Zadanie 2 (Kryptografia)

1. Zdôvodnenie

Vybrali sme si knižnicu pyca/cryptography hlavne z dôvodu, že to bola prvá, ktorú sme videli, a po prezretí stránky pre tejto knižnice, sme usúdili, že má dôkladne zdokumentované funkcionality a hlavne funkcionality, ktoré budú potrebné pre toto zadanie.
Inštaláciu sme vykonali v nami vytvorenom virtuálnom prostredí pomocou správcu balíkov pip, cez príkaz pip install cryptography.

2. Výber asymetrickej šifry a jej parametrov zdôvodnite

Keďže táto úloha požaduje použitie asymetrickej šifry, ako prvé nám na um popadla šifra RSA, ktorá je najpopulárnejšia a má silnú bezpečnosť. Taktiež, že knižnica pyca/cryptography disponuje s funkciami pre prácu s RSA šiframi.
Parameter **veľkosti kľúča** sme vyberali podľa súčasných odporúčaní, keďže veľkosti 1024 bitov a menej sú už prelomiteľné, tak sme použili veľkosť 2048 bitov, ktorá je bezpečná a nie moc výpočtovo náročná. **Verejný exponent** sme taktiež vyberali na základe odporúčania, keďže tento parameter pomáha pri generovaní kľúča, zvyčajne sa vyberajú Fermatove prvočísla, ktoré pomáhajú k výpočtovej efektivite, a odporúčaným číslom je 65537, ktoré je dostatočne veľké a stále výpočtovo efektívne.

```
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
```

Verejný kľúč sa uloží do simulovanej databázy, privátny kľúč sa “odošle” späť klientovi, v podobe upb.key súboru

3. Výber symetrickej šifry a jej parametrov zdôvodnite. Formát zašifrovaného súboru popíšte

Vybrali sme si šifrovací algoritmus AES, ktorý je považovaný na najbezpečnejší symetrický algoritmus, taktiež jeho výhodou je efektivita a výkon. Poskytuje rôzne nastavenie parametrov ako napríklad vybratie veľkosti kľúča, ktorý sme nastavili na 256 bitov, čo je odporúčané. Tento kľúč sa zašifruje verejným kľúčom používateľa user.
Taktiež sme mali možnosť nastaviť režim šifrovania - využili sme režim CBC čo znamená, že každý blok je závislý na jeho predchádzajúcom bloku, ale za to požaduje aby sa IV (initial vektor) generoval náhodne, tento initial vektor zaručuje že rovnaké plaintextové údaje

nebudú vždy šifrované rovnako taktiež sme museli pridať padding pre obsah z dôvodu, že AES šifruje dáta po blokoch, a ak je posledný blok menší ako požadovaná veľkosť, potrebujeme ho doplniť o ďalšie bajty. Bez paddingu by sa šifrované dáta nemohli správne spracovať, čo by mohlo viesť k chybám pri dešifrovaní alebo k odhaleniu vzorov v šifrovaných údajoch.
Formát zašifrovaného súboru pozostáva zo zašifrovaného symetrického kľúča, ktorý je na začiatku súboru, nasleduje zašifrovaný obsah súboru, ktorý môže a nemusí mať padding. Tieto 2 časti sú uložené v binárnom formáte. Výstup bude v súbore encrypted.bin.

4. Implementujte API volanie /api/decrypt (POST požiadavka)

V tejto úlohe si príjemca načíta svoj privátny kľúč (private key), pomocou ktorého dešifruje zašifrovaný kľúč jeho verejným kľúčom (public key, čím získa symetrický kľúč (secret key). Pomocou tohoto symetrického kľúča dešifruje obsah spomenutého súboru, ktorého výstup bude uložený v súbore decrypted.pdf. Obsah môžeme následne porovnať s originálnym súborom file.pdf, a uvidíme, že úlohy prebehli správne.

5. Implementujte API volanie /api/sign (POST požiadavka)

Užívateľ načíta svoj privátny kľúč a obsah dokumentu, tento obsah sa zahashuje nejakou funkciou, napríklad v našom prípade sme použili funkciu SHA256. Tento hash následne zašifruje užívateľ svojím privátnym kľúčom pomocou paddingu PSS (Probabilistic Signature Scheme), ktorý pridáva náhodnosť a zvyšuje bezpečnosť podpisu. Týmto vzniká digitálny podpis, ktorý sa uloží do súboru signature.bin.

6. Implementujte API volanie /api/verify/ (POST požiadavka)

Cieľom tohoto API volania je overiť pravosť digitálneho podpisu pre daného používateľa. Postupovali sme tak, že sa najprv načítal súbor a digitálny podpis z požiadavky, taktiež verejný kľúč používateľa, ktorý je uložený v databáz

Cieľom tohoto API volania je overiť pravosť digitálneho podpisu pre daného používateľa. Postupovali sme tak, že sa najprv načítal súbor a digitálny podpis z požiadavky, taktiež verejný kľúč používateľa z databázy. Obsah súboru sme zahashovali pomocou funkcie SHA256. Overenie digitálneho podpisu prebehne tak, že sa verejným kľúčom dešifruje podpis a následne výsledný hash sa porovná s hashom obsahu súboru, ktorý sme si predtým uložili. Ak sa digitálny podpis overil, vrátili sme kladnú odpoveď v JSON formáte, a keď nie, tak sa uložila záporná odpoveď.

Tento mechanizmus zaručuje, že ak bol dokument zmenený alebo ak podpis nebol vytvorený príslušným privátnym kľúčom, verifikácia zlyhá.

7. Zdôvodnite a popíšte mechanizmus ochrany integrity

Postupovali sme ako v úlohách 3 a 4 a riešili sme pridanie mechanizmu ochrany integrity.

Ako pridaný mechanizmus kontroly integrity je hashovacia funkcia, ktorá je použitá na zahashovanie súboru. Akékoľvek pozmenenie súboru zmení výsledný hash funkcie, čiže príjemca, ktorý dešifruje súbor vykoná rovnakú hash funkciu nad dešifrovaným súborom a následne vie porovnať výsledné hashe. Ak sa tieto hashe zhodujú, tak môže povedať, že súbor nebol modifikovaný. Ak sa hashe nezhodujú, tak vieme, že niekto modifikoval súbor.

Taktiež pri modifikovaní súboru môže nastať, že sa modifikuje zašifrovaný kľúč, čo sme vykonali zmenou cez hex editor (kľúč sa v našom prípade nachádza na začiatku súboru), a v takom prípade dešifrovanie zvyšku súboru neprebehne. Testovali sme aj situáciu, kde sme modifikovali obsah súboru, a v tomto prípade sa nevráti dešifrovaný súbor, ale nastane chyba, keďže sa hashe nezhodujú.