Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проектна задача по предметот Информациска безбедност

тема:

Енкрипција како сервис

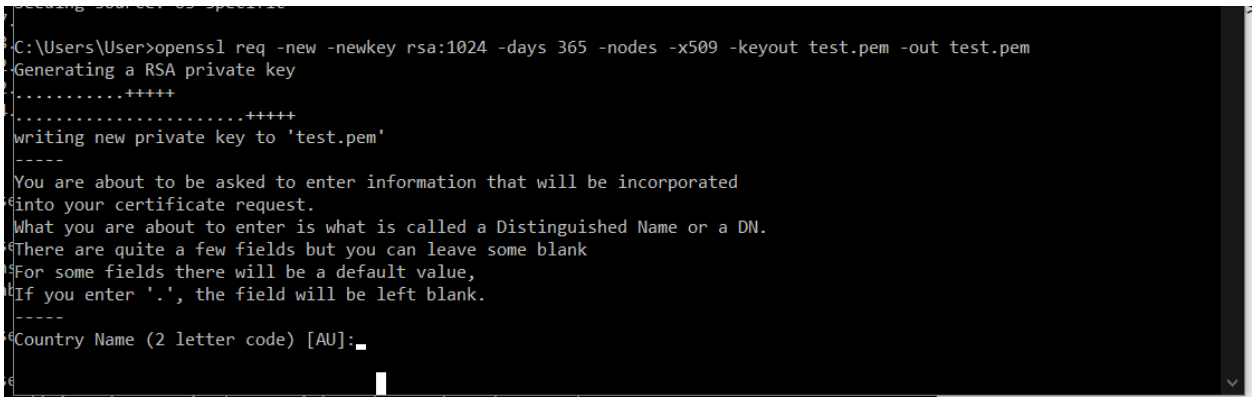
(EaaS)

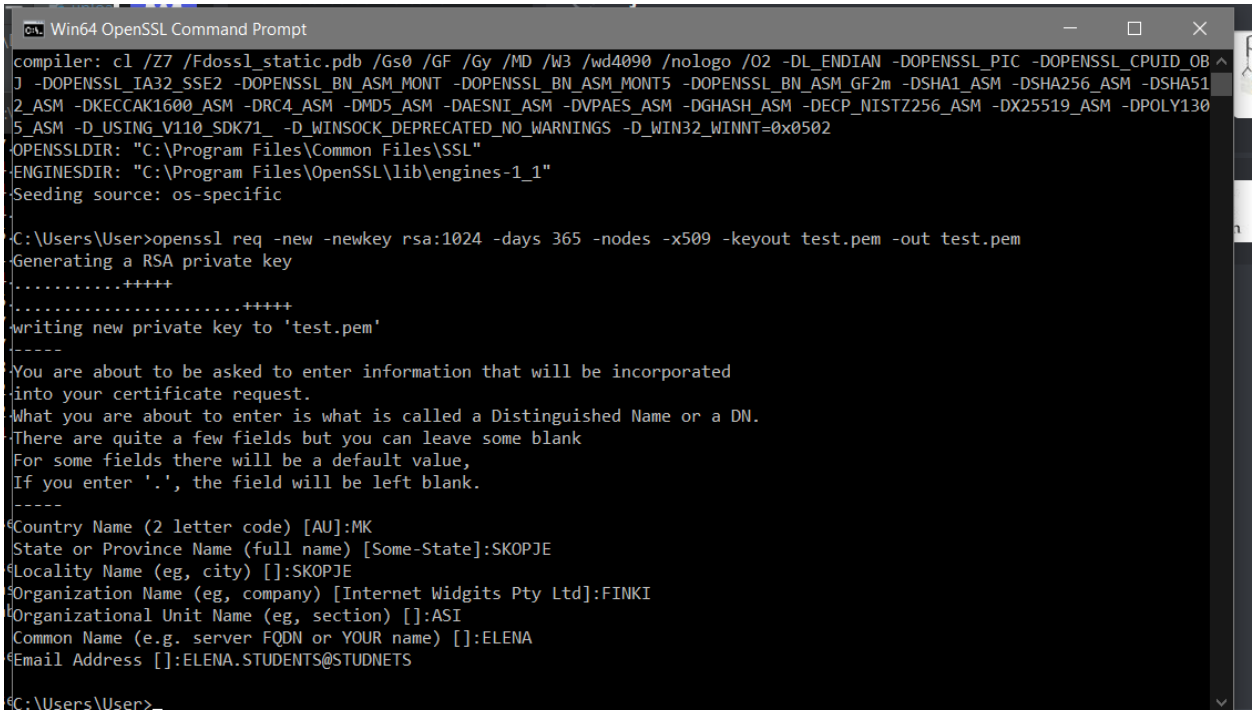
Изработиле:
176011 Елена Папазова
176008 Тијана Ѓорѓиевска
181274 Давид Здравковски
181273 Дамјан Здравковски

Документација

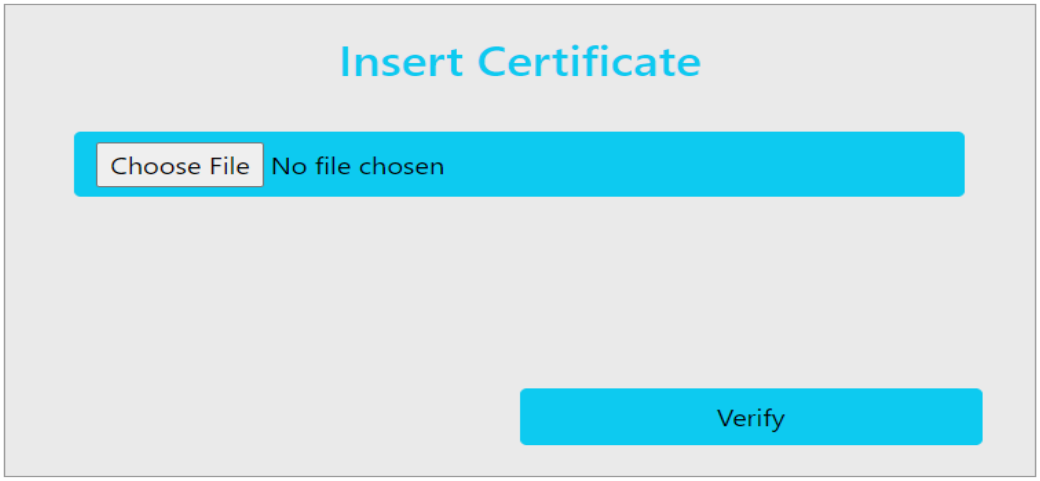
Овој проект за Енкрипција како Сервис целосно е изработен во Python Flask Frameworks, Flask е микро веб framework напишан во python. Се класифицира вака бидејќи не се потребни некои особени алатки или библиотеки за да функционира. Нема абстрактен слој за дата бази, валидации за форми или било какви други компоненти кои содржат веќепостоечки библиотеки направени од трети-лица кои што доставуваат популарни функции. Сепак, единствено нешто што Flask подржува се екстензии кои што можат да додаваат функционалности на нашата апликација и да функционираат исто како да биле напишани во Flask. Овие екстензии постојат за објектно-релационите мапери, валидација на форми, хендлирање прикачувања, различни технологии на отворена автентикација(oauth) и голем број чести framework алатки.
Поради овие екстензии и едноставноста за користење, Flask беше избран за токму овој проект. Ни овозможува да имаме едноставна форма на фронтенд-от и да ја управуваме или прилагодуваме рачно. Преку оваа форма го земаме сертификатот на корисникот и документите кои што сака да ги енкриптира. Преку POST/GET методите фронтен-от и бекенд-от комуницираат, со поставување функција на одредени рути можеме да слушаме и да ги дочекаме овие requests и да ги разработиме целосно на сервер страната. Преку декораторот @app.route од Flask правиме приватни рути преку кои кажуваме која фунцкија ќе се извршува кога се насочуваме на одредена страница.

Сертификатот го генерираме преку **openssl** алатката, каде што можеме лесно да специфицираме во каков формат сакаме да го издадеме, во овој случај x509(pem format), со 1024 битен приватен клуч.Потоа ни се бара да ги,запишуме потребните информации како име, презиме, емаил, држава и град на живеење итн, кои ќе се вметнат во сертификатот.

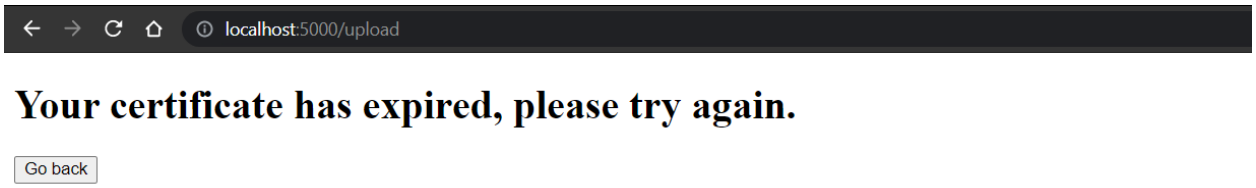


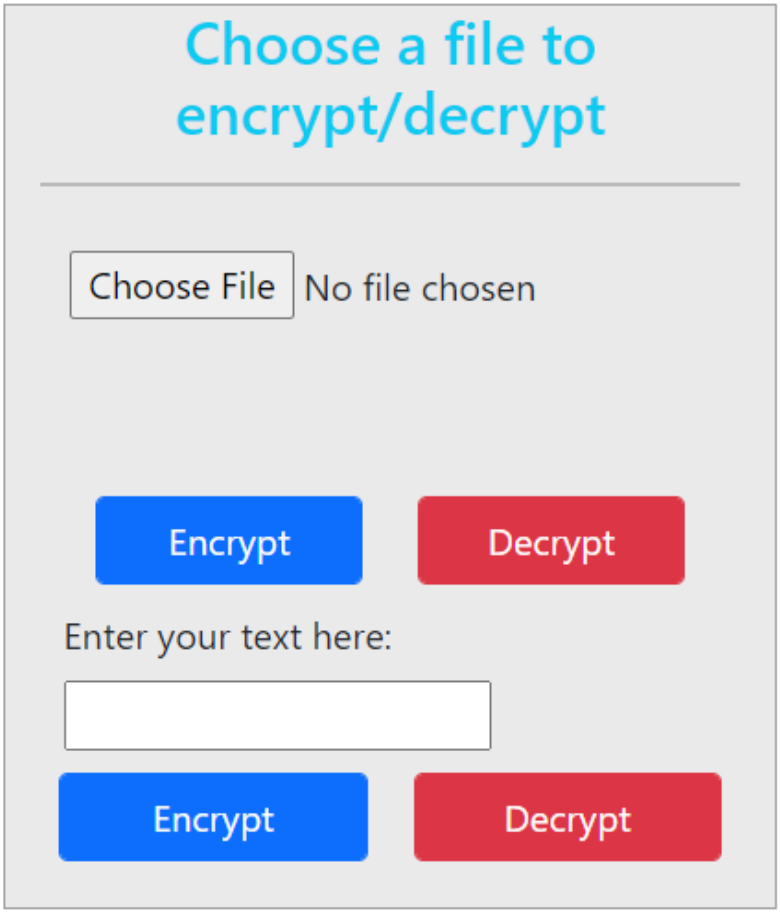


Почетната страна ни го претставува начинот на кој го внесуваме сертификатот и правиме проверка за неговата валидност. Внесувањето на сертификатот го правиме преку обичен HTML input type=”file” така што со кликање на submit во формата се испраќа Post Request со патека /upload и се повикува функцијата `upload_file()` која е точно наместена на оваа патека. Со повикот на оваа функција прикачениот сертификат се зачувува локално, т.ш. преку flask алатки го преземаме од веб прелистувачот. Сертификатот го читаме со готовата функција `x509.load_pem_x509_certificate()` од cryptography библиотеката во Python. Оваа функција го десеријализира сертификатот кој е енкодиран во base64 и ни враќа нов објект од кој можеме да го прочитаме датумот на валидност, клучот итн...



Преку функцијата check_date(cert проверуваме дали сертификатот е валиден доколку не е валиден на корисникот му се јавува порака дека сертификатот е со поминат датум, доколку е валиден со `SHA.new(cert.signature).hexdigest()` го хешираме неговиот потпис и го враќаме во хексадецимален број за да можеме да го запишеме како клуч во локалните фајлови. И за крај, со `check_cert(self)` фунцкијата проверуваме дали овој сертификат е веќе запишан во базата со неговиот доделен клуч, ако е тоа случајот, го извлекуваме неговиот клуч и го сетираме како активен, за понатамошна енкрипција/декрипција. Додека ако не постои запис за овој сертификат, генерираме единствен клуч кој само тој го има преку кој ќе може да го препознаваме и го запишуваме во keys.key фајлот како мапа од 'signature':'key'.





По успешната валидација можеме да прикачиме документ кој сакаме да го енкриптираме или да прикачиме веќе енкриптиран текст кој сакаме да го декриптираме, односно да го добиеме оригиналниот документ.

Доколку избереме документ кој сакаме да го енкрипираме со притискање на копчето Encrypt се вклучува функцијата `encrypt_file(self)` каде што со веќе претходно избраниот клуч го енкриптираме подесениот документ и потоа го зачувуваме во нов фолдер под ново име каде потоа ќе се препрати на корисникот и корисникот ќе може да го преземе.

Доколку избереме енкриптиран документ кој сакаме да го декриптираме со притискање на копчето Decrypt се вклучува функцијата `decrypt_file(self)` каде што со помош на клучот го декриптираме документот при што го зачувуваме во нов фолдер под ново име кој потоа ќе се препрати на корисникот и тој ќе може да го преземе.

Download your file here.

Go back

Во backend делот од кодот, со помош на Flask дефинираме одредени рути кои ги користиме од фронтенд-от и при секое пренасочување на овие рути, се повикува соодветната python функција со соодветната GET/POST метода. За landing page('/') рутата ја дефиниравме како

```
@app.route('/')
def index():
    return render_template('prva_strana.html')
```

Со тоа што вградената функција `render_template()` од Flask ни овозможува да вратиме целосен HTML фајл како резултат. На оваа прва страна, природно, на корисникот му е прикажана формата за валидација и при неговиот клик на submit

`<form id="login-form" class="form" method="post" enctype = "multipart/form-data" action = "http://localhost:5000/upload">`
Формата ќе испрати POST барање до наведената патека /upload, каде што соодветно ќе се повика функцијата `upload_file()` во која се првично ќе се изврши проверка со кој метод е направено барањето до нејзе и според тоа ќе се изврши одредената секвенција на акции. Во овој случај превземање на сертификатот и негова проверка.

```
@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['certificate']
        f.save("certificates/" + secure_filename(f.filename))
        with open('certificates/' + secure_filename(f.filename), 'r') as file:
            data = ''.join(file.readlines()).encode('utf-8')
            cert = x509.load_pem_x509_certificate(data, default_backend())
            print(f'UPLOAD FILE HEXDIGEST FROM CERT_SIGNATURE: {SHA.new(cert.signature).hexdigest()}', file=sys.stdout)
            if not check_date(cert):
                return render_template('not_valid.html')
            global handler
            handler.set_cert_hash(SHA.new(cert.signature).hexdigest())
            handler.check_cert()
        return redirect('/functions')
```

Со вградената класа request од Flask, можеме да пристапиме до сите фајлови кои што се прикачени на прелистувачот, овде специфично бараме од прелистувачот фајл прикачен со име ‘certificate’. Ова име ќе биде исто како дадената вредност на name атрибутот во input тагот: `<input type="file" id="formFile" name="certificate">`, на овој начин можеме да имаме повеќе input тагови и специфично да избираме кој сакаме да го повлечеме. Потоа го зачувуваме сертификатот локално во сервер, со `secure_filename(f.filename)` вградената Flask функција која ни овозможува безбедно да го зачуваме фајлот, без да се повикаат малициозни команди преку неговото име. Овој фајл, првично го енкодираме за да ја добиеме неговата содржина во битови и сета негова содржина ја конкатенираме во еден ред. Во ваков формат сега можеме да го испратиме во `x509.load_pem_x509_certificate(data, default_backend())` вградената функција од cryptography библиотеката, која ни ја декодира целата негова содржина. И сега можеме да испратиме проверка до наша функција за неговата валидност.

```
def check_date(cert):
    return cert.not_valid_before.timestamp() < datetime.datetime.now().timestamp() < cert.not_valid_after.timestamp()
```

Ако сертификатот е валиден, корисникот ќе биде пренасочен на /functions рутата каде што ќе може да продолжи до наредната страна каде што ќе му е дозволено да прикачува нов фајл, а ако е истечен рокот, тогаш се враќа нова страна во која добива порака дека сертификатот е истечен.
При валиден сертификат се препаќа неговиот потпис во хексадецимална форма до глобалниот објект од класата `EncService` која служи како хендлер за сите проверки и обработки на фајловите или сертификатите.
Во оваа класа се чува името на документот, клучот кој што е генериран и потписот на сертификатот. Објектот примарно служи за чување глобално на сите променливи коишто ќе ни се потребни за извршување на овој проект.

```
class EncService:
    def __init__(self):
        self.key = Fernet.generate_key()
        self.cert_hash = ''
        self.doc_name = ''
        self.text = ''

    def set_cert_hash(self, cert_hash):
        self.cert_hash = cert_hash

    def set_doc_name(self, doc_name):
        self.doc_name = doc_name

    def set_text_name(self, text_name):
        self.text = text_name

    def get_doc_name(self):
        return self.doc_name
```

Во класната функција `check_cert(self)` која се повикува откако ќе се постави потписот во хендлерот, се прави проверка дали корисникот претходно има постоечки клуч изгенерирано или се валидира за прв пат и е потребно да му се издаде нов клуч. Ако клучот е пронајден во фајлот keys.key, тогаш се поставува како главен клуч и функцијата се терминира, додека ако не постои веќе запишан клуч, тогаш се повикува `write_key(self)`функција која го запишува клучот во горенаведиот фајл во форма на: `keys_file.write('\n' + self.cert_hash + ":" + self.key.decode('latin1'))`

Во наредната страна, кога корисникот успешно ќе се валидира, на сличен начин се превземаат и фајлот за енкрипција и декрипција. Овој пат го пренасочуваме корисникот на друга т.н меѓу-рута во која ќе се провери дали корисникот сака да го енкриптира или декритптира овој фајл.

```
@app.route('/doc_upload', methods=['POST'])
def doc_upload():
    print(f'BUTTON CLICKED: {request.form["action"]}', file=sys.stdout)
    f = request.files['document']
    f.save("documents/" + secure_filename(f.filename))
    global handler
    handler.set_doc_name(secure_filename(f.filename))
    return redirect('/encrypt') if request.form['action'] == 'Encrypt' else redirect('/decrypt')
```

Ова е овозможено со додавање на атрибут ‘action’ на нашите копчиња во кое секое копче би имало соодветна вредност според неговата намена. Во оваа функција додатно се

додава името на фајлот којшто е прикачен од корисникот во нашиот глобален објект и понатаму го пренасочуваме корисникот на /encrypt или /decrypt соодветно.

```python
@app.route('/encrypt')
def encrypt():
    global handler
    handler.print_details()
    handler.encrypt_file()
    return render_template('download_file.html')
```

```python
@app.route('/decrypt')
def decrypt():
    global handler
    handler.print_details()
    handler.decrypt_file()
    return render_template('download_file.html')
```

Овие две функции се релативно симетрични, во двете ја повикуваме соодветна класна фунцкија за енкрипција и декрипција, сега веќе го имаме поставено името на документот и можеме слободно да манипулираме со него. И на крај се рендерира HTML страната за симнување на фајлот за корисникот.

```python
def encrypt_file(self):
    fernet = Fernet(self.key)
    with open(f'documents/{self.doc_name}', 'r') as file:
        enc_data = fernet.encrypt(''.join(file.readlines()).encode('utf-8'))
        print('ENCRYPTING DATA: ', enc_data, file=sys.stdout)
        with open(f'modified/client_file', 'wb') as enc_file:
            enc_file.write(enc_data)

def decrypt_file(self):
    fernet = Fernet(self.key)
    with open(f'documents/{self.doc_name}', 'r') as file:
        dec_data = fernet.decrypt(''.join(file.readlines()).encode('utf-8'))
        print('DECRYPTING DATA: ', dec_data, file=sys.stdout)
        with open('modified/client_file', 'wb') as dec_file:
            dec_file.write(dec_data)
```

Исто како и фунцкиите на рутите, класните функции за енкрипција и декрипција се исто така симетрични, каде што во променливата fernet го иницијализираме симетричниот криптувач со клучот кој што го извлековме или изгенериравме. Потоа овој фајл се запишува на сервер во фолдерот /modified под името ‘client_file’. На корисничката страна веќе ќе му биде прикажан линкот за симнување на овој фајл, и при клик на тој линк ќе се пренасочи корисникот на рута со името на фајлот, во овој случај client_file:

```python
@app.route('/<path:filename>', methods=['GET', 'POST'])
def download(filename):
    print('filename: ', filename, file=sys.stdout)
    return send_file('modified/client_file', as_attachment=True)
```

И овде со готовата функција `send_file()` фајлот ќе се симне локално на корисничката машина, каде што тој ќе може да се справа со него како што сака.
На сличен начин работи и внесувањето на текст, во тој случај текстот се вчитува од испратената форма со командата `text = request.form['text']` каде притоа во input тагот за текст ќе се специфицира атрибут ‘name’ со вредност ‘text’, наредно на исти начин преку хендлер објектот се повикува функција за да се енкриптира или декриптира според кликнатото копче. И секако на крај се рендерира HTML страната за симнување на фајлот.