Webová aplikace pro šifrování souborů

Jakub Oral

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jakub Oral**
Osobní číslo:	**A15061**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Webová aplikace pro šifrování souborů**
Téma anglicky:	**A File Encryption Web Application**

Zásady pro vypracování:

1. **Popiště možnosti implementace webových aplikací a nástroje.**
2. **Poreferujte o existujích nástrojích a aplikacích pro vzdálené šifrování.**
3. **Vyberte šifrovací algoritmy a rozepište, které budou implementovány.**
4. **Navrhněte způsob implementace.**
5. **Implementujte webovou aplikaci pro vzdálené šifrování souborů.**
6. **Vhodně reprezentujte výsledky a možnosti využití a úprav do budoucna.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **PAAR, Christof a Jan PELZL. Understanding cryptography: a textbook for students and practitioners. Berlin: Springer, 2010, xviii, 372 s. DOI: 978-3-642-04101-3. Dostupné také z: http://www.springerlink.com/content/ph2608/?p=0f30ce760cef42ef9f05cab0001587a0&pi=9**
2. **ŽÁČEK, Petr. Návrh nové symetrické šifry pro mobilní zařízení. Zlín: Univerzita Tomáše Bati veZlíně, 2014, 142 s. 3 s. příloh. Dostupné také z: http://hdl.handle.net/10563/30001. UniverzitaTomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav elektroniky a měření. Vedoucí práceMalaník, David.**
3. **Python 3.6.3 documentation [online]. 2017 [cit. 2017-11- 21]. Dostupné z:https://docs.python.org/3/**
4. **Django [online]. Django Software Foundation and individual contributors, 2017[cit. 2017-11- 21]. Dostupné z: https://www.djangoproject.com/**
5. **JAWORSKI, Michał a Tarek ZIADÉ. Expert Python Programming Second Edition.Second edition. Birmingham: Packt Publishing, 2016. ISBN 978-1-78588-685- 0.**
6. **BENDORAITIS, Aidas. Web Development with Django Cookbook. Second Edition.Birmingham: Packt Publishing, 2016. ISBN 978-1- 78588-677- 5.**
7. **HOLOVATY, Adrian. a Jacob. KAPLAN-MOSS. The definitive guide to Django: Webdevelopment done right. 2nd ed. New York: Distributed to the book trade worldwide bySpringer-Verlag, c2009. ISBN 978-1- 4302-1937- 8.**

Vedoucí bakalářské práce: **Ing. Petr Žáček**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **15. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

.....
podpis diplomanta

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a samotnou realizací webové aplikace pro šifrování souborů. Teoretická část této práce se zabývá možnostmi implementace takovéto aplikace pomocí webových nástrojů k tomu dostupných. Popisem již existujících aplikací pro vzdálené šifrování souborů. Odůvodněním výběru zvoleného nástroje pro tvorbu takovéto aplikace a následně její využití v praktické části této práce. Dále je v teoretické části rozebrána problematika moderních šifer, které mohou být v tomto ohledu využity pro šifrování souborů a jejich možnosti implementace v takovéto aplikaci. V praktické části se práce zabývá samotnou implementací této aplikace, jejím zprovozněním přes webové rozhraní. V závěru se práce zabývá popisem bezpečnosti celé aplikace včetně zabezpečení komunikace a popisuje zde možnost dalších úprav aplikace do budoucna.

Klíčová slova: AES, DES, 3DES, Smallie, šifrování, dešifrování, webová aplikace, kryptografie, Python, Flask, framework

ABSTRACT

This Bachelor thesis is about design and implementation of web application for file encryption. The theoretical part deals with possibilities of implementation of such application using web tools available for this purpose. Description of existing applications for remote file encryption. Rationalization of choosing used tools for creating such an application and then its use in the practical part of this work. The theoretical part also deals with modern ciphers which can be used for file encryption, their problems and implementation possibilities in such application. In practical part is described the implementation of application itself, its launching through the web interface. At the end, this thesis deals with security of this application including communication security . It describes possibilities of further modifications of the application.

Keywords: AES, DES, 3DES, Smallie, encryption, decryption, web application, cryptography, Python, Flask, framework

Touto cestou bych chtěl moc poděkovat zejména vedoucímu mé diplomové práce Ing. Petru Žáčkovi za cenné rady a konzultace ohledně bakalářské práce a poskytnutí zdrojových souborů pro implementaci šifry Smallie. Také bych chtěl poděkovat mé přítelkyni, která mi byla vždy oporou a mé rodině za toleranci při psaní této práce. Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Jelikož v dnešním světě je čím dál větší problém udržet nějakou informaci v tajnosti, je důležité myslet na bezpečnost dat. Existuje řada předpisů, nařízení a zákonů na ochranu osobních údajů a s nimi spojených záležitostí. Proto je nutné zajistit, aby veškeré citlivé údaje byly bezpečně chráněny. Ať už se jedná o seznamy jmen, přes čísla bankovních účtů, nebo o další osobní údaje nebo třeba plány nových výrobků a know-how. Všechny tyto informace mohou být zneužity, pokud se dostanou do rukou lidem, kteří vědí, jak je zneužít.

V historii první šifrované zprávy byly předávány již nějakých 480 let př.n.l. v dobách Řecko – Perských válek. Velice významnou roli v historii sehrál i Julius Caesar, který vytvořil vlastní substituční šifru, která spočívala pouze v posunutí abecedy o tři pozice.

V dnešní době známe mnoho technik pro šifrování zpráv, a proto si je v této práci také dále přiblížíme. Cílem této práce je vytvořit webovou aplikaci pro bezpečné šifrování zpráv na straně serveru a problematiku s tvorbou webové aplikace spojenou, což obnáší i nutnost znalosti fungování jednotlivých moderních šifer, které budou implementovány. Například DES, 3DES a AES. Dále jsme se také zaměřili, na již známé šifrovací webové aplikace, které jsou částečně vzorem pro tuto práci. Největší odlišností od těchto aplikací je implementace zcela experimentální šifry navržené Ing. Petrem Žáčkem, která funguje rozdílně než aktuálně využívané standardizované šifry. Tyto šifry jsou v dnešní době dostatečně bezpečné, ovšem může dojít k jejich prolomení, což by bylo dost závažným problémem pro velkou část světa. Zároveň se práce věnuje vývoji aplikace samotné v programovacím jazyku Python 3.x a implementací pomocí některého frameworku podporující webové aplikace v Pythonu.

I. TEORETICKÁ ČÁST

1 MOŽNOSTI IMPLEMETNACE WEBOVÝCH APLIKACÍ A NÁSTROJE K TOMU URČENÉ

Hned v úvodu této práce si musíme ujasnit spoustu pojmů, se kterými se setkáme během práce. Prvním z nich je rozdíl mezi webovou aplikací a webovou stránkou, dále si musíme něco povědět o vývojových prostředích, frameworcích, a poté se dostaneme k samotným aplikacím pro šifrování souborů. Následně si povíme něco o šifrování a šifrách.

1.1 Webová aplikace nebo webová stránka

Mnoho lidí si nedokáže ani v dnešní době představit rozdíl mezí tím co znamená pojem webová aplikace a webová stránka. Většina lidí totiž rozdíl mezi těmito věcmi ani nevidí. Což je na jednu stranu dobře a je tím dosaženo požadovaného výsledku, o který se programátor snažil. Podíváme-li se na webovou stránku, jedná se o tu část, kterou si zobrazí uživatel ve webovém prohlížeči a kterou vidí, tzv. front end. Pokud bychom se podívali na takovouto stránku detailněji, můžeme tvrdit, že webových stránek existuje naprosté minimum, jelikož už i stránka, která se jen nezobrazí, ale dokáže provádět určité operace, např. animace, prokliky a další, tak už je ve své podstatě také webová aplikace, pokud k těmto úkonům nevyužívá jen HTML a CSS. Nyní přicházíme k pojmu webová aplikace, jedná se tedy o samotné pozadí webové stránky tzv. backend. Tato část webu nám provádí veškerou logiku, která je implementována. Ve své podstatě se tedy jedná o to, že webová stránka nám slouží jen jako rozhraní mezi uživatele a serverem, kde se odehrávají veškeré operace, které webová aplikace provádí.

1.2 Vývojová prostředí

Doba pokročila a není již tak snadné vytvořit kvalitní webovou aplikaci bez užití různých frameworků a dalších doplňků. Programátorovi již nestačí sednout si k počítači, otevřít poznámkový blok a začít psát kód. Takovouto činnost dokáže provádět jen velmi úzká skupina lidí na světě, většina ostatních programátorů využívá různá vývojová prostředí, která umožňují zvýrazňovat klíčová slova používaná při tvorbě kódu. Vývojová prostředí označovaná též jako IDE, nám totiž nejen umožňují upravovat kód jako v klasickém editoru, ale zároveň chápou i logiku kódu, který vytváříme. V případě, že pracujeme s proměnnými, je IDE schopno nám nabídnout proměnné, které můžeme použít, také nám nabízí nápovědu k jednotlivým instrukcím, případně umožní zobrazit manuálové stránky, pokud jimi disponuje.

Nabízí se nám i možnost si kód zkompilovat a následně otestovat, případně se máme i možnost si kód tzv. debugovat, takže můžeme testovat jen určité části kódu, respektive nám umožní efektivní hledání chyb v kódu. K této možnosti můžeme používat i tzv. breakpointy, jsou to body, které si umístíme do kódu, a program se při provádění kódu na těchto bodech zastaví a my můžeme nahlédnout například do obsahu proměnných. Jedná se tedy o dnes již skoro nepostradatelnou věc, při vývoji softwaru, jelikož díky tomu můžeme využívat velké množství doplňků a pracovat mnohem efektivněji. [1]

1.2.1 Vývoj webových aplikací

Pro vývoj webových aplikací existuje v dnešní době velká řada možností. V podstatě každý si najde nejvhodnější jazyk, v jakém může aplikace vyvíjet. Do základního výčtu patří například JavaScript, PHP, Python, JQuery a další. S využitím patřičných frameworků potom není problém, vytvořit v podstatě skoro cokoliv co člověka napadne.

Několikrát zde již zmiňujeme slovo framework, ale zatím jsme se nesetkali s vysvětlením tohoto pojmu. Jedná se o jakousi podpůrnou strukturu při programování a vývoji aplikací. Frameworky obsahují již hotové části kódu, různé již hotové programy, různé knihovny API, které nám slouží například pro korektní komunikaci hardwaru a softwaru, různé návrhové vzory a metody. Programátor se tedy může soustředit na řešení svého problému a nemusí se zabývat již vymyšlenými věcmi, ty si jednoduše pomocí frameworku vygeneruje. [2]

JavaScript je programovací jazyk, který je syntaxí velice blízký jazyku *„C“,* je tedy označován jako tzv. „C-like“. Za zásadní rozšíření javascriptu se zasloužila společnost Google, která jej použila při tvorbě svého emailového klienta Gmail. Významným rozdílem například oproti PHP je ten, že javascript běží přímo v prohlížeči klienta a tudíž může docházet k dynamickým změnám. javascript je objektově orientovaný programovací jazyk a mezi jednu z výhod patří také možnost uložit do proměnné nejen hodnotu ale i funkci. Případy využití tohoto jazyka můžeme vídat každý den a to v podobě načítacího kolečka na webu, společnost Facebook pomocí javascriptu zajišťuje dynamické zobrazování nových zpráv na stránce bez nutnosti obnovit prohlížeč pomocí *„F5“.* Jako zajímavost je třeba dobré také uvést verzi známé hry Angry Birds pro prohlížeč Google Chrome, která je taktéž napsána v javascriptu. [3]

Na straně serveru potom stojí programovací jazyk PHP. Pomocí toho objektově orientovaného jazyka dokážeme vytvářet aplikace také, jenže na rozdíl od javascriptu, se vše vykonává na straně serveru a k uživateli jsou posílány již výsledky. Tudíž nic se nevykonává

dynamicky v prohlížeči u klienta. PHP nám vyřeší tedy například to, jak bude stránka vypadat, co se na ni má generovat ještě předtím než se nám zobrazí. Typickým příkladem může být přihlašování se do nějaké například privátní sekce stránky. V tomto případě se využívá také přístupů do databáze, díky této možnosti můžeme ukládat informace práce do databáze a nemusíme vytvářet například velké datové soubory obsahující tyto data.[4]

Nyní se pojďme konečně podívat na programovací jazyk Python, kterému se bude tato práce věnovat především. Pomocí samotného Pythonu si moc při vývoji webových aplikací nepomůžeme a je nutno si vybrat některý framework, se kterým budeme pracovat. Python jako programovací jazyk je oproti ostatním programovacím jazykům relativně jednoduchý. Jedná se o objektově orientovaný jazyk a beztypový jazyk. Beztypový jazyk znamená, že nezáleží, co do proměnné uložíme, protože jazyk sám rozpozná, o jaký typ proměnné se jedná podle jejího obsahu, zjistí tedy, jestli jde o integer, float, double, atd., pokud uložíme číslo, případně, zda se jedná o textové řetězce nebo proměnou s obsahem nějaké struktury. Python podporuje výjimky, což se nám může skvěle hodit pro zajištění regulérního běhu programu. Může pracovat s datovými soubory, jako jsou CSV nebo XML, můžeme využít práce s databázemi, vytvářet grafické rozhraní nebo můžeme program strukturovat jako vícevláknový. [5] Že se jedná o objektově orientovaný programovací jazyk, znamená, že můžeme k proměnným přistupovat jako k objektům, například můžeme vytvořit třídu zaměstnance, který bude mít nějaké jméno, příjmení, funkci, plat atd. a tato třída bude obsahovat vlastní metody například vypsání jména, platu, změna funkce atd.. Jestliže si takovou třídu vytvoříme, můžeme ji používat například v jiných projektech, v tom je právě výhoda objektových jazyků. Vždy máme nějaký objekt, který má své vlastnosti, metody a funkce.

Nyní se tedy pojďme pobavit o tom, jaké frameworky existují pro tvorbu webových aplikací pro Python.

Prvním ze zajímavých je framework s názvem CherryPy. Umožňuje tvorbu webových aplikací stejně jednoduše, jako tvorba normálního objektově orientovaného programu v Pythonu. Výsledkem je ovšem menší a jednodušší zdrojový kód. Tento Framework byl vyvinut již před deseti lety, a proto se dá říct, že je velmi spolehlivý a stabilní. Jednou z hlavních odlišností od ostatních frameworků je ten, že se nejedná o „full stack“ framework. To znamená, že nám neposkytne kompletní řešení všech problémů. CherryPy totiž nedokáže pracovat jako víceúrovňový systém a nedokáže také pracovat s uložištěm. Jedná se tedy spíše o backendové řešení. [6]

Druhým významným zástupcem je framework web2py. Základní vlastností web2py je ten, že je tvořen webovým administračním rozhraním, programátor si tedy nemusí nic instalovat do svého počítače, stačí si pouze stáhnout balíček s tímto frameworkem, který obsahuje veškeré komponenty potřebné pro zprovoznění. Pomocí administračního rozhraní lze nahrát novou aplikaci, navrhnout novou aplikaci, spravovat již existující aplikaci, testovat běžící aplikaci nebo třeba komunikovat s databází. To vše jde udělat i přes samotný shell operačního systému nebo shell Pythonu. [7]

1.2.2 Framework Flask

Jedná se o mikro framework, který je velmi přívětivý k uživateli. Je založen na WSGI utility knihovně Werkzeug a využívá šablonovací systém Jinja2. Pro vytvoření nejjednodušší webové aplikace s tímto frameworkem stačí necelých 5 minut. Tento Framework pracuje jako všechny výše zmíněné frameworky s programovací jazykem Python. Šablonovací systém spolu s flaskem zajistí to, že dokážeme zobrazovat webové stránky, které se nazývají šablony. Uživatel tedy při přístupu na server generuje požadavek na zobrazení stránky, pomocí základního python skriptu dokáže Framework vyhodnotit tento požadavek a vykreslí patřičnou šablonu. Můžeme tak propojovat velké množství skriptů v pythonu se šablonami. Pro velkou čistotu kódu a jednoduchost frameworku jej budeme používat v našem projektu. Existuje velké množství doplňků, které můžeme s tímto frameworkem použít. [8]

2 WEBOVÁ APLIKACE PRO ŠIFROVÁNÍ SOUBORŮ

Jedním ze základních způsobů zabezpečení dat je ochrana heslem a jeho následné využití jako klíče pro šifrování. Existuje řada aplikací, která dokáže zašifrovat soubor, složku, případně celý diskový oddíl a opatřit přístup k ní heslem.

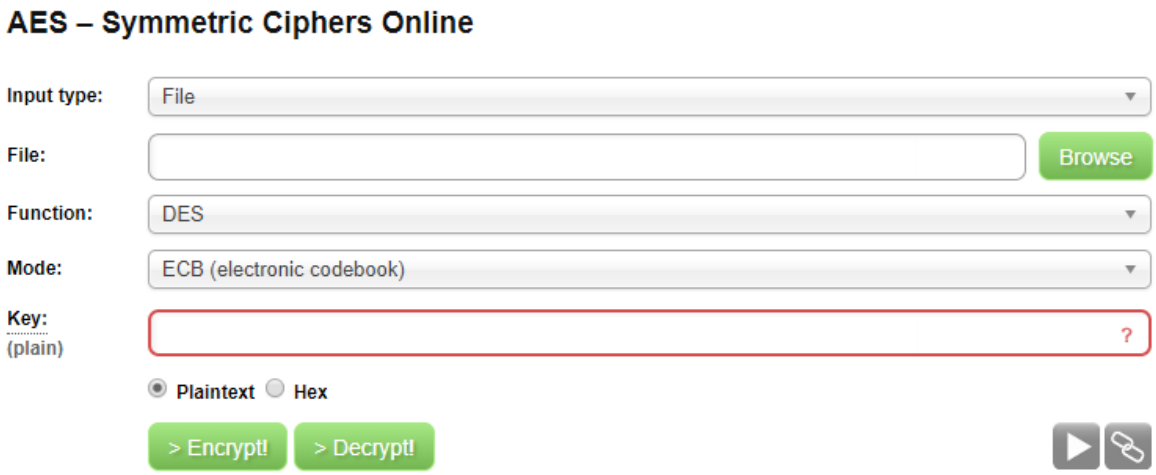
Jedním z takových základních nástrojů je BitLocker, software obsažený v některých verzích operačního systému Windows Server a v některých verzích Windows. BitLocker dokáže šifrovat jednotlivé oddíly pevného disku, např. tedy dokážeme pomocí něj zašifrovat flash disk. Při pokusu o přečtení dat ze zašifrovaného média dojde k dotazu na požadované heslo a následně teprve dojde ke zpřístupnění dat. V případě ztráty takového média je větší pravděpodobnost, že se nikdo nedostane k údajům, které jsme potřebovali šifrovat.[9]

Za jednu z předních šifrovacích aplikací by se dal označit software VeraCrypt, jako pokračovatele známe aplikace TrueCrypt. Dochází zde k šifrování souborů pomocí námi zvoleného šifrovacího algoritmu a zároveň si můžeme vybrat hashovací algoritmus, jako například SHA256 nebo SHA512. Aplikace šifruje soubory jako tzv. kontejnery tak, že se s nimi dá poté dále pracovat jako s normálním souborem, můžeme je přesouvat, kopírovat nebo smazat. Pokud ale chceme soubor otevřít, musíme to provést v softwaru VeraCrypt, dojde zde k zadání patřičného hesla, případně souborů s klíči a se souborem se poté pracuje jak s diskovým oddílem. Zároveň děšifrování souborů probíhá pouze v paměti RAM, nikdy nedochází k ukládání dešifrovaných souborů na disk. Vezmeme-li třeba zašifrovaný video soubor, který chceme spustit, tak načteme patřičný zašifrovaný soubor do VeraCrypt, dále musíme otevřít soubory pro dešifrování nebo zadat heslo. Posléze dojde k načítání souboru do paměti RAM a přehrávání ve standardním přehrávači. [10]

Online aplikací pro šifrování existuje velká řada, nejedná se o nic velice složitého, většina těchto aplikací pracuje na principu zadání textu, který požadujeme zašifrovat, následně si vybereme z množství šifrovacích algoritmů a zadáme klíč, pomocí kterého chceme náš text zašifrovat. Následně se provede samotné šifrování a získáme zpátky zašifrovaný text. Existují ale také aplikace, které dokáží zašifrovat v podstatě skoro jakýkoliv soubor. Takovýchto online aplikací není mnoho, proto byl jeden z důvodů pro vytvoření aplikace v této práci. Jestliže se podíváme na věc tak, že potřebujeme mít veškeré informace, které chceme šifrovat v bezpečí a chránit je před někým, tak zřejmě nechceme, aby se data ukládala do vzdáleného uložiště na server, o kterém nic nevíme. Neznáme bezpečnost tohoto serveru, nejsou

nám známi informace, jaké si ponechává a tak dále. Proto je potřeba mít dostatečně zabezpečený už samotný přenos souborů, které chceme šifrovat.

Jednou z takovýchto aplikací nalezneme na adrese *http://aes.online-domain-tools.com*. Tato webová aplikace nám umožňuje šifrovat jak otevřený text, který zadáme do textového pole, tak umožňuje také volbu šifrování souborů.



Obrázek 1 – Šifrovací aplikace aes.online-domain-tools.com

Po načtení souboru si vybereme, jakou šifru budeme chtít použít pro zašifrování našich dat, na výběr je z velké řady šifrovacích algoritmů ať už se jedná o šifry AES, DES, 3DES, Blowfish, RC4, atd. Dále si můžeme zvolit mód šifrování, který si budeme přát použít, popis těchto módů bude uveden dále v této práci, a zadáme heslo, pomocí kterého si přejeme naše data zašifrovat. První věc, na kterou ale můžeme narazit již při vstupu na stránku je to, že stránka nepoužívá komunikaci pomocí zabezpečeného protokolu HTTPS. Když tedy klikneme na tlačítko zašifrovat, můžeme v komunikaci se serverem vidět, že naše heslo, které jsme zadali je při přenosu v komunikaci viditelné. Zároveň ani pole pro zadávání hesla neobsahuje nahrazování znaků symboly při jejich vstupu z klávesnice a bezpečnostní slabina je tedy i v tom, že někdo může přečíst naše heslo již při jeho samotném zadávání. Jak probíhá šifrování na straně serveru, bohužel nemáme možnost zjistit, ale už na první pohled můžeme poukázat na nedostatky, které tato webová aplikace obsahuje.

Použití zabezpečeného spojení pomocí protokolu HTTPS je jedním z důležitých prvků při komunikaci mezi klientem a serverem. V případě, kdy použijeme přenos s certifikací SSL, máme větší pravděpodobnost toho, že námi odeslaná zpráva nebude přečtena nikým jiným než příjemcem, kterému jsme zprávu odeslali. Tato skutečnost je zajištěna dvojicí klíčů, privátním a veřejným, veřejný klíč je poskytnut všem kdo se chtějí účastnit komunikace a slouží

pro zašifrování zprávy a každý klient má svůj privátní a veřejný klíč. Privátním klíčem potom dochází k rozšifrování příchozí zprávy. Tento způsob šifrování je aplikací tzv. asymetrické kryptografie. U této aplikace je i velice pravděpodobné, že dochází při šifrování dat k lokálnímu ukládání.

3 ŠIFROVACÍ ALGORITMY

Dříve než se objevili vůbec první šifry, lidé se snažili své zprávy nějakým způsobem skrývat. Existovali různé triky jak zprávu doručit a nemusela být nějak šifrovaná, protože ji stačilo ukrýt na místo, o kterém věděl jen odesílatel a příjemce zprávy. Všem těmto metodám, kdy zprávu nešifrujeme, ale pouze ji skrýváme před zrakem všech, se nazývá steganografie. V moderní době je steganografie také stále populární a to především pro svoji jednoduchost oproti složitým šifrovacím algoritmům. Není problém například do rodinného alba z dovolené uložit citlivé údaje z firemních serverů a následně je vynést ven z firmy bez jakéhokoliv podezření. V průběhu let docházelo k vývoji kryptografie, ta se již zabývá samotným šifrováním zpráv, zde už nejde o to zprávu skrýt, ale změnit její podobu tak, aby bez znalosti nějakého klíče byla nečitelná. První šifra vznikla 500 let před naším letopočtem, byla jí šifra ATBASH, jednalo se o velice jednoduchou transpoziční šifru, kdy klasická abeceda o 26 znacích byla postavena naopak. [11]

$$abcdefghijklmnopqrstuvwxyz$$

$$zyxwvutsrqpomnlkjihgfedcba$$

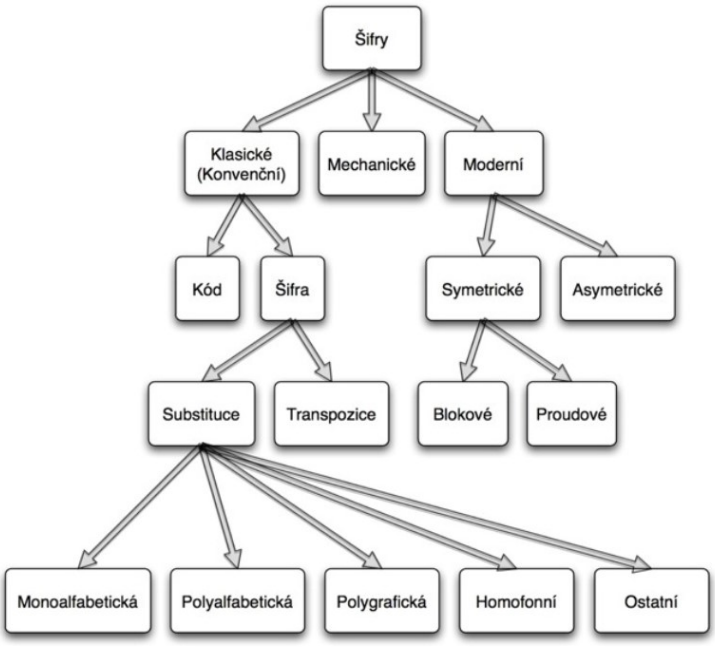
V 19. století došlo k prolomení šifer s klíči, takže můžeme vidět, že tyto šifry byli relativně dlouho dobu bezpečné. Během 1. a 2. světové války došlo k velkému rozvoji komplikovanějších šifer a různých šifrovacích mechanických systémů. Typickým příkladem je stroj Enigma, který byl vynalezen v roce 1918 německým rodákem Arthurem Scherbiusem. [12]

V roce 1973 došlo k objevení kryptografie s veřejným a privátním klíčem. Vznikl tak tedy nový pojem asymetrická kryptografie. Algoritmy pro šifrování a dešifrování v symetrické kryptografii jsou podstatně jednodušší a rychlejší, je to zapříčiněno tím, že pro šifrování i dešifrování používáme jen jeden klíč. Nevýhodou je ten, že musíme tento klíč nějak sdílet s ostatními účastníky, kterým chceme zprávu předat. Tím může snadno dojít k jeho odchycení a následnému prolomení šifry. S čím více uživateli budeme chtít komunikovat, potřebujeme mít pro každého unikátní klíč, což je značně náročné. U asymetrické kryptografie nám odpadá nutnost přenášet klíče pro dešifrování, jelikož komunikace probíhá s pomocí dvou klíčů, veřejného a privátního. Veřejný klíč nám slouží pro zašifrování zprávy a pomocí privátního klíče si každý účastník komunikace dokáže zprávu rozšifrovat. To také znamená, že na rozdíl od symetrické kryptografie nepotřebujeme mít tolik klíčů pro více uživatelů. Nevýhod je hned několik, mezi první patří velká výpočetní složitost, jelikož potřebujeme

počítat složité algoritmy pro šifrování i dešifrování. V případě některých šifer pracujeme například s 16-místnými prvočísly, se kterými provádíme složité aritmetické operace. Proto jsou také tyto šifry pomalejší. A dále také musíme dobře zabezpečit, udržovat a kontrolovat náš veřejný klíč. Kdokoliv by tento klíč totiž pozměnil, nebylo by dále možné zprávu číst a mohl by se k ní dostat někdo jiný.

Jak už to tak bývá, postupný vývoj došel ke vzniku hybridní kryptografie, která kombinuje výhody obou metod a snaží se eliminovat jejich špatné vlastnosti. V principu dochází k tomu, že hlavní data, kterých jsou velké objemy, jsou šifrovány pomocí symetrické šifry a následně klíč k této symetrické šifře je zašifrován asymetrickou šifrou. Dochází tak k velkému zrychlení celého procesu. Jelikož rychlá symetrická šifra nám zajistí bezpečnost přenášených dat a asymetrickou šifrou, zabezpečíme jen klíč, takže se stává také velmi rychlou. Následně jsou data a klíč k symetrické šifře zabaleny do jednoho balíku a odeslány. [11]

Šifry můžeme rozdělit podle obrázku níže. Tato práce se bude zabývat moderními šiframi, které si dále přiblížíme a následně se budeme věnovat jejich principům a naprogramování v praktické části této práce.



Obrázek 2 – Základní rozdělení šifer [11]

3.1.1 Moderní kryptografie

Tato práce se bude zabývat dále jen symetrickou kryptografií. Jak můžeme z obrázku výše vidět, tato kryptografie se dále dělí na šifry proudové a šifry blokové. Šifry proudové jsou

takové, kde zpracováváme otevřený text bit po bitu. Toto zpracování je výhodné především ve chvílích, kdy na počátku šifrování neznáme kompletní zprávu a potřebujeme zajistit šifrování otevřeného textu. V dnešní době se tedy takovéto šifrování využívá k šifrování přenosů dat, hovorů, šifrování komunikace přes wifi a mobilní komunikaci. Nejčastější operací při provádění šifrování proudových šifer je operátor XOR (exkluzivní OR). Klasická operace OR, nám zajišťuje disjunkci a operace XOR tedy zajišťuje exkluzivní disjunkci. Níže je uvedeno porovnání pravdivostní tabulky OR a pravdivostní tabulky logického operátoru XOR pro dva vstupy. Můžeme si všimnout, že operátor XOR zajišťuje, že na výstupu bude logická úroveň H pouze ve chvíli, kdy je pouze jedna vstupní proměnná v úrovní H. Všechny ostatní stavy budou indikovat na výstupu logickou úroveň L. Na rozdíl od logického operátoru OR, který má na výstupu logickou úroveň L jen ve chvíli, kdy jsou všechny vstupní proměnné v úrovni L.

XOR		
X_0	X_1	Y_0
L	L	L
L	H	H
H	L	H
H	H	L

Tabulka 1 – Pravdivostní tabulka XOR

OR		
X_0	X_1	Y_0
L	L	L
L	H	H
H	L	H
H	H	H

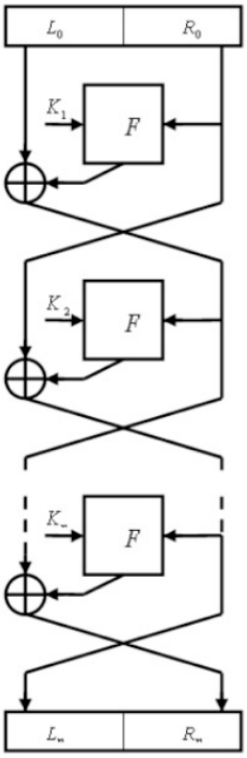
Tabulka 2 – Pravdivostní tabulka OR

Mezi nejznámější zástupce těchto proudové kryptografie patří šifry RC4, FISH, A5/1, A5/2, HELIX,…

Tato práce se bude dále zabývat blokovými šiframi. Tyto šifry mohou být použity pouze v případě, kdy známe již na počátku šifrování celý otevřený text, který potřebujeme zašifrovat. Blokové šifry si následně otevřený text rozdělí na bloky určité velikosti a doplní poslední blok vhodným způsobem na požadovanou délku. Šifrovaný text se získává nejčastěji z otevřeného textu pomocí tzv. rundové funkce, probíhá zde tedy iterační proces. Dochází tedy

k určité operaci s daty a tato operace se několikrát opakuje. Vstupem do takovéto rundové funkce je klíč a výstup z předchozí iterace. Blíže bude tento princip vysvětlen u jednotlivých šifer, kterými se budeme zabývat. Specifikem blokových šifer je to, že otevřený text i výsledný šifrovaný text mají pevnou délku. Což vychází z požadavku na nutnost znát celý otevřený text již na začátku šifrování.

Většina blokových šifer používá při svých operacích tzv. Feistelovu strukturu. Feistelova šifra pak následně definuje způsob, jakým jsou data zpracovávána v šifrovacích systémech, nejedná se tedy o konkrétní algoritmus. Jak můžeme vidět na příkladu, který je uveden na obrázku níže, pomocí Feistelovy šifry dojde k rozdělení otevřeného textu na dvě poloviny L_0 a R_0. Tyto části jsou postupně zpracovávány pomocí rundových (Feistelových) funkcí s využitím podklíče. V každé další iteraci se stává pravá část otevřeného textu levou, a levá část pravou. [11]



Obrázek 3 – Feistelova struktura [11]

Základní rozdíly mezi blokovými šiframi jsou především tyto: „

- Velikost bloku – 64, 128, 192, 256, 312, 512 (obvykle násobky čísla 8 → bajt, 8 bitů)
- Délka klíče – často stejná délka, jako je délka bloku
- Operace prováděné nad bloky – XOR, sečtení modulo, násobení modulo, substituce, transpozice (permutace), bitový posun
- Počet rund (provedení sledu operací nad blokem) – 8.5, 10, 12, 14, 16
- Struktura rundy, poskládání operací
- Principem generování inicializačního vektoru
  Režimem činnosti, generování klíče pro následující blok dat “ [13]

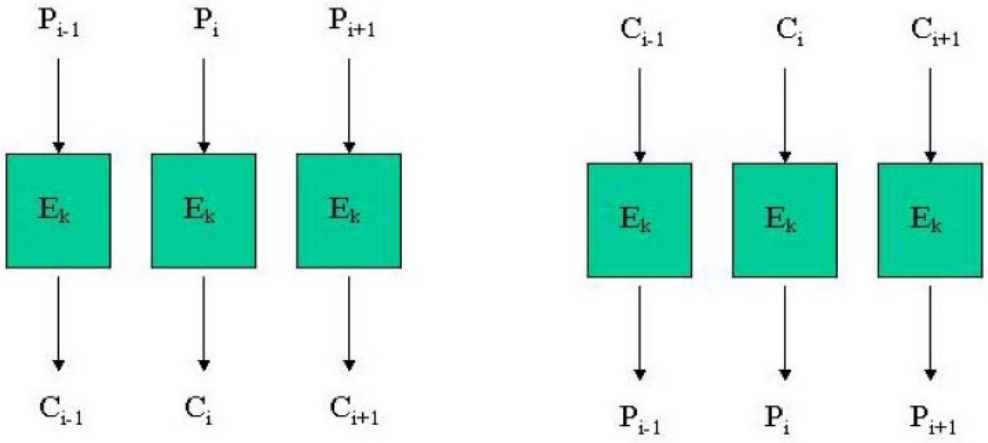
Pokud se podíváme na šifrovací aplikace, které má běžný uživatel dostupné na internetu, můžeme dojít k tomu, že v dnešní době je stále v popularitě šifrovací algoritmus AES a 3DES. Většina těchto aplikací obsahuje možnost šifrovat pomocí různých algoritmů. V žebříčku pěti nejpoužívanějších šifer můžeme nalézt šifru 3DES, AES a Blowfish.[14] Jelikož se jedná o šifry v dnešní době hojně používané a zároveň jim v průběhu studia bylo věnováno hodně času, rozhodli jsme se použít v aplikaci právě šifry 3DES, AES a pro úplnost i z historického hlediska jednu z prvních bezpečných, ačkoliv nyní už ne, blokových šifer a to šifru DES. Další šifrou, kterou bude tato aplikace obsahovat je nová bloková šifra Smallie, která byla a stále je vyvíjena Ing. Petrem Žáčkem na Fakultě aplikované informatiky ve Zlíně v rámci doktorského studia. [13]

3.1.2 Správa klíče – režim činnosti

Jedním ze základních principů celé kryptografie je, aby nikdy nebyl při šifrování stejných textů (dat) použit stejný klíč. Na tento problém naráží především moderní blokové symetrické šifry, u kterých jsou šifrovaná data delší než délka bloku. Nastával by zde problém, že by různé po sobě jdoucí bloky byli šifrovány pomocí stejného klíče, vystupovali by zde tedy stejné sady rundových podklíčů. Z tohoto důvodu byli standardizovány různé režimy činnosti, které popisují, jakým způsobem se šifrují data, jejichž délka je větší než délka bloku a jakým způsobem se zde pracuje s tzv. inicializačním vektorem. Jedná se o určitý stochastický (náhodný nebo pseudonáhodný) prvek, který nám pomáhá spolu s klíčem zajistit, že pro stejná data se stejným klíčem dostaneme různý výstup. [11] Uvedeme zde tedy několik režimů činnosti.

3.1.2.1 Režim činnosti ECB – Electronic Code Book

Jedná se o zcela základní režim, bez jakékoliv správy klíče a tudíž pro všechny bloky je použit ten samý klíč.[13]



Obrázek 4 – Režim činnosti ECB [11]

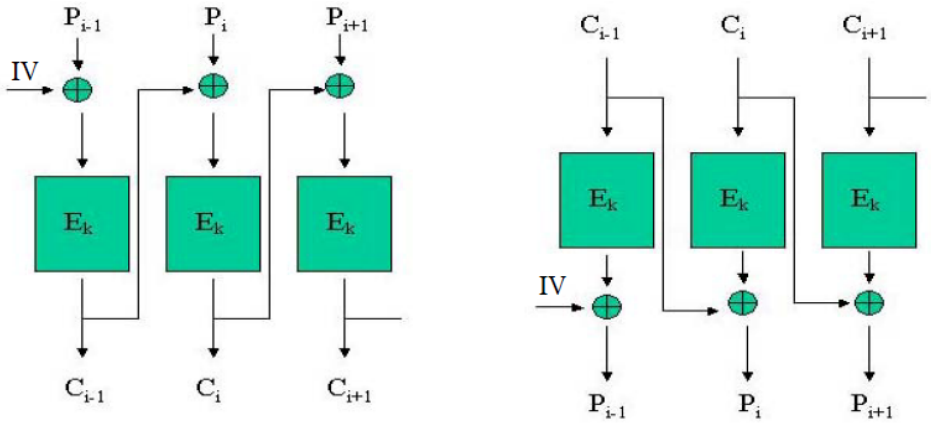
$P_i - vstupní\ otevřený\ text$

$E_k - šifrovací\ algoritmus$

$C_i - výstupní\ šifrovaný\ text$

3.1.2.2 Režim činnosti CBC – Cipher Block Chaining

V tomto režimu dochází k provádění logické funkce XOR mezi již zašifrovanými daty a otevřeným textem. K této operaci dochází vždy před vstupem do šifrovacího algoritmu. Výjimku tvoří první blok, kde nemůžeme použít předchozí výstup, jelikož žádný nemáme. Za tímto účelem právě využíváme již zmíněný inicializační vektor. [13] Schématická ukázka činnosti režimu činnosti je znázorněna na následujícím obrázku.



Obrázek 5 - Režim činnosti CBC [11]

$P_i - vstupní\ otevřený\ text$

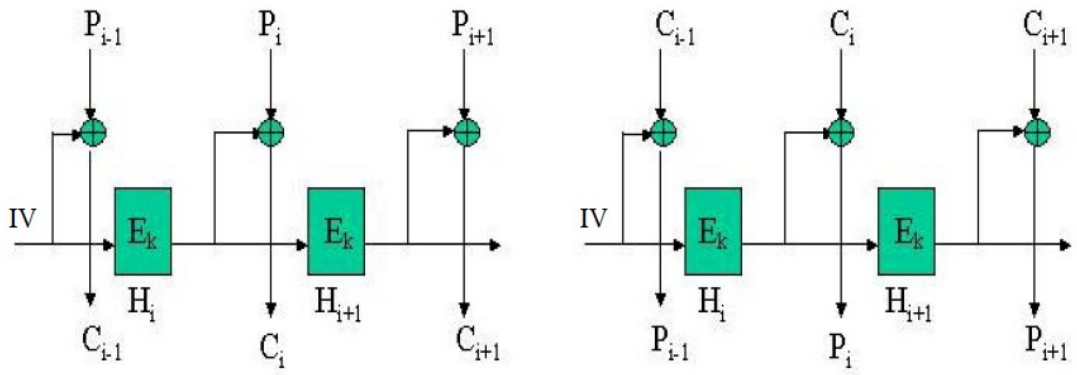
$E_k - šifrovací\ algoritmus$

$C_i - výstupní\ šifrovaný\ text$

$IV - inicializační\ vektor$

3.1.2.3 Režim činnosti OFB – Output FeedBack

Tento režim činnosti má za úkol simulovat proudovou šifru, dochází zde k provádění operace XOR mezi bloky nezašifrovaných dat a mezi výstupem blokové šifry. To znamená, že blokové šifra zde působí jako generátor pseudonáhodné posloupnosti. Do prvního bloku je opět místo výstupu blokové šifry použit jako jeden ze vstupů inicializační vektor.[11] Schématická ukázka činnosti režimu činnosti je znázorněna na následujícím obrázku.



Obrázek 6 - Režim činnosti OFB [11]

$P_i - vstupní\ otevřený\ text$

$E_k - šifrovací\ algoritmus$

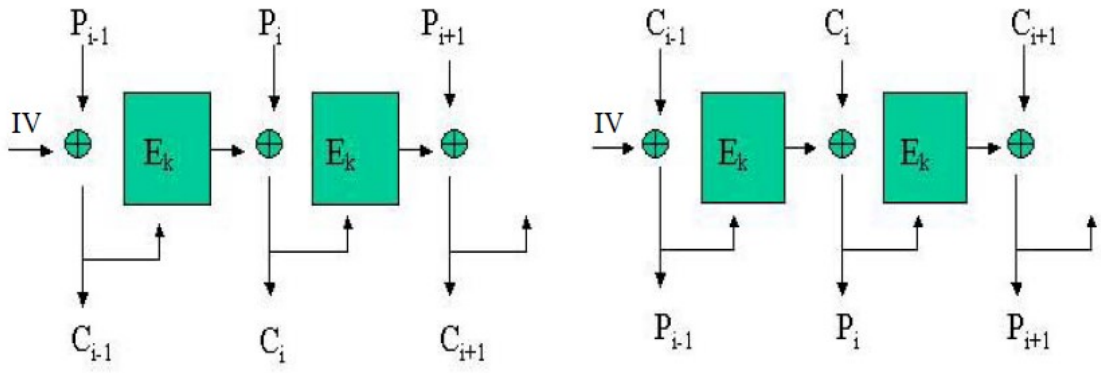
$C_i - výstupní\ šifrovaný\ text$

$H_i - Hodnota\ pseudonáhodné\ posloupnosti$

$IV - inicializační\ vektor$

3.1.2.4 Režim činnosti CFB – Ciphertext FeedBack

Princip je z velké části velice podobný jako OFB – Output FeedBack. Opět se jedná o simulační režim proudové šifry, je zde použit logický operátor XOR mezi vstupním otevřeným textem a výstupem šifrovacího algoritmu blokové šifry. U prvního bloku je opět místo výstupních dat blokové šifry použit inicializační vektor. [13] Schématická ukázka činnosti režimu činnosti je znázorněna na následujícím obrázku.



Obrázek 7 - Režim činnosti CFB [11]

$P_i - vstupní\ otevřený\ text$

$E_k - šifrovací\ algoritmus$

$C_i - výstupní\ šifrovaný\ text$

$IV - inicializační\ vektor$

3.1.3 Režim PKCS7

V režimu PKCS7, jsou doplňovány bloky dat velice jednoduše. Doplňované data jsou tvořena vždy hodnotou počtu bytů, které se doplňují. Můžeme to znázornit pomocí následující posloupnosti. Pokud tedy potřebujeme doplnit N bytů, budeme mít N bytovou hodnotu N krát.[15]

01
02 02
03 03 03
04 04 04 04
atd.

Pokud tedy budeme mít data o velikosti bloku 4 byty, bude vypadat zpráva následovně

$$\ldots | DD\ DD\ DD\ DD\ |DD\ \mathbf{03\ 03\ 03}$$

3.1.4 Režim ISO/IEC 7816

Princip toho paddingu je velice jednoduchý. Vždy dojde k doplnění číselné hodnoty 80 na konec dat, které potřebujeme doplnit a následně je zpráva doplněna až do velikosti bloku pomocí nulových byte 00. Takže pokud tedy budeme doplňovat například 4 bytovou zprávu, která má v posledním bloku jen jeden byte, doplníme zprávu následovně. [15]

$$\ldots | DD\ DD\ DD\ DD\ |DD\ \mathbf{80\ 00\ 00}$$

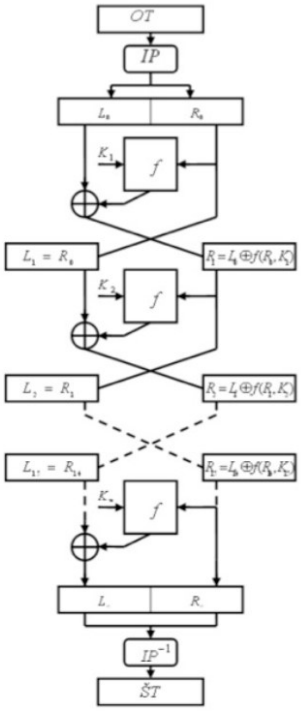
3.1.5 Režim ANSI X.923

Tento režim nám zajistí doplnění bloku dat pomocí nulových bytů. Poslední doplněný byte obsahuje informaci o tom, kolik bytů bylo doplněno. Byty, které se doplňují, jsou tvořeny nulami. Pokud tedy budeme uvažovat 4 bytovou zprávu, v posledním bloku, který bude obsahovat například jen jeden byte dat a bude třeba jej doplnit na požadovanou délku 4 byty, bude situaci vypadat tak, že se doplní dva bloky 00 a poslední třetí blok 03 a právě trojka znázorňuje, kolik bytů dat se muselo doplnit. Celá situace je znázorněna níže.[15]

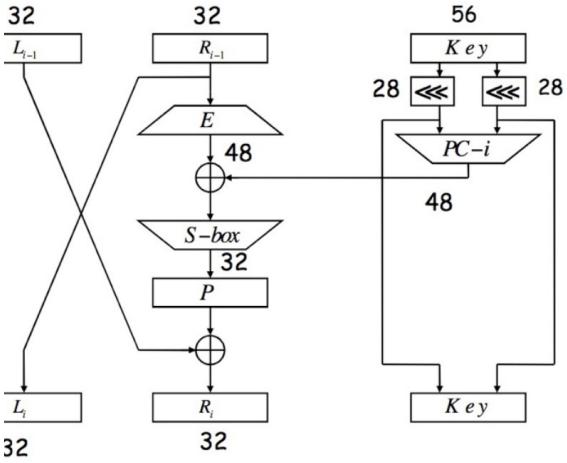
$$\ldots \mid DD\ DD\ DD\ DD \mid DD\ \mathbf{00\ 00\ 03}$$

3.1.6 Šifra DES

Tato šifra byla jednou ze základních blokových šifer. Klíč u této šifry je tvořen pomocí 64 bitů, šifrovací algoritmus ovšem využívá pouze 56bitů, jelikož zde dochází ke kontrole správnosti a k tomu slouží každý 8 bit, který je paritní. Šifrovací algoritmus je postaven na S-Boxech, které provádí substituci, neboli nahrazování jednotlivých hodnot bitů, jinou hodnotou z tabulky a P-Boxech, ty se starají o transpozici, tudíž zde probíhá záměna jednotlivých bitů v bloku. Operace v šifrovacím algoritmu jsou provedeny 16 krát, tudíž je zde provedeno 16 rund. [13] Průběh šifrovacího algoritmu je znázorněn na následujícím schéma.



Obrázek 8 – Šifrování pomocí algoritmu DES [11]



Obrázek 9 – Průběh šifrování v průběhu jedné rundy [11]

V roce 1997 byla společností RSA DSI vypsána soutěž o 10 000 dolarů, za rozluštění zprávy zašifrované pomocí šifry DES. Z toho důvodu vznikl program, který dokázal běžet na pozadí počítačů, které zrovna neprováděli žádné operace. Tento software měl za úkol hledat klíč k dešifrování zprávy. Do toho projektu se zapojilo kolem 78 tisíc počítačů. Ovšem toto prolomení klíče trvalo příliš dlouho. V roce 1998 vznikl software DES Cracker, který dokázal nalézt klíč již do 5 dnů. [11]

3.1.7 Triple DES

S ohledem na prolomení šifry DES vznikla v roce 1998 nová šifra s názvem Triple DES. Rozdíl oproti šifře DES je ten, že je algoritmus použitý pro šifru DES aplikován 3x za sebou. Velikost klíče vzrostla na trojnásobek také, tudíž užitečných zde bylo 168 bitů klíče. V tomto ohledu existuje několik různých variant této šifry.

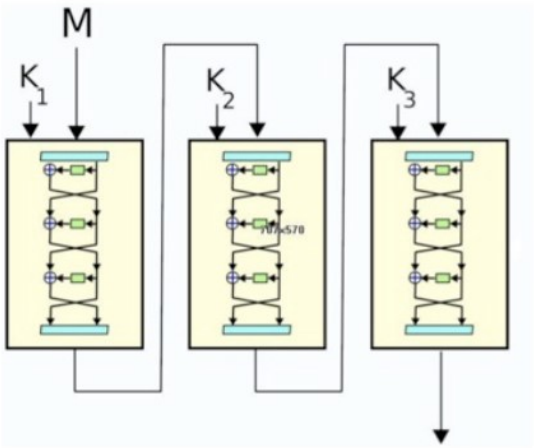
DES EEE3 – zde jsou použity tři různé klíče pro šifrování v každé rundě.

DES EDE3 – v této metodě jsou také použity tři různé klíče, zpráva je pomocí prvního klíče zašifrována, pomocí druhého klíče dešifrována a pomocí třetího klíče opět zašifrována.

DES EEE2 – v tomto algoritmu jsou použity jen dva klíče, nejdříve je zpráva zašifrována prvním klíče, následně je zašifrována druhým klíčem a nakonec je opět šifrována prvním klíčem.

DES EDE2 – nejpoužívanější metoda je právě tato, kdy se používají opět dva klíče, pomocí prvního klíče probíhá zašifrování zprávy, pomocí druhého klíče se zpráva dešifruje a následně je opět zašifrována pomocí prvního klíče. [11]

Níže je uvedeno schéma principu šifrování pomocí Triple DES šifry.



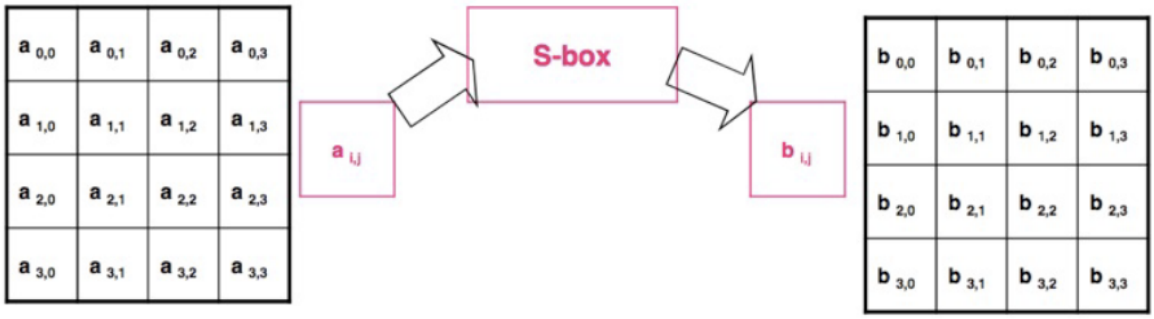
Obrázek 10 – Schéma šifrování v šifře 3DES [11]

3.1.8 Šifra AES

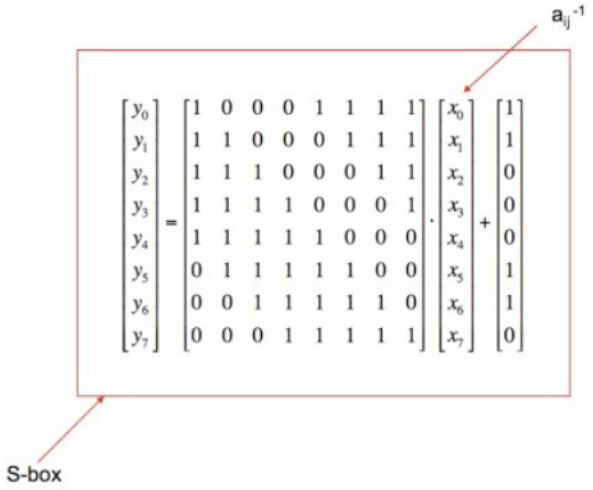
Vznik této šifry se pojí s rokem 1998, stejně jako šifra Triple DES. Jelikož došlo k prolomení klasické šifry DES, byla snaha o vytvoření nových šifer. Šifrovací algoritmus zpracovává data po blocích o velikosti 128, 192 nebo 256 bitů. V závislosti na velikosti těchto bloků jsou proměnlivé i délky klíčů, ty jsou ve velikostech 128,192 a 256 bitů. Rychlost šifrování dat pomocí tohoto algoritmu přesahuje 45 MB/s. Zároveň se s délkou bloků a délkou klíče mění i počet rund, ve kterých je šifrování prováděno, standardně 10, 12 nebo 14 rund. [11]

Při šifrování pomocí AES jsou použity následující operace:

ByteSub Transformation – jedná se o nelineární vrstvu, ve které je prováděna substituce s jednotlivými bajty dat. Tato vrstva nám zajišťuje odolnost proti lineární a diferenciální kryptoanalýze. [13]

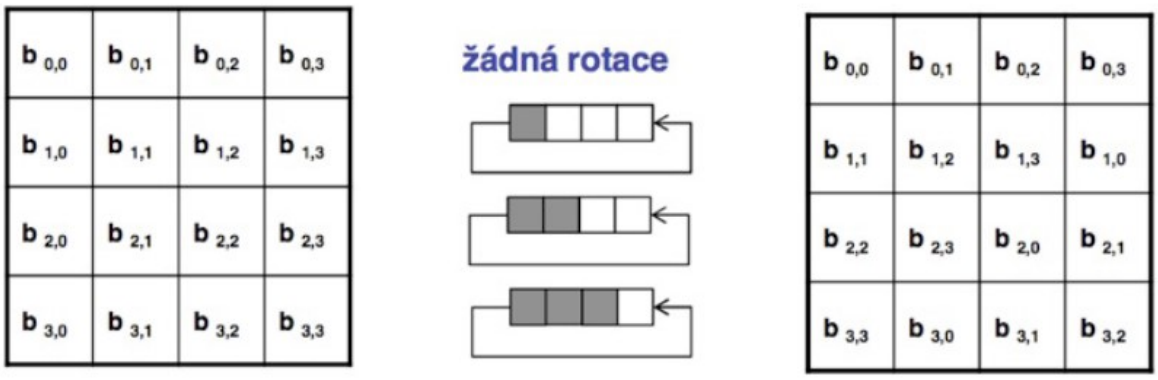


Obrázek 11 – ByteSub Transformation AES [11]



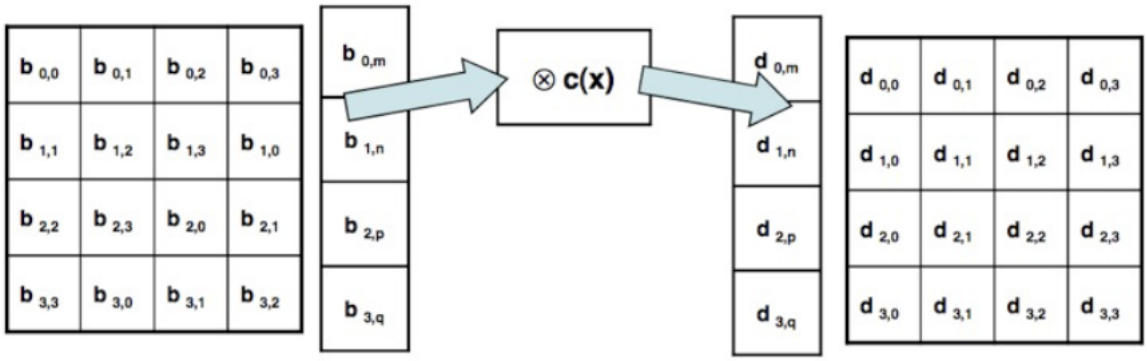
Obrázek 12 – Struktura S-Boxu u AES [11]

ShiftRow Transformation – jde o lineární vrstvu, která provádí v tabulce posun bitů v cyklickém pořadí. [11]



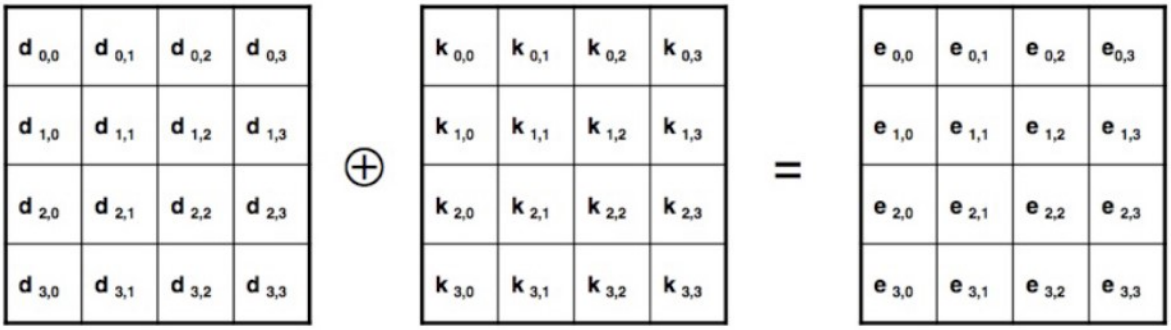
Obrázek 13 – ShiftRow Transformation AES [11]

MixColumn Transforamtion – v této operaci se provádí mixování jednotlivých buněk tabulky, jednotlivé sloupce jsou násobeny polynomem a dále je na ně použita logická operace XOR. [11]



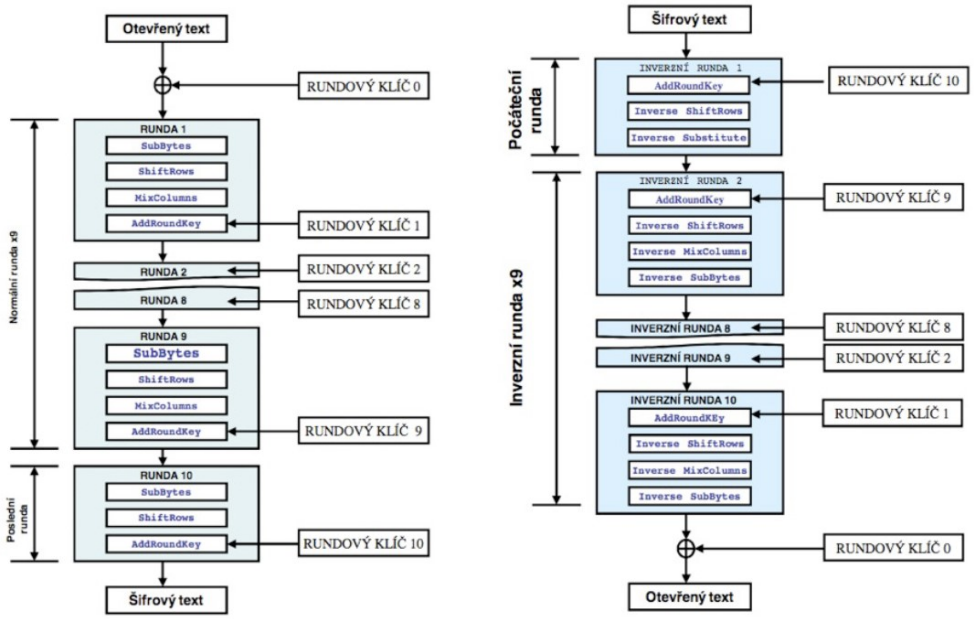
Obrázek 14 - MixColumn Transforamtion AES [11]

Add Round Key – dojde k přičtení rundového klíče pomocí logické funkce XOR. Tento klíč je dán pomocí algoritmu pro plánování rundového klíče. [11]



Obrázek 15 - Add Round Key AES [11]

Dále je zde znázorněn celý průběh šifrování a dešifrování pomocí algoritmu AES pro 10 rund.



Obrázek 16 – Šifrování a dešifrování pomocí AES [11]

3.2 Šifra Smallie

Jedná se o experimentální šifru vyvíjenou Ing. Petrem Žáčkem. Šifra byla implementována ve verzi označené číslem 1.0.1. Tato šifra je odvozena od jeho diplomové práce [13]. Šifra smallie zde nebude rozebrána do podrobna, jelikož je stále ve fázi vývoje a podrobný popis bude obsahem disertační práce Ing. Petrem Žáčkem. Nicméně, cílem zde bylo provést první implementaci a to zejména ke sběru statistických údajů pro další analýzu. Z výše uvedených důvodů byla provedena implementace na straně serveru, aby zatím nebylo možné číst její zdrojové kódy. Každopádně zde uvedeme krátký popis.

Šifra je symetrická a bloková, tudíž využívá stejný klíč pro šifrování i dešifrování. Klíč k šifrování se odvozuje od klíčových souborů o minimální velikosti 128 bajtů libovolného obsahu. Šifra se chová polymorfně, to znamená, že její struktura a většina funkcionalit je závislá na vstupních parametrech a nastavení. Při změně zejména vstupního klíče (hesla) dojde k nastavení šifry a následně dochází i k nastavení podmínek pro šifrování každého nového bloku otevřených dat. Účelem tohoto fungování je znemožnění útočníkovi znát strukturu a průběh šifrování bez znalosti vstupního klíče. Tudíž délka klíče je pro každý blok vypočítávána zvlášť na základě řídicích parametrů závislých na klíči, konkrétně v rozmezí 101 až 111 bajtů (808 až 888 bitů). Velikost šifrovaného bloku je závislá na klíči, tudíž šifra je schopna šifrovat bloky libovolné délky, a proto není potřeba paddingu.

Některé vstupní parametry jsou možné měnit uživatelem, což zajistí nárůst množiny různých „typů“ a nastavení šifry. Nicméně v případě změny parametrů uživatelem je nutné nastavení stejných parametrů i při dešifrování. V opačném případě dešifrování proběhne, ale nezískáme data jako při původním nastavení. Pokud je uživatel nezmění, neznamená to, že nedojde ke specifickému nastavení, protože i v defaultním nastavení parametrů se využívá vstupní klíč. Parametry, které lze měnit uživatelem jsou následující a mohou nabývat hodnot <0, 255>:

- Parametr pro počet rund
- Řídicí proměnná pro určení počtu rund
- Řídicí proměnná pro určení délky bloku
- Parametr X pro přepočet řídicích proměnných
- Parametr N ovlivňující výpočet následujícího klíče

Speciální parametr je parametr přesnost, který je dostupný pouze pro režim činnosti PM-DC-LM k nastavení inicializačního vektoru režimu činnosti. Tento parametr může mít hodnoty na intervalu <9, 14>.

Dále šifra využívá dvou režimu činnosti blokových šifer, a to následující:

- PM -> původně upravený návrh z diplomové práce
- PM-DC-LM -> experimentální polymorfní režimy činnosti

[13]

3.3 Použití SSL certifikátů

Protokol SSL nám zajišťuje důležité funkce při komunikaci mezi klientem a serverem. Jelikož při komunikaci chceme docílit toho, aby klient komunikoval právě a jen s tím serverem, na který odeslal dotaz a čeká na něj odpověď. Toto nám právě pomáhá zabezpečit SSL protokol. Další a zároveň hlavní funkcí tohoto protokolu je šifrování dané komunikace. Při navázání spojení nám protokol SSL zprostředkuje to, že se klient a server domluví na šifrovacím algoritmu, který bude zabezpečovat komunikaci a zabezpečí výměnu klíčů. To, že je při komunikaci používáno šifrované spojení, které používá právě protokolu SSL, můžeme poznat podle toho, že před názvem webové stránky vidíme zkratku HTTPS://. Tato zkratka znamená „Hypertext transfer protocol secure“. To ovšem neznamená, že klient je zcela v bezpečí, pokud používá šifrované spojení, ale můžeme říct, že se jedná o mnohem bezpečnější komunikaci, jelikož vše co si vyměňuje klient a server je šifrováno. Všechny dnes

již důvěryhodné instituce jako banky, úřady, eshopy velkých značek, atd., používají právě při komunikaci nějaký certifikát pro ověření pravosti, zda uživatel komunikuje se serverem, se kterým chce komunikovat. Certifikáty vydávají instituce, které si za vystavení takovéhoto certifikátu účtují poplatek. Společnost, která si chce nechat vystavit takovýto certifikát, doloží veškeré potřebné údaje pro ověření pravosti a následně jí je udělen certifikát. Uživatelským zařízením je následně tento certifikát při přístupu na stránku stažen a ověřen přes některou z webových databázích, které obsahuje již defaultně od instalace, pro ověřování pravosti certifikátů. V případě, že je vše v pořádku, prohlížeč nás nechá postupovat dále na stránku, pokud ovšem dojde k neshodě nebo není certifikát nalezen, prohlížeč nám to oznámí a my se můžeme rozhodnout, zda skutečně chceme na zadanou stránku vstoupit a komunikovat s ní. [16]

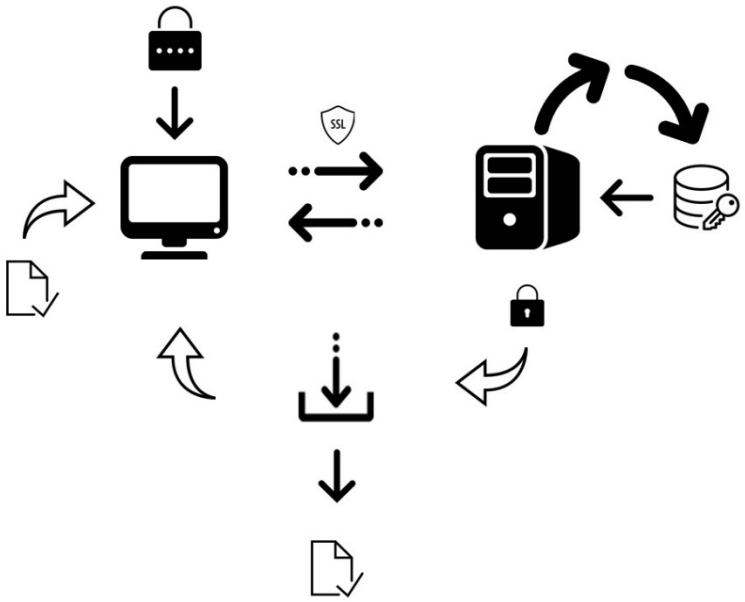
II. PRAKTICKÁ ČÁST

4 ÚVOD DO PRAKTICKÉ ČÁSTI

V praktické části bakalářské práce budeme popisovat návrh aplikace, samotnou implementaci algoritmů a celkový popis činností, které jsme museli vykonat, abychom byli schopni webovou aplikaci zprovoznit, a které všechny úkony byli třeba před samotným spuštěním aplikace. Budeme se zde tedy zaobírat veškerými činnostmi, které jsme prováděli za účelem vytvoření webové aplikace pro šifrování souborů.

4.1 Návrh webové aplikace

Mezi důležité stavební prvky této práce musí patřit bezesporu kvalitní návrh webové aplikace. Je potřeba si rozmyslet všechny kroky, které musíme udělat, abychom byli schopni získat provozuschopnou webovou aplikaci pro šifrování souborů. Jedná se o webovou aplikaci, tudíž budeme muset vytvořit celou strukturu. Nebude nám stačit vytvořit jen obyčejnou webovou stránku, kde budeme něco zobrazovat, ale musíme dokázat tyto data i na úrovní serveru zpracovat.



Obrázek 17 – Návrh webové aplikace diagram

Budeme tedy potřebovat vytvořit webovou stránku. Bude se jednat o jakousi úvodní stranu, tato stránka by měla být co nejjednoduššího vzhledu a měla by zaujmout, to především proto, aby koncového uživatele přesvědčila o tom, že použití této webové aplikace je opravdu správná volba. Tudíž by se zde mělo nalézat jen to nejnutnější jako je například nadpis, který

nám bude říkat, že se jedná o aplikaci pro šifrování souborů. Dále by se na této stránce měl nacházet určitý prvek, sloužící pro navigaci na další stránky. Poté nejspíše budeme využívat určitý druh menu, jenž bude obsahovat položky, které nám budou tvořit odkazy na další stránky naší aplikace. Určitě mezi tyto stránky bude patřit odkaz na úvodní stránku. Pokud by se uživatel dostal někam, kam nechtěl, bude stačit jen kliknout na určité tlačítko a dostane se zpět na domovskou stránku. Jelikož bude aplikace sloužit pro šifrování a dešifrování, tak zde nalezneme zcela jistě tyto dvě stránky.

V těchto jednotlivých sekcích, bychom jistě opět rádi na první pohled informovali nadpisem o tom, na které stránce se nacházíme. Dále by zde opět neměl chybět navigační panel pro rychlou orientaci mezi jednotlivými stránkami. Zcela jistě zde pak budou uvedeny konkrétní řídící prvky pro provedení šifrování a dešifrování. Mohli bychom zde tedy nalézt prvky pro volbu šifrovacího algoritmu, dále bychom zde měli mít možnost zvolit si režim činnosti šifry. Jelikož každý šifra dokáže pracovat s určitými délkami klíče. Je tedy vhodné, abychom si mohli zvolit délku klíče, která nám bude nejvíce vyhovovat. Dalším prvkem by měla být možnost, zvolit si metodu doplnění délky vstupních dat na jednotlivé bloky. Dalším důležitým prvkem na této stránce by také mělo být pole pro zadávání uživatelského klíče, pomocí kterého budeme šifrovat a dešifrovat data. Toto pole by mělo být ošetřeno dostatečně, aby nebylo možno klíč přečíst nebo jej nějak získat při komunikaci. Zajisté zde budeme potřebovat také vstup pro uživatelské soubory. Pokud bychom neměli způsob jak soubory dostat na server, nebyla by také žádná možnost tyto data šifrovat nebo dešifrovat. Poté už jen budeme potřebovat potvrzovací tlačítko, které nám bude sloužit k tomu, abychom data odeslali na server pro následné zpracování. Na stránce pro šifrování ještě budeme potřebovat zřídit zaškrtávací tlačítko pro možnost potvrzení toho, že uživatel souhlasí s uložením zašifrovaných dat pro analytické účely. Implementovat budeme šifry AES, DES, 3DES a novou šifru Smallie. Jelikož první tři šifry se shodují ve většině vstupních parametrů, můžeme použít stejnou stránku pro šifrování a dešifrování pro všechny tyto tři šifry. Ovšem šifra Smallie využívá spoustu dalších prvků, které si uživatel může navolit, z toho důvodu pro tuto šifru bude vhodnější zřídit vlastní stránky pro šifrování a dešifrování. Tuto volbu provedeme především z důvodu přehlednosti stránky.

Nyní se podívejme opět na to, co jsme napsali výše. Máme navrženu část, kde řešíme to, co vidí uživatel při vstupu na stránku. Nyní ale musíme vyřešit to, jak získat data od uživatele a jak je zpracovat. Používat budeme programovací jazyk python a framework postavený na

tomto programovacím jazyce s názvem Flask. Musíme tedy získat data, která uživatel nahraje pomocí formuláře, a volby, které si nastaví při šifrování resp. dešifrování. Budeme proto muset vytvořit skript, který bude zajišťovat tuto obsluhu pomocí nějaké metody pro předávání parametrů mezi uživatelem a serverem. Dále tyto data budeme potřebovat nějak zpracovávat. K tomu nám bude nejspíše sloužit další skript, který bude využívat šifrovací knihovny pythonu, budeme muset zabezpečit zajištění korektní délky klíče a doplňování vstupních dat na správnou délku, abychom byli schopni je rozdělit do patřičných délek bloku. Pokud vyřešíme tyto skutečnosti, můžeme data zašifrovat resp. dešifrovat, pomocí funkcí z patřičné knihovny. Následně už jen musíme vyřešit, jakým způsobem data předat zpět uživateli. Zároveň musíme při šifrování zajistit uložení se správným jménem souboru, abychom při zpětném dešifrování měli možnost získat opět správnou příponu souboru. Tím bychom měli rozmyšlenu celou aplikaci a můžeme se pustit do práce na webové aplikaci s využitím všech dostupných technologií.

4.2 Python 3.x

Základním stavebním prvkem pro tvorbu naší webové aplikace byla nutnost provést instalaci samotného programovacího jazyku Python 3.X.. V našem případě jsme zvolili Python 3.4. Volba této verze měla jedno významné opodstatnění, jelikož pro naši práci budeme využívat framework pro tvorbu webové aplikace, ale zároveň potřebujeme používat standartní balíček knihoven pyCrypto. Při zkušebním zprovoznění s Pythonem ve verzi 3.6 jsme se setkali s problémem, že obsahuje již aktualizovanou knihovnu pyCrypto a framework použitý pro tvorbu webové aplikace nebyl schopný s těmito knihovnami pracovat.

4.2.1 pyCrypto knihovna

Jedná se o knihovnu kryptografických funkcí, tato knihovna obsahuje pro nás velmi důležité funkce, jelikož jsme se rozhodli, že v naší práci budeme používat šifry AES, DES, 3DES a Smallie. Najdeme zde funkce pro generování náhodných čísel, tuto funkci využijeme především při tvorbě inicializačního vektoru. Dále jsou zde již přesně definované funkce pro šifrování i dešifrování pomocí námi zvoleného šifrovacího algoritmu. Stačí nám tedy pouze zadat klíč, pomocí kterého budeme chtít zprávu zašifrovat, následně jsou dalším vstupním parametrem vstupní data, tedy data, která chceme zašifrovat a následně už jen inicializační vektor, který nám pomáhá se šifrováním v nultém bloku. Tuto knihovnu jsme si nainstalovali, jak již bylo zmíněno, při instalaci samotného pythonu do PC. V případě, že bychom

tuto knihovnu nezískali při samotné instalaci pythonu, můžeme využít instalátor „*package management system*“, též známý pod zkratkou „*pip*“. Pomocí tohoto systému můžeme lehce přidávat, ale i mazat balíčky s knihovnami pro python. Většina distribucí pythonu obsahuje již v základu tento systém a můžeme jej tedy lehce vyvolat pomocí příkazové řádky. Pro instalaci nového balíčku nám stačí napsat

$$pip\ install\ název\_balíčku$$

Nebo v případě, že potřebujeme nějaký balíček odinstalovat, stačí opět použít podobný příkaz a to

$$pip\ uninstall\ název\_balíčku$$

4.3 Framework Flask

Jedná se o velmi jednoduchý balíček pro tvorbu webových aplikací s pomocí programovacího jazyka Python. Tento framework nám zajistí základní komunikaci python skriptů se serverem a dokáže zajistit vykreslování šablon pomocí HTML a dalších podpůrných souborů. Získání tohoto mikro-frameworku je velice snadné stačí opět do konzole napsat

$$pip\ install\ flask$$

Následně se nám zobrazí všechny dostupné a potřebné balíčky, které si musí flask stáhnout pro své fungování a dojde k samotné instalaci.

Nyní máme vše nainstalováno pro samotnou práci na našem projektu a můžeme tedy začít. Základním a elementárním souborem našeho projektu bude soubor s názvem „*app.py*“. Tento soubor bude obsahovat základní strukturu naší aplikace, bude nám udávat co se má stát při přístupu na stránku. Do již tedy vytvořeného souboru vepíšeme tento kód.

```
from flask import Flask,render_template

app = Flask(__name__)

@app.route('/')
def main():
    return "Webová aplikace pro šifrování souborů"

if __name__ == "__main__":
    app.run()
```

Obrázek 18 – Základní kód ve flasku

Zdrojový kód, který je uveden na obrázku výše sám o sobě nedokáže udělat vůbec nic, musíme ho nejprve nahrát na webový server, který nám dokáže generovat webovou stránku pomocí tohoto skriptu. Pojďme si ale ještě popsat, co tento kód vlastně znamená. První řádek kódu nám zajistím načtení balíčků z názvem flask a render_template, první zmíněný nám zajišťuje základní funkce frameworku, bez něj bychom nebylo schopni vůbec vytvořit ani základ webové aplikace. Druhý zmíněný balíček, render_template, jsme zatím nevyužili, ten nám slouží pro vykreslení webové stránky, tzv. template. Tuto funkci popíšeme později, až ji budeme využívat. Na druhém řádku máme kód, který nám definuje, že se jedná o aplikaci flasku. Samotný uživatelský program začíná až od *„@app.route“*, zde definuje cestu ke stránce. Tudíž pokud napíšeme jako my jen „*@app.route('/')*“, bude se jednat o domovskou stránku, která se vygeneruje po otevření stránky bez jakéhokoliv doplňku. Kdybychom třeba napsali „*@app.route('/home')*“, naši požadovanou stránku, kterou by nám funkce generovala, bychom našli na adrese, kterou nám vygeneruje server, v našem případě *http://127.0.0.1:5000/home*. Dále už je obsažena samotná funkce bez vstupního parametru a dojde tedy k jejímu spuštění hned po přístupu na stránku a vrátí nám požadovaný obsah, který obsahuje instrukce return. Musíme si tedy spustit konzoli a přemístit se do adresáře, kde máme soubor „*app.py*“. Následně do konzole zapíšeme příkaz a dojde ke spuštění serveru, jako je ukázáno na následujícím obrázku.

```
C:\Users\kubao\Desktop\Bakalařka\flask-project\flask-project>python app.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Obrázek 19 – Log z konzole spuštění serveru

Na adrese, kterou nám flask vypsal do konzole, následně nalezneme naši webovou stránku. Na této stránce nebude zatím umístěno nic zvláštního, jen text, který jsme uvedli do příkazu return.

Obrázek 20 – Webová stránka po prvním spuštění

Stránka neobsahuje žádný odkaz na kaskádové styly, to znamená, že se text zobrazí ve výchozím nastavení, které má nastaveno prohlížeč jako výchozí formátování pro text, který neobsahuje žádný stylopis. Je to úplně stejné, jako kdybychom v HTML kódu napsali paragraf, který by vypadal následovně

$$< p > Webová\ aplikace\ pro\ šifrování\ souborů\ </p >$$

Nyní tedy dokážeme vytvořit pomocí frameworku webový server a následně na tuto stránku něco vypsat. Nyní by bylo vhodné, abychom vysvětlili, jak framework flask pracuje s adresářovou strukturou. Pokud chceme použít pro vykreslení soubor html, který nám bude reprezentovat nějakou webovou stránku, nemůžeme ho jen tak vložit do kořenového adresáře projektu a následně ho pomocí příkazu ve skriptu „*app.py*“ vypsat. Toto by nám nefungovalo. Flask totiž defaultně obsahuje jistou strukturu, kterou prohledává za účelem nalezení daného souboru, který potřebuje pro svou činnost. Již jsme se zmínili o knihovně render_template. Právě tato knihovna se nám stará o načtení a následné vykreslení html dokumentu na webovou stránku. Každý dokument s příponou .html, musí být umístěn v adresáři s názvem „*template*“. Tento adresář je výchozí, do kterého Flask vstoupí pro nalezení požadovaného dokumentu HTML. Přistoupí tedy do adresáře template a dotazuje se, jestli zde není soubor například index.html, který hledá. Pokud je jeho požadavku vyhověno a nalezne daný soubor tak jej vykreslí na webovou stránku. V případě že by byl soubor umístěn v jiném podadresáři, adresáře template, musí to být uvedeno v cestě, kterou předáváme právě příkazu render_template. Například pokud bychom hledali soubor index.html a ten byl umístěn v adresářové struktuře „ ./template/home “, museli bychom napsat

$$return \quad render\_template(\ /home/index.html\ )$$

Nyní už tedy víme jak je to se soubory HTML, které nám slouží pro vykreslení webové stránky. Bylo by tedy vhodné si ukázat, jak to funguje v našem kódu. Nejdříve je ovšem nutné si vytvořit jakousi kostru webu, do které budeme tvořit jednotlivé prvky. V dnešní době, kdy chceme mít možnost následně stránky upravovat po grafické stránce, je pro nás výhodné používat soubory s kaskádovými styly označované též jako CSS. Tyto kaskádové soubory se styly nám poskytují ve velmi přehledné formě styly všech prvků na stránce. Nám následně stačí jen využívat správných tříd, můžeme tedy využívat stejné styly pro různé prvky, pokud je to možné a vyhnout se tak velice elegantně duplicitám v kódu. Samozřejmě můžeme použít styly v přímo v HTML kódu. Ovšem toto řešení není příliš šťastné z pohledu následných úprav, snažíme se je tedy využívat co nejméně, abychom například při následné editaci nemuseli upravovat HTML kód, ale stačí nám jen upravit CSS soubor.

Nyní si tedy již můžeme vytvořit soubor index.html, který bude tvoři základ naší webové stránky, a budeme nám udávat, co se bude na webové stránce zobrazovat po přístupu na ni. Ve skriptu „*app.py*“, toho nemusíme měnit příliš, stačí nám změnit jeden řádek v celém

kódu. Instrukce return nám nyní nebude vracet text, ale bude nám pomocí příkazu render_template, vykreslovat právě dokument index.html. Obsah upraveného souboru *„app.py“*, je uveden na následujícím obrázku.

```
from flask import Flask,render_template

app = Flask(__name__)


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

if __name__ == "__main__":
    app.run()
```

Obrázek 21 – Obsah app.py pro vykreslení HTML dokumentu

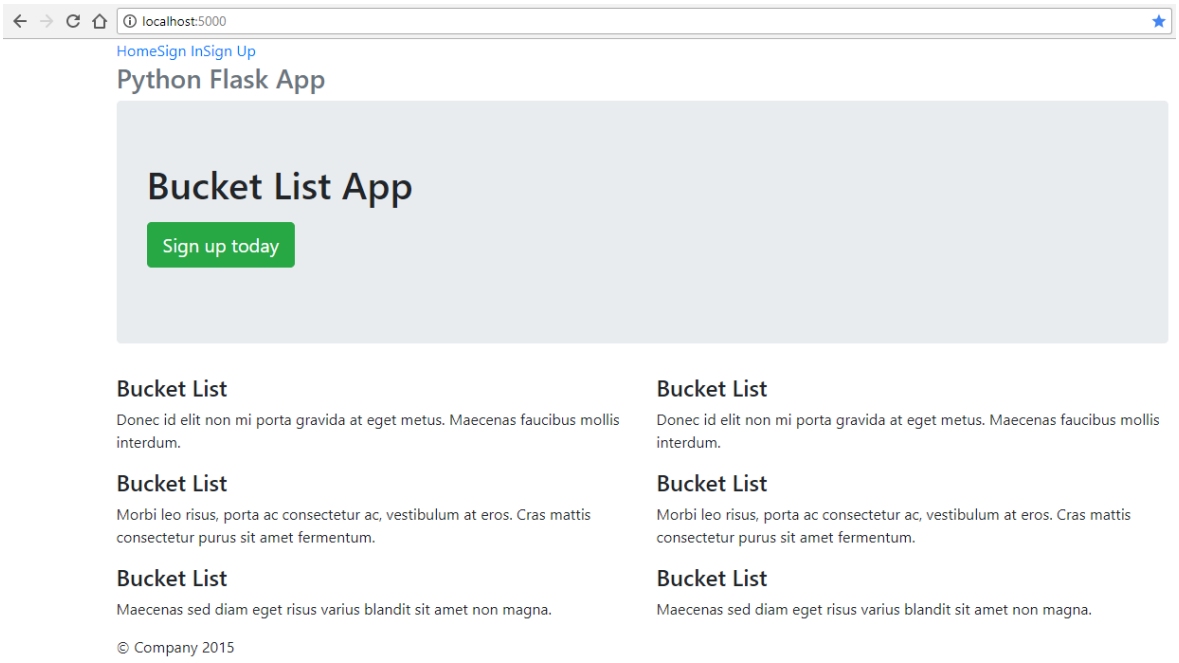
Nyní si tedy vytvoříme v naší složce s projektem novou složku, která se bude jmenovat *„template“* a v této složce si vytvoříme nový soubor s názvem *„index.html“*. Pokud bychom nyní spustili server a zobrazili si naši stránku, viděli bychom jen prázdnou bílou stránku. Pro rychlou ukázku si stáhneme z webu Bootstrap jejich šablonu se zdrojovým kódem stránky. Jedná se o open source projekt, a proto můžeme využít jejich zdrojové kódy pro ukázku v naší práci. [17]

Do našeho nového html dokumentu tak vložíme vzorový kód z bootstrap projektu. Tento kód obsahuje naimportované CSS soubory z bootstrapu. Tento kód je umístěn v hlavičce dokumentu. [17]

```
<head>
    <title>Python Flask Bucket List App</title>

       <link  href="http://getbootstrap.com/dist/css/bootstrap.min.css"
rel="stylesheet">

      <link href="http://getbootstrap.com/examples/jumbotron-
      narrow/jumbotron-narrow.css" rel="stylesheet">

</head>
```

Jak si můžeme všimnout, projekt bootstrap obsahuje všechny soubory v online podobě, tudíž není problém použít kaskádové styly z toho projektu jen odkázáním se na server bootstrapu.

Naše webová stránka nyní po spuštění serveru bude vypadat zcela jinak, než když jsme ji navštívili naposled. Ve skriptu app.py jsme toho nezměnili příliš, ale jelikož jsme v podstatě změnili zcela obsah, který chceme vykreslit, tak je změna na první pohled ihned znatelná. Z toho plyne hned jedna velká výhoda. V případě, že budeme potřebovat něco změnit v naší aplikaci, stačí nám vytvořit nový html soubor, případně můžeme provádět změny v jiných skriptech. Tato provázanost celého systému je velice výhodná především ve chvíli, kdy na projektu pracuje více lidí a každý se může věnovat svojí části. Na obrázku níže je uvedena webová stránka po vykreslení námi nově vytvořeného souboru index.html.



Obrázek 22 – Webová stránka po připojení souboru HTML

Dalším adresářem, který bude dobré využívat a framework jej podporuje je adresář static. V tomto adresáři můžeme mít uloženy css soubory, případně soubory javascriptu, pokud bychom je využívali. V našem případě zde budeme mít uloženy css soubory a dále si v této složce budeme uchovávat soubory, které bude uživatel pomocí rozhraní na webové stránce nahrávat na server pro statistické účely. Nyní si tedy stáhneme pouze soubory s kaskádovými styly, které jsme importovali přes odkazy ze serverů bootstrapu a umístíme si je do této složky a do podadresáře „*css*“. Nyní nám jen stačí přepsat dva řádky, které nám importovali css soubory z webu bootstrapu v index.html a nahradit je tímto výrazem.

```
<link href="/static/css/bootstrap.min.css" rel="stylesheet">
```

Nyní jsme již zcela nezávislý na jakýchkoliv úpravách v css souborech na serverech bootstrapu a můžeme si provádět veškeré změny sami, bez rizika, že budou kaskádové styly na webu například aktualizovány a nám by se změnila podoba některých prvků.

5 VYTVOŘENÍ ŠABLON PRO WEBOVOU APLIKACI

Šablonami v tomto případě neuvažujeme nějaké vzory, do kterých budeme doplňovat různé informace, jak by mohlo být z názvu patrné. V případě použití toho frameworku se šablonami označují jednotlivé webové stránky, které se budou zobrazovat uživateli. V této části tak budeme řešit front-endové řešení aplikace. Ukážeme si, co budou obsahovat jednotlivé stránky naší aplikace a jak tyto prvky provážeme s naší aplikací, o tom poreferujeme následně v další části. Nejprve si tedy vytvoříme šablonu s názvem $index.html$. Jak se budeme na šablony odkazovat v samotné aplikaci, si ukážeme následovně. Jelikož samotný princip odkazování již byl zmíněn výše.

Do titulku stránky umístíme název „*Encryption application"*. Uživateli se tedy při přístupu na stránku zobrazí tento nápis v názvu záložky v prohlížeči. Jak jsme tedy již uvedli při návrhu, naše aplikace by měla na úvodní straně obsahovat nadpis, který bude říkat, o jakou stránku se jedná, jestli jde jen o statickou stránku nebo v našem případě o webovou aplikaci. Z toho důvodu bude hlavní nadpis na stránce obsahovat nápis *„WEB APPLICATION"*. Uživatel tak hned při vstupu na stránku pozná, že nejde jen o webovou stránku, kde si může přečíst nějaké informace, ale bude zde moci provést i nějakou operaci. Další nedílnou součástí úvodní stránky bude navigační menu, pro velice snadnou a rychlou orientaci. Zde budeme mít pouze tři položky „*Home, Encryption a Decryption"*. A také zde budeme mít umístěnu nějakou patičku, která bude obsahovat copyright na tuto webovou aplikaci a také odkaz na stránku, kde bude umístěna nápověda k této webové aplikaci. Přejděme tedy k samotnému vytvoření této stránky. Na zdrojovém kódu níže můžeme vidět veškeré prvky, které jsme použili na stránce. Pojďme si je tedy popsat.

```
<!DOCTYPE html>
<html lang="cs">

<head>
    <title>Encryption Application</title>

</head>

<body>
    <div>
    <p class ="nadpis w3-animate-opacity stred">WEB APPLICATION</p>

<div >
  <a href="/">HOME</a>    
  <a href="encrypt">ENCRYPTION</a>    
  <a href="decrypt">DECRYPTION</a>
</div>

    <footer>
        <p>&copy; Univerzita Tomáše Bati Zlín, Fakulta aplikované informatiky</p><p>Jakub Oral 2018</p>
        <a href="help">HELP</a>
    </footer>
    </div>

</body>

</html>
```

Obrázek 23 - Webová stránka index.html

Jak můžeme vidět ve struktuře, jsou zde obsaženy všechny nutné prvky pro tvorbu HTML stránky jako označení, že se jedná o html dokument, hlavička tohoto dokumentu a také tělo toho dokumentu, kam již doplňujeme náš kód. Můžeme si povšimnout, že v hlavičce máme již naše zmíněné označení „*Encryption Application*“. V těle dokumentu máme umístěn nadpis, v oddílu $< div >$ máme umístěno samotné menu a dále v patičce dokumentu copyright a stránku s nápovědou. Když se nyní podíváme na stránku, uvidíme něco takovéhoto.

WEB APPLICATION

HOME ENCRYPTION DECRYPTION

© Univerzita Tomáše Bati Zlín, Fakulta aplikované informatiky

Jakub Oral 2018

HELP

Obrázek 24 – Stránka index.html bez CSS

Stránka není nijak graficky stylizovaná pomocí css souboru, což se nám ovšem nelíbí. Takováto stránka by nikoho nezaujala a ani nepřesvědčila o tom, že by takovouto aplikaci měl vůbec používat. Je proto nutné připojit ke stránce nějakou soubor s kaskádovými styly. Za tímto účelem, jsme si vytvořili vlastní soubor s názvem „*encryption_app.css*“. Zde budeme umisťovat veškeré styly k naší stránce. Po aplikování tohoto souboru, můžeme vidět změnu jak v html kódu, tak i na samotné webové stránce. V html kódu nám přibyly jednotlivé třídy

(„*class*“), do kterých následně v CSS souboru zapisujeme jednotlivé styly prvků. Na obrázku níže je tedy uveden upravený html kód i vzhled webové stránky.

```
<!DOCTYPE html>
<html lang="cs">

<head>
    <title>Encryption Application</title>


    <link href="/static/css/encryption_app.css" rel="stylesheet">
    <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>

<body>
    <div class="telo">
    <p class ="nadpis w3-animate-opacity stred">WEB APPLICATION</p>

<div class="row2 stred">
  <a href="/">HOME</a>    
  <a href="encrypt">ENCRYPTION</a>    
  <a href="decrypt">DECRYPTION</a>
</div>

    <footer class="footer stred">
        <p>&copy; Univerzita Tomáše Bati Zlín, Fakulta aplikované informatiky</p><p>Jakub Oral 2018</p>
        <a href="help">HELP</a>
    </footer>
    </div>

</body>

</html>
```

Obrázek 25 – HTML kód index.html s CSS

Obrázek 26 – Webová stránka index.html s aplikovaným CSS

Stránka se nám zcela proměnila a nyní vypadá díky použití souboru se styly zcela jinak. O této funkcionalitě jsme se zmiňovali již výše. Můžeme také vidět, že používáme css soubor z webu w3schools.com. Pomocí něj zajišťujeme animaci nadpisu, na který jsme tím aplikovali animaci fade-in, tedy že při vstupu na stránku se nadpis zobrazí z nulového krytí do 100% krytí během jedné vteřiny. Pro další zkrášlení této stránky jsme se rozhodli využít také již zmíněného javascriptu. Využili jsme existující projekt https://github.com/Vincent-Garreau/particles.js/. [18] Jedná se o particle systém napsaný v javascriptu, který nám bude zobrazovat poletující částice na naší stránce, jedná se pouze o vizuální prvek, který nemá na funkčnost aplikace žádný význam. Stáhli jsme si tedy zdrojové soubory z githubu a umístili je do adresáře „*/static/js*“. V tomto adresáři budeme mít uloženy veškeré zdrojové soubory

s javascriptem. Abychom aplikovali skript na stránku, musíme na konec dokumentu před ukončovací značku $</body>$ umístit tento kód. [18]

```
<div id="particles-js"></div>
<script src="/static/js/particles.js"></script>
<script src="/static/js/app.js"></script>
```

Nyní se nám na stránce zobrazí částice. Tímto bychom měli úvodní stránku naší webové aplikace hotovou. Následně budeme z této šablony vycházet a budeme ji používat pro tvorbu dalších podstránek.

Obrázek 27 – Hotová stránka index.html

Nyní si můžeme vytvořit šablonu pro stránku encryption. Použijeme tedy šablonu index.html, jen změníme některé věci. Jak jsme již psali, potřebujeme zadávat vstupní parametry. Budeme zde proto mít menu, sloužící pro výběr možnosti jakou šifru zvolit, jaký režim činnost, délku klíče a padding budeme chtít použít. Jak jsme napsali při návrhu, šifra Smallie je natolik odlišná od ostatních šifer, že bude mít vlastní stránky pro šifrování a dešifrování, proto v menu pro výběr šifry nebude uvedena, ale bude na těchto stránkách odkaz na možnost použití této šifry. Následně bude uživatel přesměrován na jinou stránku, kde možnost použít tuto šifru. Dále zde budeme mít pole pro zadání klíče pro šifrování a také musíme zajistit možnost načítání souboru. Následně už budeme jen potřebovat potvrzovací tlačítko, které nám zajistí odeslání požadavku na zpracování serverem. Dále zde budeme mít checkbox, kterým uživatel může povolit uložení zašifrovaných dat na server pro analytické účely, o této možnosti se zmíníme později. A také checkbox pro možnost uložit si nastavení šifry do textového souboru, při zvolení této možnosti, ale uživatel úmyslně snižuje bezpečnost celé aplikace. Finální vzhled této stránky bude následovný.

Obrázek 28 – Encryption šablona

Je vhodné také zmínit, že aplikace je navržena pro maximální komfort uživatele při práci s ní. Proto veškeré stavy, které by mohli uživatele ohrozit, jsou oznámeny pomocí vyskakovacích oken. Pokud tedy uživatel vybere šifru DES, zaprvé mu zmizí možnost výběru délky klíče, jelikož tato již zastaralá šifra podporuje jen klíč o velikosti 8 bytů a zároveň je varován, že se chystá použít starou šifru, která již není vhodná pro zabezpečování souborů. V případě volby šifry 3DES dojde k deaktivaci jedné položky v menu pro výběr délky klíče, kdy tato šifra využívá jen první dvě možnosti. Také při zvolení položky „Zahrnout do statistiky“ je uživatel varován na operaci, která se stane. Tudíž že jeho zašifrovaná data budou uložena na server pro další statistické a analytické účely. Při vybrání položky „Exportovat konfiguraci nastavení šifry“ je varován, že zvolení této položky může mít vliv na snížení bezpečnosti aplikace, jelikož se znalostí parametrů, kterými byly data šifrovány, je pro případného útočníka snazší útok na tuto šifru. Stránka pro dešifrování bude vypadat skoro stejně, jen zde nebude umístěn checkbox, zajišťující uložení souboru pro statistické účely a možnost uložení konfigurace.

Obrázek 29 – Decryption šablona

Hlavní stránky naší webové aplikace bychom měli tedy navrženy. Ovšem je potřeba uživateli zprostředkovat jeho zašifrovaná, případně dešifrovaná data zpět. Po stisku tlačítka Encryption resp. Decryption je program směřován do funkce na zpracování těchto dat, ten si popíšeme následně. Jenže po zpracování těchto dat se potřebujeme dostat někam dále. Za tímto účelem si vytvoříme stránku s názvem $encryption.html$. Tato stránka bude obsahovat pouze nadpis, menu a dále zde bude odkaz na možnost stažení konfigurace, kterou uživatel použil při šifrování v případě, že chtěl tuto konfiguraci exportovat. To nám zajistí možnost stáhnout si textový soubor s názvem „*cipher_config.txt*“. Tento soubor obsahuje pouze informaci o tom, jakou šifru uživatel použil, jaký režim činnosti byl při šifrování použit, jaká byla zvolená délka klíče a jaká typ paddingu byl použit. A v neposlední řadě zde máme tlačítko *„Download file“*, které nám slouží pro uložení našich zašifrovaných případně dešifrovaných dat. Podívejme se tedy na zdrojový kód této stránky. Za povšimnutí stojí tlačítko

pro stažení souboru, to obsahuje akci nazvanou $onclick = "klik();\ redir();"$. To značí zavolání funkce $klik()$ a funkce $redir()$ při stisku toho tlačítka. Funkce $redir()$ jak název povídá, zajišťuje pouhé přesměrování stránky. Zdrojový kód této funkce vypadá následovně.

```
<script type="text/javascript">
      function redir(){
            window.setTimeout(function () {
              location.href = "/";
            }, 1000);
      }
</script>
```

Jedná se o funkci napsanou v javascriptu. Tato funkce nám zajistí přesměrování na úvodní stránku. Zároveň zde ale máme nastaveno zpoždění 1000ms. Toto zpoždění je zde nastaveno z důvodu použití druhé funkce. Tou je právě funkce nazvaná $klik()$. V této funkci totiž dochází k samotnému řešení ukládání souboru k uživateli.

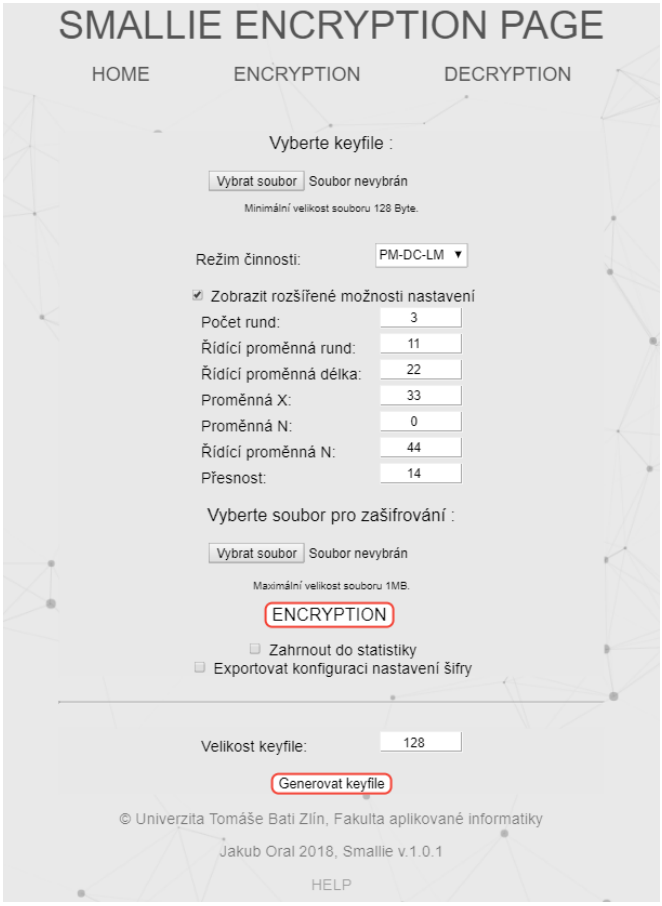
```
  <script src="/static/js/saver.js"></script>

  <script>
function klik () {
   data = "{{soubor}}";
   var sampleArr = base64ToArrayBuffer(data);
   saveByteArray("{{jmeno}}", sampleArr);
}
  </script>
```

Nejprve dojde k načtení souboru saver.js , ve kterém jsou uloženy samotné funkce pro ukládání souboru. Dále je zde také funkce, která slouží k převedení vstupních dat, která jsou typu base64 na pole bytů. To zajišťuje funkce base64ToArryBuffer. Této funkci předáme vstupní data, která si pomocí direktiv $\{\{soubor\}\}$ předáme z hlavního pythonovského skriptu. Následně se provede zpracování a takto vytvořené pole bytů se předá spolu se jménem souboru do funkce saveByteArray. Tato funkce nám pomocí vestavěného filesystému prohlížeče provede vytvoření dialogového okna a uložení samotného souboru k uživateli. Zdrojový kód javascriptu saver.js je uveden na obrázku níže. [19]

```javascript
function base64ToArrayBuffer(base64) {
    var binaryString = window.atob(base64);
    var binaryLen = binaryString.length;
    var bytes = new Uint8Array(binaryLen);
    for (var i = 0; i < binaryLen; i++) {
       var ascii = binaryString.charCodeAt(i);
       bytes[i] = ascii;
    }
    return bytes;
 }

function saveByteArray(reportName, byte) {
    var blob = new Blob([byte]);
    var link = document.createElement('a');
    link.href = window.URL.createObjectURL(blob);
    link.download = reportName;
    link.click();
}
```

Obrázek 30 – Javascript saver.js [19]

Nyní se podívejme již na stránky pro šifru Smallie, tato stránka obsahuje velice komplexní kód a nabízí nám množství nabídek, které musíme obsluhovat. Vzhled stránky pro šifrování je tedy následující a následně si jej popíšeme.



Obrázek 31 – Stránka smallie_enc.html

Jak můžeme vidět, na stránce se opět nachází nadpis a navigační menu, zároveň i patička zůstala nezměněná. Tyto prvky jsou neměnné a zůstávají na všech stránkách stejné. Zásadní změnu doznala část pro zadávání klíče pro zabezpečení dat při šifrování, jelikož šifra Smallie nepodporuje textový řetězec jako bezpečností klíč, je třeba zde nahrát tzv. *keyfile*. Jedná se o soubor, jenž bude použit v podobě bytových dat jako šifrovací klíč. Pokud uživatel nedisponuje žádným souborem, který by mohl použít jako keyfile, tzn., nemá soubor o velikosti alespoň 128 bytů, je mu umožněno si vygenerovat keyfile o velikosti až 4092 bytů. Dále zde nalezneme menu, ve kterém můžeme vybírat ze dvou režimů činnosti. Následuje checkbox, kterým si odemkneme rozšířené možnosti nastavování parametrů. U všech parametrů (počet rund, řídící proměnná rund, řídící proměnná délka, proměnná X, proměnná N, řídící proměnná N) lze hodnoty nastavovat v rozsahu 0-255. Pouze v přídě položky Přesnost můžeme nastavit hodnotu od 9 – 14. Všechny tyto záložky mají své výchozí hodnoty, které jsou použity v situacích, kdy uživatel nic nezmění v této nabídce. V případě, že je zvolen druhý režim činnosti, zobrazí se nám navíc ještě položka přesnost, kterou můžeme také nastavit. Poté už zde jsou jen klasické prvky na možnost uložit si konfiguraci a také pole pro načtení souboru s daty, které se mají zašifrovat. Tyto data mohou mít velikost maximálně 1MB a potvrzovací tlačítko pro provedení šifrování. Stránka pro dešifrování je stejná jen neobsahuje dva poslední checkboxy.

5.1 Zajištění komunikace klient – server

Samotnou velkou kapitolou této práce je nutnost zajistit komunikaci mezi webovou stránkou a samotným skriptem napsaným v Pythonu, který nám zajišťuje back-endové řešení na straně serveru. Jelikož jedním ze základních prvků většiny aplikací je ten, že dokáží reagovat na vstupy od uživatele. Kdybychom například potřebovali zpracovávat jen obyčejné obsloužení čehokoliv, budeme k tomu nejspíše potřebovat například jen tlačítko. Ale ani toto tlačítko by nebylo schopno reagovat na stisk, pokud by neobsahovalo nějakou logiku, co se má stát po jeho stisknutí. Existuje spousta možných řešení jak reagovat na událost, například můžeme reagovat dynamicky pomocí javascriptu nebo JQuery. Můžeme také reagovat na událost pomocí parametru $action =$ "", přímo v prvku umístěném v HTML dokumentu. Tato akce nám může zajistit například zavolání funkce ve skriptu. My tedy musíme zajistit doručení dat z webové stránky do našeho skriptu napsaného v pythonu. Díky tomu, že pro vykreslení webové stránky se využívá klasického HTML dokumentu, tak můžeme používat klasické tlačítka a formuláře, které nám nabízí standard HTML5. Použít můžeme dotazování

pomocí metody „*GET*“ nebo můžeme využít metodu „*POST*“. Rozdíl mezi těmito dvěma metodami je takový, že metoda GET nám předává veškeré informace, které chceme přenést jako textový řetězec v URL adrese stránky. Metoda POST nám předá tyto informace skrytě v těle požadavku. Není tedy tak snadné získat údaje, které pomocí POSTu předáváme. Jak si můžeme ukázat, standartní formulářový požadavek na vstup se jménem by vypadal například tak, že by obsahoval textové pole pro zadání jména a následně potvrzovací tlačítko, které by nám zajišťovalo, odeslání dat z formuláře a vykonání nějaké akce.

```
<form method = GET>
<input name = “name“ type = “text“ placeholder = “Jmeno“>
<br>
<input type=submit  value=Submit>
</form>
```

Pokud bychom použili tento kód, zobrazilo by se nám textové pole, které by obsahovalo nápis Jmeno, který by po kliknutí do pole zmizel a umožnil nám napsat naše jméno. Poté by pod tímto textovým polem bylo umístěno tlačítko s nápisem Submit.

Obrázek 32 – Textové pole s tlačítkem

V tomto okamžiku by jen toto tlačítko odeslalo pomocí metody GET hodnotu pole s názvem name. Tím by se nám změnila URL adresa a objevilo se nám následující.

```
…/index.html?name=Petr
```

Nyní již tedy víme, jak se nám předávají informace z formuláře. Ovšem mi jej potřebujeme nějak získat ve skriptu pro následující zpracování. K tomu ovšem musíme tento HTML kód lehce upravit a to přidat do formuláře položku „*action*“. Tato položka nám zajistí, že při tisku tlačítka se nám zavolá patřičná funkce v pythonovském skriptu. HTML kód by tedy vypadal následovně

```
<form action = “/vypis“ method = GET>
<input name = “name“ type = “text“ placeholder = “Jmeno“>
<br>
<input type=submit  value=Submit>
</form>
```

Framework flask spolu se šablonovacím systémem Jinja2, který využívá, nám zajistí, že po stisku tlačítka, se má z hlavního skriptu zavolat adresa „*/vypis"*. Tato adresa obsahuje ovšem v našem hlavním skriptu funkci pro zpracování dotazu z formuláře. Funkce tedy bude obsahovat následující kód.

```
@app.route('/vypis‘, methods=['GET'])
def vypis():
    jmeno = request.form[‘name‘]
return jmeno
```

Tato funkce nám tedy přesměruje stránku na adresu …/vypis a na této stránce nám vypíše jméno, které jsme zadali do formuláře. Načtení toho jména nám zajišťuje příkaz request.form[‘name‘], tento příkaz totiž ví, že pomocí metody GET má získat z formuláře s názvem *name* jeho hodnotu a uložit ji do proměnné *jmeno*. Na závěr funkce obsahuje návratovou hodnotu *jmeno.* Tudíž na nově vygenerované stránce uvidíme jméno, které jsme zadali do formuláře.

V naší aplikaci budeme také využívat formulářů generovaných pomocí HTML5. V našem případě ovšem uživatel bude pomocí klávesnice zadávat jen jedinou věc a tou bude heslo, pomocí kterého se mají data šifrovat. Ovšem jak jsme již popsali výše, funkce pro šifrování přijímají i další vstupní proměnné než jen heslo. Musíme zajistit ještě jiný způsob volby položek. Nechceme aplikaci zpomalovat složitými vyhodnocováními, o jakou variantu se jedná, pokud by uživatel mohl napsat své možnosti do určitého textového vstupu. V tomto způsobu zadávání by vznikalo obrovské množství chyb a přidávalo by to jen na složitosti používání aplikace. Zároveň by to snižovalo uživatelský komfort při práci s touto aplikací. Rozhodovací logiku tedy zcela v tomto případě vynecháme a použijeme raději přívětivější zadávání vstupních parametrů pomocí menu, ze kterého si uživatel vybere položku, kterou bude chtít zvolit. Ušetříme tím tak výpočetní čas na provedení šifrování a zároveň uživateli usnadníme volbu položky, kterou bude chtít vybrat. Již při návrhu aplikace jsme si řekli, které vstupní parametry budeme vyžadovat na vstupu do funkce, nyní se tedy na tyto požadavky zaměřme konkrétněji. Určitě budeme potřebovat získat heslo, pomocí kterého se provede samotné šifrování souboru. Pro tento účel nám bude sloužit klasický vstupní textový formulář, ovšem tentokrát bude tento formulář jiného typu. Nebude se totiž jednat o vstupní pole $type = "text"$, ale použijeme $type = "password"$. Použití tohoto typu pole nám zajistí, že v okamžiku, kdy do pole začneme vpisovat jednotlivé znaky, objevují se místo nich

jen symboly koleček. Tato velice nepatrná změna typu textového formuláře, nám pomůže ke zlepšení zabezpečení našeho hesla již při samotném jeho zadávání. Na vstupu bychom tedy při zadání klíčového slova „heslo“ viděli jen pět teček.

$$heslo => \cdots\cdots$$

Jak jsme tedy již uvedli výše, budeme pro předávání parametrů využívat metody POST. Veškeré části kódu, které jsme uvedli výše, zůstanou stejné, jen se nám změní položka $methods = ['POST']$. Pojďme se tedy nyní podívat již na náš konkrétní případ a projdeme si celou komunikaci mezi uživatelem a serverem. Začněme například na úvodní stránce, zde máme menu, sloužící pro odkazování se mezi jednotlivými stránkami aplikace, toto menu by za standartní situace odkazovalo přímo na jednotlivé html stránky, které by se následně zobrazovali. My jelikož tvoříme webovou aplikaci a využíváme šablonovací systém, tak tuto možnost nevyužíváme, naopak vždy odesíláme požadavek do hlavního skriptu $app.py$, který nám zpracovává náš požadavek na vykreslení nějaké stránky. Odkaz v menu tedy vypadá například následovně.

$$< a\ href = "encrypt" > ENCRYPTION $$

Tento odkaz se odkáže na funkci encrypt ve skriptu $app.py$, kde již máme napsáno následující.

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

Až nyní nám funkce encrypt vykreslí pomocí funkce $render\_template$ požadovanou html stránku. Jak můžeme vidět, jedná se o vcelku lehkou komunikaci uvnitř aplikace. Nyní si pojďme ukázat předávání parametrů z našich voleb, které jsme si vytvořili na stránce pro šifrování. Princip fungování bude všude stejný, proto si vystačíme s ukázkou na této stránce, kde máme nejvíce prvků, které potřebujeme předávat.

```
<form  action="/encryption" method=post enctype=multipart/form-data id="cipher_mode">

                <!--pole pro klic a chooser-->
<div class="heslo stred">
<input type="password"  id="password" name="password" placeholder="Put your password here!" required>

    </div>

            <div class="row2 stred" style="padding-bottom: 0px;">
    <a href="smallie_enc">Cipher smallie</a>
    </div>
    <br>



    <table class="stred">
        <tr>
    <td class="allign-left1">Typ šifry:</td>
        <td>
                <select class="allign-right1" name="cipher_mode" id="sifra" onchange="kontrola();">
                  <option value="AES">AES</option>
                  <option value="DES">DES</option>
                  <option value="DES3">3DES</option>
                </select>
        </td>
    </tr>
        <tr>
        <td class="allign-left1">Režim činnosti:</td>
 <td>
                <select class="allign-right1" name="mode_selected" >
                   <option value="ECB">ECB</option>
                   <option value="CBC">CBC</option>
                   <option value="CFB">CFB</option>
                   <option value="OFB">OFB</option>
     </select>
    </td>
        </tr>
        <tr>
       <td class="allign-left1"> Padding:</td>
 <td>            <select class="allign-right1" name="pad_type" >
                  <option value='pkcs7'>pkcs7</option>
                  <option value='iso7816'>iso7816</option>
                  <option value='x923'>x923</option>
                </select>
    </td>
        </tr>
        <tr id="key_safe">
      <td class="allign-left1"> Délka klíče:</td>
 <td>             <select class="allign-right1" id="key_safe2"name="key_safe" >
                  <option value="16">16 Byte</option>
                  <option value="24">24 Byte</option>
                  <option value="32">32 Byte</option>
                </select>
            </td>
        </tr>
    </table>
```

Obrázek 33 – Formulář v encryption

Veškeré parametry, které budeme chtít předávat, máme umístěny mezi značkami $< form >$ $</form >$.

V kódu níže můžeme vidět input který je $type = file$, díky HTML5 tak nemusíme řešit načítání souboru, ale stačí nám vytvořit tento input.

```
 <div class = "select_file stred">
<p><input type=file name=file required></p>
     <p style="font-size: 11px;">Maximální velikost souboru 30MB.</p>
                    <input type=submit  class="button encrypt_button" value=ENCRYPTION>
                   <br>

</div>
```

Obrázek 34 – Input pro načtení souboru

Prohlížeče, které podporují HTML5 si sami vytvoří dialogové okno pro načtení souboru z disku. Dále zde můžeme vidět rozbalovací menu, které obsahují jednotlivé volby. Za povšimnutí stojí především to, že formulář obsahuje $action = '/encryption'$ a $method = post$. To znamená, že při stisku potvrzovacího tlačítka, které máme typu submit se nám

odešlou data pomocí metody post do funkce nazvané encryption. Tyto operace provedené na straně webové stránky jsou v podstatě již všechny, veškerá další komunikace se odehrává na straně serveru v pythonovských skriptech. Když tedy nahlédneme do skriptu $app.py$ a podíváme se do funkce encryption uvidíme následující kód.

```
@app.route('/encryption',methods=['GET','POST'])

def encryption():

    if request.method == 'POST':
        f = request.files['file']
        key = request.form['password']
        cipher_mode = request.form['cipher_mode']
        mode_selected = request.form['mode_selected']
        key_safe = request.form['key_safe']
        pad_type = request.form['pad_type']
        statistic = request.form.get('statistic')
```

Vidíme, že cesta podporuje jak metodu GET, tak i metodu POST. Při vstupu do funkce tedy nejdříve testujeme, jestli metoda požadavku je POST, pokud ano, dojde k načítání jednotlivých vstupních proměnných tak, jak byli identifikovány v HTML kódu svými jmény ($name$). Zde je patrné, že každý prvek musíme dotazovat jinak, pro načtení souboru musíme použít metodu $request.file[]$, pro zpracování checkboxu použijeme metodu $request.form.get()$ a pro veškeré ostatní položky ať už se jedná o prvky menu nebo textové pole, do kterého zadáváme klíč, můžeme použít metodu $request.form[]$. V tuto chvíli ovšem nemáme například data ze souboru vůbec načteny, máme jen objekt souboru, tudíž musíme ještě použít následující příkaz.

```
text_encryption = f.read()
```

Tento příkaz zajistí načtení dat typu byte do proměnné *text_encryption* a my tyto data můžeme následně použít k dalšímu zpracování. Po provedení všech potřebných operací s našimi daty potřebujeme dostat tyto data zpět do šablony, kde si je převezme již zmíněný javascript, který provede uložení těchto dat. K tomuto účelu slouží ve frameworku direktiva se syntaxí {{$název\_proměnné$}}. Pro náš konkrétní případ, kdy potřebujeme předat jméno souboru,

```
return render_template('encryption.html',
soubor=text_enc_str,
jmeno=file_name,
config=konfigurace)
```

data, která si má uživatel stáhnout a parametry které byli použity pro šifrování pro případné uložení konfiguračního souboru tak zapíšeme.

Nyní tedy když v šabloně $encryption.html$ napíšeme {{$soubor$}}, získáme data, která jsme měli ve skriptu $app.py$ ve funkci encryption uloženy v proměnné $text\_enc\_str$.

5.2 Implementace šifer do aplikace

V této kapitole se bude tato práce zabývat samotnou implementací našeho skriptu pro šifrování. Bude nutností připravit data do podobny potřebné pro korektní zpracování a především dokázat korektně pracovat se souborem. Nejdůležitějším krokem bude si do skriptu $app.py$ naimportovat náš skript $sifry.py$, který tvoří rozhraní mezi knihovnou pyCrypto a právě zdrojovým skriptem celé aplikace, vytvoření tohoto skriptu bude popsáno později.

```
from sifry import AES_Cipher, DES_Cipher, DES3_Cipher

from sifry import Valid_control
```

Nejideálnější bude popisovat vše přímo na zdrojovém kódu. Načtení vstupních proměnný jsme si již ukázali, proto tuto část kódu vynecháme a budeme se věnovat ostatním částem.

```
jmeno,koncovka = os.path.splitext(f.filename)
jmeno = jmeno.replace(" ","_")
file_name = jmeno + "__"+koncovka[1:]

text_encryption = f.read()

file_size = sys.getsizeof(text_encryption)

if file_size > 30000000:
    return render_template("decrypt.html", errormsg="Zadali jste příliš velký soubor!!!")

if cipher_mode == "DES":
    key = Valid_control.key_length_control(key, cipher_mode,8)  # kontrola klice aby mel spravnou delku v sifry.py to samé s OT
else :
    key = Valid_control.key_length_control(key, cipher_mode,key_safe)  # kontrola klice aby mel spravnou delku v sifry.py to samé s OT

text_encryption = Valid_control.text_length_control(text_encryption,cipher_mode,pad_type)

if cipher_mode == "AES":
    text_encrypted = AES_Cipher.encryption(key,text_encryption,mode_selected)
elif cipher_mode == "DES":
    text_encrypted = DES_Cipher.encryption(key,text_encryption,mode_selected)
elif cipher_mode == "DES3":
    text_encrypted = DES3_Cipher.encryption(key,text_encryption,mode_selected)

if statistic == "true":
    cas = time.strftime("%x")
    cas = cas.replace("/","_")
    jmeno = file_name +"_" + cas + "_" + cipher_mode + "_" + mode_selected + "_" + str(key_safe) + "_" + pad_type
    with open(path+"/static/encrypted/"+ jmeno + ".encrypted", "wb") as statistic_file:
        statistic_file.write(text_encrypted)

file_name = file_name + ".encrypted"
text_encrypted = base64.b64encode(text_encrypted)
text_enc_str = text_encrypted.decode('ascii')
konfigurace = "Typ sifry: "+cipher_mode + " " + "Mod sifry: " + mode_selected + " "+"Delka klice: " + str(key_safe) +" Byte" +" "+ "Typ paddingu: " + pad_type
konfigurace = base64.b64encode(konfigurace.encode('ascii'))
konfigurace = konfigurace.decode('ascii')
return render_template('encryption.html',soubor=text_enc_str,jmeno=file_name,config=konfigurace, configAllowed = configure)
```

Obrázek 35 – Funkce encryption v app.py

Jako první si musíme převést délku klíče na číslo, jelikož všechny informace načtené metodou $request.form()$ jsou typu string. Dále budeme potřebovat uchovat jméno a příponu souboru, jelikož bychom při dešifrování nevěděli, jakého typu byl původní soubor, a nebyli bychom schopni jednoduše rozpoznat, na jaký typ máme soubor převést. Proto si v proměnné

$file\_name$ uchováme jméno souboru včetně přípony, původní symbol tečky nahradíme dvojitým podtržítkem. O načtení bytových dat do proměnné $text\_encryption$ jsme se již bavili, a proto je potřeba tyto data zkontrolovat na naše omezení, že vstupní soubor nesmí být větší než 30MB. V případě, kdy by tento soubor byl větší než zvolená mez, bude uživateli vypsáno chybové hlášení a stránka bude znovu načtena pro možnost zašifrovat menší soubor. Dále následuje rozhodovací podmínka if, pokud se totiž jedná o šifru DES, musíme použít délku klíče jen 8 bytů, jelikož tato šifra nepodporuje ani jednu z námi zvolených možností, které byli obsaženy v menu. Po rozhodovací podmínce tedy zavoláme příslušnou funkci ze třídy $Validity\_control$ v tomto případě funkci $key\_length\_control$ tato funkce přijímá na svém vstupu tři vstupní parametry a to vlastní klíč, který má být použit pro šifrování, následně typ šifry, která byla zvolena a také délku klíče, kterou požadujeme. Tato funkce nám vrátí již upravený klíč, který můžeme následně použít pro samotné zašifrování zprávy. V dalším kroku zajistíme úpravu délky vstupních dat pomocí funkce $text\_length\_control$. Na vstup funkce musíme poslat vstupní data, která se mají upravit, typ šifry a typ paddingu, který chceme použít. Následuje rozhodovací podmínka, pomocí které se vybere šifrovací algoritmus pro příslušnou šifru. Vstupními parametry funkcí pro šifrování je klíč, pomocí kterého budeme šifrovat, již upravené data, která jsou upravena na patřičnou délku bloku a také režim činnosti. Jestliže uživatel zvolí šifru DES, je varován vyskakovacím dialogem, že hodlá použít zastaralou šifru. Tuto funkcionalitu nám zajišťuje javascript, který kontroluje, která šifra byla zvolena. Pokud se nám provedou všechny operace korektně, dojde k vrácení zašifrovaných dat do proměnné $text\_encryption$. Dále následuje vyhodnocení proměnné, která obsahuje hodnotu 0 nebo 1 v případě že byl zaškrtnut checkbox, kterým uživatel potvrzuje, že chce uložit jeho zašifrované data pro statistické účely. Za tímto účelem si pomocí knihovny $time$ získáme aktuální datum a vytvoříme si název souboru, pod kterým si budeme tyto data archivovat. Název obsahuje veškeré potřebné informace, název souboru, tak jak si jej může uživatel stáhnout, ale navíc obsahuje také datum, kdy byly data šifrovány, typ šifry, která byla použita, režim činnosti, délku klíče a také typ paddingu. Následně se nám pomocí standartní metody pro ukládání souborů vytvoří nový soubor s tímto názvem a příponou $.encrypted$ do adresáře $static/encrypted/$. Nyní si do proměnné $file\_name$ uložíme jméno souboru včetně jeho přípony. Následně pro další zpracování pomocí javascriptu musíme naše daty typu byte převést pomocí funkce $b64encoded()$ z knihovny $base64$ na byte data tohoto typu. Jelikož funkce potřebuje získat tyto data na vstupu jako string musíme provést ještě jednu konverzi pomocí funkce

$decode('ascii')$. Ten samý postup musíme provést i s textem, který nám udává použitou konfiguraci použitou při šifrování dat. Nakonec již můžeme pomocí funkce return odeslat data do šablony, kde se provede uložení dat pomocí javascriptové funkce.

Funkce pro dešifrování funguje na podobném principu, odlišuje se jen tím, že v této funkci chybí některé části. Zdrojový kód této funkce bez načtení vstupních parametrů je uveden na obrázku níže. Můžeme vidět opět omezení na podmínku, že soubor musí být menší než 30MB.

```
  #načtení souboru který zadá uživatel
text_nacteny = f.read() #cteni souboru
file_size = sys.getsizeof(text_nacteny)

if file_size > 30000000:
    return render_template("decrypt.html", errormsg="Zadali jste příliš velký soubor!!!")


if cipher_mode == "DES":
    key = Valid_control.key_length_control(key, cipher_mode,8)  # kontrola klice aby mel spravnou delku v sifry.py to samé s OT
else :
    key = Valid_control.key_length_control(key, cipher_mode,key_safe)  # kontrola klice aby mel spravnou delku v sifry.py to samé s OT
try:
    if cipher_mode == "AES":
        text_decrypted = AES_Cipher.decryption(key,text_nacteny,mode_selected,pad_type)
    elif cipher_mode == "DES":
        text_decrypted = DES_Cipher.decryption(key,text_nacteny,mode_selected,pad_type)
    elif cipher_mode == "DES3":
        text_decrypted = DES3_Cipher.decryption(key,text_nacteny,mode_selected,pad_type)
except:
    return render_template('decrypt.html',errormsg=" Tento klíč nepatří k tomuto souboru !!! ")

jmeno, koncovka = os.path.splitext(f.filename)
if "__" in jmeno :
    jmeno = jmeno.replace("__",".") #ziskam puvodni nazev souboru s koncovkou
    jmeno, koncovka = os.path.splitext(jmeno) #ziskam koncovku a nazev souboru

file_name =jmeno + koncovka
text_decrypted = base64.b64encode(text_decrypted)
text_dec_str = text_decrypted.decode('ascii')

konfigurace = "Typ sifry: " + cipher_mode + " " + "Mod sifry: " + mode_selected + " " + "Delka klice: " + str(
    key_safe) + " Byte" + " " + "Typ paddingu: " + pad_type
konfigurace = base64.b64encode(konfigurace.encode('ascii'))
konfigurace = konfigurace.decode('ascii')
return render_template('encryption.html', soubor=text_dec_str, jmeno=file_name, config=konfigurace)
```

Obrázek 36 – Funkce pro dešifrování v app.py

Můžeme si povšimnout opět stejné kontroly a úpravy délky klíče, to proto, že tento klíč opět zadává uživatel pro dešifrování souboru. Dále je zde opět podmínka, která rozhoduje, jaká šifra je vybrána a podle toho vybírá funkci, pomocí níž se provede dešifrování, zde je již několik odlišností, tato funkce má na vstupu celkem čtyři parametry na rozdíl od funkce pro šifrování. Je to dáno tím, že potřebujeme znát i typ paddingu, který byl použit při šifrování, abychom mohli data upravit na konci dešifrování do původní podoby. Dále zde můžeme najít ošetření chyby pro případ, že uživatel zadá nesprávné heslo. Dojde k vykreslení šablony pro dešifrování souboru a zároveň se uživateli zobrazí upozornění, že zadal nesprávný klíč. Následuje opět vytvoření názvu souboru, kdy ovšem v tomto případě odstraníme příponu $.encrypted$ a nahradíme ji příponou, která byla v názvu souboru oddělena dvojitým podtržítkem. Dále provedeme převedení na data typu $base64$ a zpětně na string. Opět se vytvoří

konfigurace pro případ, že by si někdo chtěl uchovat i tuto informaci a data jsou odeslána pomocí návratové hodnoty do šablony.

Se šifrou Smallie jsme museli také implementovat nové skripty, které se starají o práci šifry, stejně jako soubory z knihovny pyCrypto.

from Smallie import GenerujKeyFile

from Smallie import Smallie

Nás budou zajímat především dva tyto zdrojové soubory a to je soubor *„Smallie.py“* a soubor *„GenerujKeyFile.py“*. Jelikož u této šifry dochází k šifrování pomocí klíčového souboru. Pokud by uživatel neměl vlastní keyfile, který by mohl využít pro šifrování, může si pomocí funkce vygenerovat nový keyfile o velikosti od 128 bytů do velikosti 4092 bytů. Za tímto účelem máme v hlavním skriptu *„app.py“* vytvořenu funkci *„generateKey()“*. Kód této funkce je uveden na obrázku níže.

```
@app.route('/keyfilegenerate',methods=['POST'])

def generateKey():

    keySize=request.form['keyFileSize']
    keySize=int(keySize)
    generatedKey = GenerujKeyFile.VygenerovatKeyFile(keySize)

    generatedKey = base64.b64encode(generatedKey)
    generatedKey = generatedKey.decode('ascii')
    return render_template('smallie_enc.html', keyFile = generatedKey)
```

Obrázek 37 – Funkce pro generování keyFile

Funkce převezme jako vstupní parametr jen velikost klíčového souboru, který si uživatel zvolil na stránce a následně vrátí byte hodnotu, kterou opět pomocí javascriptu nabídneme uživateli ke stažení. Poté dojde k přesměrování zpět na stránku pro šifrování šifrou Smallie a uživatel může dokončit šifrování jeho souboru. Podívejme se tedy na funkci pro šifrování. Zdrojový kód této funkce zde vzhledem k jeho rozsahu nebudeme uvádět, ale jen si popíšeme jednotlivé části. Najdeme zde opět načtení jednotlivých prvků jako u předchozích šifer. Získáme si tentokráte ale dva soubory, jeden sloužící jako keyfile a druhý obsahující samotná data pro zašifrování. Provedeme si opět vytvoření nového názvu souboru, jako tomu bylo u předchozích šifer, dále provedeme kontrolu velikosti klíče, jelikož potřebujeme mít na vstupu klíč o minimální velikosti 128 bytů. Dále provedeme kontrolu pro velikost samotných dat, která se mají šifrovat, ta je omezena pro šifru Smallie na velikost 1MB. Poté už následuje

samotné odeslání dat do šifrovací funkce. Veškeré vstupní parametry máme vypsány zde a popsány jsou v teoretické části práce.

```
text_encrypted = Smallie.SifrovaniDesifrovani(file,keyFile,encrypt/decrypt,re-
zim_sifry,rundy,promennaN,ridiciRundy,ridiciDelka,promennaX,ridiciN,presnost)
```

Dále už probíhají jen klasické režie, jako tomu bylo u předchozích šifer, dojde ke kontrole, zda byl vybrán checkbox pro uložení konfigurace, dále zda byl vybrán checkbox pro uložení souboru se zašifrovanými daty pro statistické účely. Nakonec proběhnou převody dat na formát, který jsme pomocí javascriptové funkce schopni doručit uživateli, jako soubor ke stažení. Funkce pro dešifrování je stejná jako funkce pro šifrování, jelikož funkce, kterou voláme ze skriptu Smallie.py je obousměrná a stačí jí změnit třetí vstupní parametr a funkce může sloužit jak pro šifrování tak i dešifrování.

5.3 Zabezpečení komunikace mezi uživatelem a serverem

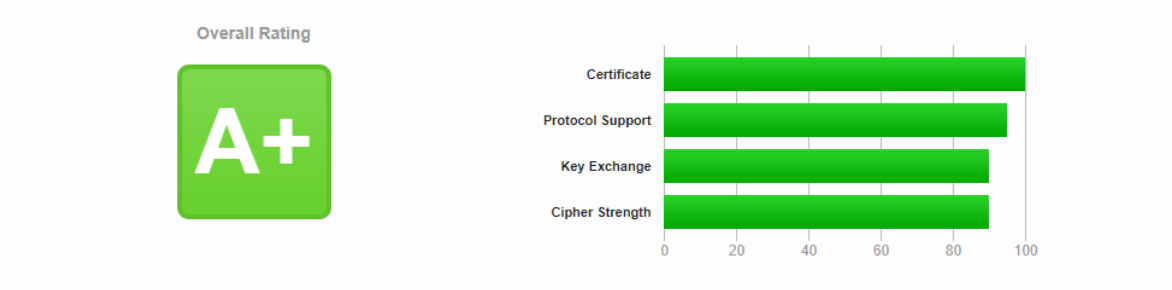
Velice důležitou částí této práce je právě bezpečnost celé aplikace. Samozřejmě můžeme vytvořit webovou aplikaci pro šifrování souborů, která bude umět načíst soubor, zašifrovat jej a následně předat zpět uživateli v podobě zašifrovaných dat. Jenže narazíme hned na několik aspektů, které se neslučují s bezpečností a důvěryhodností celé aplikace. Nevíme, kdo všechno může odposlouchávat přenos našich dat ještě před jejich šifrováním už během přenosu na server, kde se data zpracovávají. Dále jako koncový uživatel nikdy nepozná, zda se jeho data ukládají někam na server. Můžeme vidět, jak webová stránka komunikuje ze serverem, jestli používá metodu POST nebo metodu GET. Takže na tyto jednotlivé rizika se musíme zaměřit.

Jako první se tedy podívejme na naši webovou aplikaci. Jednoznačné rozhodnutí padlo v otázce komunikace již při práci s formuláři. Pole pro zadávání hesla bylo nutno nastavit typu „*password*“. To nám zajistilo, že se zobrazují jen zástupné symboly místo znaků, které zadáváme. Tyto drobnosti nám napomáhají ke zlepšení bezpečnosti a důvěryhodnosti celé naší webové aplikace. Dále bylo nutné veškeré informace, které chceme předat na server a sdělit je skriptu pro následné zpracování, předávat pomocí metody POST, jelikož tato metoda nepřenáší data v URL odkazu, ale předává si je jako proměnné. To je pro náš případ výhodné, nechtěli bychom, aby uživatel viděl své tajné heslo v odkazu. Zaprvé by to byla výrazná bezpečností slabina a také by to nepůsobilo nijak důvěryhodně. Výše popsané prvky by ale i tak byly snadno napadnutelné, proto musíme do naší komunikace zařadit další prvek

a tím je použití SSL certifikátu pro šifrovaný přenos pomocí protokolu HTTPS. Princip SSL byl popsán již v teoretické části. Proto se zde pobavíme spíše o implementaci v našem případě. Framework flask, který využíváme k tvorbě naší aplikace, podporuje již existující skript, který se jmenuje *„flask_sslify.py“*. Nám tedy stačí v naší aplikaci použít tento skript, který sám řeší komunikaci pomocí HTTPS protokolu. Do našeho kódu tak stačí přidat jeden import, který se postará o načtení toho skriptu, a dále přidáme následující řádek.

```
sslify = SSLify(app, subdomains=True)
```

Po provedení tohoto příkazu dojde k nastavení zabezpečení SSL pro naši aplikaci nazvanou *„app“* a všem jejich podstránek. Jestliže si nyní obnovíme naši stránku, uvidíme již, že je přidán certifikát, pomocí kterého se ověřuje zabezpečení stránky a komunikujeme se serverem pomocí protokolu HTTPS. Pro ověření nastavení certifikátu jsme použili webovou stránku „*Qualys. SSL Labs*“, která se věnuje této kontrole. Tuto webovou stránku můžeme nalézt na adrese *https://www.ssllabs.com/index.html*. Provedli jsme tedy SSL Server test. Výsledky tohoto testu jsou uvedeny na obrázku níže.



Obrázek 38 – SSL Protocol test

Nyní tedy můžeme přistoupit k řešení práce se soubory, které chce uživatel zpracovat. Jak jsme si již řekli, veškeré operace by měli probíhat ideálně na straně prohlížeče u uživatele v počítači. Vzhledem k použití frameworku Flask a programovacího jazyka Python jsme se rozhodli provádět veškeré výpočetní operace na straně serveru a zbytek se snažíme provádět právě v prohlížeči uživatele. Na počátku celé práce jsme pracovali s myšlenkou soubor načíst od uživatele a pracovat s jeho obsahem jen jako s proměnnou, tato část zůstala zachována, jelikož je to vyhovující pro bezpečnost, ovšem dále jsme chtěli soubor nabídnout uživateli ke stažení, pomocí standartního formuláře z HTML5, který podporuje možnost dialogového okna a následně stažení souboru k uživateli. V tomto případě bychom nebyli limitování velikostí souboru a mohli bychom soubor nabídnout snadno ke zpětnému stažení. Jenže v tomto případě jsme museli soubor po skončení šifrování uložit na server do určitého adresáře, následně soubor z tohoto adresáře navázat na stahovací dialog a po stažení tento soubor

smazat. Toto řešení se jevilo jako nejvhodnější a jediné možné řešení jak vyřešit tento problém pomocí HTML. Jenže my si nepřejeme ani dočasné ukládání souborů na serveru, jediná možná varianta kdy zašifrovaná data uložíme na server a necháme je tam uložené je v případě, kdy uživatel vybere možnost, že jeho zašifrovaná data budou použita pro analytické účely. My jsme se ovšem rozhodli jít cestou, kdy nebudeme tedy vůbec nic ukládat ani dočasně, a proto jsme museli zvolit javascript a práci s „*file system api"*. Toto api nám umožňuje pracovat se souborem v uživatelově počítači. Tím řešíme problém toho, že bychom museli někde něco ukládat mimo uživatelské koncové zařízení. Práce s api je velice jednoduchá a pomocí javascriptu poměrně snadno implementovatelná. Jediná potíž, na kterou narážíme, je podpora starších prohlížečů, se kterými ale už dnes pracuje jen minimum uživatelů a dá se také předpokládat, že tito uživatelé nebudou využívat naši webovou aplikaci. Implementaci tohoto api jsme provedli již dříve. Uživateli se tedy data ukládají přímo do jeho zařízení. Pomocí výše zmíněných opatření zvýšíme bezpečnost naší aplikace vždy o nějaký stupeň. Nelze říci, že naše aplikace je bezpečná, a proto byste ji měli jako uživatelé používat, ale můžeme tvrdit, že se snažíme, aby naše aplikace byla co nejbezpečnější. Přesně to znázorňuje náš graf, který jsme zmínili na počátku návrhu naší aplikace. Načteme uživatelská data, získáme heslo pro provedení šifrování/dešifrování a následně provedeme komunikaci se serverem, která je zabezpečena pomocí certifikátu SSL. Na straně serveru se provede šifrování/dešifrování a opět bezpečně se data předají zpět uživateli ke stažení. Dále jsme přidali jako prvek zabezpečení omezení na maximální velikost souboru, kdy soubory, které se šifrují šiframi AES, DES nebo 3DES mohou mít maximální velikost 30MB a soubory, které šifrujeme pomocí šifry Smallie mohou mít velikost 1MB. Jde především o zabezpečení proti vytěžování serveru. Mohlo by to totiž dojít k útoku na náš server a mohl by být zahlcen velkými soubory a následně by zkolaboval.

6 PROGRAMOVÁNÍ APLIKACE

Nyní se pojďme již zaměřit na samotné programování naší webové aplikace pro šifrování souborů. V přechozích kapitolách jsme si vysvětlili všechny důležité informace, které potřebujeme znát. Již v předchozí části jsme si vypracovali tyto body:

- Reprezentace jednotlivých šifrovacích algoritmů
- Vytvoření grafické vizualizace aplikace
- Zajištění komunikace klient-server
- Implementace šifer do webové aplikace
- Zabezpečení komunikace mezi uživatelem a serverem

6.1.1 Vytvoření souboru pro reprezentaci šifer

Jak jsme se již zmínili na začátku, budeme používat knihovnu pyCrypto, vytvořenou speciálně pro kryptografické účely. Tato knihovna nám ušetří velké množství práce s vytvářením všech šifrovacích algoritmů. Jednalo by se opět o to, dělat práci, kterou už někdo někdy udělal. Začneme například šifrou AES.

Vytvoříme si tedy třídu, kterou nazveme AES_Cipher, tato třída bude v sobě obsahovat funkce pro šifrování a dešifrování. Mohli bychom vytvořit přímo funkce pro šifrování a dešifrování typu AES_Cipher_Encryption a AES_Cipher_Decryption, jenže tento přístup je nepřehledný a při implementaci například v rozsáhlejších projektech nebo jiných skriptech bychom museli implementovat celý soubor, který by obsahoval všechny šifry, tím by docházelo ke zpomalování algoritmu a prodlužovali bychom vybavovací dobu na požadavek. Proto strukturujeme jednotlivé šifrovací algoritmy do vlastních tříd a v nich teprve vytváříme funkce pro šifrování a dešifrování.

Šifrovací funkce musí získávat na vstupu určité parametry, pokud bychom této funkci nedodali požadované vstupní parametry, museli bychom po spuštění funkce nějak získat vstupní proměnné pro šifrovací algoritmus. Tudíž bychom museli zařadit například nějaký dotaz na načtení vstupu z klávesnice nebo načtení souboru z cesty, kterou jsme zadali buď přímo jako fixní adresu ve funkci, ze které se bude číst vždy nebo opět načíst tuto cestu jako vstup z klávesnice. Pro náš případ ovšem budeme do této funkce posílat vstupní parametry, abychom je následně zpracovali. Jako první vstupní proměnná je v našem případě zvolena proměnná „*key*“. Jak již napovídá název, tato proměnná v sobě ponese hodnotu klíče, který bude použit pro šifrování a následně také pro dešifrování, bude se jednat o textový řetězec,

pomocí kterého se zpráva bude šifrovat. Druhým vstupním parametrem bude pro náš případ již samotný vstupní text, ten se uloží do proměnné soubor, který se má zašifrovat. A třetí vstupní proměnnou je pomocná proměnná, která nám bude následně určovat, jaký režim činnosti se má použít. V teoretické části jsme se o těchto režimech již zmiňovali, jedná se o režimy (ECB, CBC, CFB, OFB). V tuto chvíli máme tedy vytvořenou funkci a její vstupní proměnné. Nyní můžeme tedy přistoupit k jejich zpracování. Jako první si převedeme vstupní proměnnou key a soubor na proměnné typu byte. Jelikož šifrovací algoritmus, který budeme používat, vyžaduje vstupní proměnné typu byte. Nyní si musíme vytvořit inicializační vektor do proměnné „*iv*“, za tímto účelem již budeme potřebovat naimportovat nějakou knihovnu. Zároveň si rovnou můžeme importovat i knihovnu pro práci se šifrou AES. Proto na začátek celého kódu doplníme řádek s importem

$$from\ Crypto\ import\ Random$$

$$from\ Crypto.Cipher\ import\ AES$$

Výraz importování znamená, že z adresáře Crypto, resp. podadresáře Cipher, potřebujeme naimportovat soubor Random a soubor AES, který obsahuje funkce, které budeme využívat. Nyní již tedy můžeme vytvořit inicializační vektor

$$iv\ =\ Random.new().read(AES.block\_size)$$

Funkce Random z knihovny Crypto potřebuje jako vstupní parametr délku bloku šifry, kterou budeme používat, z toho důvodu jí musíme předat parametr AES.block_size. Vytvoří se nám tedy inicializační vektor, který bude mít velikost stejnou jako velikost bloku šifry AES. Díky tomu ho můžeme využívat pro šifrování. Následně si vytvoříme jednoduchou rozhodovací logiku pomocí podmínek if, pro rozhodování o zvoleném režimu činnosti šifry. Každé splnění podmínky potom obsahuje příslušnou variantu šifry. Takže například pokud zvolíme režim ECB, bude proveden následující příkaz

$$cipher\ =\ AES.new(key, AES.MODE\_ECB, iv)$$

Do proměnné cipher se nám připraví šifra z klíče, pomocí určitého módu a s pomocí inicializačního vektoru. V dalším kroku již provedeme samotné šifrování zprávy. Do této zprávy musíme zahrnout i unikátní inicializační vektor, který jsme si vytvořili za účelem šifrování a dešifrování zprávy. Proto zapíšeme

$$msg\ =\ iv\ +\ cipher.encrypt(soubor)$$

Proměnná „*msg*“ bude následně obsahovat již zašifrovanou zprávu. Dále nám stačí jen tuto zprávu odeslat zpět do hlavního programu pomocí příkazu return. Celý kód funkce encryption, třídy AES_Cipher bude vypadat, tak jak je uvedeno na následujícím obrázku.

```
class AES_Cipher:

    def encryption(key,soubor, mode):

        key = str.encode(key)
        iv = Random.new().read(AES.block_size)
        if mode == "ECB":
            cipher = AES.new(key, AES.MODE_ECB, iv)
        elif mode == "CBC":
            cipher = AES.new(key, AES.MODE_CBC, iv)
        elif mode == "CFB":
            cipher = AES.new(key, AES.MODE_CFB, iv)
        elif mode == "OFB":
            cipher = AES.new(key, AES.MODE_OFB, iv)
        msg =iv+cipher.encrypt(soubor)
        return msg
```

Obrázek 39 - Šifrovací algoritmus pro AES_Cipher

Na podobném principu funguje i dešifrovací algoritmus. Opět se zde provede převod klíče na byte, jelikož klíč musí uživatel zadat pro dešifrování, bez jeho znalosti nebude možnost zprávu zpětně rozšifrovat. Dále se vytvoří inicializační vektor pro dešifrování, opět pomocí stejné funkce, jako ve funkci pro šifrování. Opět proběhne stejný rozhodovací cyklus pomocí if podmínek. Následně se ovšem provede pomocí něj již dešifrování.

$$soubor = cipher.decrypt(soubor)$$

Od dešifrovaného výsledku odečteme inicializační vektor a vypíšeme finální zprávu po dešifrování. Pokud vše proběhlo bez problému, měli bychom získat původní zprávu.

```
def decryption(key,soubor, mode,pad_type):
    key = str.encode(key)
    iv = Random.new().read(AES.block_size)

    if mode == "ECB":
        cipher = AES.new(key, AES.MODE_ECB, iv)
    elif mode == "CBC":
        cipher = AES.new(key, AES.MODE_CBC, iv)
    elif mode == "CFB":
        cipher = AES.new(key, AES.MODE_CFB, iv)
    elif mode == "OFB":
        cipher = AES.new(key, AES.MODE_OFB, iv)

    size_iv = AES.block_size
    soubor = cipher.decrypt(soubor)
    soubor = Padding.unpad(soubor, AES.block_size, style=pad_type)
    soubor = soubor[size_iv:]


    return soubor
```

Obrázek 40 - Dešifrovací algoritmus pro AES_Cipher

Stejným způsobem jsou vyhotoveny šifrovací a dešifrovací algoritmy i pro šifry DES a 3DES. Můžeme si povšimnout, že knihovna Crypto je napsána tak, že je ve své podstatě jedno jakou šifru chceme vytvořit a stačí nám tedy jen vybrat správné funkce.

6.1.2 Ošetření vstupních parametrů šifrovacích funkcí

Jednou z velice důležitých částí je vždy správné ošetření stavů, které mohou nastat. V našem případě máme na vstupu několik parametrů, které jsme si již představili dříve. Jedná se o klíč, kterým bude zpráva následně šifrována, dále samotná data, která mají být zašifrována a šifrovací režim. V zájmu možnosti nabídnout uživateli ještě vyšší bezpečnost jeho dat, nám dále přibude další vstupní parametr a tím bude délka klíče, jelikož pouze šifra DES se omezuje na délku klíče o velikosti 8 bitů a šifra 3DES využívá pouze klíče od délce 16 nebo 24 bitů. Zbylá šifra AES nám může nabídnout klíče o velikosti 16,24 nebo 32 bitů. Jenže co když uživatel zadá klíč, který bude delší než požadovaný počet bitů, nebo naopak zadá klíč kratší? Na tuto variantu samozřejmě musíme myslet a musíme se postarat, aby ani v takovém případě nedošlo k pádu aplikace a uživatel přesto směl použít klíč, který si sám zvolil a chce jej z nějakého důvodu použít. Pojďme se tedy podívat na to jak tento problém řešit. V našem skriptu „*sifry.py*“ jsme si vytvořili novou třídu pojmenovanou „*Valid_control*“. Tato třída bude obsahovat následně veškeré funkce, které budeme potřebovat pro ošetření všech vstupních parametrů předtím, než je pošleme k samotnému zpracování do funkcí pro šifrování a dešifrování. Takže jako první si vytvoříme funkci, která nám bude hlídat délku klíče. Jako

vstupní proměnné této funkce bude „*key“*, tato proměnná bude obsahovat klíč, který zadá uživatel. Další proměnnou bude „*cipher“*, zde bude uložena informace, kterou šifru budeme používat, je to z důvodu rozhodování, zda budeme volit klíč o délce 8,16,24 nebo 32 bitů. A poslední informací, kterou potřebujeme znát před samotným zpracováním je právě informace o délce klíče, který chceme použít. Tuto zprávu si předáme do proměnné „*delka_klice“*. Díky tomu, že již máme všechny potřebné informace na vstupu, můžeme provést samotné zpracování klíče. Jak jsme již říkali, nezáleží na tom, jestli uživatel zadá dlouhý nebo krátký klíč. První operací, která je se vstupním klíčem provedena, je cyklus while. Zde se nám provede v několika iteracích prodloužení klíče na délku 32 bitů. Toto prodloužení je provedeno opakovaným přičtením klíčového slova na konec původního klíče. Pokud zadáme klíč, který bude delší než 32 bitů, v tomto cyklu se nestane s klíčem nic a pokračuje se dále, jelikož není splněna podmínka pro provádění cyklu a to, že klíč musí být menší než 32 bitů. Následně se dostaneme k rozhodovacím podmínkám if, kde se postupně dotazujeme, o jakou šifru se jedná, vezmeme tedy obsah proměnné *„cipher“*, a porovnáváme jej s žádanou hodnotou. V případě uspokojení podmínky, například pro šifru AES dojde ke vstupu dovnitř podmínky. Zde dojde k uložení nového klíče do proměnné „*key“*. Jelikož jsme si na vstupu získali hodnotu délky klíče, můžeme napsat rovnou příkaz pro získání prvních n-znaků z řetězce. Následně takto vytvořený klíč vrátíme zpět do hlavního programu, odkud jsme tuto funkci zavolali. Z toho tedy vyplívá, že pokud je uživatelský klíč příliš krátký, dojde k jeho naduplikování a následnému zkrácení na požadovanou délku tak, aby šifrování s tímto klíčem mohlo proběhnout. V případě, že uživatel zadá delší klíč než je požadovaná délka, dojde jen k jeho zkrácení. Uživatel tuto skutečnost nepozná, jelikož celou tuto logiku za něj vyřeší funkce k tomu určená a může použít své heslo o libovolné délce. Program však využije jen délku slova, kterou nutně potřebuje a ostatní část bude nepoužita. Celé toto ošetření délky klíče je uvedeno na obrázku níže.

```
def key_length_control(key,cipher,delka_klice):
    while 'true':
        if len(key)>= 32:
            break
        key+=key

    #Ošetření délky klíče 8-bitů
    if cipher == "DES":
        key = key[:8]



    #Ošetření délky klíče pro 16,24,32 bitů
    if cipher == "AES":
        key=key[:delka_klice]

    #Ošetření pro 3DES šifru a klíče 16 a 24 bitů
    if cipher == "DES3" and delka_klice == 32:
        key = key[:24]
    elif cipher == "DES3":
        key = key[:delka_klice]

    return key
```

Obrázek 41 - Ošetření délky klíče

Prvkem, který se u blokových šifer objevuje, je velikost bloku šifry. Na tuto skutečnost nesmíme zapomínat ani v naší práci, a proto musíme vědět, že i vstupní text musí mít nějaký rozměr, abychom jej dokázali následně zpracovat. Proto jsme si ve třídě Valid_control vytvořili funkci s názvem „*text_length_control*“. Tato funkce je velice jednoduchá a využívá dalšího prvku z knihovny Crypto. Jedná se o třídu nazvanou „*Padding*“. Tato třída obsahuje funkce „*pad*“ a „*unpad*“. Jako vstupní parametry přijímají tyto funkce vstupní data v bytech, délku bloku a styl, kterým se mají data doplnit. Funkce „$Padding.pad()$“, nám zajistí doplnění vstupních dat na patřičnou délku. Jelikož existuje několik standardů pro takovéto doplňování bloků, museli jsme se postarat, aby naše řešení pokrývalo všechny možnosti, které nám třída Padding poskytuje. Můžeme tedy pomocí jednoduchého menu, které si vytvoříme v html dokumentu volit mezi módy (PKC7, ISO7816 a X923). Každý tento mód se chová jinak při doplňování délky bloku. Níže na obrázku můžeme vidět, jak je řešena kontrola délky vstupních dat.

```
def text_length_control(vstupni_text,cipher,pad_type):
    if cipher == "AES":
        vstupni_text = Padding.pad(vstupni_text, AES.block_size, style= pad_type)
    elif cipher == "DES":
        vstupni_text = Padding.pad(vstupni_text, DES.block_size, style= pad_type)
    elif cipher == "DES3":
        vstupni_text = Padding.pad(vstupni_text, DES3.block_size, style= pad_type)


    return vstupni text
```

Obrázek 42 - Kontrola délky bloku dat

Jak si můžeme povšimnout, nechybí zde rozhodování kontroly pro jednotlivé šifry. Mezi jednotlivé vstupní parametry patří vstupní data, typ šifry, kterou budeme používat a mód paddingu.

6.2 Implementace aplikace na webový server

Rozhodli jsme se, že naše aplikace poběží na serveru. Jelikož momentálně nemáme k dispozici server, na kterém bychom mohli celou aplikaci zprovoznit, rozhodli jsme se využít hostingu zdarma který poskytuje portál pythonanywhere.com. Postup jakým jsme tedy zprovoznili naši aplikaci na tomto serveru, byl následující. Služba Pythonanywhere podporuje jak programovací jazyk Python tak právě framework Flask. Prošli jsme tedy jednoduchým konfiguračním rozhraním na webu, pomocí kterého jsme vytvořili nový projekt, který podporoval již dvě zmíněné technologie. Dále byl postup velice jednoduchý, stačilo vytvořit si stejnou adresářovou strukturu jako při tvorbě celé aplikace v počítači, kde jsme testovali celou aplikaci na localhostu. Následně jen stačilo upravit některé importy a nahrát celou aplikaci na server pomocí tlačítka Reload page. Nyní již máme tedy i přístup k naší webové aplikaci v online podobě a nalezneme jej na adrese: https://encapp.pythonanywhere.com/ .

7 ZHODNOCENÍ VÝSLEDKŮ PRÁCE

Tato práce se zaměřovala na webovou aplikaci pro šifrování souborů. Naším cílem bylo vytvořit funkční webovou aplikaci, která bude využívat námi zvolených šifrovacích algoritmů. Tohoto zadání jsme splnili ve všech jeho bodech a aplikace je provozuschopná a do budoucna bude jistě dále rozvíjena. V úvodní části této práce jsme se věnovali obeznámení se s problematikou webových aplikací, dále jsme se zajímali o možnosti, které existují v oblasti vývoje webových aplikací za pomocí programovacího jazyku Python. Po obeznámení se, se všemi skutečnostmi sloužícími pro vývoj webových aplikací jsme se pustili do výběru vhodných šifer, které jsme se následně rozhodli implementovat do naší aplikace. Jednalo se o šifry AES, 3DES, DES a Smallie. První dvě jsme zvolili s ohledem na používání těchto šifer i v ostatních aplikacích a zároveň jsou tyto šifry stále dostatečně bezpečné a vhodné pro použití v naší aplikaci. Šifra DES byla zvolena ke dvěma již zmíněným především z historického hlediska, jelikož se jednalo o velmi významný šifrovací algoritmus ve své době. Šifra Smallie byla zvolena z důvodu, že tato šifra dosud nebyla nikde implementována a použita. Tuto novou kompaktní šifru vyvíjí vedoucí této bakalářské práce Ing. Petr Žáček. Jak již bylo zmíněno, tato šifra slouží především pro šifrování malých souborů, jelikož výpočetní čas této šifry je oproti ostatním šifrám mnohonásobně vyšší. Po výběru vhodných šifrovacích algoritmů, jsme se pustili do návrhu samotné webové aplikace. Vytvořili jsme si určité předpoklady, jak bychom chtěli, aby celá aplikace vypadala, a kousek po kousku jsme se pouštěli do zpracování celé této problematiky. V celé práci byl kladen důraz především na vysokou úroveň bezpečnosti, a aby aplikace byla především takzvaně „user friendly“, tudíž uživatelsky přívětivá a řešila za uživatele veškeré spojitosti a omezení. Zároveň jsme se tedy snažili o co nejlepší grafický dojem z celé aplikace. Všechny tyto části se nám podařilo zakomponovat dohromady a z těchto jednotlivých částí tak mohla vzniknout tato webová aplikace, která by mohla do budoucna sloužit pro širší využití veřejností. Zároveň celá tato webová aplikace nabízí velké množství možností pro další zlepšení. Jelikož jsme se snažili věnovat především funkčnosti celé aplikace, neřešili jsme responzivitu na různých zařízeních, zároveň jsme neřešili například kompatibilitu se staršími prohlížeči. Toto jsou vše cesty, kterými by se mohla aplikace zlepšovat do budoucna. Dále je prostor pro zvýšení bezpečnosti, kdy by aplikace mohla zpracovávat veškerá data jen v prohlížeči u uživatele a to například pomocí javascriptu nebo jQuery. Dále by mohla být vytvořena například mobilní aplikace pro možnost využít tuto šifrovací aplikaci v mobilních zařízeních, jako jsou dnes již všudy přítomné smartphony a tablety. Vylepšením by také mohla být možnost sice

si uložit uživatelskou konfiguraci, která byla použita při šifrování, ale tuto konfiguraci ukládat jako výřez obrazovky a ukládat jej jako obrázek například ve formátu JPG. Tato skutečnost by útočníkům velice ztížila možnost si informace o nastavení šifrovací aplikace získat. Velkým vylepšením by byla varianta, kdy by uživatel zadal jen svůj email a webová aplikace by sama konfiguraci šifry odeslala na tento email například v podobě již zmíněného obrázku. Případně by byla možnost i zašifrovaná data odesílat emailem na adresu uživatele a ten by si je následně stahoval. V tomto případě by ovšem data nejspíše musela být někde dočasně ukládána a následně odesílána pomocí emailového klienta na adresu uživatele.

ZÁVĚR

Tato bakalářská práce měla za cíl vytvořit funkční webovou aplikaci pro šifrování souborů. V této práci byl zvolen jako hlavní programovací jazyk Python a jeho frameworky. Jelikož bylo naším cílem umožnit použití webové aplikace veřejnosti, bylo také zároveň nutno vytvořit funkční webovou aplikaci přístupnou z internetu. K naplnění tohoto cíle jsme se museli nejdříve seznámit s různými technologiemi, které jsme hodlali použít v naší práci, proto se v úvodu teoretické části zabýváme některými základními pojmy, se kterými jsme se následně setkávali a museli jsme s nimi být schopni pracovat. Nejdříve se tedy věnujeme možnostem implementace webových aplikaci a nástrojům, které za tímto účelem můžeme využít, proto jsme se museli zmínit nejdříve o tom, co to je webová aplikace a co nám může nabídnout na rozdíl od webové stránky. Dále bylo třeba si říci, pomocí jakých technologií můžeme ať už webové stránky nebo webové aplikace tvořit. Popsali jsme si základní programovací jazyky pro tvorbu webových aplikací jako javascript, PHP a Python. Dva první zmíněné pracují na odlišném principu, než pokud tvoříme aplikaci v Pythonu. Z toho důvodu jsme museli pro zprovoznění takovéto aplikace použít framework, který nám pomůže vytvořit takovouto aplikaci. Následně jsme si tedy popsali jednotlivé frameworky, které nám mohou za tímto účelem sloužit, a pokusili jsme se vybrat ten nejvhodnější pro naši aplikaci. Z důvodu, nepotřebujeme využívat propojení s databázovým systémem, mohli jsme volit z jednodušších frameworků a nepotřebovali jsme komplexní systém jako třeba Django. V dalším kroku jsme zmínili různé šifrovací aplikace, které jsou v dnešní době dostupné a pracují na podobném principu, jako naše aplikace. Tento bod byl docela oříšek, protože najít aplikaci, která by uměla podobné věci, jako naše aplikace se zdálo nemožné, většina aplikací dokáže šifrovat pouze text, který uživatel zadá do vstupního pole a následně je uživateli vrácen tento text v druhém textovém poli. Tento nelehký úkol se nám ale zdařil a my jsme se tak mohli zaměřit na aplikaci pro šifrování souborů, která uměla šifrovat soubory, ale už v žádném případě není bezpečná. V následující části práce jsme poreferovali o vývoji kryptografie a šifrovacích algoritmech. Z výsledků našeho zkoumání jsme vybrali vhodné šifry, které by měli být v naší práci implementovány. Tyto šifry jsme si následně patřičně nastudovali a detailně popsali. Na závěr teoretické části jsme se zmínili o bezpečnosti komunikace mezi uživatelem a serverem s využitím šifrovaného přenosu pomocí HTTPS protokolu s využitím SSL certifikátu. Následující část této bakalářské práce se tak již mohla plně věnovat tvorbě samotné webové aplikace pro šifrování souborů. Zde jsme si na úvod nejdříve zkusili vytvořit představu o tom, jak bychom chtěli, aby naše aplikace vypadala a jaké prvky by měla obsahovat.

Následně jsme se věnovali popisu technologií, které použijeme při tvorbě této aplikace, část jsme věnovali popisu programovacího jazyka Python, kde jsme se setkali s problémy s kompatibilitou knihovny, kterou jsme potřebovali použít pro náš případ. Tento problém by tedy byl jedním z bodů, které by se dali řešit do budoucna jako další část práce. Následně jsme si tedy popsali i knihovnu, kterou jsme použili při tvorbě toto aplikace, jednalo se o knihovnu pyCrypto, která slouží pro šifrování za využití šifrovacích algoritmů. Tato knihovna nám nejvíce vyhovovala vzhledem ke zvoleným šifrovacím algoritmům. Následně jsme věnovali část popisu samotného frameworku Flask, který jsme při tvorbě aplikace použili. Použitím tohoto frameworku jsme si hodně usnadnili práci, jelikož tento framework za nás řeší komunikaci mezi front-end a back-end logikou celé aplikace. Popsali jsme si základní funkcionality celého frameworku a pustili jsme do práce na naší aplikaci. Vytvořili jsme si základní skripty v pythonu, které se nám starají o komunikaci s knihovnami pro šifrování, ošetřili jsme si délky klíčů a doplňování na patřičnou délku bloku pomocí paddingu. Tyto režimy doplňování na délku bloku jsme popsali pro ujasnění způsobu funkce každého z nich již v teoretické části práce. Další část práce se již věnovala návrhu grafické stránky celé webové aplikace, navrhli jsme si jednotlivé strany, které se uživateli zobrazují, a také jsme je provázali základní logikou celé aplikace, abychom byli schopni se mezi těmito stránkami přepínat. Dále jsme museli zajistit komunikaci mezi uživatelem a serverem. Potřebovali jsme totiž získávat od uživatele určité informace a také data, pomocí kterých jsme následně mohli provést šifrování a také následné vrácení souboru uživateli. Vyřešit stahování souboru zpět k uživateli do počítače se z počátku zdálo celkem náročné, jelikož požadavek na bezpečnost nám neumožňoval ukládat uživatelské data na server. Následně jsme tento problém ale vyřešili implementací vhodného javascriptu, s využitím filesystému webových prohlížečů. V dalším kroku jsme provedli samotnou implementaci námi vytvořeného rozhraní pro šifrovací funkce do aplikace, tudíž jsme již po vyhotovení tohoto kroku byli schopni zašifrovat soubory a nabídnout je uživateli zpět ke stažení. V neposlední řadě jsme se pobavili a bezpečnosti celé naší aplikace, popsali jsme jednotlivé kroky, které jsme provedli pro zvýšení bezpečnosti celé naší aplikace, ať už se jedná právě o bezpečné stahování souborů, zabezpečení uživatelského klíče nebo využití protokolu SSL pro zabezpečení spojení mezi uživatelem a serverem. Věnovali jsme se také popisu implementace celé aplikace na webový server pythonanywhere.com, který nám poskytl hosting pro naši aplikaci zcela zdarma v rámci 3 měsíční zkušební doby. Na závěr celé práce jsme pohovořili o krocích, které jsme podnikly během celé naší práce, a popsali jsme si možnosti vylepšení aplikace do budoucna. Při této

práci jsme se setkali se spoustou problémů, které jsme museli řešit. Ovšem díky těmto problémům jsme se dostávali více do hloubky celé logiky věci a pomohli nám k detailnějšímu a mnohem lepšímu pochopení celé aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] BITTNER, Jan. Technologie pro vývoj webových aplikací: Úvod a porovnání. IT Network [online]. Praha, 2014 [cit. 2018-01-22]. Dostupné z: https://www.it-network.cz/html-css/webove-aplikace/technologie-pro-vyvoj-webovych-aplikaci-uvod-a-porovnani

[2] Framework [online]. San Francisco, 2017 [cit. 2018-01-24]. Dostupné z: https://cs.wikipedia.org/wiki/Framework

[3] ČÁPKA, David. Úvod do JavaScriptu. IT Network [online]. Praha, 2013 [cit. 2018-01-23]. Dostupné z: https://www.itnetwork.cz/javascript/zaklady/javascript-tutorial-uvod-do-javascriptu-nepochopeny-jazyk

[4] ČÁPKA, David. Úvod do PHP a webových aplikací. IT Network [online]. Praha, 2015 [cit. 2018-01-22]. Dostupné z: https://www.itnetwork.cz/php/zaklady/php-tutorial-uvod-do-webovych-aplikaci

[5] BÍLEK, Petr. Úvod do jazyka Python [online]. Praha, 2005, 2015 [cit. 2018-01-23]. Dostupné z: http://www.sallyx.org/sally/python/python1.php

[6] Top 13 Python web frameworks software [online]. 2017 [cit. 2018-01-24]. Dostupné z: https://www.predictiveanalyticstoday.com/top-python-web-framework-software/

[7] What is web2py? [online]. Chicago, 2016 [cit. 2018-01-24]. Dostupné z: http://www.web2py.com/init/default/what

[8] RONACHER, Armin. Flask web development, one drop at a time. Flask [online]. 2018 [cit. 2018-04-10]. Dostupné z: http://flask.pocoo.org/

[9] Jak BitLocker funguje? [online]. Dublin, 2013 [cit. 2018-01-24]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/hh831507(v=ws.11).aspx

[10] Introducing [online]. Dublin, 2016 [cit. 2018-01-24]. Dostupné z: https://vera-crypt.codeplex.com/wikipage?title=Introduction

[11] ŠENKEŘÍK, Roman. Přednášky z předmětu Kryptologie. 2005-2016

[12] KUBÍČKOVÁ, Klára. Nepokořitelná Enigma: K prolomení nerozluštitelné šifry pomohla náhoda [online]. Praha, 2016 [cit. 2018-01-26]. Dostupné z: http://www.stoplusjednicka.cz/nepokoritelna-enigma

[13] ŽÁČEK, Petr. *Návrh nové symetrické šifry pro mobilní zařízení*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2014, 142 s. 3 s. příloh. Dostupné také z: http://hdl.handle.net/10563/30001

[14] 5 Common Encryption Algorithms and the Unbreakables of the Future [online]. USA: StorageCraft Technology Corporation, 2018 [cit. 2018-05-02]. Dostupné z: https://blog.storagecraft.com/5-common-encryption-algorithms/

[15] Using Padding in Encryption [online]. Australie: DI Management Services, 2016 [cit. 2018-05-02]. Dostupné z: https://www.di-mgt.com.au/cryptopad.html

[16] HANÁK. *Vysvětlení SSL certifikátů: Co jsou, jak fungují a proč je používat* [online]. Brno, 2016 [cit. 2018-02-21]. Dostupné z: https://www.master.cz/blog/co-jsou-ssl-certifikaty-a-ssl-protokoly-jak-funguji-vysvetleni-navod/

[17] Bootstrap 4 Tutorial [online]. Refsnes Data, 2018 [cit. 2018-05-02]. Dostupné z: https://www.w3schools.com/bootstrap4/default.asp

[18] GARREAU, Vincent. Particle.js [online]. Github, 2015 [cit. 2018-05-02]. Dostupné z: https://github.com/VincentGarreau/particles.js/

[19] Saving binary data as file using JavaScript from a browser [online]. stackoverflow, 2014 [cit. 2018-05-02]. Dostupné z: https://stackoverflow.com/questions/23451726/saving-binary-data-as-file-using-javascript-from-a-browser

[20] W3schools.com [online]. Refsnes Data, 2018 [cit. 2018-04-24]. Dostupné z: https://www.w3schools.com/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

3DES	Triple Data Encryption Standard
AES	Advance Encryption Standard
B	Byte
CBC	Cipher Block Chaining
CFB	Ciphertext FeedBack
CSS	Cascading Style Sheets
DES	Data Encryption Standard
ECB	Electronic Code Book
EDE	Encryption-Decryption-Encryption
EEE	Encryption-Encryption-Encryption
HTML	HyperText Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IV	Inicializační vektor
MB	Mega byte
OFB	Output FeedBack
PIP	Package management system
SSL	Secure Sockets Layer
WIFI	Wireless Fidelity
WSGI	Web Server Gateway Interface

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář **Text bakalářské práce** – obsahuje text bakalářské práce ve formátu PDF.
- Adresář **Zdrojové kódy** - obsahuje všechny zdrojové kódy, které jsme vytvořili při návrhu aplikace.
- Adresář **Odkaz** – obsahující textový soubor s webovou adresou, kde je aplikace dostupná na internetu.