**Zadanie 1**

Zaimplementuj funkcję szyfrowania i deszyfrowania algorytmu AES, korzystając z co najmniej 1 biblioteki kryptograficznych dla wybranego przez Ciebie języka/języków:

* Przypadek 1: z użyciem ECB, padding PKCS#5
* Przypadek 2: z użyciem CBC, padding PKCS#5
* Przypadek 3: z użyciem CTR

We wszystkich przypadkach 16 bajtowy wektor inicjujący (IV) oraz klucz mają być wybierane losowo. Implementację umieść na swoim repozytorium.

Poniżej znajdziesz 3 szyfrogramy (zakodowane w formacie HEX) i klucze do ich odszyfrowania. Używając swojej implementacji odszyfruj je i umieść rozwiązanie w swoim repozytorium (w pliku txt).

Zaszyfruj dowolny ciąg swoim skryptem we wszystkich zaimplententowanych wersjach algorytmu  i umieść wraz z kluczami i wektorami inicjującymi w repozytorium.

**Dodatkowo odpowiedz na pytania** (odpowiedzi umieść w repo):

1. Jeśli dostępne są źródła wybranych przez Ciebie bibliotek, to znajdź w nich implementacje, których używałeś i porównaj je. Jeśli źródło nie jest dostępne to zastanów się jak mógłbyś porównać działanie zawartych w nich algorytmów.
2. Jak można chronić kod bibliotek i innych implementacji, tak, żeby był odporny na inżynierię wsteczną?
3. Która biblioteka działa szybciej? Dlaczego? Jak to porównałeś?
4. Na jakiej stronie znalazłeś informacje odnośnie podatności w bibliotekach kryptograficznych, z których skorzystałeś?
5. W przypadku użycia klucza o długości większej niż 128 bitów dla biblioteki javax.crypto pojawia się wyjątek: java.security.InvalidKeyException: Illegal key size. Z czego wynika?
6. Które z trybów wiązania bloków w algorytmie AES są bezpieczne, a które nie?

**Info:**  Do implementacji AES użyj bibliotek w wersjach nieposiadających publicznie znanych podatności.

Poniżej znajdziesz również funkcję paddingu, użytą w procesie szyfrowania. Korzystając z innej biblioteki może (ale nie musi) pojawić się błąd typu:

|  |
| --- |
| Given final block not properly padded |

Spróbuj wtedy odszyfrować ciąg w trybie NoPadding, padding=none, etc.

|  |
| --- |
| Python padding: |
| pad = lambda s: s + (BS - len(s) % BS) \* chr(BS - len(s) % BS)  unpad = lambda s: s[0:-ord(s[-1])] |

|  |
| --- |
| KeyECB:  32461636d541e5e8dc97e5c2c51acfd7  MessageECB:   |
| KeyCBC:  1a57247fb590dabbb09682ee2ee394c8  IV:  e7f0a472369951ce7af3d26903da07ed  MessageCBC:   |
| KeyCTR:  fb202e894b38838b29f878d02e0df633  IV:  43b51aeb42d826e71726de23b7384077  MessageCRT:   |

**Zadanie 2**

Zaimplementuj skrypt obliczający logarytm dyskretny (log mod p) korzystając z algorytmu *meet-in-the-middle*. Niech g będzie elementem w Zp\* i dane jest h należące do Zp\*, takie, że h=gx, gdzie 1x240. Twoim zadaniem jest znalezienie x.

Skrypt jako dane wejściowe ma przyjmować p,g,h a ma zwracać x.

|  |
| --- |
| p=13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084171 |
| g=7986892892644964809678573127386712039532852301584169807905118050675340956203921770492145296166905084126855959303199420887184894225814618909287510943703491 |
| h=10353923830460929735328496784361539060361238653685678824483529175983950553832878320450730127803739628395851941601331144145295269915794187037493934868513335 |

**Info:** Do rozwiązania zadania najlepiej użyć środowiska wspierającego arytmetykę modularną i obliczenia na liczbach o zwiększonej precyzji.

* Java: BigInteger (mod, modPow,  modInverse)
* Python: gmpy2, numbthy
* C: GMP

**Odpowiedz na pytanie:**

Jak inaczej można obliczyć logarytm dyskretny? Czy Twoja metoda jest lepsza od algorytmu *meet-in-the-middle*? Dlaczego?

Zadanie 2  
Do obliczania logarytmy istnieje metoda Baby-Step-Giant-Step jednak po zaimplementowani go w pythonie po 10 godzinach obliczeń algorytm nie uzyskał wyniku. Co sugeruje ze algorytm meet In the Middle jest lepszy.