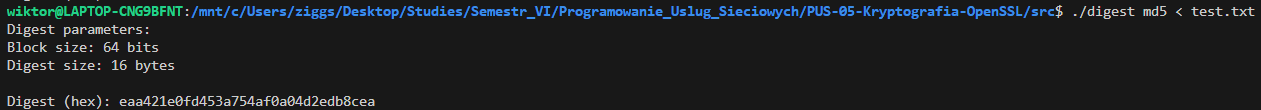
**ZADANIE 4**

Program oblicza skrót kryptograficzny wiadomości podanej na standardowym wejściu, używając algorytmu skrótu, którego nazwa jest podana jako argument linii poleceń.

****

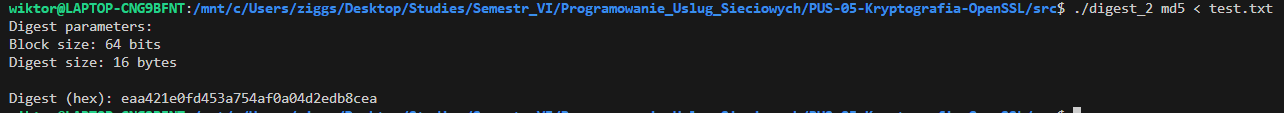
****

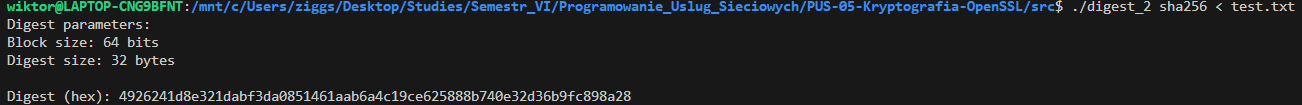
Wynikowy skrót jest zgodny z obliczeniami narzędzia openssl dgst dla tych samych danych i algorytmu. W tym przypadku jest to algorytm SHA256.

****

****

A tutaj przykład innego algorytmu (MD5) który również daje prawidłowe wyniki.

****

****

Po modyfikacji programu tak aby algorytm przechodził w pętli znak po znaku również widzimy te same wyniki dla obydwu algorytmów.

**ZADANIE 5**

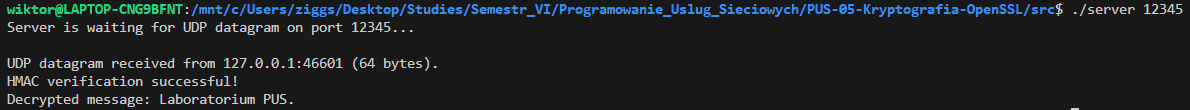
**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

Przykład wywołania programu przy pomocy dwóch różnych algorytmów

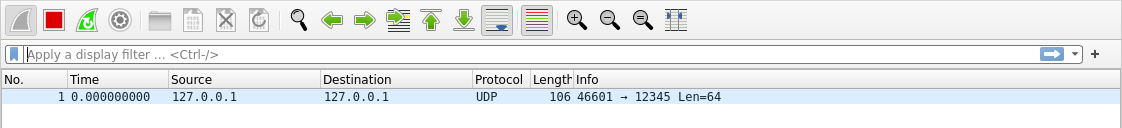
Główna zmiana polegała na zastąpieniu mechanizmu obliczania skrótu opartego na kontekście (EVP\_MD\_CTX, EVP\_DigestInit\_ex, Update, Final\_ex) pojedynczym wywołaniem funkcji HMAC(). Dodano również definicję klucza key i jego długości key\_len, które są niezbędne do obliczenia HMAC.

**ZADANIE 6**

****

****

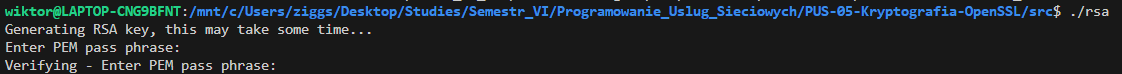
Po połączeniu clienta do serwera pojawia się komunikat o otrzymaniu datagramu, następnie "HMAC verification successful!" i na końcu "Decrypted message: Laboratorium PUS.".

****

W wiresharku możemy zobaczyć pojedynczy pakiet UDP wysłany z klienta do serwera. W jego zawartości są dane binarne.

Sniffer pokazuje jedynie zawartość datagramu UDP, która składa się z zaszyfrowanej wiadomości oraz kodu HMAC. Bez znajomości tajnych kluczy nie można ani odczytać oryginalnej wiadomości "Laboratorium PUS.", ani potwierdzić jej autentyczności na podstawie przechwyconych danych.

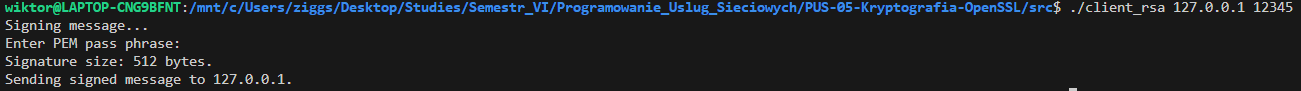
**ZADANIE 7**



Generujemy kluczę i ustawiamy hasło



Serwer oczekuje informacji od klienta



Uruchamiamy klienta i wpisujemy ustawione hasło w celu wczytania zaszyfrowanego pliku. Po pomyślnej autoryzacji klient generuje podpis i wyświetla komunikat o jego rozmiarze