Kódolás és IT biztonság
(VIHIBB01)
Laboratóriumi gyakorlat

Kriptográfiai programkönyvtárak használata

Szerző:
Buttyán Levente

2020. november 6.

Tartalomjegyzék

1. A gyakorlat célja **2**

2. Elméleti háttér **2**

3. PyCryptodome **7**

4. Feladatok **8**

4.1. Feladat (vezetett) 9
4.2. Feladat 11
4.3. Feladat 11
4.4. Feladat 13

1. A gyakorlat célja

Ebben a laborgyakorlatban egy valódi kriptográfiai programkönyvtárat és API-t fogunk használni egyszerű, parancssoros alkalmazások készítéséhez, melyek kriptográfiai műveleteket végeznek. A gyakorlat célja, hogy kézzel fogható tapasztalatot szerezzetek a kriptográfiai programkönyvtárak használatában, valamint hogy élőben is lássátok azon kriptográfiai algoritmusok (legalább egy részhalmazának) működését, gyakorlati alkalmazását, melyekről az előadásokon szó volt. Konkrétan, ebben a gyakorlatban a PyCryptodome[1] kriptográfiai programkönyvtárat fogjuk használni és Python nyelven készítünk alkalmazásokat. Ezek egyike az AES blokkrejtjelezőt fogja használni CBC módban, a másik pedig az RSA nyilvános kulcsú rejtjelezőt OAEP formázással, valamint az RSA digitális aláírást PSS módban. Használni fogjuk még az SHA-256 hash függvényt és a PBKDF2 jelszó alapú kulcsderiváló függvényt, melyekről szintén volt szó az előadásokon. Bár nem minden kriptográfiai primitívet fogunk kipróbálni, a gyakorlat elvégzése után mégis elég jó képetek lesz arról, hogy hogyan kell a PyCryptodome programkönyvtárat használni, és képesek lesztek, a dokumentáció segítségével, az abban implementált többi algoritmus használatára is. Továbbá, mivel a különböző kriptográfiai programkönyvtárak és API-k hasonlítanak egymásra (pl. hasonló absztrakciókat használnak), ezért a gyakorlat elvégzése után könnyen meg tudjátok tanulni majd más kriptográfiai programkönyvtárak használatát is.

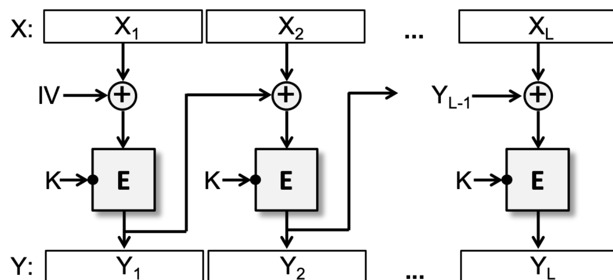
2. Elméleti háttér

Kriptográfiai algoritmusok segítségével olyan információ-biztonsági feladatokat tudunk megoldani, mint az adatok bizalmasságának megőrzése vagy az adatok integritásának védelme. Az alapvető kriptográfiai algoritmusokat kriptográfiai primitíveknek is hívjuk. Ezek a következők: szimmetrikus és aszimmetrikus kulcsú rejtjelezők; kriptográfiai hash függvények; üzenethitelesítő kódok, vagy MAC függvények; és digitális aláírás sémák. Ebben a gyakorlatban egy szimmetrikus kulcsú rejtjelezőt (AES), egy aszimmetrikus kulcsú rejtjelezőt (RSA), egy hash függvényt (SHA-256), egy MAC függvényre épülő kulcs deriváló algoritmust(PBKDF2), és egy digitális aláírás sémát (RSA) fogunk használni.

[1] https://github.com/Legrandin/pycryptodome/

A blokkrejtjelezők, mint pl. az AES, egy több bájtból álló *blokkot* képesek egy lépésben rejtjelezni. A bemeni blokkot *nyílt szöveg blokknak*, a kimeneti blokkot *rejtett szöveg blokknak* hívjuk. A bementi és a kimeneti blokkok mérete azonos, ezt hívjuk a rejtjelező *blokkméretének*. Az AES blokkmérete pl. 16 bájt (128 bit). A bemeneti blokkon kívül, a blokkrejtjelezőknek van egy másik bemenete is: a *kulcs*. Ez szintén egy több bájtból álló vektor, mely tipikusan véletlen bájtokat tartalmaz. Az AES esetében a kulcs mérete változó, 16, 24, vagy 32 bájt (128, 192, vagy 256 bit) lehet.

Mivel a védeni kívánt adat mérete általában nagyobb, mint a blokkrejtjelező blokkmérete, ezért szükségünk van olyan módszerekre, melyekkel nagyobb mennyiségű adatot tudunk kódolni, illetve dekódolni, a blokkrejtjelezővel. Ezeket a módszereket *blokkrejtjelezési módoknak* hívjuk. Létezik néhány jól ismeret, szabványos blokkrejtjelezési mód, mint pl. a CBC (Cipher Block Chaining) mód, melynek működését (kódolás esetén) az 1. ábra szemlélteti.



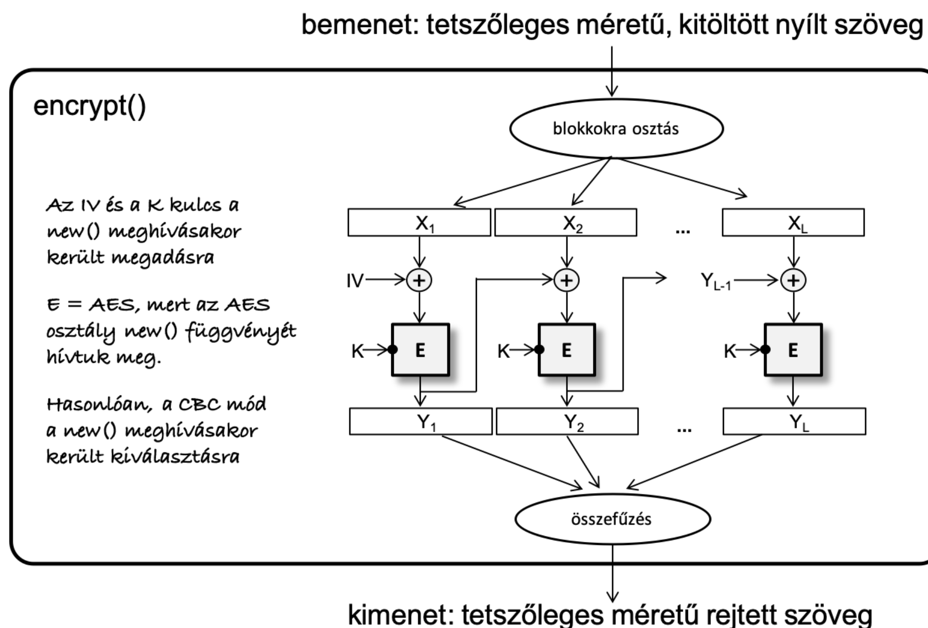
$$Y_i = E_K(X_i \oplus Y_{i-1})$$

1. ábra. A CBC blokkrejtjelezési mód működése (kódolás).

Ahogy az ábrán látható, a bementet blokkokra osztjuk, és a blokkokat egyesével dolgozzuk fel oly módon, hogy mielőtt egy nyílt blokkot rejtjelezünk a blokkrejtjelezővel, hozzá XOR-oljuk az előző rejtjeles blokkot. Az első blokk feldolgozásához szükségünk van egy speciális „előző” blokkra, amit IV-nek (Initial Vector) nevezünk. Az IV-vel szemben az a követelmény, hogy az legyen egy nem megjósolható véletlen blokk, ezért azt általában egy kriptográfiai véletlenszám generátorral állítjuk elő. Továbbá, mivel az IV-re

szükség van a dekódolásnál is, ezért azt el kell juttatni a dekódolást végző entitáshoz. Ezt tipikusan úgy oldjuk meg, hogy az IV-t a kódolt adathoz csatoljuk ECB (Electronic Codebook) módban rejtjelezett formában, ahol a rejtjelezéshez használt kulcs ugyanaz a kulcs amivel magát az adatot is rejtjelezzük (CBC módban).

Előfordulhat, hogy az adat mérete nem egész számú többszöröse a rejtjelező blokkméretének, ezért az adaton rejtjelezés előtt általában *kitöltést* (angolul padding) alkalmazunk, mely során oly módon fűzünk extra bájtokat az adat végéhez, hogy az így megnövelt méret a blokkméret többszöröse legyen. Léteznek jól ismert, szabványos kitöltési sémák, mint pl. a PKCS#7 padding. A dekodolás során a kitöltést fel kell ismerni és el kell távolítani, így visszanyerve az eredeti adatot.



2. ábra. A kripto könyvtárak API-ja általában elrejti a használt blokkrejtjelezési mód (pl. CBC) részleteit.

A kriptográfiai programkönyvtárak nem mindig rejtik el az IV generálást és a kitöltés alkalmazását az API absztrakciói mögé, ami azt jelenti, hogy ezeket a feladatokat gyakran a könyvtárat használó programozónak magának

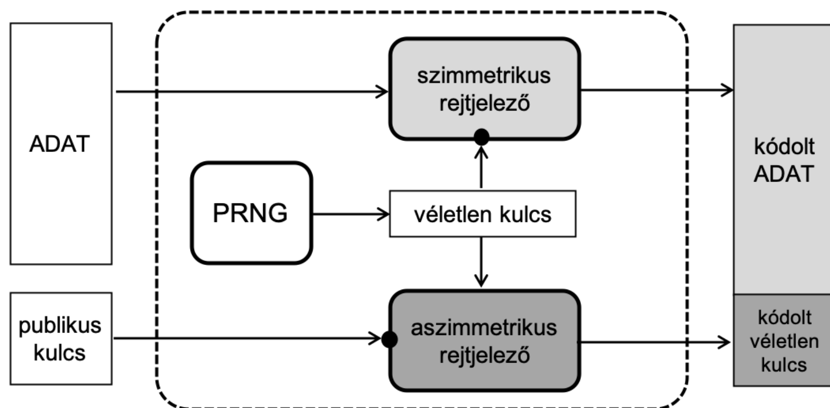
kell ellátnia. Az IV generálásához tipikusan egy (a kriptográfiai programkönyvtárban található) kriptográfiai véletlenszám generátort használhatunk, a kitöltés megoldásához pedig egy ismert padding sémát implementálhatunk. Miután ezeket a feladatokat megoldottuk, létrehozhatjuk a blokkrejtjelező egy példányát a megfelelő API hívás segítségével, és felparaméterezhetjük a blokkrejtjelezőnket az alkalmazni kívánt blokkrejtjelezési móddal (pl. CBC), a korábban generált IV-vel, és a rejtjelezéshez használni kívánt kulccsal. Miután ez megtörtént, a kitöltéssel ellátott nyílt bemeneti adaton meghívhatjuk a blokkrejtjelezőnk kódoló (pl. `encrypt()`) függvényét. A kriptográfiai programkönyvtár implementációja gondoskodik ezek után a bemenet blokkokra osztásáról, a blokkok kódolásáról, majd a rejtjeles blokkok összefűzéséből, és a kódoló függvény kimeneteként megkapjuk a rejtjelezett adatot. Ezt a színfalak mögött zajló folyamatot szemlélteti a 2. ábra CBC mód használata esetén. Ezután a programozónak kell gondoskodni az IV ECB módban történő rejtjelezéséről és a rejtjelezett IV csatolásáról a rejtjelezett adathoz. A dekódoló oldalon ezek a műveletek (vagy azok inverze) fordított sorrendben kerülnek végrehajtásra annak érdekében, hogy vissza tudjuk állítani az eredeti adatot.

Konkrét kulcs értékeket sosem szabad a programban megadni, mert ekkor bárki aki a programhoz (akár forráskód, akár bináris szinten) hozzáfér, hozzájuthat a kulcshoz is. A kulcsokat ezért vagy a futó programnak magának kell generálnia vagy kívülről kell azokat a futó programba juttatni (pl. megfelelően védett fájlból beolvasni). Generálhatunk kulcsokat egy felhasználó által begépelt jelszóból is, aminek megvannak a saját problémái, ezért körültekintően kell eljárni. Jelszóból mindig egy biztonságos jelszó alapú kulcsgeneráló függvénnyel kell a kulcsokat származtatni, nem pedig saját, alkalmi megoldásokat használni erre a célra (mint például a jelszó egyszerű hash-elése, ami nem biztonságos). Egy jól ismert jelszó alapú kulcsgeneráló függvény például a PBKDF2.

A konvencionális rejtjelezők, mint pl. az AES, szimmetrikus kulcsúak, ami azt jelenti, hogy a kódoláshoz és a dekódoláshoz ugyanazt a kulcsot kell használni. Vannak azonban aszimmetrikus kulcsú rejtjelezők is, mint pl. az RSA, ahol a kódoló és a dekódoló kulcs nem azonos. Ezeknél a rejtjelezőknél a kódoló és a dekódoló kulcs egy *kulcspárt* alkot, amit egy kulcspár generáló függvénnyel állítunk elő. A kódoló kulcs nyilvánosságra hozható, ezért szokták nyilvános kulcsnak is nevezni, a dekódoló kulcs azonban mindig titkos marad, ezért szokták privát kulcsnak is hívni. Az aszimmetrikus kulcsú rejtjelezők hasonlítanak a blokkrejtjelezőkhöz abban az értelemben,

hogy ezek is egy több bájtból álló blokkot rejtjeleznek egyszerre.

Az aszimmetrikus kulcsú rejtjelezők több nagyságrenddel is lassúbbak lehetnek, mint a szimmetrikus kulcsú rejtjelezők. Ezért tipikusan nem használjuk őket nagy méretű adatok rejtjelezésére. Ehelyett hibrid rejtjelezést alkalmazunk, ami azt jelenti, hogy az adatot egy szimmetrikus kulcsú rejtjelezővel (pl. AES-sel CBC módban) rejtjelezzük egy véletlen generált kulccsal, majd csak ezt a véletlen kulcsot rejtjelezzük az aszimmetrikus kulcsú rejtjelezővel (pl. RSA-val) az adatot később dekódolni szándékozó entitás publikus kulcsával. Végül a kódolt adatot és a kódolt véletlen kulcsot összefűzzük és együtt tároljuk vagy együtt juttatjuk el őket a dekódolás helyére. Ezt szemlélteti a 3. ábra. Dekódolásnál először a véletlen kulcsot dekódoljuk a privát kulcs segítségével az aszimmetrikus kulcsú dekódoló eljárást alkalmazva, majd ezzel a kulccsal dekódoljuk az adatot a szimmetrikus kulcsú dekódoló eljárással.



3. ábra. A hibrid rejtjelezés illusztrációja, ahol a szimmetrikus és az aszimmetrikus kulcsú rejtjelezést együtt használjuk a hatékonyabb (gyorsabb) kódolás megvalósítása érdekében.

A digitális aláírás egy olyan aszimmetrikus kulcsú kriptográfiai primitív, amivel az adatok integritását lehet megvédeni, valamint az adatok eredetének hitelességét lehet biztosítani. A digitális aláírás sémák is kulcspárokat használnak; ebben az esetben, a privát kulcsot használjuk az aláírás generálásához, és a publikus kulcsot használjuk az aláírás ellenőrzéséhez. Maga a digitális aláírás olyan mint egy ellenőrző összeg, amit az aláírás generáló függvénnyel számolunk ki, majd az adathoz csatoljuk. Ellenőrzés során az aláírás el-

lenőrző függvény segítségével ellenőrizzük az aláírás helyességét az adaton.

Az aszimmetrikus kulcsú rejtjelezőkhöz hasonlóan, a digitális aláírás sémák is lassúak, ezért a gyakorlatban nem írunk alá nagy adatokat közvetlenül, hanem az adatot először hash-eljük, majd az így kapott, sokkal kisebb méretű hash értéket írjuk alá. Ebben az esetben nagyon fontos, hogy a hash-elést egy ütközésellenálló hash függvénnyel végezzük, mint pl. a SHA-256.

A kulcspárokat, vagy azok elemeit gyakran tároljuk fájlokban. Ekkor fontos, hogy a privát kulcsot mindig rejtjelezett formában tároljuk, különben bárki aki hozzáfér a kulcsot tároló fájlhoz azonnal hozzáférne a titkos privát kulcshoz is, ami így kompromittálódna. A legtöbb kriptográfiai programkönyvtár támogat olyan kulcsmenedzsment funkciókat, melyek lehetővé teszik a kulcspárok, vagy azok elemeinek szabványos formában (pl. PEM vagy DER) történő exportálását, és az exportált privát kulcs jelszóval történő védelmét. Utóbbi azt jelenti, hogy az exportált privát kulcs a jelszóból generált kulccsal kerül rejtjelezésre.

3. PyCryptodome

A PyCryptodome egy Python nyelven írt kriptográfiai programkönyvtár, ami kriptográfiai algoritmusok implementációját tartalmazza, és számos hasznos kiegészítő funkciót támogat, mint pl. kitöltés (padding), kulcs export/import, jelszó alapú kulcsgenerálás, stb. A programkönyvtárat könnyen lehet telepíteni a következő paranccsal:

```
pip install pycryptodome
```

A PyCryptodome egy objektum orientált könyvtár, amiben a különböző kriptográfiai primitívek, mint pl. a rejtjelezők és a hash függvények, objektumként vannak reprezentálva, aminek vannak attribútumai és függvényei (metódusai). Egy primitív használatához a programozónak létre kell hoznia egy példányt a primitívet reprezentáló objektum osztályból, amit az adott osztály new() függvényével lehet megtenni (pl. cipher = AES.new(...)). Ezután a szokásos módon lehet elérni a létrehozott objektum példány attribútumait és meghívni függvényeit (pl. cipher.encrypt(...)). A new() függvénynek bemenetként lehet megadni az adott primitív inicializáló paramétereit (pl. cipher = AES.new(key, AES.MODE_CBC, iv)).

A PyCryptodome könyvtárral történő megismerkedés leghatékonyabb módja

az API dokumentáció[2] olvasgatása, és az abban található példa programok kipróbálása. Ez a gyakorlat az API dokumentáció következő részeinek ismeretét feltételezi, így **ezek elolvasása a gyakorlat megkezdése előtt kötelező**:

- AES blokkrejtjelező: https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html
- CBC mód: https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#cbc-mode
- Véletlenszám generátor: https://pycryptodome.readthedocs.io/en/latest/src/random/random.html
- Kitöltési sémák: https://pycryptodome.readthedocs.io/en/latest/src/util/util.html#crypto-util-padding-module
- PBKDF2 jelszó alapú kulcsderiváló függvény: https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html#Crypto.Protocol.KDF.PBKDF2
- SHA-256 hash függvény: https://pycryptodome.readthedocs.io/en/latest/src/hash/sha256.html
- RSA kulcspár generálás: https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html
- RSA-OAEP rejtjelező: https://pycryptodome.readthedocs.io/en/latest/src/cipher/oaep.html
- RSA-PSS digitális aláírás: https://pycryptodome.readthedocs.io/en/latest/src/signature/pkcs1_pss.html

4. Feladatok

Ez a laborgyakorlat 4 feladatot tartalmaz, melyek közül az első vezetett módon kerül megoldásra a gyakorlat során. Az első két feladatban az AES blokkrejtjelezőt kell használni CBC módban rejtjelezésre és dekódolásra. A

[2] https://pycryptodome.readthedocs.io/en/latest/src/api.html

második két feladat hibrid rejtjelezéssel és dekódolással, valamint digitális aláírás generálással és ellenőrzéssel foglalkozik, és az AES mellett, az RSA-OAEP és az RSA-PSS algoritmusokat kell használni a megoldáshoz. Minden feladatban egy, már elkezdett, de még hiányos programot kell kiegészíteni működő alkalmazássá. A megkezdett programok tartalmazzák a parancssori argumentumok kezelését, tartalmaznak továbbá néhány hasznos kiegészítő függvényt, sok hasznos kommentet, és tartalmazzák a megvalósítandó fő funkcionalitás egy részét. A hiányzó részek aláhúzás karakterekkel (___) vagy pont karakterekkel (...) vannak jelölve. Az aláhúzás egy hiányzó elemet jelöl (pl. egy hiányzó függvény paramétert vagy függvény nevet), a pontozás viszont több hiányzó elemet jelent (pl. egy hiányzó program sort vagy nagyobb program részletet). A kiegészítésre szoruló részeket egyelőre kommentekben helyeztük el, melyek a `# TODO:` címkét tartalmazzák. A megoldás során, ezeket a részeket ki kell kommentezni, a `# TODO:` címkét el kell távolítani, és a programot megfelelően ki kell egészíteni.

4.1. Feladat (vezetett)

A 4.1 és 4.2 feladatokban egy Python programot kell írni, ami az AES rejtjelezőt használja CBC módban fájlok rejtjelezésére és dekódolására. A következő fájlok adottak a feladathoz:

- `_aes_cbc.py` – a hiányos program kezdemény amit ki kell egészíteni,
- `test_plaintext1.txt` – egy teszt input fájl a 4.1 feladathoz,
- `test_ciphertext2.crypted` – egy teszt input fájl a 4.2 feladathoz.

A program a következő parancssori inputokat kapja:

- az elvégzendő művelet jele (e = rejtjelezés (encryption) vagy d = dekódolás (decryption)),
- egy jelszó amiből az AES kulcsot generáljuk (deriváljuk),
- az input fájl neve, ami a rejtjelezendő vagy dekódolandó adatot tartalmazza,
- az output fájl neve, amibe a rejtjelezés vagy dekódolás eredményét írjuk.

A programnak röviden a következőket kell csinálnia: beolvassa az input fájl tartalmát, az AES rejtjelezővel, a megadott jelszóból generált kulccsal, CBC módban rejtjelezi vagy dekódolja a tartalmat (attól függően, hogy milyen művelet lett megadva a parancssorban), majd kiírja az eredményt az output fájlba.

A 4.1 feladatban a program rejtjelező részét kell befejezni. Javasolt a `_aes_cbc.py` fájlt lemásolni és átnevezni `aes_cbc.py`-ra, és a továbbiakban ezen a másolaton dolgozni (így az eredeti hiányos fájl megmarad és rendelkezésre áll, ha esetleg nulláról újra kellene kezdeni a feladat megoldását).

Először nézzük meg az adott hiányos programot és próbáljuk megérteni hogy milyen parancssori inputokat vár a program, és ezeket az inputokat hogyan dolgozza fel. Ezután nézzük meg a rejtjelező részt, olvassuk el a kommenteket, és egészítsük ki a programot a megfelelő függvény nevekkel és paraméterekkel:

1. Az első részfeladat a nyílt szöveg kitöltése PKCS#7 padding alkalmazásával:

   ```
   # apply PKCS7 padding on the plaintext
   padded_plaintext = Padding.pad(plaintext, AES.block_size, style='pkcs7')
   ```

2. A következő részfeladat a kulcs származtatása a jelszóból:

   ```
   # derive a 32-byte key from the passphrase using PBKDF2
   # with a random salt and iteration count 1000
   key = KDF.PBKDF2(passphrase, salt, count=1000, dkLen=32)
   ```

3. Ezután egy IV-t kell generálnunk, ami egy véletlen blokk kell legyen:

   ```
   # generate random IV ...
   iv = Random.get_random_bytes(AES.block_size)
   ```

4. Mostmár minden készen áll ahhoz, hogy létrehozzuk azt a AES rejtjelezőt, amivel CBC módban rejtjelezünk majd:

   ```
   # ... and create an AES-CBC cipher object
   cipher_CBC = AES.new(key, AES.MODE_CBC, iv)
   ```

5. A következő, és egyben utolsó feladat az IV és a kitöltött nyílt szöveg rejtjelezése:

```
# encrypt the IV in ECB mode and the padded plaintext in CBC mode
encrypted_iv = cipher_ECB.encrypt(iv)
ciphertext = cipher_CBC.encrypt(padded_plaintext)
```

Mikor a rejtjelező rész teljesen készen van, állítsuk a `TEST` változót a program elején `True` értékre, és futtassuk a programot rejtjelezés funkcióval a `test_plaintext1.txt` teszt input fájllal, az `adishamir` jelszót használva. A program kiírja a képernyőre a rejtjelezés eredményének utolsó 16 bájtját hex formátumban. Ez az amit a 4.1 feladat megoldásaként be kell adni.

4.2. Feladat

A 4.2 feladatban az `aes_cbc.py` program dekódoló részét kell kiegészíteni a kommentekben adott instrukciókat követve. Mikor a dekódoló rész teljesen készen van, futtassuk le a programot dekódolás funkcióval a `test_ciphertext2.crypted` teszt input fájllal, a `ronrivest` jelszót használva. A program kiírja a képernyőre a dekódolás eredményének első 16 bájtját hex formátumban. Ez az amit a 4.2 feladat megoldásaként be kell küldeni.

Tanulmányozd a program azon részeit is melyek adva voltak az elejétől fogva, mert ezek tartalmazzák pl. az IV rejtjelezését az AES rejtjelezővel ECB módban. Érdemes megnézni, hogyan van ez a rész megírva, és hogyan történik a rejtjelezett IV kiírása (a rejtjelező részben), valamint beolvasása és feldolgozása (a dekódoló részben).

4.3. Feladat

A 4.3 és 4.4 feladatokban egy olyan Python programot kell fejleszteni, ami hibrid rejtjelezést és dekódolást használ, valamint opcionálisan digitális aláírást generál és ellenőriz. A fenti funkciók megvalósításához a program az AES-CBC, az RSA-OAEP, és az RSA-PSS algoritmusokat használja. A feladathoz a következő fájlok adottak:

- `_hybrid.py` – a hiányos program kezdemény amit ki kell egészíteni,
- `test_plaintext3.txt` – egy teszt input fájl a 4.3 feladathoz,
- `test_ciphertext4.txt` – egy teszt input fájl a 4.4 feladathoz,
- `test_pubkey.pem` – egy teszt publikus kulcsot tartalmazó PEM fájl,

- `test_keypair.pem` – egy teszt kulcspárt tartalmazó PEM fájl; mivel a kulcspár tartalmazza a privát kulcsot, ezért az jelszóval védett (és a jelszó: `crysys`).

A program a következő parancssori inputokat kapja:

- az elvégzendő művelet jele (k = kulcspár generálás, e = rejtjelezés (encryption), vagy d = dekódolás (decryption)),
- az input fájl neve ami a rejtjelezendő vagy dekódolandó adatot tartalmazza,
- az output fájl neve amibe a végrehajtott rejtjelezés vagy dekódolás eredményét írjuk,
- a publikus kulcsot tartalmazó PEM fájl neve, amire a rejtjelezéshez (vagy a digitális aláírás ellenőrzéshez) van szükség,
- a privát kulcsot (azaz az azt tartalmazó kulcspárt) tartalmazó PEM fájl neve amire a dekódoláshoz (vagy a digitális aláírás generáláshoz) van szükség.

A programnak röviden a következőképpen kell működnie: A program beolvassa az input fájl és a kulcs fájlok tartalmát, és elvégzi a kért műveletet (rejtjelezés vagy dekódolás) az inputon. Hibrid rejtjelezést használunk, azaz az inputot egy véletlen generált szimmetrikus kulccsal rejtjelezzük AES-sel CBC módban, majd a szimmetrikus kulcsot rejtjelezzük a megadott publikus kulccsal RSA-OAEP algoritmussal. Dekódolásnál először a szimmetrikus kulcsot dekódoljuk RSA-OAEP algoritmussal a megadott privát kulccsal, majd az így kapott szimmetrikus kulccsal dekódoljuk az inputot AES-sel CBC módban. A rejtjelezés eredményét Base64 kódolással kell kiírni az output fájlba. Ha rejtjelezésnél a parancssori inputok között volt privát kulcsot tartalmazó fájl is, akkor a programnak digitális aláírást is kell készítenie és azt szintén bele kell írni az output fájlba Base64 kódolással. Dekódolás esetén, ha az input fájlban található digitális aláírás, akkor a parancssori bementek között kell legyen egy publikus kulcsot tartalmazó fájl is, és a digitális aláírást ezzel a publikus kulccsal kell ellenőrizni a dekódolás előtt.

A 4.3 feladatban a hiányos program rejtjelezés részét kell kiegészíteni. Javasolt a `_hybrid.py` fájlt lemásolni és átnevezni `hybrid.py`-ra, és a továbbiakban ezen a másolaton dolgozni (így az eredeti hiányos fájl megmarad

és rendelkezésre áll, ha esetleg nulláról újra kellene kezdeni a feladat megoldását). Először vizsgáld meg a megadott hiányos programot: milyen parancssori inputokat vár és hogyan dolgozza azokat fel? Vizsgáld meg a program fájl elején található segéd függvényeket is, melyek kulcsok mentését és betöltését végzik. Érdemes megvizsgálni továbbá a kulcspár generálást végző függvényt és megérteni annak működését. Miután a program szerkezetével illetve a megadott részekkel megismerkedtél, utána van értelme elkezdeni a hiányzó részek megírását. Ehhez kövesd a kommentekben adott utasításokat és tanácsokat.

Mikor a rejtjelező rész teljesen készen van, állítsd a `TEST` változót a program elején `True` értékre, és futtasd a programot rejtjelezés funkcióval a `test_plaintext3.txt` teszt input fájllal és a `test_pubkey.pem` teszt publikus kulccsal. A program 32 karaktert ír ki a képernyőre, és ez az amit a 4.3 feladat megoldásaként be kell küldeni.

A feladat részeként vizsgáld még meg a program által generált output fájl tartalmát. Mivel minden kódolás eredményét Base64 kódolásban írtuk az output fájlba, ezért a fájl ASCII karaktereket tartalmaz, így bármilyen szövegszerkesztővel olvasható vagy a képernyőre nyomtatható a `less` vagy a `cat` parancsokkal. Látható, hogy a különböző output elemek (azaz a rejtjelezett AES kulcs, a CBC mód által használt IV, maga a rejtjelezett tartalom, és az aláírás) speciális elválasztó sorokkal vannak elválasztva (pl. `--- ENCRYPTED AES KEY ---`), ami a beolvasásnál segít majd az egyes elemek megtalálásában a következő feladatban.

4.4. Feladat

A 4.4 feladatban a program dekódoló részét kell befejezni a kommentekben adott tanácsokat és instrukciókat követve. Ebben a feladatban már teljes hiányzó sorok vannak (ezeket ... pontozás jelöli), melyeket meg kell írni. Mikor elkészült a dekódoló rész, futasd a programot dekódolás módban a `test_ciphertext4.txt` input fájllal és a `test_keypair.pem` privát kulcsot tartalmazó fájllal (emlékeztetőül: a privát kulcs a `crysys` jelszóval van védve, amit a program kérésére be kell írni a billentyűzeten). A program kiírja a képernyőre a dekódolás eredményének utolsó 16 bájtját hex formátumban. Ez az amit a 4.4 feladat megoldásaként be kell küldeni.

Érdemes még megvizsgálni az input beolvasását és feldolgozását. Látható, hogy az elválasztó sorok alapján hogyan azonosítja a program az egyes elemeket (kódolt kulcs, IV, kódolt adat, és aláírás).