Opis i implementacija algoritama kriptiranja u programskom jeziku Python

Jović, Antonio

Sveučilište u Rijeci, Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Antonio Jović

Opis i implementacija algoritama kriptiranja u programskom jeziku Python

Završni rad

Mentor: prof. dr. sc., Ana Meštrović

Rijeka, 8. rujna 2023.

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 21.6.2023.

Zadatak za završni rad

Pristupnik: Antonio Jović

Naziv završnog rada: Opis i implementacija algoritama kriptiranja u programskom jeziku Python

Naziv završnog rada na engleskom jeziku: Description and implementation of encryption algorithms in the Python programming language

Sadržaj zadatka:

Zadatak završnog rada jest dati kratki pregled algoritama kriptiranja. Potrebno je detaljno opisati odabrane algoritme te u okviru praktičnog dijela zadatka potrebno je implementirati odabrane algoritme u programskom jeziku Python.

Mentor

Prof. dr. sc. Ana Meštrović

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 22.6.2023.

(potpis pristupnika)

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

Sažetak

Cilj završnog rada je primarno opisati modele različitih algoritama kriptiranja, odnosno objasniti načine kriptiranja za svaki opisani algoritam te međusobno usporediti njihovu učinkovitost u zaštiti podataka, i primijeniti ih u aplikaciji za razmjenu poruku.

Teorijski dio rada se sastoji od opisa odabranih algoritama kriptiranja.Za svaki algoritam je opisan i matematički model koji taj algoritam koristi, te primjena i razmjena ključeva za svaki algoritam. Odabrani su poznatiji algoritmi koji su u današnje vrijeme najčešće zastupljeni u praksi.

Praktični dio rada se sastoji od izrade aplikacije u kojoj se dva klijenta sigurno povezuju preko web socket-a (hrv. mrežna utičnica; pojem se koristi manjoj mjeri) kako bi mogli komunicirati. Klijenti se spajaju na istu aplikaciju koja je pokrenuta na lokalnom domaćinu (engl. localhost) te se preko dvije otvorene sesije preglednika prijavljuju sa korisničkim imenom i lozinkom, te po uspješnoj prijavi spajaju na socket. Komunikacija u prednjem planu izgleda kao i svaka druga; jedan klijent šalje poruku i ta poruka se prikazuju na oba klijenta, dok se u pozadini aplikacije generira ključ koji će se koristiti za autorizaciju klijenata te se provodi enkripcija i dekripcija poruke. Klijent može birati koju vrstu enkripcije želi, ali ju u prednjem planu ne vidi.

Za izradu aplikacije je korišten programski jezik Python, python modul tkinter, ugrađeni modul za generiranje elemenata sučelja, python modul pycryptodome, modul koji sadrži razne algoritme kriptiranja, modul pickle, koji služi za deserijalizaciju podataka te modul socket, potreban za ostvarivanje konekcije i modul threading, za višenitno izvršavanje programa.

Ključne riječi: enkripcija, dekripcija, algoritam, ključ, izvorni tekst, šifrirani tekst, socket, Python, tkinter, Crypto

Sadržaj

1. Uvod

U današnje vrijeme, potreba za sigurnošću na Internetu nikada nije bila veća jer postoje mnoge usluge za koje se očekuje da su sigurne, u koje se svakodnevno unose osobni i povjerljivi podaci što je svojevrsni sigurnosni rizik. Takvi podaci mogu biti ukradeni i upotrijebljeni u zlonamjerne svrhe, ako onaj koji ih ukrade ima smislene podatke. U tu svrhu, kako bi se osigurali podaci korisnika u raznim sustavima od zlonamjernih napada i dohvaćeni podaci učinili neupotrebljivima, koristi se postupak enkripcije, koji se u modernoj informatici pojavljuje krajem drugog tisućljeća (Burr i suradnici 2001).

Enkripcija, postupak je koji pretvara izvorni niz znakova, u praksi poznatiji kao plaintext (nadalje će se koristiti pojam izvorni tekst), u šifriran niz znakova, u praksi poznatiji kao cyphertext (nadalje će se koristiti pojam šifrirani tekst) (NIST 2015). Obratno, proces pretvaranja šifriranog niza znakova u izvorni niz naziva se dekripcija. Idealno, dohvaćeni šifriran niz znakova će kao izvorni niz moći pročitati samo ona osoba koja je autorizirana da ju pročita i neće biti čitljiva osobi koja ih presreće. U slučaju da postoji samo jedan način enkripcije podataka, tako kriptirane podatke bi onda bilo i lako za dekriptirati, zbog čega se pojavljuju različiti algoritmi kriptiranja, matematički niz radnji koji generira pseudo nasumičan enkripcijski ključ, niz znakova koji je pohranjen u datoteku ključeva, po kojem se dvije strane mogu autorizirati. Ključeve se može podijeliti na asimetrične i simetrične (NIST 2020).

Zadnjih nekoliko godina, pojavom ideje kvantnih računala[1], potreba za boljim algoritmima kriptiranja postaje veća jer kvantna računala mogu jednostavno „probiti“ današnje algoritme, zbog čega neki od algoritama kao što su AES ili DSA postaju zastarjeli i nemaju svrhu.

Kroz rad se prolazi kroz algoritme koji su najčešće, još uvijek u uporabi i koji se smatraju standardima kriptiranja kao što su AES, RSA i ECDSA te ih se kroz teorijski dio opisuje i prati kroz kôd.

[1] Kvantno računalo je računalo koje iskorištava kvantno mehaničke fenomene. U malim razmjerima, fizička materija pokazuje svojstva i čestica i valova, koristeći specijalizirani hardver koji podržava pripremu i manipulaciju kvantnih stanja.

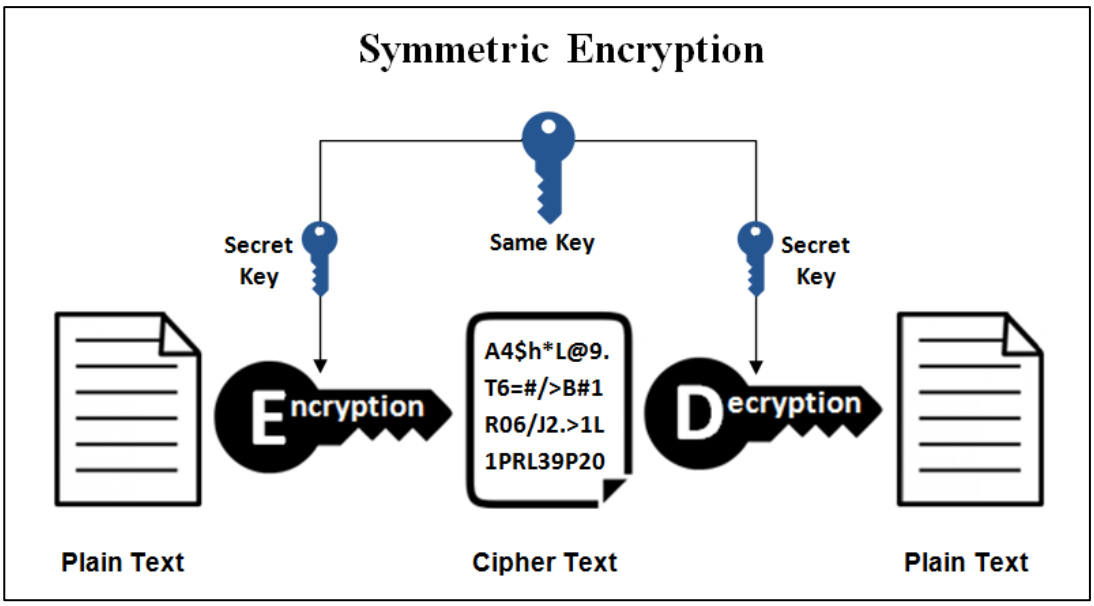
2. Ključevi i vrste ključeva

Ključevi, u kriptografiji, obično su neki nasumični nizovi slova i brojeva koji se pohranjuju u datoteke, te se nakon njihove obrade preko algoritma, mogu kriptirati ili dekriptirati poslani odnosno dobiveni podaci. Složenost ključa ovisi vrsti algoritma preko kojeg se generira te i o veličini bitova koja se koristi za generiranje. Veličina bitova sigurnosne snage u većini algoritama, koju preporuča organizacija NIST u dokumentu iz 2019. godine, predlaže: „da je najmanja potrebna veličina bitova od 112 bitova (prije 2014. godine je najmanja veličina bila 80 bitova)“ (NIST 2019.). Sigurnosna snaga i veličina bitova ključa nisu iste jer različiti algoritmi koriste različite veličine bitova ključeva za ostvarivanje sigurnosne snage.

Ključeve, prema vrsti, može se podijeliti na simetrične i asimetrične.

2.1 Simetrični ključevi

Simetrični ključevi, ključ je koji se koristi za izvođenje kriptografske operacije i njene inverzne (npr. za kriptiranje, dekriptiranje, stvaranje kôda za provjeru autentičnosti poruke ili provjeru kôda za provjeru autentičnosti poruke). Svrha simetričnog ključa je da služi kao jedan ključ i za šifriranje izvornog teksta i dešifriranje šifriranog teksta.



Slika 1. Shema korištenja simetričnog ključa. Izvor: (https://www.ssl2buy.com/ n.d.)

Prema shemi, oba korisnika prilikom inicijalizacije komunikacije podijele tajni ključ pomoću kojeg mogu kriptirati i dekriptirati. Nakon dijeljenja ključa, poslani izvorni tekst se kriptira u šifrirani tekst te se na drugom klijentu dekriptira ponovno u izvorni tekst.

Korištenje simetričnih ključeva se najčešće pojavljuje kad nije potrebna iznimna sigurnost podataka već brzina enkripcije odnosno dekripcije. Dvije najčešće upotrebe simetričnih ključeva su

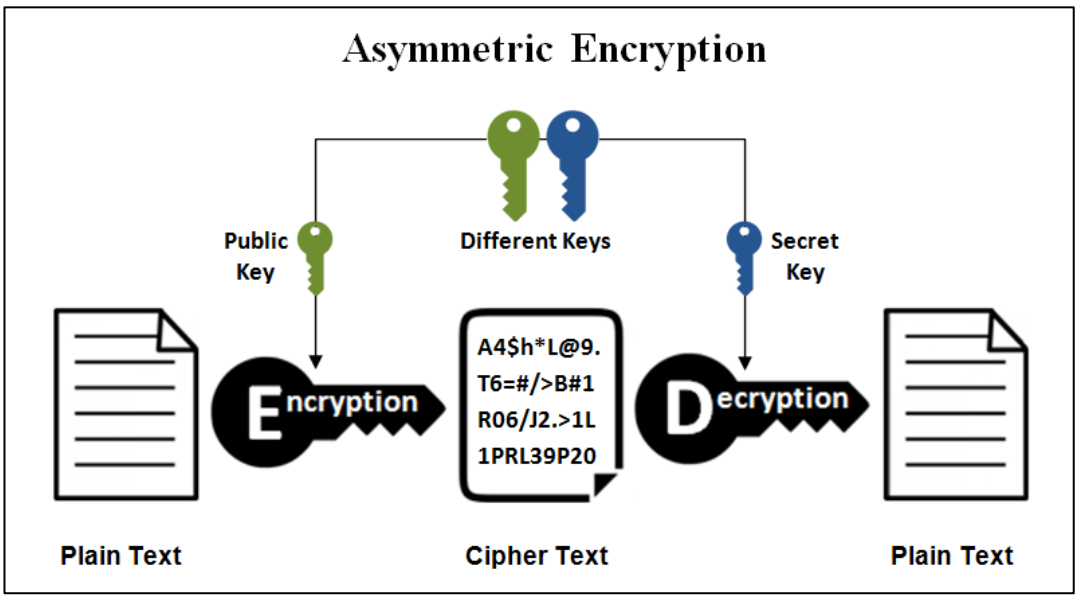
u bankarstvu, kada je potrebno kriptirati odnosno dekriptirati osobne podatke prilikom transakcija, i kod pohrane podataka dok se podaci ne koriste.

Najpoznatiji algoritmi koji koriste simetrični ključ su AES, ChaCha20, Twofish, Blowfish, DES, dvostruki DES (2DES) i trostruki DES (3DES). Prema NIST-u: „jedino AES i 3DES su ostali u praktičnoj upotrebi za ostvarivanje sigurnosti“ (NIST 2023), obzirom da je po simetričnom ključu jednostavnije za dekriptirati podatke.

2.2 Asimetrični ključevi

Asimetrični ključevi, dva su povezana ključa, a oni se nazivaju javni ključ i privatni ključ, koji se koriste za izvođenje komplementarnih operacija, kao što su enkripcija i dešifriranje ili generiranje potpisa i verifikacija potpisa.

Asimetrični ključ funkcionira tako da koristi par povezanih ključeva, jedan javni ključ i jedan privatni ključ, za kriptiranje i dekriptiranje podataka i autorizaciju klijenta. Javni ključ može koristiti bilo koji klijent i on je javno dostupan svima za kriptiranje podataka, odnosno da podatke može dekriptirati samo onaj klijent kojem su namijenjeni sa svojim privatnim ključem. Privatni ključ, također poznat i kao tajni ključ, koristi samo onaj klijent koji je inicijalizirao konekciju, odnosno svaki klijent ima različito generiran privatni ključ, ali vrijednosti kao što su eksponent ili korišteni prosti broj za kreiranje ključa moraju biti jednaki. Javni ključ se može koristiti i za kriptiranje i za dekriptiranje, dok javni ključ može samo kriptirati podatke.



Slika 2. Shema korištenja asimetričnog ključa. Izvor: (https://www.ssl2buy.com/ n.d.).

Algoritmi koji koriste asimetrični ključ su RSA, DSA i ECDSA, te algoritmi za razmjenu ključeva kao što je Diffie-Hellman ili X25519, ali postoje i razni drugi koji se koriste u manjoj mjeri. Od svih navedenih algoritama, jedino ECDSA ima potrebnu kompleksnost, odnosno ima dodatne mogućnosti u kôdu da poveća kompleksnost ključa, kako bi usporio „razbijanje“ algoritma kod napada kvantnog računala.

Trenutno se, od navedenih algoritama, ECDSA se koristi prilikom generiranja digitalnih potpisa i potpisivanja digitalnih potpisa te kod blockchain tehnologija[2] za potvrdu identiteta prilikom transakcija plaćanja kripto valutama te kod infrastrukture javnog ključa, dok se RSA koristi u svrhu izdavanja digitalnih certifikata. Osim kod navedenih primjena, asimetrični se ključevi, u većoj mjeri ECDSA, još koriste i kod nekoliko različitih protokola kao što je sigurni HTTP (odnosno HTTPS), SSL-a i TLS-a.

HTTPS (engl. skr. Hypertext Transfer Protocol Secure), poboljšana je verzija običnog HTTP-a. HTTPS je proširenje HTT protokola (HTTP), proširen je sa TLS-om (engl. skr. Transport Layer Security), odnosno prije TLS-a je bio proširen s SSL-om (engl. skr. Secure Sockets Layer) (Franklin 2020). Svrha HTTPS-a je potreba da se provjeri autentičnost web stranice, odnosno kako bi se očuvao integritet podataka klijenta koji trenutno provodi vrijeme na stranici. Zbog korištenog sigurnosnog sloja i enkripcije podataka, klijent je zaštićen od neželjenog prisluškivanja (engl. eavesdropping) komunikacije između klijenta i povezanog servera.

TLS, kriptografski je protokol koji pruža sigurnu komunikaciju preko mreže (Wozniak i drugi 2007). TLS ima cilj da održava integritet i autentičnost podataka korištenjem potpisanih certifikata. Dakako, komunikacija je moguća i bez TLS-a, ali time poslužiteljska web stranica gubi na autentičnosti, odnosno takva web stranica ima prefiks HTTP umjesto HTTPS.

SSL, preteča TLS-a, sigurnosni je protokol za uspostavljanje sigurne veze između poslužitelja i klijenta (Wozniak i drugi 2007). Kao i TLS, SSL kriptira podatke koji se šalju preko mreže, te može generirati certifikate koji održavaju komunikaciju sigurnom. Zbog svojih nedostataka, kao što je opozivanje certifikata prije datuma isteka ili Hearthbleed bug-a kod kojeg ovaj bug omogućuje napadačima da ukradu privatne ključeve (https://www.manageengine.com/key-manager/help/ssl-vulnerability.html n.d.). NIST je odredio 1999. godine da je SSL zastarjeli protokol za mrežnu komunikaciju (Singhal i drugi 2007).

[2] Blockchain - suradnička knjiga otporna na neovlašteno rukovanje koja održava evidenciju transakcija. Transakcijski zapisi (podaci) grupirani su u blokove. Blok je povezan s prethodnim uključivanjem jedinstvenog identifikatora koji se temelji na podacima prethodnog bloka.

3. Opis i implementacija odabranih algoritama kriptiranja

3.1 Rivest–Shamir–Adleman algoritam

Rivest–Shamir–Adleman algoritam, čije ime je dobio po tri kreatora algoritma zbog čega je češće poznatiji kao RSA, asimetrični je ključ koji koristi algoritam javnog ključa (Baker 2023). Sigurnost RSA algoritma oslanja se na praktične poteškoće rastavljanja umnoška dva velika prosta broja. Razbijanje RSA enkripcije poznato je i kao RSA problem. Trenutno, ne postoje objavljene metode za poraz sustava ako se koristi dovoljno velik ključ. RSA je relativno spor algoritam te se zbog toga često ne koristi za izravno kriptiranje korisničkih podataka. Češća upotreba RSA algoritma je za prijenos zajedničkih ključeva za kriptografiju sa simetričnim ključem, koji se zatim koriste za skupno kriptiranje odnosno dekriptiranje.

3.1.1 Opis izvedbe algoritma RSA

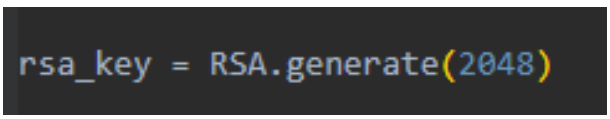
RSA algoritam se sastoji od četiri koraka: generiranje ključa, dijeljenje ključa te kriptiranje i dekriptiranje podataka.

Kako bi se generirao ključ, potrebno je odabrati dva velika prosta broja: neka su to P=61 i Q=71. Zatim ih je potrebno pomnožiti: n = P×Q = 4331. Osim dva prosta broja, potreban je i mali eksponent e koji će biti cijeli broj i ne smije biti faktor od Φ(n), koji se računa tako da oduzmemo od svakog prostog broja broj 1 i pomnožimo ih, čime se dobije da je Φ(n) = (P-1)(Q-1) = 4200. Neka je e = 9 (on zadovoljava prethodno definirane uvjete) i sada je moguće generirati privatni ključ: d = (k*Φ(n) + 1) / e za neki cijeli broj k; neka je k = 3 iz čega proizlazi da je d = 1400.

3.1.2 Primjena algoritma RSA u kôdu

U Python-u postoji već gotov modul pycrytodome, koji je verzija za Windows operativni sustav (*pycrypto* je modul koji je podržan samo na Linux operativnim sustavima). Unutar modula se nalaze razne funkcije za manipulaciju RSA ključevima pa nije potrebno da se unutar kôda unosi cijeli algoritam.

Želi li se kriptirati ili dekriptirati poruke, prvo je potrebno generirati RSA ključ kao na slici 3:



Slika 3. Generiranje RSA para ključeva

Funkcija *generate()* stvara RSA par ključeva veličine 2048 bita (može biti još i 1024 ili 4096) i koristi za eksponent broj 65537 (eksponent se može i implicitno izmijeniti, ali je kod generiranje

on interno zadan na navedeni broj, koji je prema pycryptodome dokumentaciji standardan po NIST-u).

Nakon generiranja, potrebno je parove ključeva podijeliti na javni i privatni ključ. Funkcijom *export_key()* se izvozi privatni ključ, a kombinacijom *publickey().export_key()* se izvozi javni ključ. Navedeni ključevi se zatim pohranjuju u datoteke ključeva, svaki posebno u svoju, kao što je i prikazano na slici 4:

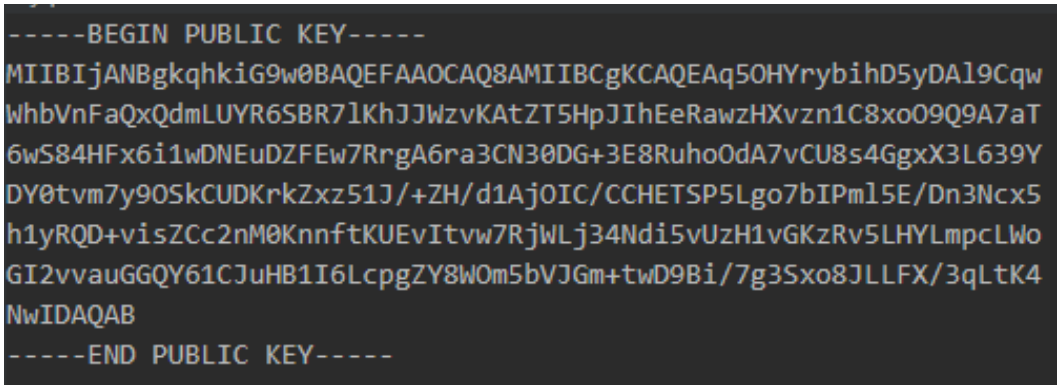
```
def generate_RSA_keys(key):

    if not (os.path.isfile("priv_key.pem") or os.path.isfile("pub_key.pem")):

        priv_key = key.export_key()
        pub_key = key.publickey().export_key()

        priv_key_pem = open('priv_key.pem', 'wb')
        priv_key_pem.write(priv_key)
        priv_key_pem.close()
        pub_key_pem = open('pub_key.pem', 'wb')
        pub_key_pem.write(pub_key)
        pub_key_pem.close()

    pub_key_pem = open('pub_key.pem', 'rb')
    priv_key_pem = open('priv_key.pem', 'rb')
    rsa_priv = RSA.importKey(priv_key_pem.read())
    rsa_pub = RSA.importKey(pub_key_pem.read())

    decipher_rsa = PKCS1_OAEP.new(rsa_priv)
    cipher_rsa = PKCS1_OAEP.new(rsa_pub)

    return decipher_rsa, cipher_rsa
```

Slika 4. Pohrana javnog i privatnog RSA ključa

Osim izvoza, tako pohranjeni ključevi se zatim mogu i uvesti kako bi se sa istim ključevima moglo kriptirati odnosno dekriptirati. Kriptiranje se vrši preko javnog ključa, ali prije nego bi se moglo kriptirati potrebno je preko funkcije *new()* iz modula *PKCS1_OAEP*, modul pomoću kojeg se stvara novi objekt za šifriranje (engl. cipher object) s kojim se šifriraju poruke s obzirom na koji ključ je postavljen u funkciji *new()*. Dekriptiranje, ali i kriptiranje, moguće je preko privatnog ključa.

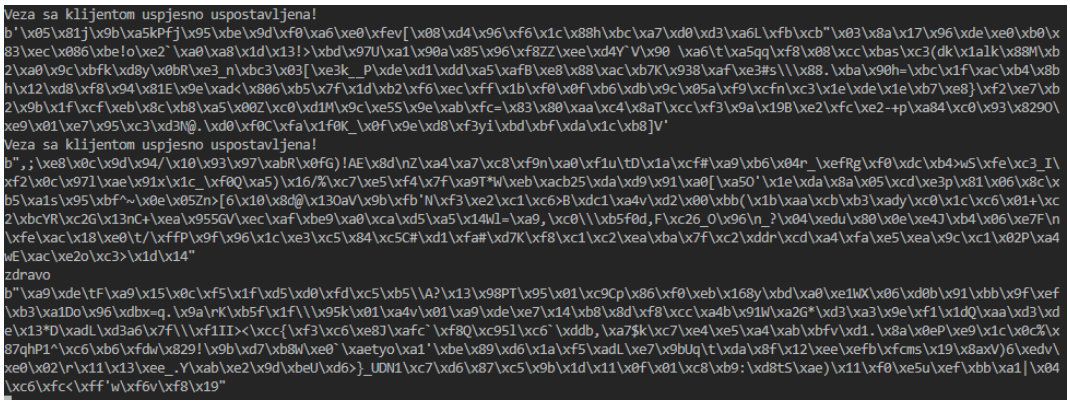
Nakon izvoza i pohrane ključeva, na slikama 5. i 6. su prikazani pohranjeni ključevi u datoteke ključeva:

Slika 5. Pohranjeni privatni RSA ključ



Slika 6. Pohranjeni javni RSA ključ

Nakon što su ključevi pohranjeni, sada ih je moguće koristiti kod slanja poruka. Kod kriptiranja poruka, potrebno je koristiti prethodno prikazani cipher_rsa te pomoću funkcije *encrypt()* kriptirati poruku. Obzirom da nakon spajanja klijenata slijedi pozdravna poruka od servera, ona se također kriptira, pa je kriptirane poruke moguće vidjeti na slici 7:



Slika 7. Kriptirane poruke javni RSA ključem

Na slici se vidi da su dvije kriptirane poruke različite, ali one se klijentima jednako prikazuju U ovom slučaju, moguće je zanemariti ispis: „Veza sa klijentom uspjesno uspostavljena!“, iako ona je dobar indikator da su dva klijenta uspješno uspostavili razmjenu poruka sa serverom.

3.2 Napredni standard enkripcije, AES

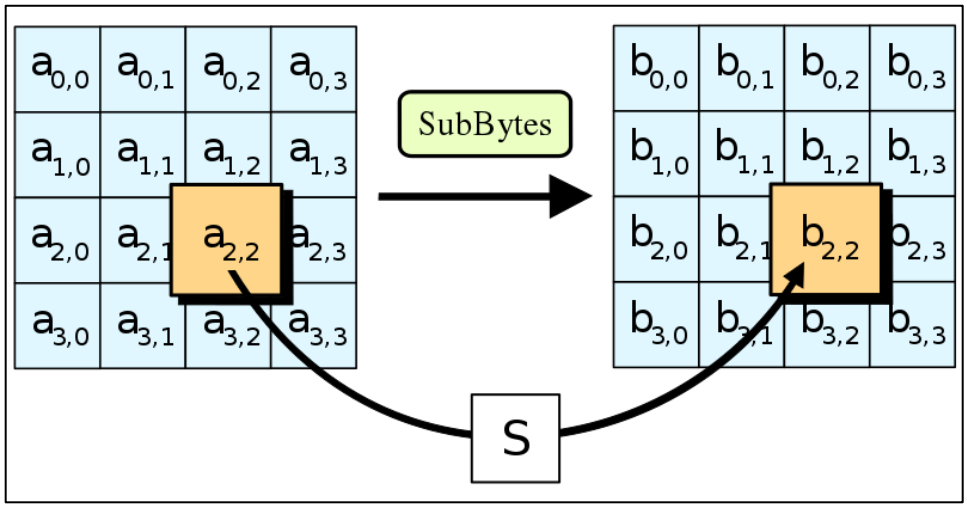
Napredni standard enkripcije (engl. Advanced Encryption Standard, AES), standard je također poznat i kao Rijndaelov algoritam, simetrično blok šifriranje koje može obrađivati blokove podataka od 128 bita, koristeći ključeve duljine 128, 192 i 256 bita (NIST 2023). Osim navedenih, AES je dizajniran za rukovanje dodatnim veličinama blokova i duljinama ključeva. AES je usvojen i standardiziran algoritam za kriptiranje podataka Američkih federalnih dokumenata. Osim primjene u američkoj vladi, Microsoft ga zadnjih nekoliko godina primjenjuje kod Bitlocker-ovog kriptiranja i dekriptiranja ugrađenog diska računala u svrhu osiguravanja podataka. Trenutna upotreba AES-a je temeljena na 256 bita obzirom da je ona kompleksnija od ostalih. AES je također trenutno jedini aktualni i standardiziran algoritam simetričnog ključa koji se još uvijek učestalo koristi.

3.2.1 Opis izvedbe algoritma AES

AES koristi princip supstitucijsko-permutacijske mreže, niz je povezanih matematičkih operacija koje se koriste u algoritmima blok šifriranja; Takva mreža uzima blok plaintext-a i ključa kao ulaze i primjenjuje nekoliko izmjeničnih krugova zamjenskih kutija (S-kutija) i permutacijskih kutija (P-kutija) za proizvodnju bloka šifriranog teksta, odnosno ovisno koji se broj bitova ključa odabere, toliko će postojati i rundi (engl. rounds) kroz koje će algoritam prolaziti: 128 bita je 10 rundi, 192 bita je 12 rundi i 256 je 14 rundi. AES koristi konačno polje koje je veličine 4×4, od

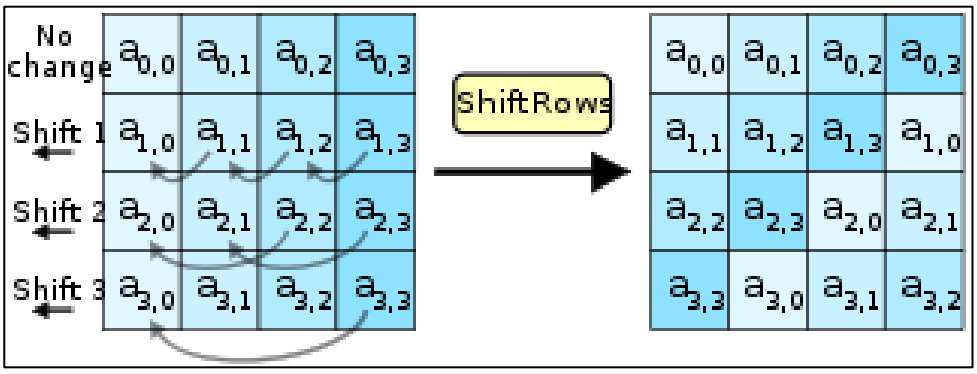
kojih svako polje ima po 1 bajt (ukupno 16 bajtova) te se takvo polje naziva stanjem. Svaka runda se sastoji od 4 koraka: *SubBytes*, *ShiftRows*, *MixColumns* i *AddRoundKey*.

Zamjena bajtova korak je implementira zamjenu. U ovom koraku svaki bajt je zamijenjen drugim bajtom. Izvodi se pomoću tablice pretraživanja koja se također naziva S-box. Ova zamjena se vrši na način da se bajt nikada ne zamjenjuje sam po sebi niti drugim bajtom koji je dodatak trenutnom bajtu. Rezultat ovog koraka je matrica od 16 bajtova (4 x 4) kao prije.



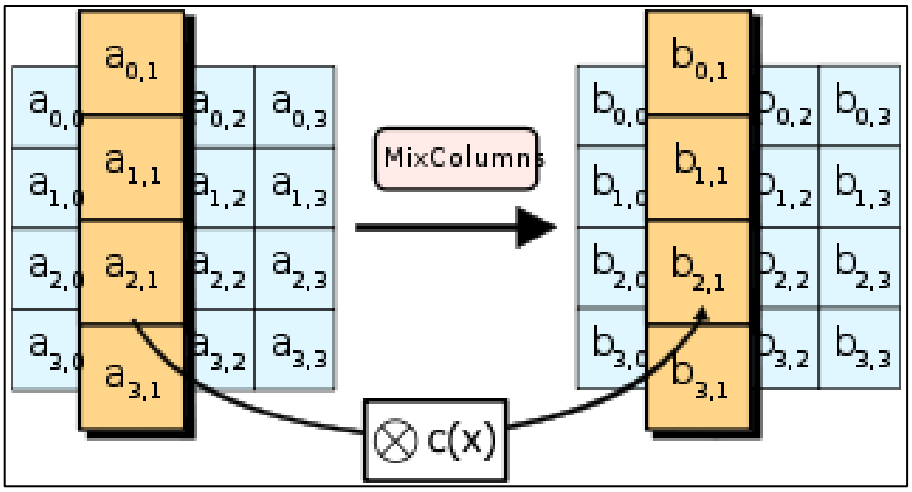
Slika 8. Zamjena bajtova kod AES algoritma. Izvor: (Matt Crypto 2023)

Pomak redaka, korak je u kojem se svaki redak stanja, odnosno bajt, pomiče kružno. Prvi red nije pomaknut. Svaki bajt drugog retka pomaknut je jedan ulijevo. Slično, treći i četvrti redak pomaknuti su za pomake od dva odnosno za tri bajta. Prema tome, prvi redak se ne mijenja, dok u drugom retku svaki bajt je pomaknut za jedan bajt u lijevo, zbog čega se na prvom mjestu sad nalazi $a_{1,1}$, a na zadnjem se nalazi onaj koji je bio prvi. Obzirom na veličinu AES ključa, onoliko rundi će se izvršiti. U slučaju da je AES-128, odnosno 10 rundi, nakon 10 rundi će u drugom retku se na prvom mjestu nalaziti $a_{1,2}$, u trećem retku ponovno je $a_{2,0}$ i u četvrtom je $a_{3,3}$.



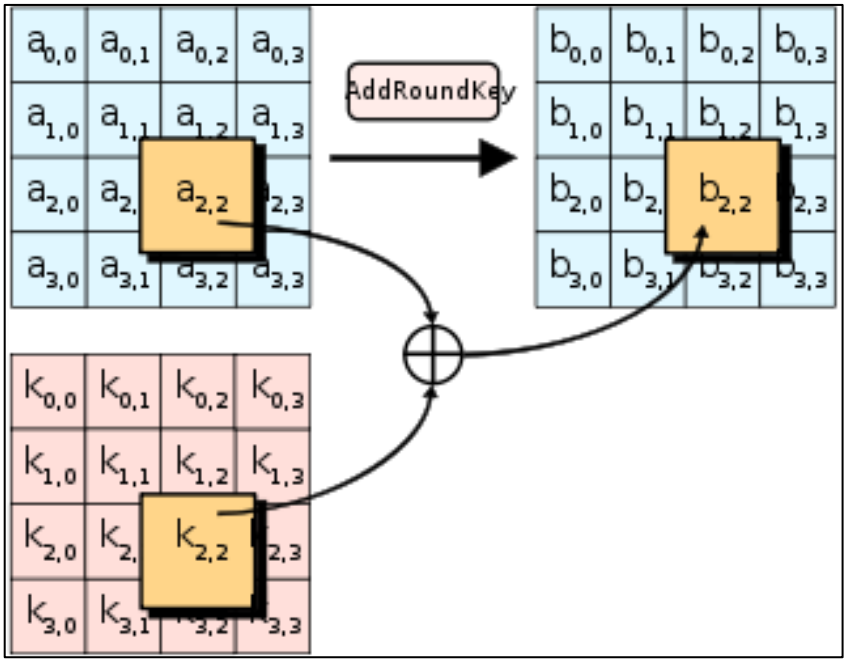
Slika 9. Pomicanje redaka kod AES algoritma. Izvor: (Matt Crypto 2023)

Kombiniranje stupaca, korak je kod kojeg se vrši množenje matrice. Svaki stupac se množi s određenim konstantnim polinomom i stoga se položaj svakog bajta u stupcu mijenja kao rezultat. Ovaj korak se preskače u zadnjoj rundi.



Slika 10. Množenje matrice kod AES algoritma. Izvor: (Matt Crypto 2023)

Kao konačni korak, preostaje dodavanje ključa runde. Na kraju svake runde, oba ključa runde iz prvog stanja i drugog stanja, gdje drugo stanje k predstavlja podključ koji je izveden za svaku rundu iz glavnog ključa ključa korištenjem Rijndaelova rasporeda ključeva, se svaki ključ runde pomoću isključive disjunkcije (XOR) kombinira sa podključem runde, gdje je podključ runde na istoj poziciji u matrici kao i ključ runde. Kombinacijom se, nakon odrađivanja svih rundi, dobiva šifra kojom će se kriptirati podaci.



Slika 11. Dodavanje kružnog ključa kod AES algoritma. Izvor: (Matt Crypto 2023)

3.2.2 Implementacija algoritma AES u kôdu

Obzirom da je AES simetrični algoritam, on generira isti ključ za kriptiranje i dekriptiranje. U Python-u, kao što je RSA generirao svoj objekt za šifriranje, tako i AES generira svoj. Problem koji nastaje kod AES-a u ovakvoj vrsti programa jest ta što je kriptiranje i dekriptiranje sa AES objektom moguće samo jednom i teže je uporabljivo. Svakako, algoritam je implementiran u kôdu kako bi se prikazala njegova primjena.

Prvo je potrebno generirati tajni AES ključ, odnosno neki znakovni niz pretvoren u niz bajtova, preko kojeg će se generirati objekt za šifriranje. Osim ključa, koristi se i „soljenje“ (engl. salting), odnosno uzima se neki znakovni niz i pretvara u niz bajtova kako bi se dodatno zakomplicirao objekt za šifriranje i onemogućili ili najmanje daleko usporilo „razbijanje“ objekta za šifriranje.

```
aes_key = Random.get_random_bytes(16)
aes_salt = Random.get_random_bytes(16)
```

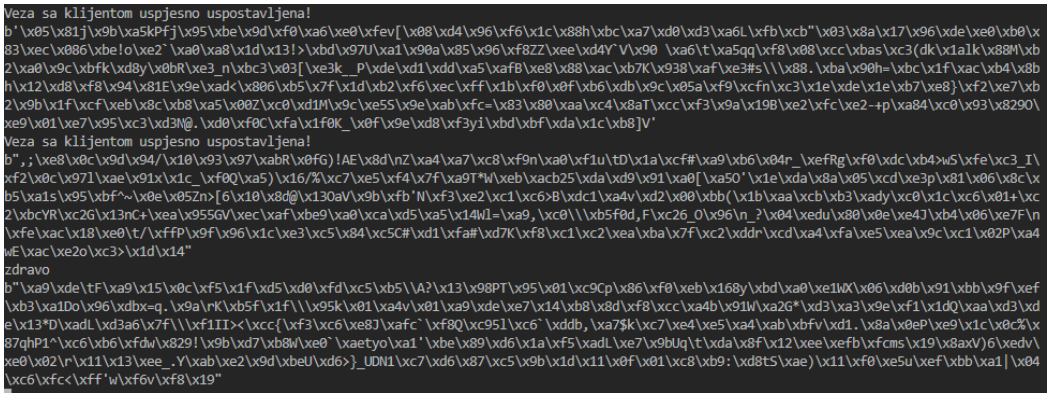
Slika 12. Generiranje tajnog ključa i dodatnog znakovnog niza primjenom algoritma AES

Kako bi se dodatno zakomplicirao objekt, pomoću modula PBKDF2 se derivira ključ iz ta dva znakovna niza. Pomoću već navedene funkcije new(), generira se objekt za šifriranje iz deriviranog ključa i odabira načina šifriranja, kojih postoji nekolicina. U kôdu je korišten način MODE_CFB, odnosno način koji šifrira blokove, način koji predstavlja prethodno obrađeno teorijsko objašnjenje.

```
def generate_AES_key(aes_key, aes_salt):

    kdf = PBKDF2(aes_key, aes_salt)
    cipher_aes = AES.new(kdf, AES.MODE_CFB)
    decipher_aes = AES.new(kdf, AES.MODE_CFB)
    return  cipher_aes, decipher_aes
```

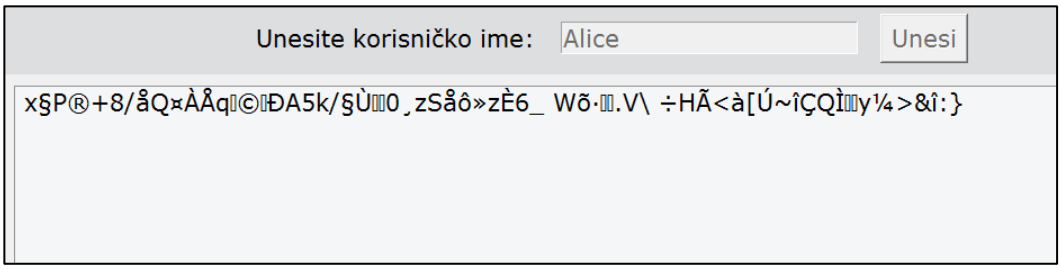
Slika 13. Generiranje objekta šifriranja koristeći derivrani ključ kod AES algoritma

Kriptiranjem pozdravne poruke, ponovno se dobiva neki nasumično generirani niz znakova:



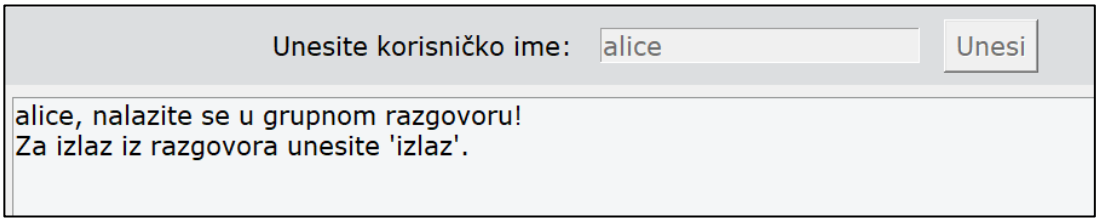
Slika 14. Pozdravne poruke kriptirane AES ključem

Kako AES traži da broj bitova koji se prihvaćaju na socket-u mora biti 1024 bita, dolazi do krivog čitanja poruke. Na slici 15 moguće je vidjeti primjer greške koja se događa kada se pozdravna poruka, koja je kod RSA dekriptiranja poruke bila jednaka izvornom tekstu, dekriptira pogrešno jer klijent dobiva nepotpunu poruku zbog neusklađenosti veličina bitova i tipa enkodiranja znakovnog niza u bitove, odnosno pripremanje poruke za slanje preko socket-a:



Slika 15. Primjer pogreške u ispisuGreška ispisa

Standardni tip enkodiranja je UTF-8, ali za potrebe AES kriptiranja i dekriptiranja gdje je privaćena poruka veličine 4096 bitova, potrebno je enkodirati kriptiranu poruku odnosno dekodirati dekriptiranu poruku sa UTF-32 čime se dobije točan izvorni tekst prikaz na slici 16:



Slika 16. Izvorni tekst nakon pravilnog enkodiranja sa UTF-32

Standardni Obzirom da je AES simetrični ključ, razgovor između dva klijenta nije moguć jer ovakak ključ služi samo komunikaciju između dvije strane, u ovom samo između slučaju servera i jednog klijenta.

3.3 Eliptični DSA

Eliptični DSA, poznatiji po kraćem ime kao ECDSA (engl. Elliptic Curve DSA), jedna je od varijanta algoritma digitalnog potpisa (engl. skr. DSA) (NIST 2020). Za razliku od običnog DSA, ECDSA koristi eliptičnu kriptografiju, pristup kriptografiji s javnim ključem koji se temelji na algebarskoj strukturi eliptičkih krivulja nad konačnim poljima. ECDSA je trenutno jedna od boljih opcija kod kriptiranja podataka jer nije jednostavna za „razbiti“, te se može teoretski duže odupirati brute force napadima kvantnih računala.

3.3.1 Opis izvedbe algoritma

Koristeći hipotetske izraze, neka postoje dva klijenta: klijent 1 i klijent 2, i klijent 1 želi poslati poruku klijentu 2. Za početak, potreban je dogovor koji će biti parametri krivulje (Krivulja, G, n), gdje su G osnovna točka primarnog reda na krivulji, a n je broj koji se množi sa G. Osim toga, potrebno je i uspostaviti Bézoutov identitet[3]. Klijent 1 generira par ključeva koji je sastavljen od privatnog ključa d_A (koji je nasumično generiran ili odabran) koji su nasumično odabrani u intervalu od [1, n-1].Potrebna je i točka krivulje koja se računa prema formuli: $Q_A = d_A \times G$. Kako bi klijent 1 potpisao poruku, potrebno je:

- dobiti izračun e = hash(p) - gdje je hash jedna od vrsta hash-iranja kao što je SHA-256, a p je poruka
- neka z bude krajnji lijevi dio e
- nasumični odabir cijelog broja u prethodno definiranom intervalu
- zatim izračunati točku krivulje (x, y)
- izračunati r = x mod n; u slučaju da je r = 0 potrebno je ponovno odabrati cijeli broj iz intervala
- izračunati $s = k^{-1} (z + rd_A) \bmod n$; u slučaju da je r = 0 potrebno je ponovno odabrati cijeli broj iz intervala
- konačno, par (r, s) je važeći potpis

3.3.2 Primjena u kodu

Kako je ECDSA algoritam koji se koristi za digitalno potpisivanje, u aplikaciji se ECDSA koristi za međusobnu autentifikaciju servera i pojedinog klijenta. Ekvivalent RSA privatnog ključa je kod ECDSA algoritma ključ za potpisivanje (engl. signing key). Obzirom da na serveru počinje konekcija, server prvi generiran ključ. Preko tog ključa je zatim moguće generirati ključ za verifikaciju (engl. verifying key), pomoću kojeg server može prepoznati klijenta. Zatim se server „potpisuje“ za ključem za potpisivanje i šalje potpis preko socket-a klijentu.

Osim potpisa, klijentu se šalje i ključ za verifikaciju, koji je ekvivalent RSA javnom ključu. Nakon poslanog ključa, server čeka da klijent verificira potpis i da klijent pošalje svoj potpis i ključ za verifikaciju:

[3] Bézoutov identitet – u matematici; Neka su a i b cijeli brojevi s najvećim zajedničkim djeliteljem d: tada postoje cijeli brojevi x i y takvi da je $a_x + b_y = d$. Štoviše, cijeli brojevi oblika $a_z + b_t$ točno su umnošci d

```
def ECDSA_autentikacija_klijenta(klijent):
    sk = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
    vk = sk.get_verifying_key()
    sig = sk.sign(b"SERVER")
    klijent.send(sig)
    time.sleep(0.2)
    klijent.send(pickle.dumps(vk))

    sig = klijent.recv(2048)
    vk = pickle.loads(klijent.recv(65536))

    try:
        vk.verify(sig, b"CLIENT")
        print("Veza sa klijentom uspjesno uspostavljena!")
    except:
        try:
            print("Veza sa klijentom nije uspostavljena...")
        except:
            print("Greska...")
```

Slika 17. Serverska ECDSA autentifikacija

Na suprotnoj strani, klijent prvo prihvaća ključ i potpis, te nakon verifikacije i potpisa, generira svoje ključeve i šalje ih serveru:

```
def ECDSA_autentikacija_servera(client):
    sig = client.recv(2048)
    vk = pickle.loads(client.recv(65536))
    try:
        vk.verify(sig, b"SERVER")
        print("Veza sa serverom uspjesno uspostavljena!")
    except:
        try:
            print("Veza sa serverom nije uspostavljena...")
        except:
            print("Greska...")

    sk = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
    vk = sk.get_verifying_key()
    sig = sk.sign(b"CLIENT")
    client.send(sig)
    time.sleep(0.2)
    client.send(pickle.dumps(vk))
    time.sleep(0.5)
```

Slika 18. Klijentska ECDSA autentifikacija

Indikator da je veza uspješno uspostavljena nije potrebno prikazivati obzirom da je prethodno, kod RSA algoritma, moglo biti viđeno da su se server i klijent međusobno verificirali.

3.4 Trostruki DES

Trostruki DES, također poznatiji i kao 3DES, blok šifra je sa simetričnim ključem, koja primjenjuje algoritam DES šifre tri puta na svaki podatkovni blok (NIST 2007). 56 bitni ključ standarda kriptiranja podataka (engl. DES) više se ne smatra primjerenim u suočenju s modernim kripto analitičkim tehnikama i snagama superračunala, zbog čega ga je i NIST ukinuo 2017. godine za korištenje kod osiguravanja povjerljivih podataka.

3.4.1 Opis izvedbe algoritma

3DES koristi jednostavan pristup povećanju kompleksnosti snage algoritma tako što uzima kratki 56 bitni ključ i zapravo koristi dva takva ključa K_1 i K_2 te tako kriptira svaki blok 2 puta: $E_{K2}(E_{K1}(\text{plaintext}))$. Pritom se može smatrati da je duljina ključa, umjesto početnih 56 bita, sada dvostruko veća i iznosi 112 bita. Ovakav pristup se naziva dvostruki DES, ali on nije dovoljan.

Umjesto toga je potrebno dodati još jedan ključ duljine 56 bita te poput prethodnog kriptiranja dodati i treći ključ, čime se efektivno dobiva složenost od 3n bita, odnosno 168 bita. Posljednja opcija je najjača, sa 168 neovisnih bitova ključa. I dalje je ranjiv na napad susreta u sredini, ali napad zahtijeva $2^{2 \times 56}$ koraka.

Obzirom da je trostruki DES simetričan algoritam poput AES-a, njegova primjena u kôdu nije prikazana već je kao njegova zamjena korišten Fernet koji će biti objašnjen u nastavku.

3.5 Fernet

Fernet, poseban je slučaj u kriptografiji, jer je dio Python modula cryptography i njega održavaju Python programeri. Nije klasični algoritam kriptiranja, ali je zasnovan na AES 128-bitnom kriptiranju. Fernet koristi simetrično kriptiranje, odnosno koristi jedan ključ za kriptiranje i dekriptiranje. Osim jednostrukog ključa, Fernet nudi mogućnost i rotacije ključa pomoću MultiFernet-a, više različitih ključeva. MultiFernet funkcionira tako da uzima dva ključa, od kojih je prvi dodani onaj s kojim se sve podaci kriptiraju. Svi ostali ključevi se rotiraju te se pomoću njih pokušavaju dekriptirati podaci.

Fernet je idealan za kriptiranje manjih paketa podataka, koji mogu stati u memoriju, dok veći podaci kao što su datoteke nisu optimalne. Fernet se koristi u Apache i Red Hat bazama podataka u oblaku.

U kôdu, primjena je slična AES primjeni. Prvo je potrebno generirati Fernet ključ kao na slici 19:

```
fernet_key = Fernet.generate_key()
```

Slika 19. Generiranje Fernet ključa

Zatim se tako generiran ključ sprema u datoteku, odnosno čita iz datoteke ako već pstoji kako bi se kreirao Fernet objekt za šifriranje:

```python
def generate_Fernet_key(fernet_key):
    if not os.path.isfile("fernet_key.key"):

        with open("fernet_key.key", "wb") as key_file:
            key_file.write(fernet_key)

    with open("fernet_key.key", "rb") as key_file:
        read_key = key_file.read()

    fernet = Fernet(read_key)

    return fernet
```

Slika 20. Pohrana i postavljanje objekta za šifriranje

Pohranjeni ključ u datoteci nije nalik klasičnog ključa već je dosta sličan hash-iranom znakovnom nizu, odnosno izgleda kao kriptirana poruka:

```
QUb5sS8eRm6QvdV7XaB4a_1reF2t-xSZvzuKFL9kUHo=
```

Slika 21. Pohranjeni Fernet ključ

Nakon povezivanja i prilikom slanja poruke, poruka se kriptira i izgleda kao na slici 22, gdje kriptirana poruka predstavlja izvorni tekst „zdravo“:

```
zdravo
b'gAAAAABk-SrP3zAbGY_N9KkBRPeENruRydmXbdY5Zf6al0MvHTLShrGmpmpdcruZbzGwFYiJW3HGq5WdWdu8vf6H-JmX6zVgg=='
```

Slika 22. Kriptirana poruka Fernet-om

4. Zaključak

U radu su prikazane implementacije i objašnjeni teorijski neki od danas najkorištenijih algoritama kriptiranja. Može se zaključiti da korištenje asimetričnih ključeva, iako kompliciranije i sporije, omogućuje veću sigurnost jer se koriste različiti ključevi za dvije komplementarne radnje, kriptiranje podataka i njihovo dekriptiranje. Iako su RSA i ECDSA oba asimetrični ključevi, njihove primjene su različite. Dok RSA omogućava i kriptiranje i dekriptiranje te generiranje certifikata, ECDSA nudi potpisivanje poruka i njihovu verifikaciju, no također nudi i generiranje i potpisivanje certifikata. Da bi se generirao RSA ključ, potrebno je baratati sa faktorizacijom velikih brojeva, dok ECDSA koristi matematička svojstva eliptičnih krivulja.

Korištenje simetričnih ključeva je jednostavnije i lakše ih je generirati, ali problem kod simetričnih ključeva je taj što se koristi jedan ključ za dvije komplementarne radnje, što ih čini učinkovitima i bržima, ali manje sigurnima i lakšima za probiti. AES, koji je standardan i jedini prepoznat kao kvalitetan simetrični ključ, nudi brzo kriptiranje i dekriptiranje podataka manje veličine. Za razliku od AES-a, koji može kriptirati i dekriptirati sa istim ključem samo u jednom smjeru, Fernet nudi mogućnost ponovne iskoristivosti istog ključa za komplementarne radnje između više spojenih klijenata odnosno poslužitelja.

U mrežnoj sigurnosti, obje vrste ključeva se pokušavaju koristiti, tako što se kombiniraju zbog svojih međusobnih prednosti, pa se tako može koristiti ECDSA za međusobnu autorizaciju zbog svoje sposobnosti sigurnog potpisivanja, te se istovremeno može koristiti AES za simetrično kriptiranje i dekriptiranje podataka.

5. Popis slika

6. Popis priloga

Prilog A - datoteke s programskim kôdom

7. Literatura

Baker, Elaine. 2023. »https://www.nist.gov/.« ožujak. Pokušaj pristupa 6. rujan 2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-175Br1.pdf.

Burr i suradnici. 2001. *A Century of Excellence in Measurements, Standards, and Techology.* Uredio David R. Lide. Washington: National Institute of Standards and Technology. https://www.govinfo.gov/content/pkg/GOVPUB-C13-310bc7b3121ed82b8f13fc15a5c9e639/pdf/GOVPUB-C13-310bc7b3121ed82b8f13fc15a5c9e639.pdf.

Elminaam, Diaa Salama, Hatem Mohamed Abdual Kader, i Mohiy Mohamed Hadhoud. 2010. »Evaluating The Performance of Symmetric Encryption Algorithms.« svibanj. Pokušaj pristupa 27. kolovoz 2023. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e2bc791344a938ead55f77a2b2fd43949dbedc6f.

n.d. »https://www.ssl2buy.com/.« Pokušaj pristupa 26. kolovoz 2023. https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences.

Matt Crypto. 2023. »https://en.wikipedia.org.« 31. kolovoz. Pokušaj pristupa 3. rujan 2023. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

—. 2023. »https://en.wikipedia.org.« *https://en.wikipedia.org/wiki/File:AES-SubBytes.svg.* 31. kolovoz. Pokušaj pristupa 3. rujan 2023. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

NIST. 2019. *https://www.nist.gov/.* Pokušaj pristupa 26. kolovoz 2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf.

—. 2023. *https://nist.gov/.* 18. kolovoz. Pokušaj pristupa 27. kolovoza 2023. https://csrc.nist.gov/Projects/block-cipher-techniques.

—. 2007. »https://nvlpubs.nist.gov/.« 4. veljača. Pokušaj pristupa 5. rujan 2023. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-45ver2.pdf.

—. 2015. »https://www.nist.gov.« svibanj. Pokušaj pristupa 6. rujan 2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf.

—. 2020. »https://www.nist.gov.« lipanj. Pokušaj pristupa 6. rujan 2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-77r1.pdf.

—. 2023. »https://www.nist.gov/.« 9. svibanj. Pokušaj pristupa 5. rujan 2023. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

—. 2020. »https://www.nist.gov/.« lipanj. Pokušaj pristupa 5. rujan 2023. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-16.pdf.

Yackel, Ryan. 2020. *https://www.keyfactor.com/.* 17. lipanj. Pokušaj pristupa 27. kolovoz 2023. https://www.keyfactor.com/blog/symmetric-vs-asymmetric-encryption/.

Yadav, Satya Prakash, Raghuraj Singh, i Vibhash Yadav. 2023. *Quantum-Safe Cryptography.* Berlin/Boston: De Gruyter.