Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Aplikační podpora výuky kryptografie s využitím PHP frameworku

Diplomová práce

Autor: Bc. Zdeněk Hejzlar
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Jana Medková Ph.D.

Hradec Králové Duben 2024

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.4.2024 Zdeněk Hejzlar

Poděkování:

Děkuji vedoucí diplomové práce Mgr. Janě Medkové Ph.D. za metodické vedení práce a časté konzultace teoretické i praktické části projektu.

Anotace

Tato diplomová práce se zaměřuje na vytvoření teoretické rešerše v oblasti užívání kryptografických primitiv v praxi a zaměřuje se na podporu výuky kryptografie na univerzitách. Z vzniklé rešerše je vytvořen seznam vybraných algoritmů, které jsou následně zkoumány po funkční stránce. Práce zahrnuje i přípravu návrhu pro novou webovou aplikaci a následnou krokovanou implementaci vybraných kryptografických algoritmů. Hlavním cílem je vytvoření užitečného nástroje pro studenty a zájemce o obor kryptografie, který pomůže porozumět základním konceptům a metodám v moderním zabezpečení dat a digitální komunikaci. Výsledkem této práce je výuková aplikace dostupná online na adrese https://educrypter.uhk.cz. Tento vzniklý projekt slouží jako užitečný prostředek pro zdokonalení znalostí v oblasti kryptografie a poskytuje uživatelům prostředí pro praktické vyzkoušení a porozumění různým kryptografickým technikám.

Annotation

This thesis focuses on the development of a theoretical investigation into the use of cryptographic primitives in practice and aims to support the teaching of cryptography in universities. From the resulting research, a list of selected algorithms is created, which are subsequently investigated from a functional point of view. The work includes the preparation of a proposal for a new web application and the subsequent step-by-step implementation of the selected cryptographic algorithms. The main goal is to create a useful tool for students and those interested in the field of cryptography to understand the basic concepts and methods in modern data security and digital communication. The result of this work is an educational application available online at https://educrypter.uhk.cz. Main goal of practical phase is to create project that serves as a useful tool for improving knowledge in the field of cryptography and provides users with an environment for hands-on testing and understanding of various cryptographic techniques.

Obsah

1 Úvod ... 1

2 Cíl a metodika práce ... 2

3 Využití kryptografických primitiv v praxi ... 4

3.1 Klasické šifry ... 4

3.2 Blokové šifry ... 6

3.3 Proudové šifry ... 9

3.4 Asymetrické šifry ... 11

4 Popis kryptografických primitiv ... 14

4.1 Klasické šifry ... 14

4.1.1 Caesarova šifra ... 15

4.1.2 Vigenèrova šifra ... 16

4.1.3 Vernamova šifra ... 17

4.2 Blokové šifry ... 18

4.2.1 DES ... 18

4.2.2 Simple DES ... 19

4.2.3 Triple DES ... 23

4.2.4 Triple S-DES ... 24

4.2.5 Blowfish ... 24

4.2.6 AES ... 26

4.2.7 Simple AES ... 28

4.3 Proudové šifry ... 29

4.3.1 A5/1 ... 30

4.4 Asymetrické šifry ... 31

4.4.1 RSA ... 31

4.4.2 Diffie-Hellman ... 33

5 Praktická část – implementace 35
5.1 Motivace pro vylepšení původního řešení 35
5.2 Volba frameworku, metodika vývoje a použité nástroje 36
5.2.1 Kontejnerizace 37
5.2.2 Implementace kontejnerizace 38
5.2.3 Ladění aplikace 39
5.2.4 Implementace nástroje k ladění aplikace 40
5.3 Návrh systému a implementace 41
5.3.1 User interface 44
5.3.2 Menu 46
5.3.3 Design jednotlivých šifer 47
5.4 Implementace algoritmů 48
5.4.1 Klasické šifry 48
5.4.2 Blokové šifry 51
5.4.3 Proudová šifra A5/1 56
5.4.4 Asymetrické šifry 58
5.5 Nasazení do produkčního prostředí 61
6 Testování aplikace 62
6.1 Otestování správnosti implementace algoritmů 62
6.2 Vyhodnocení dotazníkového šetření 65
6.3 Zpětná vazba od uživatelů 67
7 Závěr 68
8 Seznam použité literatury 69
9 Přílohy 75

Seznam obrázků

Obr. 1 – Caesarova šifra 15
Obr. 2 – Vigenèrova tabulka 17
Obr. 3 – Ukázka Vernamovy šifry 18
Obr. 4 – S-DES schéma 20
Obr. 5 – Permutace P10 20
Obr. 6 – Permutace P8 21
Obr. 7 – Úvodní permutace IP 21
Obr. 8 – Rundovní funkce S-DES 22
Obr. 9 – Rozšířená permutace (EP) 22
Obr. 10 – Permutace P4 23
Obr. 11 – Inverzní permutace IP^{-1} 23
Obr. 12 – Blowfish schéma 26
Obr. 13 – Schéma expanze klíčů S-AES 29
Obr. 14 – A5/1 posun registrů a určení šifrovacího bitu 31
Obr. 15 – Schéma RSA 32
Obr. 16 – Diffie-Hellmanova výměna klíčů 34
Obr. 17 – Schéma kontejnerizace 37
Obr. 18 – Vývojový diagram vytvoření obraz 38
Obr. 19 – Vývojový diagram vytvoření hlášení do Sentry 39
Obr. 20 – Ukázka odchycené chyby 41
Obr. 21 – MVC interakce 42
Obr. 22 - BaseCipher model 43
Obr. 23 – Dědičnost třídy Step 44
Obr. 24 - Původní design aplikace 45
Obr. 25 - Nový design 45
Obr. 26 – Změna jazyka a nové menu 46
Obr. 27 – Ukázka stránky s šifrou 47
Obr. 28 – Ukázka výpisu testovacího formuláře 48
Obr. 29 – Implementace Caesarovi šifry 49
Obr. 30 – Caesar – útok hrubou silou 50

Obr. 31 – Implementace Vigenèrovy šifry 51
Obr. 32 – Implementace S-DES 52
Obr. 33 – Substituční S-boxy S-DES 53
Obr. 34 - Feistelova funkce 54
Obr. 35 - Implementace násobení v Galoisově tělese 56
Obr. 37 - A5/1 - Vnitřní implementace inicializace registrů 57
Obr. 38 - A5/1 - Ukázka výstupu 57
Obr. 39 – Algoritmus RSA 59
Obr. 40 – Formulář Diffie-Hellman 60
Obr. 41 – Výstup algoritmu Diffie-Hellman 61
Obr. 42 – Vzorek testujících uživatelů 65
Obr. 43 – Hodnocení webové aplikace 66

Seznam zkratek

NÚKIB	Národní úřad pro kybernetickou a informační bezpečnost
NIST	Národní institut standardů a technologie
ENISA	Evropská agentura pro kybernetickou bezpečnost
IEC	Mezinárodní elektrotechnická komise
GMP	GNU Multiple Precision Arithmetic Library
PHP	PHP: Hypertext Preprocessor
JS	Javascript
CS	Code standard
DES	Data Encryption Standard
S-DES	Simplified Data Encryption Standard
T-DES	Triple Data Encryption Standard
T-SDES	Triple Simplified Data Encryption Standard
AES	Advanced Encryption Standard
S-AES	Simplified Advanced Encryption Standard
RSA	Rivest–Shamir–Adleman
GSM	Global System for Mobile Communications
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Sockets Layer
TLS	Transport Layer Security
XOR	Exclusive OR
IP	Initial Permutation
PC1/2	Permuted Choice
P10/P8	Permutation P10 / Permutation P8
SW	Switch
EP	Extended Permutation
S0/S1	S-box 0 / S-box 1
GF	Galois Field
ER	Entity relation
UI	User interface
SPA	Single Page Application
MPA	Multi Page Application

1 Úvod

V dnešní digitální éře, kde se internet stal základním zdrojem informací a prostředkem komunikace, je zabezpečení online dat a transakcí neodmyslitelnou součástí vývoje softwarových aplikací. K tomuto účelu slouží obor nazývaný kryptografie. Hlavním cílem kryptografie je šifrování dat, což umožňuje ochranu obsahu komunikace před neoprávněným přístupem. Kromě toho kryptografické algoritmy zajišťují integritu dat, což znamená schopnost detekovat neoprávněné změny během přenosu, čímž se garantuje důvěryhodnost informací. Dále kontrolují autenticitu dat, což zahrnuje ověření identity komunikujících stran. Bez těchto opatření bychom byli vystaveni zvýšenému riziku úniku citlivých dat a kybernetickým útokům.

Tato diplomová práce se soustředí na vývoj kryptografické webové aplikace v PHP frameworku, která má za cíl podpořit výuku principů kryptografických primitiv. Tvorba této práce zahrnuje důkladnou literární rešerši, která se zaměřuje na klíčové aspekty jednotlivých kryptografických algoritmů. Během rešerše jsou zkoumány existující algoritmy a techniky, jejich základní principy fungování, bezpečnostní aspekty a doporučené postupy pro jejich implementaci a použití. Práce se zabývá návrhem moderní webové aplikace a využívá nejnovější vývojářské nástroje. Řešenou problematikou je implementace krokovaných kryptografických primitiv a následné testování. Důraz je kladen na zajištění názorné vizualizace a korektnost daných implementovaných primitiv.

Cílem této práce je nejen poskytnout uživatelům přehled o kryptografických algoritmech, ale také nástroj sloužící jako vzdělávací prostředek podporující interaktivní učení a praktickou aplikaci teoretických znalostí. Implementace algoritmů přímo v kódu aplikace a možnost krokování umožňují uživatelům důkladné prozkoumání a pochopení fungování kryptografických mechanismů na hlubší úrovni. Tato práce přispívá k rozvoji vzdělávacích nástrojů v oblasti kryptografie a podporuje aktivní a interaktivní formy výuky.

2 Cíl a metodika práce

Cílem diplomové práce je navrhnout a implementovat webovou aplikaci pro podporu výuky kryptografie. Tato aplikace bude sloužit jako interaktivní prostředek pro výuku kryptografie a poskytne studentům ucelené prostředí pro studium teoretických konceptů a praktických aplikací kryptografických metod. Hlavním cílem je optimalizace a rozšíření stávajících řešení pomocí PHP frameworku, přičemž bude kladen důraz na zlepšení uživatelského prostředí a vytvoření jednoduchých vizualizací algoritmů. Aplikace bude navržena s ohledem na moderní technologické trendy a principy uživatelského designu.

V rámci práce bude provedena důkladná teoretická rešerše v oblasti kryptografie, která poskytne pevný teoretický základ pro návrh a implementaci aplikace. Budou zkoumány klíčové aspekty symetrické a asymetrické kryptografie a jednotlivých vybraných algoritmů. V praktické části práce vznikne návrh nového vzhledu webové stránky a jednoduchý koncept celého systému. Po vytvoření návrhů bude pomocí agilního vývoje přepracována původní webová aplikace s využitím PHP frameworku a následně zavedena nová navržená architektura. Jednotlivé algoritmy budou manuálně implementovány tak, aby umožnily přehledné zobrazení všech provedených kroků. K řádné vizualizaci vzniknou i nové textové podklady, schémata a doprovodné obrázky na základě provedené teoretické rešerše. Vzniklé materiály budou přidány k jednotlivým algoritmům a budou přeloženy i do angličtiny. Výsledná aplikace bude nasazena na server a vyzkoušena v produkčním prostředí. V této fázi proběhne testování uživateli, kteří vyplní dotazník se zpětnou vazbou. Odpovědi z dotazníkového šetření budou zpracovány a využity jako výstup pro testovací fázi.

Aplikace bude vytvořena pomocí PHP verze 8.3 a frameworku Laravel [1] ve verzi 9.19. Pro rozšíření funkcí bude využita knihovna GMP (GNU Multiple Precision) [2] spolu s Laravel JS Localization 1.11, Sentry 4.1 [3], Bootstrap 5.2.1 a JQuery 3.6.1. Kvalita kódu bude kontrolována pomocí coding standard (CS) a nástroje Psalm [4] ve verzi 2.8. Pro snadný deployment aplikace bude využit Docker [5] s obrazem php:8.3-apache [6]. Textové překlady na webu budou zpracovány

pomocí rozšíření translation pro PHPStorm. Pro formátování textů a vytvoření osnovy pro testovací formulář bude použit ChatGPT 3.5. Celý projekt bude veřejně přístupný z github repozitáře https://github.com/hejzlzd1/EducrypterUHK/.

Celkovým cílem je tedy vytvořit komplexní nástroj, který usnadní studentům porozumění kryptografickým principům a umožní jim praktické aplikace nabytých znalostí prostřednictvím interaktivních testovacích formulářů.

Tato diplomová práce navazuje na bakalářskou práci [7] a rozšiřuje původní řešení vytvořené výukové aplikace. Původní vytvořená aplikace se nachází na adrese https://edu.uhk.cz/~hejzlzd1.

3 Využití kryptografických primitiv v praxi

Kryptografie od pradávna slouží jako ochrana citlivých informací a zabezpečení komunikace. Od prvního užití Caesarovy šifry či jednodušších šifrovacích postupů prošla kryptografie dlouhým vývojem, až do fáze moderních systémů, které nelze prolomit v reálném čase. S nástupem moderních technologií a rostoucí komplexity digitálního světa se kryptografie stala klíčovým prvkem v oblasti zabezpečení. Nápomocným se stal i rostoucí výpočetní výkon, který umožnil tvorbu komplexnějších kryptografických systémů. Využití kryptografických primitiv v praxi představuje esenciální součást všech systémů a aplikací, které se spoléhají na ochranu dat či autentizaci.

Tato kapitola se zaměřuje na průzkum možností v oblasti využití kryptografických primitiv v reálném prostředí, jejich bezpečnost a alternativy. Také obsahuje důvody pro přidání vybraných šifer do nově vzniklé výukové aplikace. Je nutné zmínit, že volba vhodného algoritmu do nového softwaru je značně ovlivněna hladinou požadované úrovně bezpečnosti v porovnání s výkonem. Obecně je však vhodné využívat doporučených algoritmů stanovených organizacemi jako jsou Národní institut standardů a technologie (NIST), Evropská agentura pro kybernetickou bezpečnost (ENISA) či Mezinárodní elektrotechnická komise (IEC).

3.1 Klasické šifry

Klasické šifry mají v historii kryptografie významnou roli. Byly používány před rozvojem moderní kryptografie a poskytovaly základní mechanismy pro šifrování a udržení důvěrnosti komunikace. Tyto šifry využívaly jednoduché operace, jako je posun písmen v abecedě nebo substituci písmen podle klíče, což umožňovalo přeměnu čitelného textu na nečitelnou podobu, která byla bez znalosti správného klíče obtížně rozluštitelná. Použití těchto algoritmů není doporučeno v moderním prostředí, protože neodpovídají současným standardům zabezpečení. Tyto algoritmy často trpí nedostatky, které umožňují jejich snadné a rychlé prolomení pomocí moderních kryptoanalytických metod a technologií. Do webové

aplikace byli vybráni nejznámější zástupci těchto šifer, aby demonstrovali fundamentální principy a přístupy při šifrování a dešifrování. Klasické šifry jsou známé především díky Caesarově a Vigenèrově šifře či Scytale.

Principem Caesarovy šifry je posun abecedy a nahrazování znaků z nově vzniklé abecedy. Jedná se o jednoduchou ukázku monoalfabetické substituční šifry. Využití této šifry v reálném světě není bezpečné, protože je snadno prolomitelná pomocí útoků hrubou silou (anglicky „brute force“), kdy jsou testovány všechny možné posuny, což moderní počítače zvládnou velmi rychle. Implementace Caesarovy šifry do výukové webové aplikace má klíčový význam, jelikož poskytuje studentům jednoduché a srozumitelné základní koncepty kryptografie. Caesarova šifra nabízí přístupný způsob, jak pochopit principy šifrování a dešifrování zpráv pomocí klíče, což může posloužit jako vstupní bod pro další studium kryptografických algoritmů.

Vigenèrova šifra je nejznámějším zástupcem polyalfabetické šifry. Algoritmus využívá několika substitučních abeced, což značně zvyšuje bezpečnost oproti Caesarově šifře. Navzdory vyšší bezpečnosti je náchylná k prolomení pomocí kryptoanalytických technik, jako je statistická analýza, která může odhalit vzory v textu. Implementování v rámci aplikace umožní jednoduché pochopení skládání kryptografických primitiv a poskytne vhled do problematiky polyalfabetických šifer.

Vernamova šifra, známá také jako one-time pad, pracuje na principu XORování klíče se vstupními daty. Teoreticky poskytuje absolutní bezpečnost, pokud je použita správně [8], jelikož vyžaduje klíč stejně dlouhý jako vstupní text. Nicméně získání a bezpečné distribuování klíčů odpovídající délce šifrovaného textu je často nepraktické a složité. Tento algoritmus je vhodný k uvedení do problematiky šifer pracujících na binárních datech. Taktéž poukazuje na problematiku managementu klíčů.

3.2 Blokové šifry

Blokové šifry jsou kryptografické algoritmy, které se používají k zabezpečení dat prostřednictvím šifrování bloků dat o pevné délce a dostatečně dlouhých klíčů. Jedná se o jednu z nejčastěji používaných metod šifrování, která nachází široké uplatnění v různých oblastech, včetně komunikace přes internet, ochrany dat a bezpečnosti informačních systémů. Existuje mnoho různých algoritmů blokových šifer, které se liší svou bezpečností, rychlostí a dalšími vlastnostmi. Některé z nejznámějších blokových šifer zahrnují Advanced Encryption Standard (AES), Triple Data Encryption Algorithm (T-DES), Serpent, Twofish, Camellia, Skipjack (tajná šifra) či SHA-3. V současné době (2024) je organizací NIST doporučován algoritmus AES jako nejbezpečnější při správné volbě klíče.

Národní úřad pro kybernetickou a informační bezpečnost (NÚKIB) pro rok 2023 doporučil algoritmy AES, Camellia, Serpent při délce klíče 128, 192 a 256 bitů a Twofish při délce klíče 128 až 256 bitů. NÚKIB doporučuje využití delšího klíče o délce 256 bitů, jelikož šifry s kratším klíčem jsou kvantově zranitelné (prolomitelné za využití kvantového počítače). Mezi dosluhující šifry patří Triple DES s unikátními klíči o 112 bitech a Blowfish s klíčem 128 bitů. Tyto šifry je doporučeno postupně nahrazovat pomocí AES [9].

Algoritmus AES nachází uplatnění například v šifrování disků a USB za pomoci softwaru BitLocker [10], VeraCrypt či TrueCrypt. Jakožto standard je AES využíván i ve vládním a vojenském sektoru k šifrování tajných dat. Implementace tohoto algoritmu je také součástí ve webových prohlížečích, ve kterých zajišťuje HTTPS spojení [11]. Standard je často využíván v různorodých aplikacích a hardwaru, jelikož pro zrychlení šifrování existuje instrukční sada AES-NI [12], která je součástí většiny novějších procesorů. Algoritmus je široce rozšířen díky své vysoké bezpečnosti a rychlosti.

Šifra Camellia vyvinutá společnostmi Nippon Telegraph and Telephone Corporation (NTT) a Mitsubishi Electric Corporation je šifra vyvinutá v Japonsku.

Tato šifra díky svému původu nachází časté uplatnění v japonských aplikacích a v některých mezinárodních aplikacích a protokolech. Koncept tohoto primitiva byl v roce 2006 uvolněn jako open source pro projekty, jako jsou například OpenSSL, Firefox, Linux kernel a FreeBSD [13].

Serpent a Twofish jsou často využívány k zabezpečení dat v cloudovém úložišti. Uvedené šifry jsou také velmi často využívány jako alternativa pro současný standard AES, jelikož se jedná o poměrně rychlé, bezpečné a efektivní algoritmy.

V rámci implementace v aplikaci byly vybrány nejznámější algoritmy, které studentům jednoduše přiblíží koncepty symetrické kryptografie. Prvním takovým šifrovacím systémem je AES, původně známý pod názvem Rijndael. V současné době je AES široce uznávaným standardem pro symetrické šifrování. Rijndael v roce 2000 zvítězil v soutěži o nový kryptografický standard, v následujícím roce 2001 byl tento algoritmus veřejně vyhlášen standardem a doporučen k používání [16]. Pro své fungování používá AES 128bitové datové bloky a šifrovací klíče o délkách 128, 192 nebo 256 bitů. Díky kombinaci komplexních operací a dostatečně dlouhého klíče poskytuje šifra vysokou úroveň bezpečnosti.

Pro výukové účely vznikl Simplified AES (S-AES), který je značně zjednodušenou verzí AES [14]. Pomocí této verze lze lehce představit základní koncepty a návrhy plného algoritmu, což ho činí vhodným k implementaci ve výukové aplikaci. Zjednodušené představení těchto konceptů je klíčové k pochopení ostatních, úplných komplexních kryptografických systémů.

Data Encryption Standard (DES) byl navržen na počátku roku 1970 a prezentován jako standard v roce 1977. Algoritmus DES byl využíván v sektoru bankovnictví, komerční sféře, ke komunikaci po nezabezpečené lince a k šifrování dat. V současné době není považován za bezpečný kvůli krátkému 56bitovému klíči je možné šifru snadno prolomit pomocí útoku hrubou silou [11]. Pro prolomení této šifry stačí vyzkoušet 2^{56} klíčů, což s postupem technologií není na moderním hardwaru problematickým úkolem.

První prolomení šifry je dokumentováno v roce 1997, kdy při útoku hrubou silou bylo využito distribuované sítě. Tento útok trval 96 dní a byl nakonec úspěšný. Postupem času následovaly další úspěšné pokusy o prolomení šifry, které postupně optimalizací snižovaly potřebný čas k prolomení [15]. Několik let poté byla vyhlášena soutěž o nový kryptografický standard, kterým se stal algoritmus AES. Než byl stanoven nový standard, DES nahradily šifry, jako jsou například Triple DES, Blowfish a IDEA.

Pro jednodušší pochopení konceptů vznikl algoritmus S-DES [14], který je zjednodušenou verzí původního DES. Tento algoritmus sice není vhodný pro nasazení v reálných aplikacích, které požadují vysoké zabezpečení, ale v kontextu výuky kryptografie jde o jednoduchou cestu, jak představit základní koncept rozsáhlé blokové šifry, jako je DES.

V době, kdy nebyl stanoven oficiální bezpečný standard, se stal Triple DES (T-DES) dostatečnou náhradou za původní DES a byl v roce 1999 doporučen k používání. Ve vnitřní implementaci se jedná o jednoduché trojí využití DES, což má za následek zvýšení bezpečnosti. Algoritmus může využívat tři společné klíče nebo dva až tři unikátní klíče. S vyšším počtem unikátních klíčů stoupá míra bezpečnosti na úkor výkonnosti. Snížení výkonu je pak způsobeno tím, že je pro každý klíč (potažmo průchod DES) potřeba zopakovat proces generování rundovních klíčů a každý průchod algoritmem znamená zpomalení. Kvůli tomuto faktu je použití algoritmu dosti omezené [8]. Triple DES je využíván v některých situacích, kdy je potřeba zachovat zpětnou kompatibilitu s původním DES, avšak v současné době není považován za bezpečný [16] a je doporučeno se vyvarovat jeho použití [17]. Vhodnou náhradou je AES či Blowfish a jeho alternativy.

Implementace Triple DES v rámci aplikace je podstatnou částí, která studentům představí základní skládání kryptografických primitiv a vliv délky klíče na bezpečnost a výkonnost celého kryptosystému.

Dalším významným a známým algoritmem se stal Blowfish, který byl navržen roku 1993 jakožto jedna z možných náhrad pro tehdy prolomený DES. Algoritmus sloužil jakožto náhrada prolomeného standardu díky tomu, že nebyl patentován [18] a bylo jej možné využít v komerční sféře. Algoritmus je řazen mezi nejrychlejší šifry a do roku 2016 nebyly zjištěny bezpečnostní problémy v závislosti na délce klíče (32-448 bitů) [19]. Postupem dalšího vývoje v oblasti kryptografie byl upozaděn vhodnější alternativou s názvem Twofish, která byla jedním z finalistů v soutěži o nový standard. Twofish od stejného tvůrce navázal na původní strukturu Blowfish s využitím nových principů, jako jsou například MDS matice (maximum distance separable) či Pseudo-Hadamardova transformace (viz. [20]). Tento algoritmus se stal vhodnou náhradou a je doporučován i samotným autorem šifry [21].

Algoritmus Blowfish představuje ve výukové aplikaci reprezentativní příklad algoritmu založeného na Feistelově síti, poskytující názornou ukázku průběhu plné verze šifrovacích primitiv. Z hlediska historie má zvláštní význam, neboť na určitou dobu nahradil šifrovací standardy v komerční sféře. V porovnání s Twofish, který pracuje s dvojnásobnou velikostí datových bloků, je Blowfish značně jednodušší k pochopení a lépe se hodí k ilustraci kryptografických operací. Zatímco Twofish vyžaduje pokročilejší studium, Blowfish poskytuje studentům přístupnější ukázku prováděných operací.

3.3 Proudové šifry

Proudové šifry jsou klíčovým prvkem v oblasti kryptografie poskytujícím alternativu k blokovým šifrám. Oproti blokovým šifrám, které pracují s daty po pevně definovaných blocích, proudové šifry operují s daty nepřetržitě, byte po bytu či bit po bitu. Jejich základní princip spočívá v generování pseudo-náhodného proudového klíče, který je poté kombinován s původními daty za použití operace XOR. Mezi hlavní představitele proudových šifer patří Vernamova šifra, RC4, Salsa20, ChaCha20, rodina Snow (Snow2G, Snow3G) a kryptografická rodina A5

(A5/1, A5/2, A5/3) [22]. Proudové šifry jsou v současnosti nahrazovány pomocí blokových šifer, jako je například AES [23].

NÚKIB v otázce aktuálních šifer (pro rok 2023) uvádí doporučení k používání algoritmů SNOW 2.0 a SNOW 3G při délce klíče 128 a 256 bitů. Dále je doporučována šifra s názvem ChaCha20 při využití klíče o délce 256 bitů. NÚKIB taktéž uvádí zastaralou proudovou šifru s názvem Kasumi [9]. Tato šifra by měla být nahrazena bezpečnější alternativou či blokovou šifrou.

Algoritmus ChaCha20 je například využíván v protokolu Internet Key Exchange Protocol version 2 (IKEv2), který slouží k navázání bezpečného spojení mezi dvěma komunikujícími stranami skrze privátní síť (VPN). ChaCha20 je v kombinaci s autentizačním algoritmem Poly využit v protokolu IPsec [24].

Rodina algoritmů SNOW je využívána napříč zabezpečením sítě GSM. SNOW 3G je například integrován v kryptografickém systému UEA2 a UIA2, které slouží k ověřování integrity a důvěrnosti přijatých zpráv [25]. V oblasti proudových šifer byl algoritmus SNOW 2.0 standardizován a je v některých zemích využíván pro šifrování přenosu po GSM.

Proudové šifry přinášejí několik výhod, jako je rychlost a efektivita přenosu dat [26], což je zvláště užitečné při šifrování velkých objemů dat v reálném čase. Jsou také vhodné pro zařízení s omezenými prostředky, jako jsou embedded systémy, díky své schopnosti poskytnout účinné šifrování s minimálním využitím paměti a výpočetních zdrojů. Nicméně používání proudových šifer vyžaduje správné generování a synchronizaci proudového klíče mezi šifrovacím a dešifrovacím zařízením, což může být problematické, pokud není implementace provedena správně. Ztráta synchronizace navíc může způsobit ztrátu integrity dat.

Jedním z hlavních představitelů proudové šifry se stal algoritmus A5/1, vytvořený v roce 1987. Algoritmus funguje na bázi lineárních posuvných registrů se zpětnou vazbou a operaci XOR. Významným se stal díky využití v síti GSM (2G), kde

sloužil k šifrování dat přenášených mezi zařízením a základnovou stanicí. Jedná se o jednu z nejstarších proudových šifer, která byla používána ve větším měřítku. Algoritmus byl nahrazen kvůli nalezeným bezpečnostním rizikům [27]. Při tvorbě algoritmu vzniklo i několik bezpečnostně slabších variant s názvy A5/2 a A5/3, které se využívaly mimo Evropu.

Implementace šifry A5/1 ve výukové aplikaci je relevantní především kvůli demonstraci způsobu šifrování komunikace v síti GSM. Zároveň jasně ilustruje principy a výhody použití inicializačního vektoru v procesu šifrování. Tato šifra mimo jiné poskytuje důležité základy práce s lineárními posuvnými registry se zpětnou vazbou a manipulací s bitovými proudy. A5/1 tak činí důležitý základ pro nauku o proudových šifrách a jejich konceptech.

3.4 Asymetrické šifry

Asymetrické šifry jsou klíčovým prvkem v kybernetické bezpečnosti a nacházejí široké uplatnění v elektronické komunikaci. Tyto šifry umožňují bezpečné šifrování e-mailů, online chatování, VPN připojení a další formy elektronické komunikace díky zajištění ochrany přenášených dat. Dalším důležitým aspektem je vytváření digitálních podpisů, které slouží k ověření autenticity digitálních dokumentů a transakcí. Asymetrické šifry se také používají pro správu digitálních certifikátů, které zajišťují bezpečné připojení k webovým stránkám pomocí protokolu HTTPS. Pro digitální podpisy a šifrování jsou často využívány algoritmy s názvem RSA, DSA či algoritmy využívající eliptických křivek (ECC) [28].

NÚKIB v rámci asymetrických šifer pro digitální podpisy doporučuje využití algoritmů DSA s délkou klíče 3072 bitů (a více) s délkou parametru cyklické podgrupy 256 bitů. Dalším doporučeným algoritmem je EC-DSA (DSA využívající eliptické křivky) s délkou klíče 256 bitů. Mezi další uvedené algoritmy patří Rivest-Shamir-Adleman Probabilistic Signature Scheme (RSA-PSS) o délce klíče 3072 bitů a více, Elliptic Curve Schnorr Signature Algorithm (EC-Schnorr) s klíčem dlouhým 256 bitů a více. Mezi zastaralé pak patří výše uvedené šifry za využití menšího klíče.

Pro výměnu klíčů a šifrování klíčů pak NÚKIB doporučuje následující algoritmy a délky klíčů. Pro Diffie-Hellman (DH) se doporučuje použití klíčů o délce 3072 bitů a více, s délkou parametru cyklické podgrupy 256 bitů a více. Pro Elliptic Curve Diffie-Hellman (ECDH) se doporučuje délka klíčů 256 bitů a více. Elliptic Curve Integrated Encryption System – Key Encapsulation Mechanism (ECIES-KEM), Provably Secure Elliptic Curve – Key Encapsulation Mechanism (PSEC-KEM) a Advanced Cryptographic Engine – Key Encapsulation Mechanism (ACE-KEM) by měly využívat klíče o délce 256 bitů a více. Pokud jde o Rivest-Shamir-Adleman (RSA), doporučuje se použití klíčů o délce 3072 bitů a více. Tento parametr platí pro Optimal Asymmetric Encryption Padding (RSA-OAEP) a pro Key Encapsulation Mechanism (RSA-KEM). Stejně jako je tomu u algoritmů pro digitální podpisy, výše uvedené algoritmy za využití menších klíčů jsou považovány za zastaralé. [9]

V kontextu asymetrické kryptografie se často využívají protokoly pro výměnu šifrovacích klíčů, které umožňují bezpečně sdílet tajný klíč přes veřejné kanály komunikace. Tímto způsobem je dosaženo bezpečné výměny šifrovacího klíče, což je klíčový prvek v zajištění důvěrnosti a integrity dat v kybernetické komunikaci. Jedním z těchto protokolů, který je často využíván pro dohodu dvou stran na společném tajném klíči, je Diffie-Hellman (DH), který byl navržen v roce 1976. Tento protokol představil metodu pro bezpečnou výměnu šifrovacích klíčů mezi dvěma stranami přes veřejně přístupný kanál. Od té doby se stal kritickým prvkem pro zajištění bezpečnosti komunikace v moderním digitálním světě.

Protokol Diffie-Hellman se často používá pro vytvoření bezpečného spojení s webovými stránkami pomocí protokolu HTTPS, kde umožňuje výměnu klíčů pro šifrování komunikace, což zajišťuje ochranu přenášených dat. Dále je Diffie-Hellman klíčovým prvkem ve virtuálních privátních sítí (VPN), kde zajišťuje výměnu šifrovacích klíčů mezi uživatelem a druhým účastníkem komunikace. Využití tedy hojně nalézá v protokolu TLS [29].

Tento algoritmus je bezpečný, avšak existuje riziko prolomení v případě útoku Man-in-the-middle [29]. I přes fakt, že je algoritmus poměrně bezpečný a účinný, je

možné přidat další vylepšení či komplexnější akce k získání lepšího stupně bezpečnosti. Mezi takto vylepšené algoritmy patří například Triple Diffie-Hellman (3-DH) či Elliptic-curve Diffie–Hellman (EC-DH). V kontextu bezpečnosti 3-DH oproti DH využívá další 4 veřejné klíče k vytvoření společného tajemství. EC-DH v porovnání s DH využívá eliptických křivek jako matematickou strukturu pro operace s klíči [30]. Diffie-Hellmanův algoritmus pro výměnu klíčů je v kontextu výukové aplikace důležitým prvkem, který ilustruje, jakým způsobem mohou být předány soukromé klíče pro další komunikaci. Taktéž představuje princip výpočtů primitivního kořene modulo *p* a základy modulární aritmetiky.

V kontextu asymetrické kryptografie hraje také značně důležitou roli algoritmus Rivest-Shamir-Adleman (RSA) vytvořený v roce 1977, který využívá matematické obtížnosti faktorizace velkých prvočísel. Svou bezpečností se stal nejznámějším a nejpoužívanějším v oblasti asymetrické kryptografie. Díky svému využití pro šifrování a dešifrování dat, ale také pro tvorbu digitálních podpisů, se tak stal algoritmus velmi rychle populárním.

Hlavní oblastí využití RSA je zabezpečená elektronická komunikace, kde se používá pro šifrování e-mailů, online chatů a dalších forem komunikace, což zajišťuje důvěrnost a bezpečnost dat. Dále slouží k vytváření digitálních podpisů, které ověřují autenticitu digitálních dokumentů a transakcí. Také se uplatňuje při bezpečném přihlašování uživatelů do online systémů prostřednictvím RSA tokenů a klíčů (příkladem je SSL na platformě Github). V oblasti webové bezpečnosti je klíčovým prvkem pro zabezpečení webových stránek pomocí SSL/TLS protokolu [11], zajišťujícího šifrování dat přenášených mezi serverem a klientem.

Algoritmus je vhodný k implementaci, jelikož představuje problematiku prvočíselného rozkladu velkých čísel. Jedná se o velmi podstatný asymetrický kryptografický systém, který je hojně využíván v softwaru i hardwaru především pro svou vysokou míru bezpečnosti. Dalším důvodem je nutnost představení fundamentálních matematických postupů pro tvorbu šifrovacích klíčů.

4 Popis kryptografických primitiv

Následující kapitola se zabývá teoretickou rešerší vybraných kryptografických primitiv. Podklady vzniklé touto rešerší budou následně využity při implementaci jednotlivých algoritmů ve výukové webové aplikaci. V kapitole se nachází teoretické informace o klasických šifrách, blokových šifrách, proudových šifrách a asymetrických kryptografických systémech.

4.1 Klasické šifry

Pojmem klasické či tradiční algoritmy jsou často nazývány takové šifry, které se používaly v historii a bylo možné je rozepsat za pomoci tužky a papíru díky jejich jednoduchosti.

Ve světě kryptografie představují tradiční šifry základní techniky, mezi něž patří substituce a transpozice. Substituce je procedura, při níž se znaky nešifrovaného textu, nazývaného též otevřený text (anglicky "plain text"), nahrazují odpovídajícími znaky ze substituční abecedy. Substituční šifry se dále klasifikují jako monoalfabetické a polyalfabetické, což určuje počet použitých substitučních abeced a princip, podle kterého probíhá nahrazování znaků.

V rámci monoalfabetických šifer se znaky nahrazují pouze pomocí jedné neměnné substituční abecedy. Naopak polyalfabetické šifry využívají několik substitučních abeced, přičemž každá z těchto abeced je použita s různými pravidly pro volbu odpovídajícího znaku

Transpozice jsou další základní technikou používanou v tradiční kryptografii. Transpozice se zaměřuje na změnu pořadí znaků v textu, aniž by byly samotné znaky nahrazovány jinými symboly. Při použití transpozice jsou znaky původního textu přeuspořádány podle určitého klíče nebo pravidla.

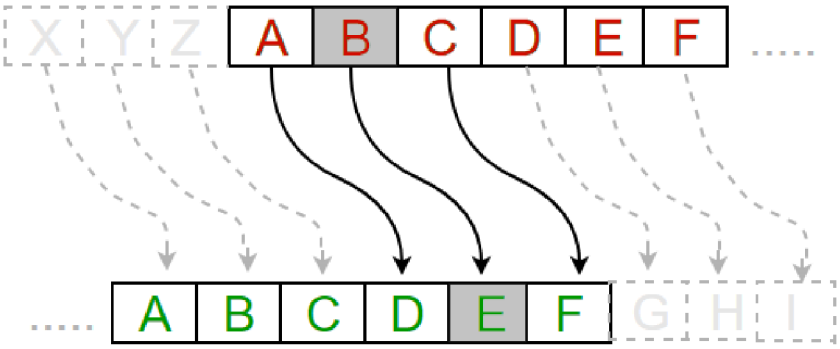
Uvedené operace tvoří základní kámen kryptografických komplexních systémů a jsou četně využívány. Integrace ukázek klasických šifer, jako je Caesarova,

Vernamova nebo Vigenèrova šifra, vytváří v navržené webové aplikaci nezbytný kontext pro pochopení historie a základních principů kryptografie. Jejich studium je prvním krokem k pochopení moderních šifrovacích protokolů a technik. Vizuální zpracování těchto šifer poskytuje uživatelům prostor pro praktická cvičení či experimentování s implementovanými šifrovacími metodami.

4.1.1 Caesarova šifra

Caesarova šifra, jakožto jedna z nejstarších a nejjednodušších klasických šifer, ilustruje základní principy substitučních abeced. Její jednoduchost spočívá v tom, že každé písmeno v otevřeném textu je posunuto o určitý počet pozic v abecedě, čímž vzniká šifrovaný text. Tento posun, historicky stanovený na tři pozice, je základním prvkem šifrovacího algoritmu [31]. Dešifrování probíhá obdobným způsobem jako šifrování, jen je posun prováděn na opačnou stranu. Princip posunutí abecedy a substituce znaků lze vidět v obrázku níže (viz. Obr. 1).

Zašifrování slova „*Trik*“, při posunu o tři znaky, může probíhat následovně. Písmeno „T“ je nahrazeno písmenem „W“, znak „r“ je nahrazen pomocí „u“, písmeno „i“ nahradí „l“ a poslední znak „k“ nahrazuje „n“. Tímto vznikne nový zašifrovaný text „Wuln“.



Obr. 1 – Caesarova šifra. Zdroj: vlastní zpracování

Tento šifrovací systém získal své jméno právě podle legendy, že Julius Caesar využíval podobný mechanismus k šifrování své osobní korespondence, což dodává této šifře historický a kulturní význam. Nicméně, i když je Caesarova šifra poměrně

snadno pochopitelná a použitelná, má své omezení a nedostatky, které ji činí nevhodnou pro moderní aplikace kryptografie.

Mezi tyto nedostatky patří snadná dešifrovatelnost prostřednictvím útoku hrubou silou (anglicky "brute force attack"), kdy je zkoumáno každé možné posunutí abecedy (tj. 26 možných variant). Další nevýhodou je, že substituované znaky se opakují, což značně ulehčuje dešifrování šifrového textu pomocí frekvenční analýzy. Aplikace frekvenční analýzy usnadňuje dešifrování šifrového textu tím, že umožňuje jednoduchou predikci písmen na základě jejich častého výskytu v daném jazyce. Podmínkou využití této analýzy je znalost jazyka, ve kterém je původní šifrovaná zpráva napsána. Pomocí výstupního zašifrovaného textu lze také provést dešifrování za pomocí kontextuální analýzy, při které můžeme sledovat určité vzory v textu a z nich dedukovat další informace. V rámci aplikace je u Caesarovy šifry metoda brute force implementována.

4.1.2 Vigenèrova šifra

Jedna z nejznámějších a zároveň jednoduchých polyalfabetických šifer je Vigenèrova šifra. V tomto schématu se sada monoalfabetických substitučních pravidel skládá z 26 Caesarových šifer s posuny od 0 do 25 [32]. Každý znak je nahrazován pomocí hodnoty získané v závislosti na vstupním znaku a znaku ze šifrovacího klíče. Tento princip lze jednoduše pochopit z Vigenèrovy tabulky (viz. Obr. 2). Ve vrchní části této tabulky je vyhledán momentálně šifrovaný znak ze vstupního textu. První sloupec slouží k vyhledání znaku z klíče. Spojnice sloupce a řádku pak vytváří nový šifrovaný znak. Tento postup lze aplikovat i na dešifrování, přičemž první řádek je využit na vyhledávání znaku šifrovaného textu.

Slovo „*TRIK*" za použití klíče „*KROK*" lze Vigenèrovou šifrou zašifrovat pomocí následujícího postupu. Pro písmeno „T" z otevřeného textu hledáme průnik v sloupci „T" a v řádku „K", čímž získáme písmeno „D". Na spojnici sloupce „R" a řádku „R" získáme písmeno „I". Znak „I" je nahrazen ze sloupce „I" a řádku „O" písmen „W". Poslední znak je nahrazen pomocí sloupce „K" a řádku „K" písmenem „U". Výsledkem šifrování je text „DIWU".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Obr. 2 – Vigenèrova tabulka. Zdroj: Michigen Technology University [33]

4.1.3 Vernamova šifra

Vernamova šifra neboli jednorázová tabulková šifra (anglicky „one-time pad“) je jednoduchý šifrovací algoritmus, který šifrovaný text vytváří pomocí exkluzivního logického součtu (XOR). Operace se provádí nad otevřeným textem a stejně dlouhým šifrovacím klíčem. Tato šifra je absolutně bezpečná [34] v případě, že je použit jednorázový náhodný klíč. Algoritmus lze používat i na různých vstupních abecedách (např. binární). Z důvodu požadavku na délku klíče je však tato šifra v praxi špatně využitelná.

Při použití této šifry je text zašifrován pomocí binárního klíče stejné délky, v případě odlišné délky je klíč zkrácen či prodloužen na délku vstupního textu. Pokud je však klíč takto upraven, je snížena bezpečnost algoritmu. Každý znak je následně kombinován s odpovídajícím znakem klíče pomocí operace XOR (viz. Obr. 3), čímž vzniká stejně dlouhý zašifrovaný/dešifrovaný binární text.

Šifrování

Otevřený text	1101001
Klíč	1101100
Zašifrovaný text (P ⊕ K)	0000101

Dešifrování

Zašifrovaný text	0000101
Klíč	1101100
Otevřený text (P ⊕ K)	1101001

Obr. 3 – Ukázka Vernamovy šifry. Zdroj: vlastní zpracování

4.2 Blokové šifry

Blokové šifry jsou kryptografické algoritmy, které pracují s daty v pevně definovaných blocích pevné délky. Každý blok dat je šifrován nebo dešifrován pomocí stejného klíče a algoritmu. Před zpracováním vstupního textu jsou data rozdělena na menší části zvané bloky, které jsou šifrovány nezávisle na ostatních blocích. Tento typ šifer se využívá pro šifrování dat, autentizaci a bezpečnou komunikaci po síti, jako je například internet (viz. kapitola Využití kryptografických primitiv v praxi). Zahrnutí blokových šifer do aplikace přináší uživatelům důležitý kontext a vhled do moderních metod šifrování používaných pro zabezpečení dat v digitálním světě.

Do aplikace jsou implementovány algoritmy, u kterých je možné jednoduše představit používané komplexní metodiky. Pro studijní účely byly použity zjednodušené verze dvou nejznámějších blokových šifrovacích algoritmů, Data Encryption Standard (DES) a Advanced Encryption Standard (AES). Upravené implementace zmíněných šifer se nazývají S-DES a S-AES. Kromě toho je v aplikaci implementován algoritmus Triple DES, který je tvořen zanořením třech průběhů S-DES algoritmu a algoritmus Blowfish.

4.2.1 DES

Data Encryption Standard (DES) je považován za jeden z prvních symetrických kryptosystémů, kde se pro šifrování i dešifrování používá stejný klíč. Algoritmus je založen na principu Feistelovy sítě [35]. DES šifruje data v blocích po 64 bitech za pomoci klíče, který má délku 56 bitů a je reprezentován jako 64bitové

číslo. Poslední bit v každém bajtu slouží jako kontrola parity pro předchozích 7 bitů, což je využíváno k identifikaci chyb. Algoritmus využívá tří základních operací: XOR, permutace a substituce [36].

Expanze klíčů je prováděna pro správné inicializování šifrovacího algoritmu a spočívá ve vytvoření 16 podklíčů o délce 48 bitů. Vstupní klíč o délce 64 bitů je upraven permutací PC1 (permuted choice), která odebere paritní bity, čímž vznikne blok o délce 56 bitů [37]. Tento blok dat je následně rozdělen do dvou 28bitových bloků a nad každým se vykoná levý posun bitů. Nad takto upravenými bloky se následně provádí permutace PC2, která tvoří podklíč rundy. Proces levého posunu je opakován na původních blocích, čímž se postupně vygenerují podklíče pro všech 16 rund.

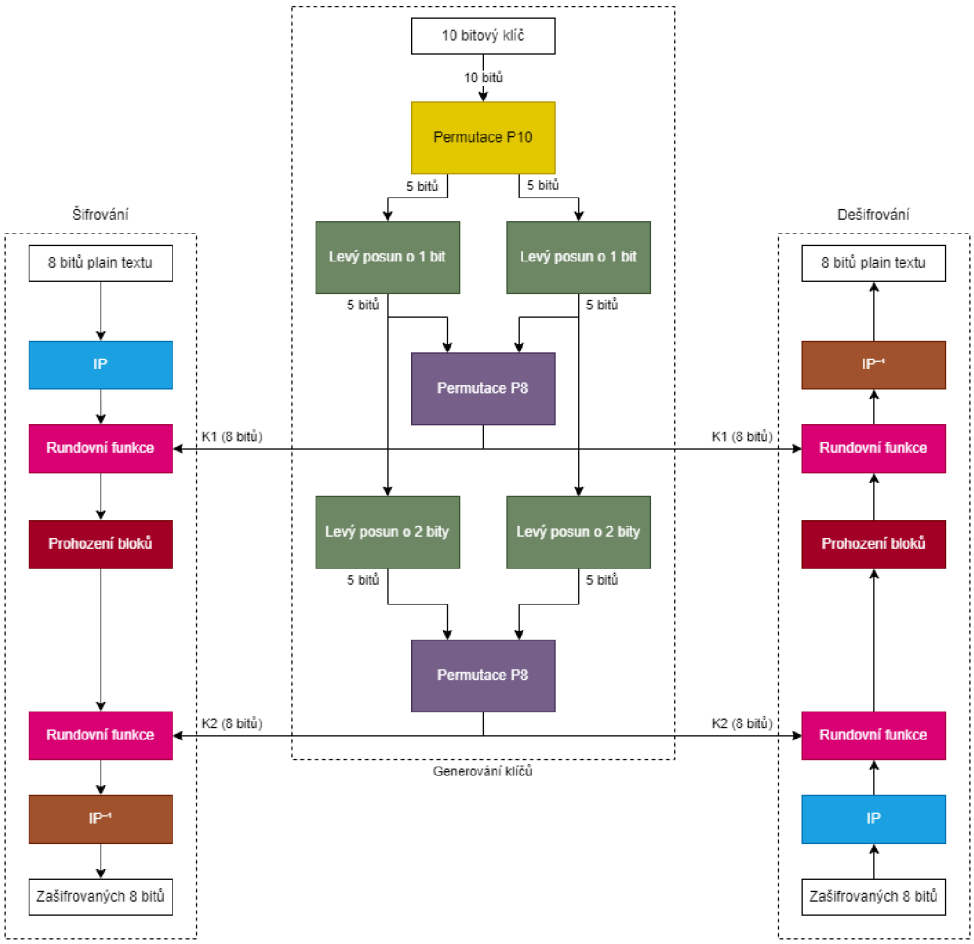
Šifrování se provádí transformací vstupního 64bitové bloku pomocí permutace IP (initial permutation). Po permutaci je vstupní blok rozdělen na dva stejně velké bloky, které procházejí paralelně 16 rundami. Rundy se skládají z permutací, XOR operace a substituce S-boxy. Po poslední rundě se provádí inverzní permutace (IP^{-1}), která vytvoří šifrovaná data.

Dešifrování je prováděno stejně jako šifrování s rozdílem, že se otočí pořadí rundovních klíčů. Toho lze jednoduše docílit inverzím postupem při generování klíče.

4.2.2 Simple DES

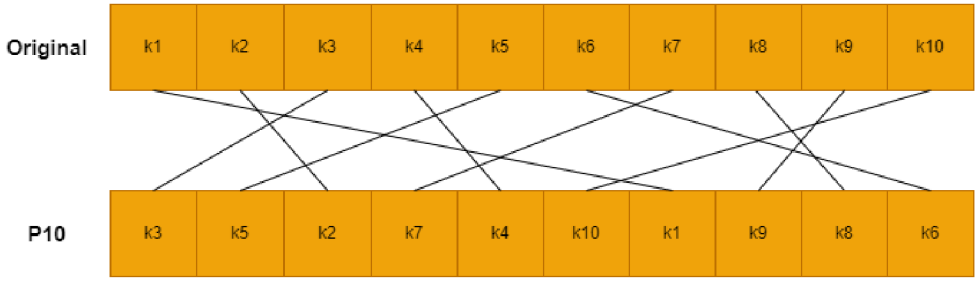
Simple DES (S-DES) [14] je zjednodušenou verzí původního Data Encryption Standard (DES), která využívá kratší klíč, menší počet rund a kratší vstupní blok. Při šifrování je postup algoritmu totožný jako u DES. Zásadním rozdílem mezi plnou verzí DES a S-DES je velikost vstupního bloku, kdy při plné verzi je využito 64 bitů, u zjednodušené verze 8 bitů. Rozdílná je i délka klíče, která je 64 bitů v plném algoritmu. Každý poslední bit v bajtu je paritní, přičemž zjednodušená verze využívá bitů jenom 10 a nezahazuje paritní bity při první permutaci klíče. Odlišná je i délka rundovních podklíčů (48 bitů v plné verzi, 8 bitů v zjednodušené). Změnou v šifrování a dešifrování pomocí S-DES je zjednodušení permutací a zmenšení S-

boxu z důvodu kratší délky zpracovávaných dat. Generování rundovních klíčů a proces šifrování/dešifrování popisuje obrázek uveden níže (viz. Obr. 4).



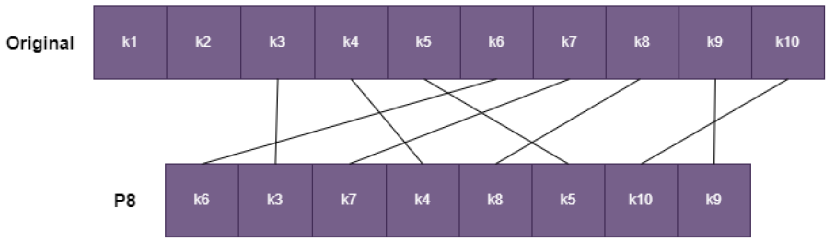
Obr. 4 – S-DES schéma. Zdroj: vlastní zpracování

Algoritmus na svém začátku provede validaci vstupních dat. Kontroluje se především délka vstupů. V případě, že je vstup kratší, je doplněn o nuly z levé strany. Pokud jsou vstupní data delší, pak jsou oříznuta z levé strany. Validovaný klíč je využit pro generování rundovních klíčů. Původní 10bitový klíč je transformován pomocí permutace P10 (viz. Obr. 5) a následně rozdělen na dva 5bitové bloky.



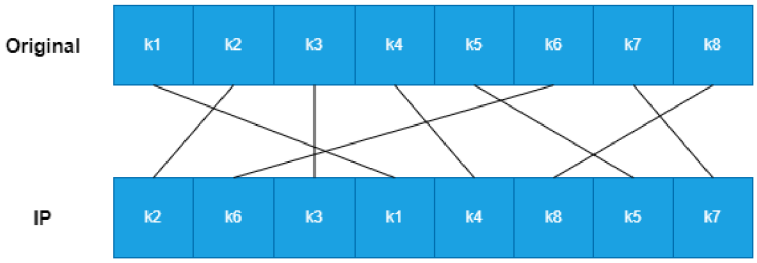
Obr. 5 – Permutace P10. Zdroj: vlastní zpracování

Na každém ze vzniklých bloků je proveden levý posun registru o 1 bit. Výsledné bloky jsou použity k tvorbě prvního rundovního klíče pomocí permutace P8. Nad bloky je také proveden další levý posun o 2 bity. Pomocí permutace P8 vzniká druhý rundovní klíč. Tuto permutaci vykresluje ilustrační obrázek níže (viz. Obr. 6).



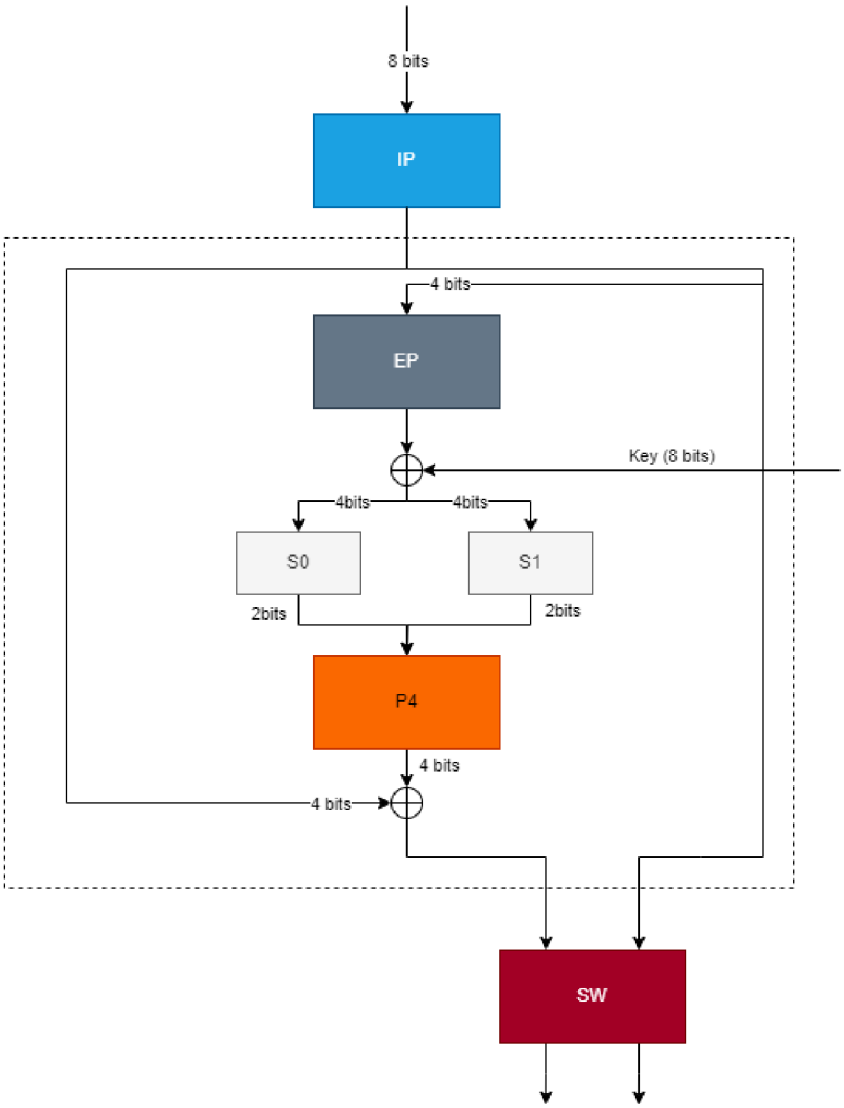
Obr. 6 – Permutace P8. Zdroj: vlastní zpracování

Prvním krokem šifrování v S-DES je transformace vstupního datového bloku pomocí úvodní permutace (IP). Tato permutace je popsána v diagramu na Obr. 7 níže.



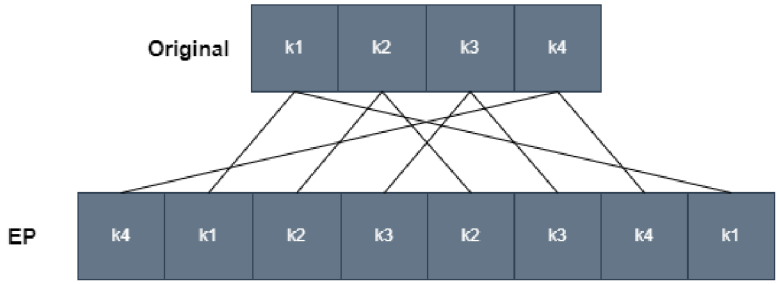
Obr. 7 – Úvodní permutace IP. Zdroj: vlastní zpracování

V další fázi vstupují data do rundovní funkce (Obr. 8), kde se data rozdělí na dva stejně veliké bloky. Pravý blok je zduplikován – jedna kopie je zachována beze změny a druhá prochází úpravou. Levý blok neprochází změnami, pouze se XORuje na konci rundovní funkce s pravým upravovaným blokem.



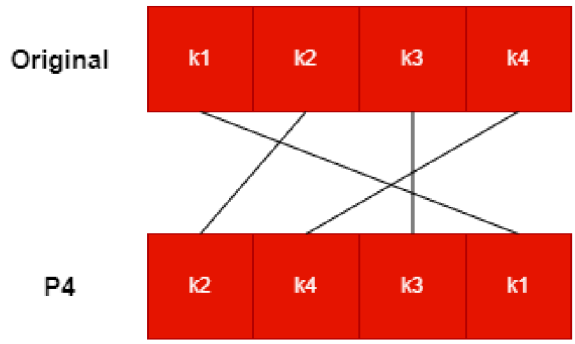
Obr. 8 – Rundovní funkce S-DES. Zdroj: vlastní zpracování

Nad pravým upravovaným blokem se provádí rozšířená permutace (EP), která je následně XORována s rundovním klíčem. Tuto permutaci lze vidět na Obr. 9.



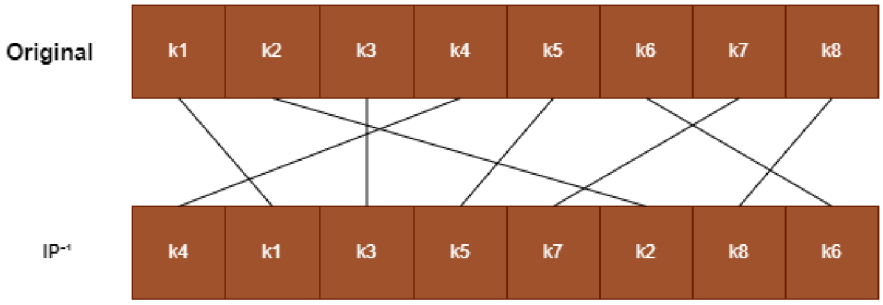
Obr. 9 – Rozšířená permutace (EP). Zdroj: vlastní zpracování

Výstupem jsou dva 4bitové bloky, na kterých se provede substituce pomocí předdefinovaných S-boxů a následná permutace P4 (Obr. 10).



Obr. 10 – Permutace P4. Zdroj: vlastní zpracování

Po těchto operacích je pravý upravený blok XORován s levým neupraveným blokem. Tento výstup je následně prohozen s pravým nemodifikovaným blokem a rundovní funkce je následně aplikována podruhé. Data z rundovní funkce jsou permutována pomocí inverzní permutace IP^{-1} (viz. Obr. 11).



Obr. 11 – Inverzní permutace IP^{-1}. Zdroj: vlastní zpracování

Proces dešifrování je u S-DES podobný jako proces šifrování. Rozdíl spočívá v otočení pořadí rundovních klíčů. V první rundě se tedy využije druhý rundovní klíč, v druhé rundě pak první rundovní klíč. Otočením pořadí rundovních klíčů se provedou všechny stejné operace, čímž se jednotlivé šifrovací operace vyloučí.

4.2.3 Triple DES

Triple Data Encryption Standard (Triple DES) byl vytvořen, aby sloužil jako bezpečnější varianta pro DES [38]. Triple DES je navržen jakožto trojnásobné zopakování algoritmu DES. V každém průchodu samostatným DES se využívá jiný klíč. Hlavní změnou je průběh šifrování, který spočívá v aplikování šifrování, dešifrování a v poslední řadě šifrování. Dešifrování je provedeno opačnou metodou

tzn. dešifrování, šifrování a dešifrování [39]. Další možností, jak složit Triple DES, je použití tří šifrovacích operací v řadě. Triple DES je rozšířen o další dva 64bitové klíče, které tak dohromady tvoří 192 bitů. Po odebrání paritních bitů se tedy jedná o 168 bitů [19].

Při implementaci tohoto algoritmu je možné využít několik klíčových strategií. První strategií je volba jednotného klíče, který se využije ve všech třech průchodech DES. Druhou variantou je využití stejného klíče pro první a poslední operaci. Pro třetí strategii, která je nejbezpečnější, je nutné využít třech unikátních klíčů.

Zašifrování datových bloků ve výukové aplikaci je dosaženo pomocí operací v tomto pořadí: šifrování, dešifrování a šifrování. Aplikace využívá principu volby tří unikátních šifrovacích klíčů. Pro dešifrování je zvolen postup inverzní k šifrování, tzn. DED.

4.2.4 Triple S-DES

Triple S-DES je zjednodušenou verzí Triple DES. Pro své fungování místo plného DES algoritmu využívá S-DES, což značně přispívá k možnosti rychle porozumět složenému kryptografickému systému. Tato verze algoritmu je stejně jako S-DES navržena pro výukové účely a není vhodná pro reálné zabezpečení dat, jelikož využívá zkrácené klíče, zmenšené bloky dat a výrazně méně komplexní substituce a permutace. Díky tomu je značně jednodušší celý systém pochopit.

4.2.5 Blowfish

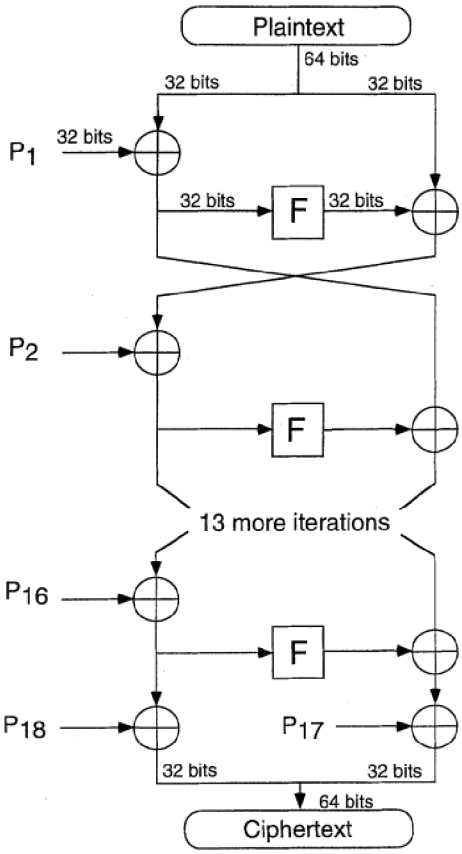
Blowfish je 64bitová bloková šifra s proměnnou délkou klíče (32 až 448 bitů). Algoritmus pro své fungování potřebuje před šifrováním/dešifrováním provést expanzi klíče. Expanze klíče převádí klíč o maximální délce 448 bitů na pole 18 dílčích klíčů o celkové délce 4168 bajtů. Šifrování dat probíhá prostřednictvím 16 rundové Feistelovy sítě. Každá runda se skládá z permutace závislé na klíči a substituce závislé na klíči a datech [18].

Proces expanze klíčů probíhá v několika krocích. Nejprve se inicializuje pole P a čtyři S-boxy pevným řetězcem, který je tvořen desetinnými čísly pocházejícími z čísla π převedených do hexadecimální soustavy (bez počáteční trojky). Volba čísla π jako základu pro pole je dána především jeho iracionální povahou a tím, že má neopakující se desetinný rozklad. Tato vlastnost pomáhá vnést do inicializačního procesu vysokou míru náhodnosti a složitosti, což zvyšuje bezpečnost algoritmu proti různým kryptografickým útokům. Po inicializaci se každý prvek pole P XORuje s odpovídajícími bity klíče (32 bitů). Tento proces se opakuje cyklicky, dokud nejsou všechny prvky pole P upraveny podle klíče.

Následně se zašifruje řetězec nul pomocí algoritmu Blowfish s využitím vytvořených podklíčů. Výsledný šifrovaný text nahradí první dva prvky pole P. Poté se tento šifrovaný text opět zašifruje algoritmem Blowfish a výsledek nahradí další dva prvky pole P. Tento proces se opakuje, přičemž se postupně nahrazují všechny prvky pole P a následně všechny čtyři S-boxy, vždy novými výstupy algoritmu.
Celkem je třeba provést 521 iterací k vytvoření všech potřebných podklíčů.

Při procesu šifrování je datový blok rozdělen na dvě části, přičemž každá má délku 32 bitů. Následně se provádí 16 rund, ve kterých se levá část XORuje s podklíčem a na konci rundy se prohodí s druhým blokem. Výstup XORu je vstupem do Feistelovy funkce (F), která vytváří výstup pro XOR s pravým blokem. Na konci rundy se bloky prohodí a pokračuje se další rundou. Po 16. rundě je levý blok XORován s 18. podklíčem a pravý blok se XORuje se 17. podklíčem. Výstup obou operací se spojí, čímž vzniká šifrovaný text. Toto schéma je vyobrazeno na Obr. 12.

Feistelova funkce vstupní data (32 bitů) rozdělí na 4 části po 8 bitech. Nad vzniklými bloky dat se provede substituce, přičemž první blok se kombinuje s druhým, výsledek je následně XORován s výstupem třetí substituce a vše je zkombinováno s výstupem čtvrté substituce do výsledného výstupu o 32 bitech.



Obr. 12 – Blowfish schéma. Zdroj: Bruce Schneier [18]

Dešifrování je v případě Blowfish podobné jako u algoritmu DES. Provádí se stejný postup jako při šifrování, přičemž jediným rozdílem je prohození pořadí dílčích (rundovních) klíčů.

4.2.6 AES

AES (Advanced Encryption Standard) je symetrický blokový šifrovací algoritmus, který podporuje libovolnou kombinaci délky dat a klíče o délkách 128, 192 a 256 bitů. AES umožňuje délku dat 128 bitů, které jsou rozděleny do čtyř datových bloků. Tyto bloky jsou považovány za pole bytů a jsou organizovány jako matice o rozměrech 4×4, která je také nazývána stavovou maticí. Nad těmito maticemi jsou prováděny různé transformace. Pro úplné šifrování je použito proměnné množství rund, N = 10, 12, 14 v závislosti na délce klíče 128, 192 a 256 bitů [40]. Každé runda je podrobena substituci S-boxem, posunu řádku matice a násobení matic v tzv. Galoisově tělese [41]. Výstup je pak kombinován s klíčem rundy. Galoisovo těleso a operace v něm jsou detailně popsány v [41].

Generování rundovních klíčů (expanze) probíhá v několika krocích. Prvním krokem je rozdělení původního klíče na části, tzv. slova (anglicky „words“) po určitém počtu bajtů v závislosti na délce klíče. Nad jednotlivými slovy je následně prováděno několik operací, jako je substituce bajtů pomocí vyhledávací tabulky, rotace bajtů a XORování s konstantami. Výsledkem expanze klíčů je sada rundovních klíčů, které jsou následně využity v procesu šifrování či dešifrování.

Proces šifrování 128bitového otevřeného textu probíhá v 10 rundách, přičemž na počátku je otevřený text rozdělen do čtyř bloků v podobě stavové matice. Stavová matice je následně XORována s odpovídajícími rundovními klíči. Takto upravená matice vstupuje do rundy, ve které se provede bytová substituce S-boxem (*SubBytes*). Výstupní data jsou transformována pomocí posunutí řádků (*ShiftRows*), kdy poslední tři řádky jsou posunuty o *i* pozic doleva, kde *i* určuje číslo řádku. Nad maticí stavů je proveden proces *MixColumns*, což je lineární transformace, která provádí násobení každého sloupce 4x1 pevně danou maticí. Každý sloupec je násoben polynomem v Galoisově tělese GF(2^8) modulo polynomu $m(x) = x^8 + x^4 + x^3 + x + 1$. Tato operace transformuje každý sloupec do nové podoby, která je lineárně nezávislá na původním sloupci. Na konci rundy je datový blok XORován s rundovním klíčem (*AddRoundKey*). Po průchodu 9 rund je provedena runda finální, která obsahuje stejné operace s absencí operace *MixColumns*, jejímž výsledkem je šifrovaný text.

Proces dešifrování probíhá za pomoci obráceného pořadí rundovních klíčů. Po XOR klíče s daty se provádí rundy. V každé rundě se využívají inverzní operace v odlišném pořadí. První operací je prohození řádků stavové matice, následuje substituce pomocí S-boxu a XOR dat s rundovním klíčem. Runda končí operací *MixColumns*, která je v poslední rundě vynechána, čímž následně vznikne otevřený text.

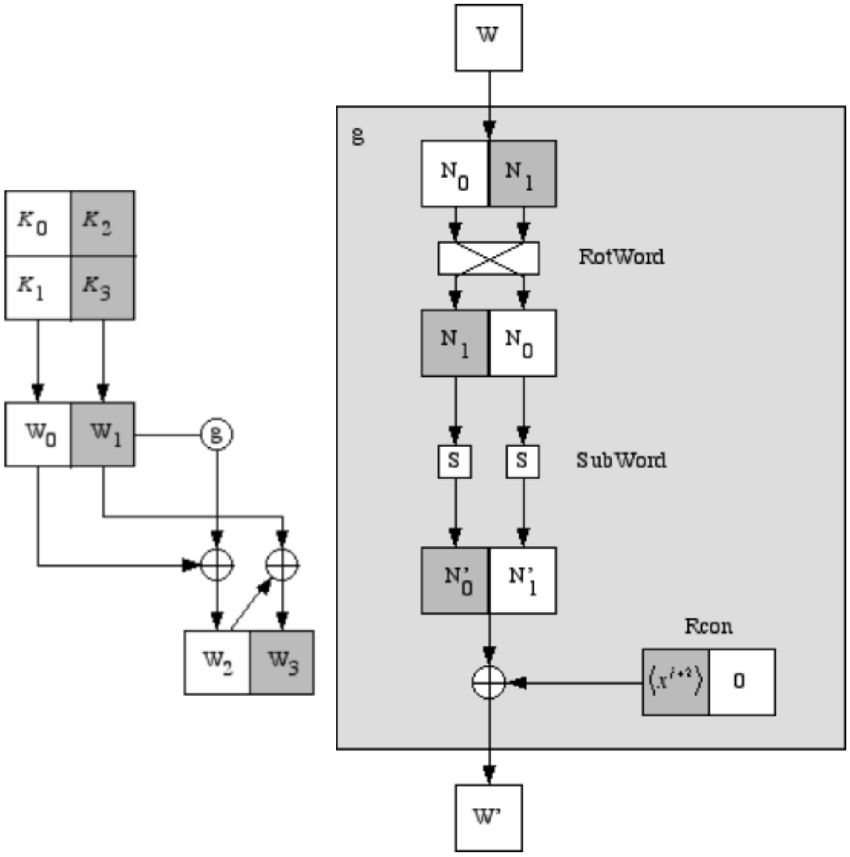
4.2.7 Simple AES

Simple AES (S-AES) [42] je zjednodušená verze plného algoritmu AES. Algoritmus se liší především v délce vstupních dat (16bitů), délce vstupního klíče (16 bitů) a počtu rund. V plném algoritmu se počet rund odvíjí od délky klíče, v případě S-AES se jedná o 2 rundy. Dalším rozdílem je nutnost počítání konstant při expanzi klíče, v případě S-AES lze využít již předem vypočítaných konstant *1000* v první rundě a *0011* v druhé rundě. Délka klíčů vyžaduje upravení transformace posunutí řádku, ve které se posouvá pouze poslední řádek stavové matice. Operace *MixColumns* taktéž vyžaduje změny, jelikož je nutné pracovat v $GF(2^4)$ s polynomem $m(x)=x^4+x+1$. Využití Galoisova násobení v tělese konečných prvků způsobí, že výstupní data budou 4bitová čísla. Maximální vypočtené číslo násobením je 15. Vyššího čísla nelze na 4 bitech dosáhnout, proto je potřeba ho redukovat pomocí modulo x^4+x+1. Galoisovo násobení je zde značně zjednodušeno, jelikož není nutno počítat výsledky násobení s maticí konstant, ale lze využít jednoduchou substituční tabulku s předem vypočítanými výsledky (jedná se o nízké hodnoty, tabulka je tedy poměrně malá).

Expanze klíčů v S-AES produkuje tři podklíče za pomoci dvou rund. Prvním klíčem je zvolen vstupní klíč bez úprav. Jedná se o podklíč, který je použit při první transformaci v šifrovací operaci. Tvorba podklíčů je započata rozdělením vstupního 16bitového klíče na dva 8bitové bloky (slova). Bloky jsou v rundách expandovány a vytváří 2 nová slova, která jsou následně zkombinována a vytváří rundovní klíč. Vytvořené bloky jsou následně předány jako vstup do nové rundy. Schéma níže (viz. Obr. 13) vyobrazuje postup tvorby prvního rundovního klíče.

Ve schématu K_{0-3} znázorňuje jednotlivé byty klíče, w_{0-3} ve schématu vyobrazují slova, přičemž w_0 a w_1 lze v druhé rundě nahradit výstupem w_2 a w_3, výstup w_2 a w_3 na konci rundy nahradí w_4 a w_5. Blok *g* ve schématu vyobrazuje operace prováděné nad vstupním slovem. V první části je vstup rozdělen na dvě poloviny a provede se jejich prohození pomocí operace *RotWord*, která nahrazuje operaci *ShiftRows* v plné verzi AES. Dalším krokem je substituce S-boxem označena

jako *SubWord*, která nahrazuje operaci *SubBytes* v plné verzi AES. V poslední části se obě poloviny slova XORují s konstantou *Rcon*. Výstupní slovo je následně XORováno s nemodifikovaným slovem, čímž vznikne rundovní klíč. Proces je následně opakován další rundou, do které vstupují upravená slova z předchozí rundy. Po dokončení generování dostatečného počtu klíčů je expanze klíčů u konce.



Obr. 13 – Schéma expanze klíčů S-AES. Zdroj: Lana Holden [42]

4.3 Proudové šifry

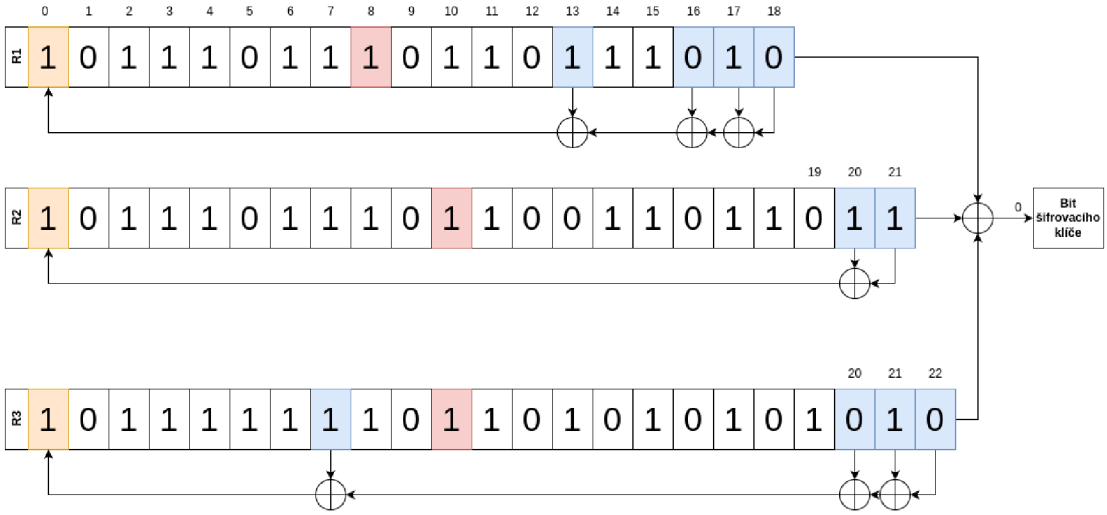
Proudové šifry (anglicky „stream ciphers“), jsou symetrické šifry, které při šifrování pracují s jednotlivými bity či s malými skupinami bitů. Často jsou konstruovány na principu lineárních posuvných registrů se zpětnou vazbou. Při šifrování otevřeného textu jsou jednotlivé bity XORovány s pseudonáhodně vygenerovaným proudem klíče (anglicky „keystream“) [43]. Pro generování tohoto klíče je často používán tzv. inicializační vektor (zkráceně IV), kterým se inicializují posuvné registry, ze kterých je pomocí vstupního klíče vytvářen proud klíče.

4.3.1 A5/1

Samotná komunikace pomocí A5/1 probíhá v tzv. rámcích (anglicky „frames“). Jako vstup využívá tajný klíč o 64 bitech a 22bitový inicializační vektor (číslo rámce). Výstupem šifry je proudový klíč o 228 bitech, který je XORován s aktuálně přenášenými daty. Pro své fungování využívá tří posuvných registrů o délce 19, 22 a 23 bitů, které krokují nepravidelně pomocí majoritního bitu. Majoritní bit se určuje z bitů v registrech, které se nachází na 9. (pro první registr) a 11. pozici (pro zbylé registry).

V prvním kroku se inicializují registry pomocí nul, které se následně transformují za použití šifrovacího klíče. Toto se provádí v *n* cyklech, kde *n* označuje délku klíče. Registry jsou v každé iteraci cyklu krokovány pravidelně a následně se odpovídající bity XORují s prvním bitem v každém registru. Následuje 22 iterací, kdy jsou registry pravidelně krokovány a odpovídající bit inicializačního vektoru je XORován s prvním bitem v každém registru. Poslední částí je sto iterací, ve kterých jsou registry nepravidelně krokovány v závislosti na majoritním bitu.

Tvorba proudového klíče probíhá pomocí pravidelného krokování registrů dle majoritního bitu. Tento princip určuje, který z registrů bude odkrokován a vhodně posunut se zpětnou vazbou. Nový bit, který je do registru vložen, se počítá za pomoci XOR operace bitů z předem určených pozic. První registr využívá bitů na pozici 14, 17, 18 a 19. U druhého registru se jedná o pozice 21 a 22. Poslední registr pak využívá 8., 21., 22. a 23. bitu. Výstupní bit proudového klíče je určen XOR operací posledních bitů v registru. Proces krokování registrů a určování bitu šifrovacího klíče je vyobrazen na schématu níže (viz. Obr. 14).



Obr. 14 – A5/1 posun registrů a určení šifrovacího bitu. Zdroj: vlastní zpracování

Šifrování a dešifrování probíhá bit po bitu XORováním vstupních dat a odpovídajícího bitu v proudovém klíči. Po iteraci přes celý vstupní text vzniká šifrovaný text.

4.4 Asymetrické šifry

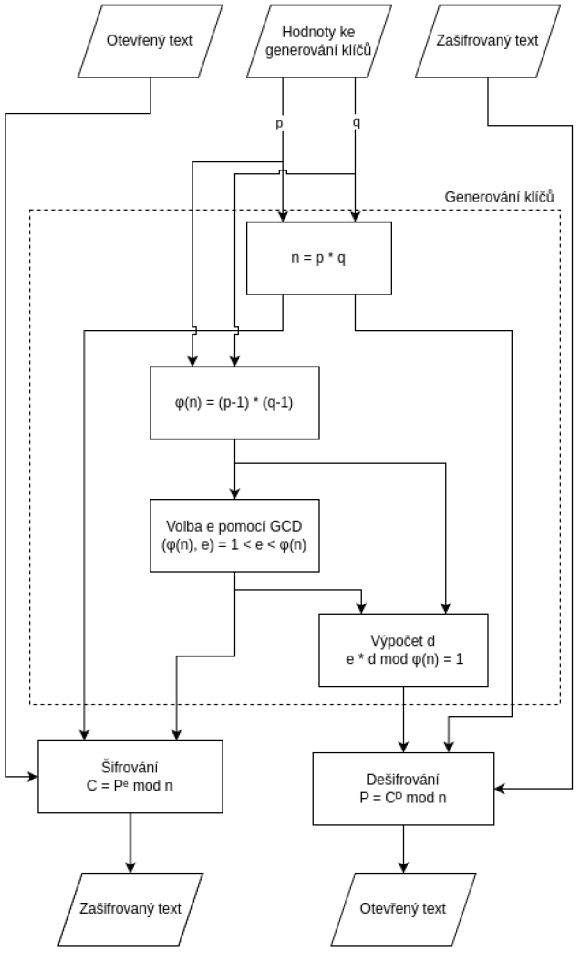
Asymetrické šifry v kryptografii představují komplexní přístup např. k výměnám klíčů, tvorbě digitálních podpisů a šifrování dat. Asymetrické šifry využívají dvou klíčů, přičemž jeden je veřejný a druhý soukromý. Veřejný klíč je využit k dešifrování dat a je znám oběma stranám komunikace, přičemž soukromý slouží k šifrování dat a je znám pouze jednomu účastníkovi komunikace. Veřejný klíč tak lze sdílet přes nezabezpečenou síť, jako je například internet.

4.4.1 RSA

RSA (Rivest-Shamir-Adleman) je jedním z nejznámějších a nejstarších asymetrických kryptografických systémů, který je možné použít k digitálním podpisům ale i k šifrování dat. Tento algoritmus využívá dvou velkých prvočísel, která po vynásobení tvoří velmi velké číslo. Bezpečnost algoritmu se zakládá na obtížnosti prvočíselného rozkladu velkých čísel. Pro klíč o velikosti pouze 768 bitů by prolomení šifry znamenalo výpočty po dobu 1500 let [28], jelikož zatím neexistuje dostatečně výkonný hardware, který by tyto výpočty urychlil.

Generování klíčů probíhá pomocí náhodného zvolení dvou různých prvočísel p a q. V další fázi se vypočítá modul N pomocí vzorce $N = p * q$, čímž vznikne velké číslo. Pro toto nově vzniklé číslo se zvolí $e \geq 3$ tak, aby e bylo nesoudělné s $\varphi(N)$, kde $\varphi(N) = (p-1) \cdot (q-1)$. Po volbě čísla e se spočítá d pomocí Euklidova algoritmu, detailněji popsaného v [44], tak aby $e \cdot d \equiv 1 \bmod \varphi(N)$. Veřejný klíč je tvořen dvojicí (n, e) a soukromý klíč je dvojicí (n, d).

Šifrování za využití veřejného klíče (n, e) probíhá vzorcem $C \equiv M^e \pmod n$, kde C je šifrový text a M je otevřený text (ASCII znak). Dešifrování se provádí pomocí soukromého klíče (n, d) podle vzorce $M \equiv C^d \pmod n$. Proces generování klíčů, dešifrování a šifrování je možné vidět na obrázku níže (viz. Obr. 15).



Obr. 15 – Schéma RSA. Zdroj: vlastní zpracování

V kontextu webové aplikace bylo nutné přidat ověřování, zda je zadané číslo k (p či q) prvočíslem. Této validace lze docílit pomocí jednoduchého cyklu, který pro všechna čísla i od 2 do $\sqrt{k}$ testuje, zda je k dělitelné číslem i bez zbytku. V případě, že toto tvrzení platí, pak číslo k není prvočíslem. Tento přístup je vhodný pouze pro malá čísla, jelikož je výpočetně velmi náročný. V aplikaci je vstup omezen na desetimístné prvočíslo, větší by v rámci výuky postrádalo smysl, čili lze použít tuto jednoduchou validaci. V reálném prostředí by však bylo nutné využít efektivnější metodu [45].

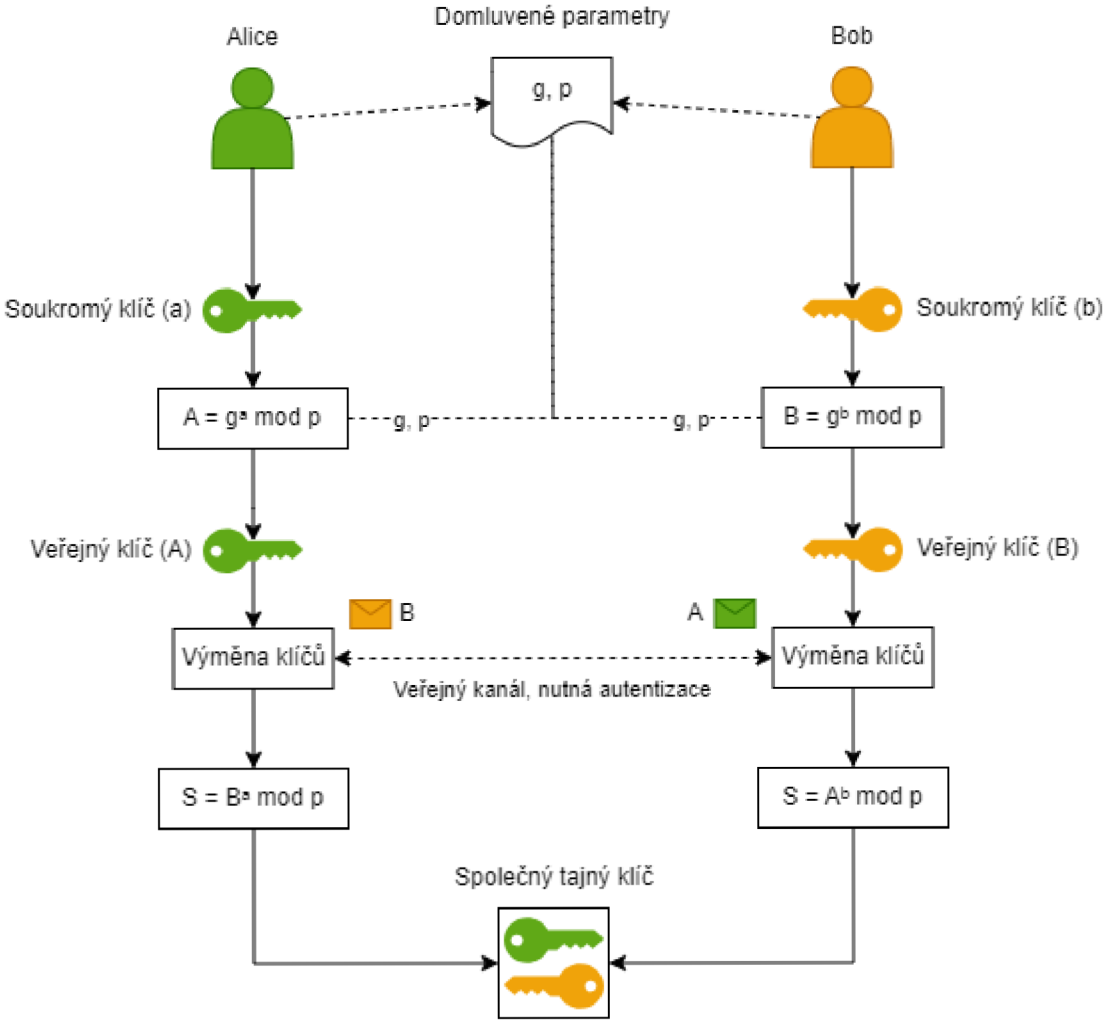
4.4.2 Diffie-Hellman

Diffie-Hellman je algoritmus sloužící pro výměnu klíčů mezi dvěma komunikujícími stranami. Koncept výměny klíčů spočívá v tvorbě dvou veřejných klíčů při předem domluvených parametrech, které jsou následně posílány přes veřejný kanál. Z veřejných klíčů si díky veřejným parametrům můžou obě strany vypočítat společný tajný klíč, který je následně využit k další komunikaci prostřednictvím symetrické šifry. Diffie-Hellman ve své implementaci využívá modulární aritmetiky a obtížnosti diskrétního logaritmu pro tvorbu společného tajného klíče.

Před zahájením protokolu výměny klíčů se obě komunikující strany domluví na společných veřejných parametrech p a g. Parametr p je náhodně zvolené prvočíslo, přičemž g je primitivním kořenem modulo p. Primitivní kořen modulo p je takové číslo g, pro které platí, že pro každé celé číslo a existuje celé číslo k, takové, že $g^k \equiv a \ (mod\ p)$ [46].

Aby byl následující popis algoritmu jednodušeji uchopitelný, jsou komunikující strany pojmenovány Alice a Bob. Po domluvení společných veřejných parametrů Alice a Bob vygenerují své soukromé klíče. Alice vygeneruje klíč a pomocí kterého následně vypočítá svůj veřejný klíč dle vzorce $A = g^a \ mod$ p. V případě Boba je veřejný klíč vypočítán ze soukromého klíče b pomocí vzorce $B = g^b \ mod\ p$. Vypočítané klíče A,B si uživatelé vymění skrze nezabezpečený veřejný kanál, jako je například internet. Po obdržení Bobova veřejného klíče Alice vypočítá společný

tajný klíč vzorcem $S = B^a \ mod \ p$. Bob tento klíč získá obdobně vzorcem $S = A^b \ mod \ p$. Tento společný tajný klíč S je následně využit Alicí a Bobem pro komunikaci symetrickou šifrou. Proces výměny klíčů je zjednodušeně zobrazen na Obr. 16.



Obr. 16 – Diffie-Hellmanova výměna klíčů. Zdroj: vlastní zpracování

5 Praktická část – implementace

Následující kapitola se zabývá důkladným rozborem původního řešení, které bylo vyvinuto v rámci zmíněné bakalářské práce [7], následovným navržením nového designu a architektury systému. Prvním krokem je identifikace silných a slabých stránek současného systému, což poskytne podklady pro vylepšení a optimalizaci. Návrh nového designu a architektury je zaměřen na zvýšení modularizace, škálovatelnosti a udržitelnosti kódu. To může zahrnovat rozdělení funkcionality do samostatných komponent, definování jasných rozhraní a použití vhodných návrhových vzorů.

Dalším krokem je analýza nástrojů pro zlepšení kvality kódu. Jsou zvoleny a integrovány vhodné nástroje pro statickou analýzu kódu a aktivní hlášení kritických chyb tak, aby bylo dosaženo vyšší úrovně kvality a stability kódu. Poté jsou popsány implementace jednotlivých kryptografických primitiv, přičemž je kladen důraz na správnou funkčnost a názornost výstupů.

5.1 Motivace pro vylepšení původního řešení

Při praktické výuce kryptografie je velice důležité, aby student pochopil fundamentální principy daných algoritmů. V tomto aspektu může být nápomocné si celý postup vyzkoušet na příkladových datech. Takto rozepsaný postup je optimální následně porovnat s řešením vytvořeným pomocí softwaru.

Původní verze webové aplikace využívala pouze komplexních knihoven určených k tvorbě výstupních dat a kód byl psán v základním PHP. Tato skutečnost se ukázala jako neoptimální v rámci rozšiřování aplikace o další funkce, jelikož neumožňuje komplexnější tvorbu modelů a jednoduchou práci s nimi. Kvůli těmto implementačním obtížím bylo nutné přepsat aplikaci s využitím frameworku, což dovolilo vytvořit vlastní implementaci jednotlivých kryptografických algoritmů. K takto vytvořeným třídám lze následně jednodušeji přidat mechanismy pro krokování, ale zároveň i efektivně přidávat nové algoritmy. Využití frameworku také umožnilo použití nástrojů, které slouží ke kontrole kvality kódu a umožňují tak psát

čistý a udržitelný kód. Dalšími výhodami je i využití vzoru MVC, optimalizovaná tvorba překladů a jednoduché přidávání dalších částí systému.

5.2 Volba frameworku, metodika vývoje a použité nástroje

Zásadní částí vylepšení aplikace se stala volba použitého frameworku. V této oblasti je poměrně velký výběr a je potřeba vybírat dle rozsahu a využití. Kandidáty se tak staly populární frameworky, jako je například Yii, Symfony či Nette. V závislosti na rozsahu a využití byl vybrán Laravel [1] s integrovaným šablonovacím nástrojem Blade, který předpřipravuje vykreslené šablony na serverové úrovni. K rozhodnutí, který framework použít, zásadně přispěla praxe s použitím dané nadstavby a možnost vypnout nevyužité komponenty, jako je například správa uživatelských účtů.

Pro vypracování projektu byla použita upravená metodika agile, která spočívá v rozplánování tvorby jednotlivých částí systému na tzv. tasky. Při této metodě je práce rozdělená do sprintů s jasně danou časovou délkou. Tato praxe se často využívá na rozsáhlých projektech, na kterých se podílí více vývojářů za účelem usnadnění a zrychlení vývoje. Navzdory tomu, že charakter tohoto projektu omezuje využití aspektu omezených sprintů, je tato metodika efektivní pro jednoduchý management práce.

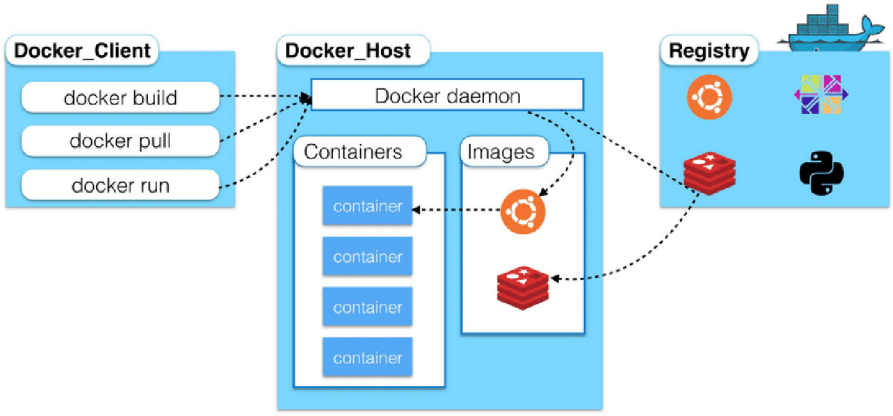
K dosažení bezpečnosti a stability aplikace bylo využito statického analyzátoru s názvem Psalm [4]. Tento nástroj umožňuje detekovat problematické struktury v kódu a odhalit běžné chyby před samotným spuštěním aplikace. Díky analyzátoru je možné zvýšit kvalitu a efektivitu vývoje, jelikož jsou identifikovány i nekonzistence, nevyužité proměnné či nespecifikované datové typy. Oprava těchto nalezených problémů tak přispívá k lepší čitelnosti kódu, udržitelnosti a rozšiřitelnosti aplikace. V kontextu vývoje bylo využito i nástrojů pro škálování, spolehlivost běhu či detekci chyb v produkčním prostředí. Tyto nástroje jsou popsány níže.

5.2.1 Kontejnerizace

V kontextu rostoucí složitosti moderních softwarových aplikací a jejich prostředí je nezbytné hledat efektivní způsoby, jak zajistit konzistenci a spolehlivost běhu. Zde vstupuje do hry kontejnerizace aplikací. Odpovídá totiž na výzvy spojené s diverzitou knihoven, operačních systémů a prostředí, které mohou vést k nekompatibilitě a následným fatálním výpadkům při nasazení aplikace.

Tyto potíže lze vyřešit jedním z nejrozšířenějších nástrojů s názvem Docker. Jeho základní princip spočívá v izolaci aplikací a všech potřebných závislostí do tzv. kontejnerů, což umožňuje vytváření konzistentního prostředí bez ohledu na použitou platformu hosta. Díky tomu, že veškeré součásti aplikace a k ní potřebné služby jsou zahrnuty do jednoho kontejneru, je možné snadno replikovat prostředí, ve kterém aplikace operuje. Tento přístup eliminuje potenciální problémy spojené s rozdílnými verzemi knihoven či operačními systémy. Mezi další výhody patří také modularita a škálovatelnost, což umožňuje v případě potřeby rozdělit aplikaci na tzv. mikroslužby a jednotlivě nasazovat jejich instance.

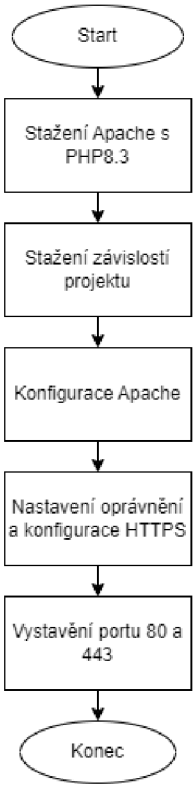
Na Obr. 17 lze vidět, jak funguje kontejnerizace projektu. Docker client vyobrazuje uživatelské vstupy do docker engine. „*Docker host*“ znázorňuje vnitřní fungování enginu, kde „*daemon*“ vytváří ze staženého image kontejner s projektem a danou službu spouští již ve vytvořeném kontejneru. Položka „Registry“ vyobrazuje předpřipravené image služeb, které může „*docker engine*“ stahovat.



Obr. 17 – Schéma kontejnerizace. Zdroj: Fawaz, Paraiso & Challita, Stephanie & al-dhuraibi, Yahya & Merle, Philippe [47]

5.2.2 Implementace kontejnerizace

Prvním krokem při implementaci kontejnerizace se stalo vytvoření vhodného obrazu se službami. Postup při tvorbě tohoto obrazu je popsán v souboru „*Dockerfile*“, jenž se nachází ve složce docker/php. V této složce je taktéž předpřipravený soubor *php.ini* s nastavením PHP. *Dockerfile* obsahuje příkazy, které se musí nad staženým obrazem se službou provést. Tento soubor obsahuje samotné stažení webového serveru Apache [6] s předinstalovaným PHP verze 8.3. Následuje stažení závislostí projektu (např. javascripty pro Laravel framework) s instalací užitečných nástrojů pro správu verzování GIT. Dále se nastavují potřebné doplňky jako GMP [2] pro práci s velkými čísly. V poslední řadě soubor obsahuje nastavení samotného Apache, přiřazení oprávnění k projektu a nastavení https za pomoci certifikátu a soukromého klíče. Po provedení všech potřebných operací se otevře port 80 a 443. Tento proces je stručně vyobrazen v diagramu (viz. Obr. 18).

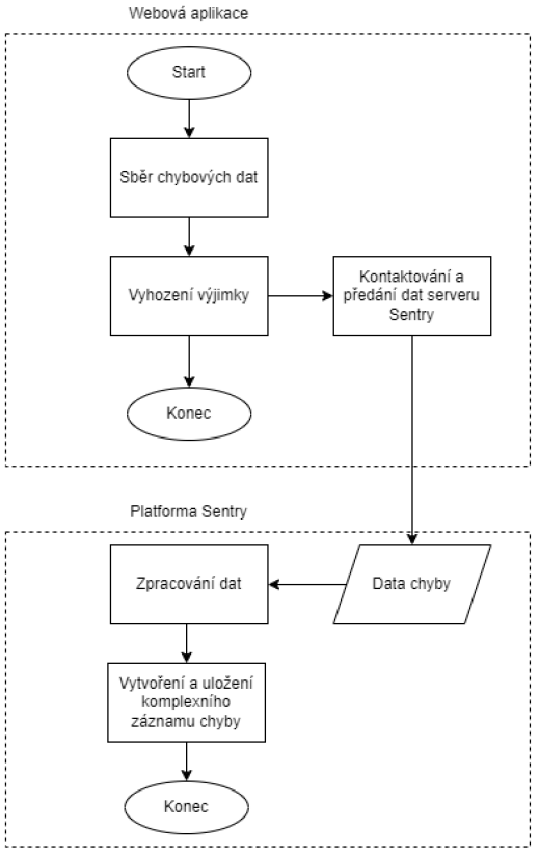


Obr. 18 – Vývojový diagram vytvoření obrazu. Zdroj: vlastní zpracování

Samotný popis kontejneru a definice spouštění se nachází v souboru „*docker-compose.yml*“. Zde jsou nadefinovány datové zdroje kontejneru, způsob restartování při selhání a název obrazu, který má kontejner použít. Tento soubor se využívá pro zapnutí kontejneru s webovou aplikací pomocí příkazu „*docker-compose up -d*“.

5.2.3 Ladění aplikace

V oboru softwarového vývoje je nemožné vytvořit kód, který by byl naprosto bezchybný. Proto se velice často využívají nástroje, které dokáží chyby odchytit a důkladně zadokumentovat. Jedním z takových nástrojů je produkt s názvem „Sentry“ [3]. Tato platforma je klíčovým nástrojem pro efektivní ladění aplikací, který vývojářům poskytuje komplexní informace o chybách a výjimkách v kódu v reálném čase. Tento nástroj umožňuje sledovat chyby v aplikaci okamžitě po jejich vzniku a poskytuje detailní informace o prostředí, ve kterém chyba nastala, včetně zásadních detailů potřebných k jejímu replikování a opravě. Postup vytvoření chybového hlášení je popsán ve vývojovém diagramu (viz. Obr. 19).



Obr. 19 – Vývojový diagram vytvoření hlášení do Sentry. Zdroj: vlastní zpracování

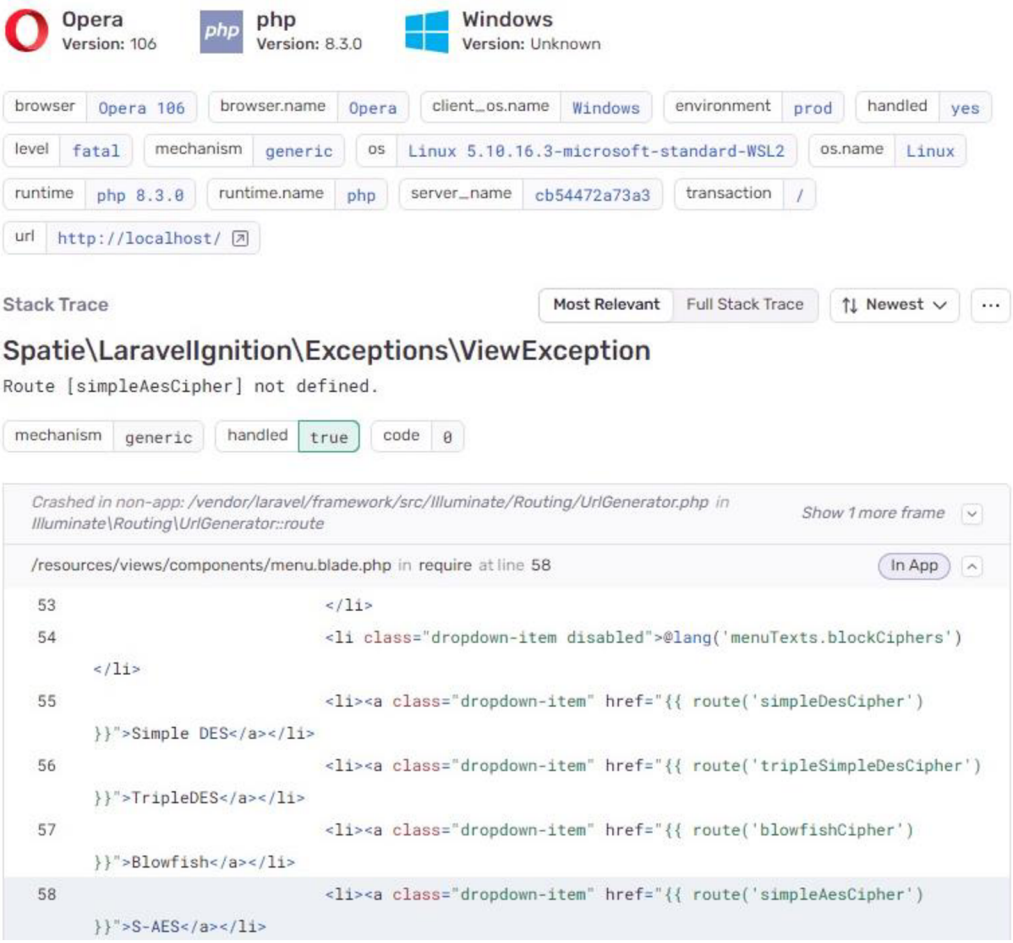
Díky funkcionalitám, jako je sledování chyb v reálném čase, nastavení upozornění na kritické události a možnost sledování vývoje chyb v čase, umožňuje Sentry vývojářům okamžitě reagovat na vzniklé problémy a minimalizovat jejich dopad na uživatele. Navíc nabízí možnost analýzy vývoje chyb v různých verzích aplikace, což umožňuje identifikovat opakující se problémy a trendy, které mohou vést k jejich řešení a zlepšení celkové kvality aplikace.

5.2.4 Implementace nástroje k ladění aplikace

Integrace dokumentování chyb pomocí Sentry spočívá v několika jednoduchých krocích. Prvním důležitým krokem při implementaci je instalace Sentry SDK. Tuto část lze zajistit pomocí Composer příkazu, který si přidá závislost na službě do svého seznamu a stáhne vše potřebné. V rámci tohoto kroku probíhá i implementace Sentry JS SDK pro hlášení chyb z Javascriptu. Instalaci je důležité zakončit konfigurací přístupového klíče, který lze získat po založení projektu v rozhraní platformy. Po instalaci SDK je nutné inicializovat Sentry v kódu aplikace. To zahrnuje volání inicializačních funkcí SDK a nastavení konfiguračních parametrů.

Pro otestování, zda je služba nainstalována správně, lze využít příkazu pro odeslání manuálně vytvořené testovací chyby. Pokud se tato chyba zobrazí v rozhraní platformy, pak je vše nastavené správně. Dále je potřeba vyzkoušet funkčnost implementace v projektu vyvoláním serverové chyby. V případě, že se i takto vytvořená chyba zobrazí, je možné integraci začít využívat bez dalších problémů.

Zachycené chyby je následně možné zobrazit na webové stránce Sentry pomocí seznamu. Rozkliknutím jednotlivých položek se lze dostat k detailním informacím (viz. Obr. 20). Mezi získaná data patří například popis prostředí, ve kterém došlo k chybě. Velkou výhodou je i označená část kódu, která způsobila vyhození výjimky. Stránka také obsahuje detailní výpis hlavičky ze zaslaného požadavku od klienta či četnost výskytu dané chyby za posledních 30 dní.



Obr. 20 – Ukázka odchycené chyby. Zdroj: vlastní zpracování

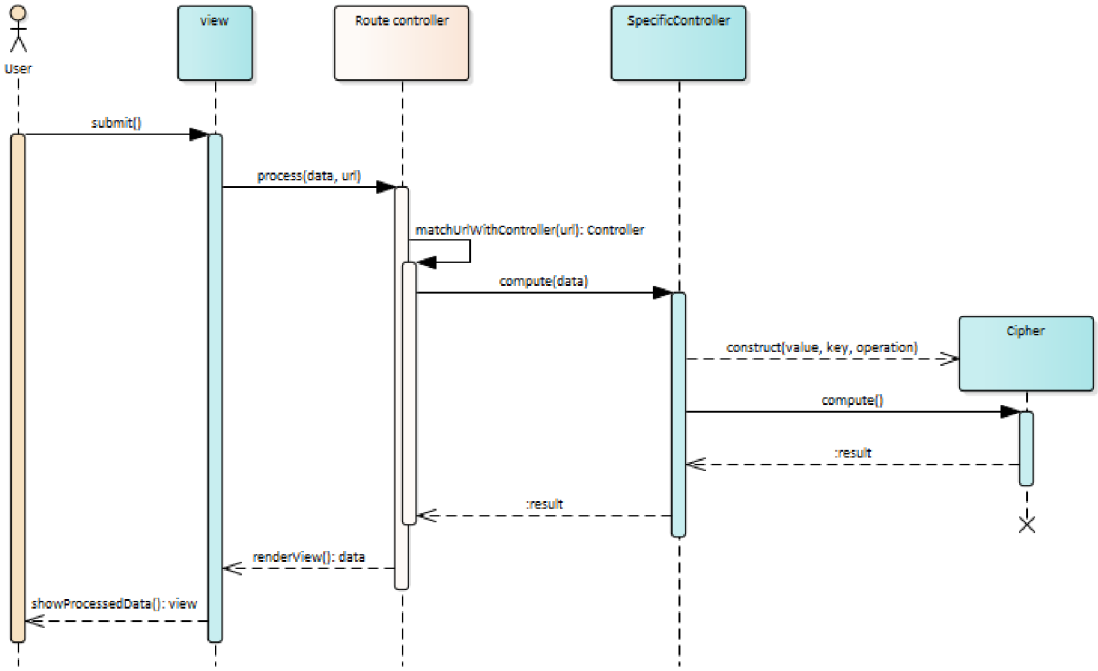
5.3 Návrh systému a implementace

Implementace webové aplikace se nachází na adrese https://educrypter.uhk.cz. Případně je možné aplikaci samostatně stáhnout z veřejného Github repozitáře a vyzkoušet na vlastním systému pomocí instalačních pokynů uvedených v souboru *readme.md* https://github.com/hejzlzd1/EducrypterUHK.

Celý systém je postaven na architektonickém vzoru Model-View-Controller (MVC), který je používaný při vývoji softwarových aplikací. Rozděluje aplikaci na tři hlavní části: model, pohled a řadič. Model reprezentuje datovou stránku aplikace a zajišťuje logiku pro manipulaci s daty. Pohled je vizuální reprezentací dat z modelu a zajišťuje uživatelské rozhraní aplikace. Řadič je prostředníkem mezi modelem a

pohledem a zpracovává uživatelské vstupy, upravuje stav modelu a aktualizuje zobrazení v pohledu.

Použití MVC výrazně zlepšuje modularitu, rozšiřitelnost a údržbu aplikace, neboť umožňuje oddělení datové logiky, uživatelského rozhraní a řízení toku aplikace. Tento koncept se často využívá při vývoji webových a desktopových aplikací, kde je důležité oddělit logiku aplikace od prezentace dat a uživatelské interakce. Obecný princip fungování MVC lze vidět na Obr. 21.



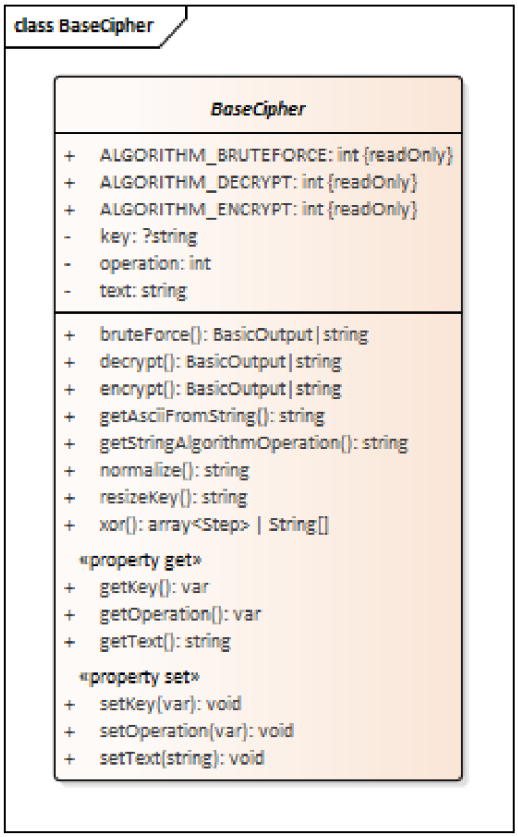
Obr. 21 – MVC interakce. Zdroj: vlastní zpracování

Před samotným vývojem aplikace je důležité připravit návrh systému a jeho částí. Navrhnout systém lze pomocí ER diagramu, jenž slouží k formalizaci struktury datového modelu aplikace a definuje entity (objekty) a vztahy mezi nimi. Cílem takového návrhu je zachytit hlavní entity, jejich atributy a utvořit strukturovaný pohled na části aplikace. Díky správně vytvořenému návrhu je možné systém jednodušeji rozšiřovat.

V rámci tvorby návrhu datových entit vzniklo několik tříd, které tvoří základní kámen webové aplikace. Tyto modely jsou následně dle potřeby rozšiřovány,

modifikovány či nahrazeny komplexnější variantou. V následujícím textu jsou stručně popsány.

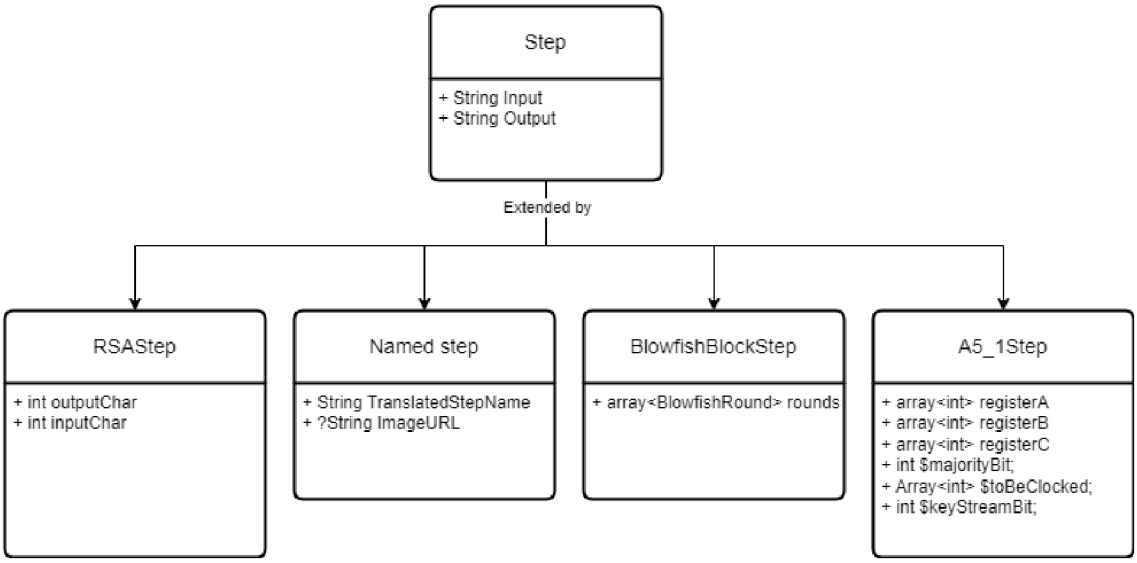
BaseCipher je abstraktní třídou kryptografické entity, ze které se odvíjí další rozdělení. Využití spočívá v dočasném uchování informací, jako jsou vstupní data, klíče k šifrám či operace prováděné nad vstupem. Taktéž drží informace v podobě potřebných konstant důležitých pro správný chod aplikace. Definuje i názvy funkcí pro šifrování, dešifrování a implementuje některé obecné metody. Návrh třídy je popsán v diagramu na Obr. 22.



Obr. 22 - BaseCipher model. Zdroj: vlastní zpracování

Pro vykreslování výstupu byl vytvořen generalizovaný model výstupních dat *BasicOutput*, který je rozšiřován v závislosti na komplexnosti vybraného algoritmu. Tento základní model obsahuje vstupní hodnoty, seznam kroků a výstupní hodnotu algoritmu. Krokování je realizováno pomocí obecné třídy „*Step*“, která obsahuje data před provedením operace a data po provedení operace. Tato třída má potomka „*NamedStep*“, který je více specifický, a obsahuje navíc název prováděné akce a

případný odkaz na obrázek se schématem. Pro komplexnější šifry s více kroky je z klasické třídy *Step* vytvořen derivát (viz. Obr. 23) obsahující specifické atributy v závislosti na potřebách algoritmu.

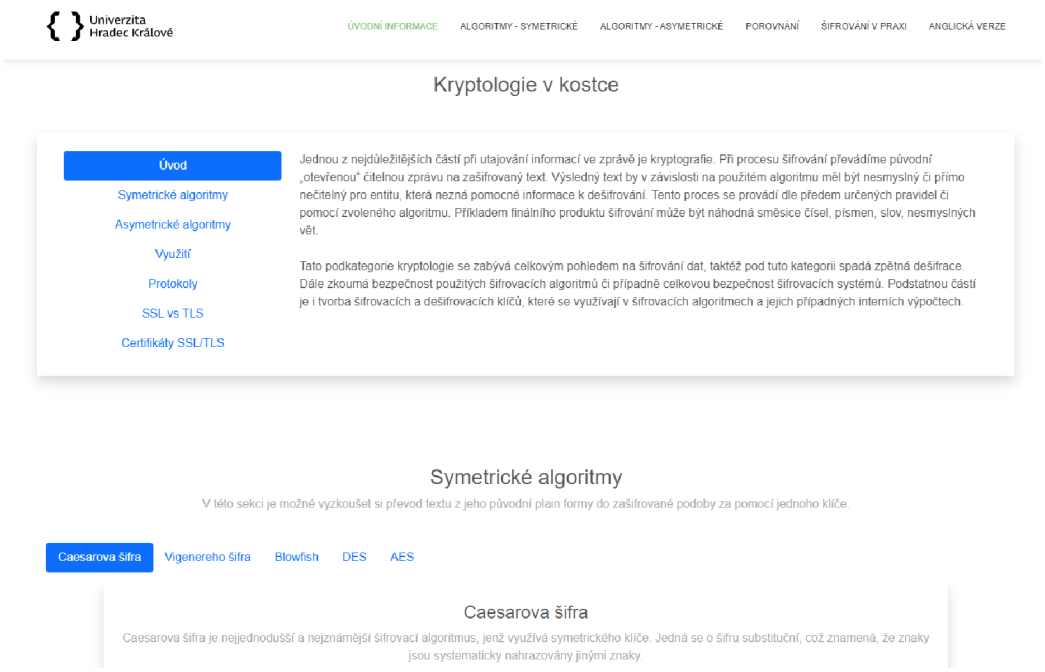


Obr. 23 – Dědičnost třídy Step. Zdroj: vlastní zpracování

5.3.1 User interface

Důležitou součástí webových aplikací je přehledné, intuitivní a užitečné uživatelské prostředí neboli user interface (UI). Mezi řešené problematiky spadá i poměr informací obsažených na stránce. Pokud je zobrazeno příliš málo informací, může se stát, že se uživatel při hledání příslušných informací unaví. Pokud je zobrazeno příliš mnoho informací, může být uživatel zmaten při přístupu k příslušné části. Výsledkem je nakonec špatný dojem ze systému, pro který bylo uživatelské rozhraní navrženo [48].

Retrospektivní analýza staré verze aplikace poukázala na nepřehlednost a zahlcení uživatele informacemi. Tento problém s použitelností vznikl především kvůli zaměření návrhu na „single page application“ (SPA) design. Původní design lze vidět na Obr. 24.



Obr. 24 - Původní design aplikace. Zdroj: vlastní zpracování

Uživatelský požitek bylo možné vylepšit rozdělením jednotlivých kryptografických algoritmů do několika stránek a využít návrhu „multi page application" (MPA). Celkový vzhled staré aplikace vyžadoval přepracování do modernějšího a více přehledného vizuálu. Úpravy, které lze vidět na Obr. 25, taktéž nastaly z hlediska responzivity na menších zařízeních. Úvodní stránka se nyní skládá z informací systematicky rozdělených do několika „informačních bloků", které obsahují teoretické podklady a ilustrativní obrázky.

Obr. 25 - Nový design. Zdroj: vlastní zpracování

5.3.2 Menu

Úpravou prošlo i menu, které je nyní rozděleno na tři sekce. Úvodní stránka obsahuje odkazy na všechny bloky, které se na ní vyskytují. Po stisknutí položky v menu se pohled přesune na danou část obsahující vyžadovanou informaci.

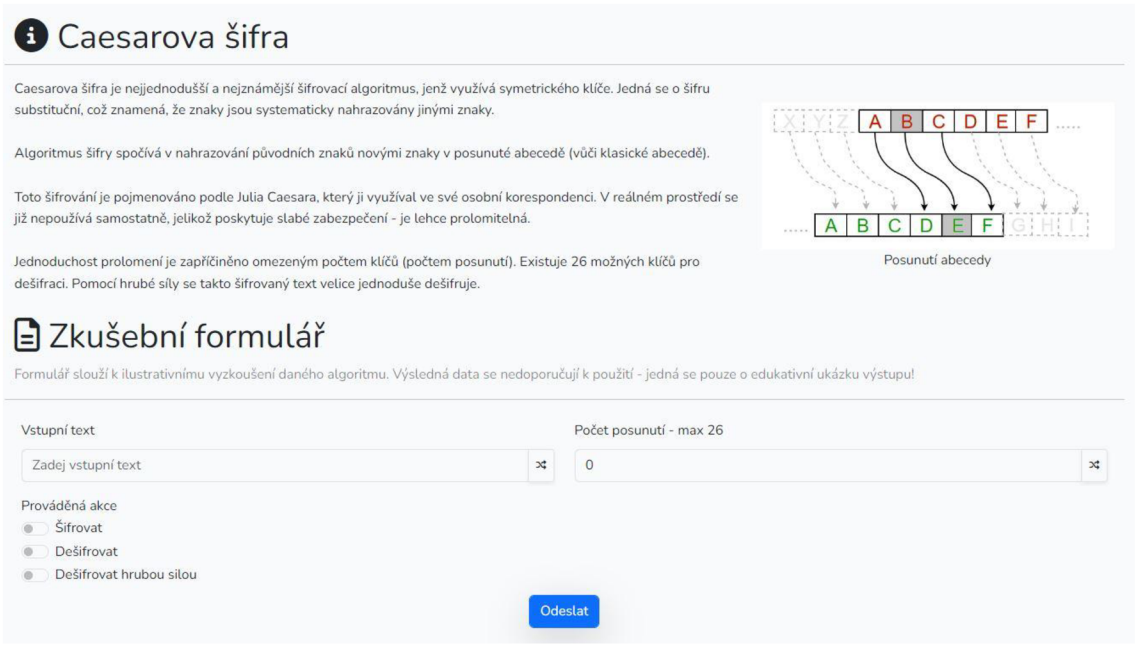
Rozbalovací nabídka „Symetrické algoritmy“ obsahuje odkazy na všechny dostupné šifry. Menu obsahuje tři kategorie v podobě klasických, blokových a proudových šifer, a odkazy na implementované verze algoritmů. Položka „Asymetrické algoritmy“ obsahuje odkaz na RSA a algoritmus pro výměnu klíčů Diffie-Hellman.

Změna nastala i v systému překladů, nyní lze jednoduše přepínat mezi anglickou a českou variantou kdekoliv na webu. Ke změně jazyka slouží tlačítko v pravé spodní části aplikace (viz. Obr. 26). Po stisknutí se provede změna jazyka, o které je uživatel informován prostřednictvím vyskakovacích zpráv na pravé straně uživatelského rozhraní. Tato změna se projevuje i ve schématech.

Obr. 26 – Změna jazyka a nové menu. Zdroj: vlastní zpracování

5.3.3 Design jednotlivých šifer

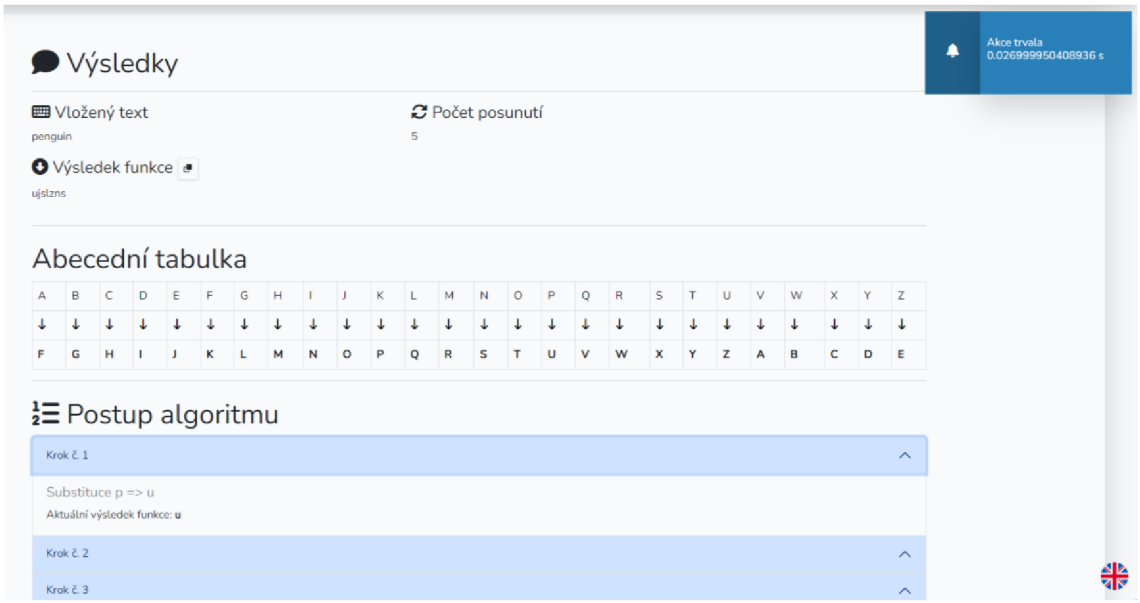
Šifrovací algoritmy se nyní zobrazují na samostatné stránce a obsahují teoretické podklady a formulář k vyzkoušení dané šifry. Informační část stránky obsahuje textový popis algoritmu, který je doplněn o relevantní schémata. Druhou komponentu stránky tvoří formulář sloužící k vyzkoušení zobrazené šifry na vlastních testovacích datech. Pro rozkrokování je nutné vyplnit vyžadované vstupy a zvolit prováděnou operaci. Vstupní pole obsahují na pravé straně tlačítka k vytvoření náhodných dat. U algoritmů, které vyžadují specifický formát dat, jsou přidány otazníky obsahující informace o validačních podmínkách. Toto uživatelské rozhraní je možné vidět na Obr. 27.



Obr. 27 – Ukázka stránky s šifrou. Zdroj: vlastní zpracování

Po odeslání formuláře dojde ke zpracování všech vstupních dat. Výsledkem je pak obnovení stránky a přidání nového bloku s komplexním rozpisem jednotlivých kroků provedené operace. Dokončení šifrování či dešifrování je oznámeno pomocí modré vyskakovací zprávy, ve které se nachází délka provedení v sekundách. Červená oznamovací okna mohou obsahovat informace o špatném formátu vstupních dat – v takovém případě se provádění operace zruší.

Nově vytvořený blok s výsledkem obsahuje záhlaví se vstupními informacemi, které mohou být v případě požadavků algoritmu modifikovány. V případě úpravy vstupních dat je tato skutečnost oznámena pomocí dodatečného pole s novými daty. Dále záhlaví obsahuje výstup algoritmu, který lze jednoduše zkopírovat pomocí tlačítka vedle nadpisu. Tato funkce slouží ke zlepšení uživatelského požitku, jelikož lze následně tento text vložit zpět do formuláře a provést nad ním novou operaci (např. dešifrování). Ve výsledném bloku se dále nachází případné doplňující informace a postup algoritmu. Jednotlivé kroky lze rozkliknout a zobrazit tak další podstatné informace (viz. Obr. 28).



Obr. 28 – Ukázka výpisu testovacího formuláře. Zdroj: vlastní zpracování

5.4 Implementace algoritmů

5.4.1 Klasické šifry

Díky implementaci klasických šifer vznikla první obecná třída z návrhu *CipherBase*. Tuto třídu následně rozšiřují jednotlivé klasické šifry a další obecné kategorické třídy. Výstupním modelem se stala třída „*BasicOutput*“, která obsahuje vstupní data, výstupní data, seznam kroků algoritmu a případné pomocné informace, jako je např. posunutá abeceda.

V algoritmech Caesarovy a Vigenèrovy šifry jsou vstupní texty před samotným procesem šifrování či dešifrování upraveny pomocí funkce „*normalize()*“. Tato funkce nahrazuje nežádoucí znaky v podobě diakritiky a symbolů, které se nenachází v běžné abecedě. Normalizací se zajistí, že operace probíhá pouze na platných znacích z ASCII, což je jedním z předpokladů pro funkčnost obou algoritmů.

Caesarova šifra

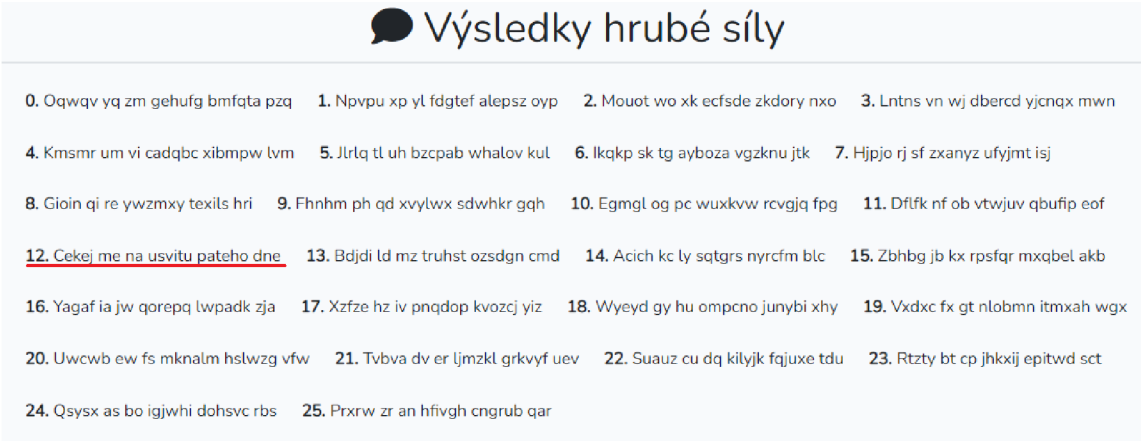
V případě Caesarovy šifry je vstupní text zpracováván znak po znaku pomocí funkce „*shiftInput*“, která přijímá počet posunů (shift) jako parametr. V první části cyklu je kontrolováno, zda je současně vybraný znak mezerou, v takovém případě se ponechává a pokračuje se dalším znakem. Pokud se nejedná o znak mezery, cyklus pokračuje převodem znaku na ASCII hodnotu. Dle velikosti písmena se přiřadí k proměnné *$firstChar* číslo 65 či 97 (65 reprezentuje v ASCII „A“, 97 znak „a“). Výstupní hodnota je vypočítána pomocí vzorce níže.

$new = (($char + $shift – $firstChar) % 26) + $firstChar

Modulo 26 zajišťuje, že v případě překročení abecedy je ukazatel vrácen na první pozici v abecedě. Hodnota je převedena na ASCII znak a přidána do výstupu, který je následně vrácen a zobrazen v UI. Postup je jednoduše vizualizován uživateli a lze vidět na Obr. 29.

Obr. 29 – Implementace Caesarovi šifry. Zdroj: vlastní zpracování

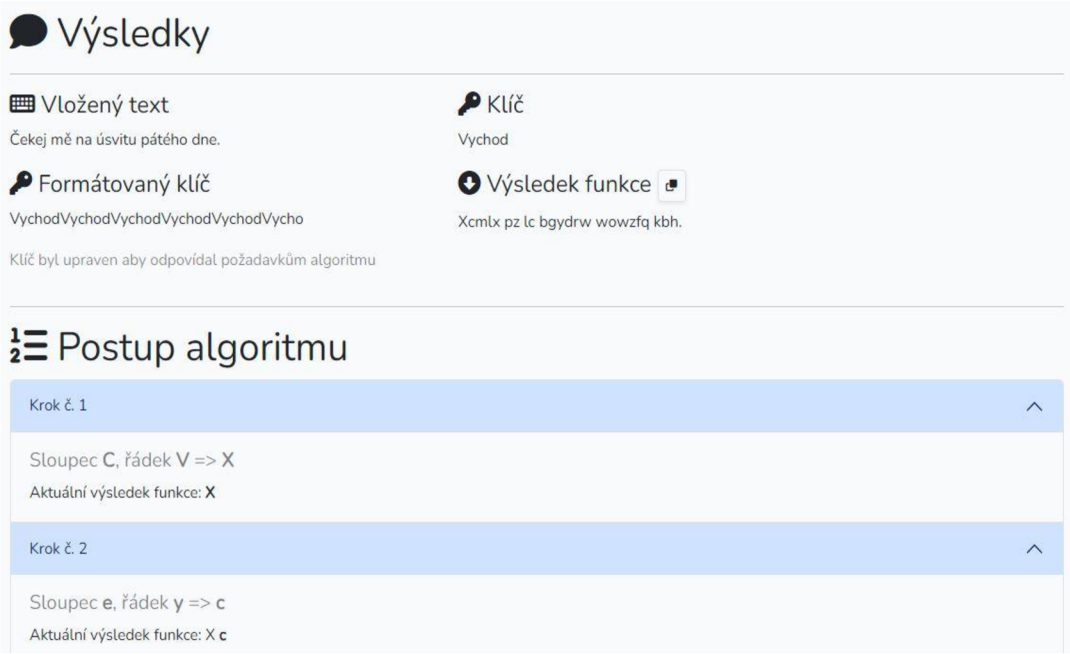
Pro situace, kdy v Caesarově šifře není k dispozici klíč k dešifrování textu, je implementována funkce „*bruteForce()*“. Tato funkcionalita vypíše všechny výsledky dešifrovacích pokusů pomocí cyklu (od 0 do 25). Tyto výsledky jsou následně vizualizovány pomocí přehledného bloku (viz. Obr. 30), ze kterého lze rozpoznat správný původní text a počet posunutí.



Obr. 30 – Caesar – útok hrubou silou. Zdroj: vlastní zpracování

Vigènerova šifra

Třída Vigenere ve své vnitřní implementaci funguje na obdobném principu jako Caesarova, ale využívá pro své šifrování textového klíče, který v kombinaci se vstupním textem určuje výběr zašifrovaného znaku pomocí Polybiovy mřížky. Oproti Caesarově šifře implementuje pokročilejší validaci vloženého klíče, který musí být stejně dlouhý jako vstupní text. V případě, že je klíč kratší, se pak při inicializaci objektu šifry konstruktorem duplikuje do požadované délky (viz. Obr. 31. Pokud je klíč prodloužen či zkrácen, systém zobrazí oznamovací hlášku o provedené úpravě. Výstupem je seznam, který ukazuje, jak provést šifru na Polybiově mřížce (viz. Obr. 31).



Obr. 31 – Implementace Vigenèrovy šifry. Zdroj: vlastní zpracování

Vernamova šifra

Vytvoření algoritmu pro Vernamovu šifru nevyžadovalo žádný komplexnější přístup, jelikož se jedná o jednoduchou operaci XOR. V třídě *CipherBase* byla vytvořena metoda s názvem „*xor*“, která provádí stejnojmennou operaci nad binárním řetězcem a je dále používána v blokových šifrách. Pro jednoduchou možnost krokování je tato operace prováděna bit po bitu. Při implementaci byl také vytvořen validátor binárních klíčů, který kontroluje délku klíče a případně jej zkrátí či doplní o nuly z levé strany na požadovanou délku. Další validační kód byl přidán pro kontrolu binárního řetězce ze vstupních dat a je používán u dalších komplexnějších částí systému.

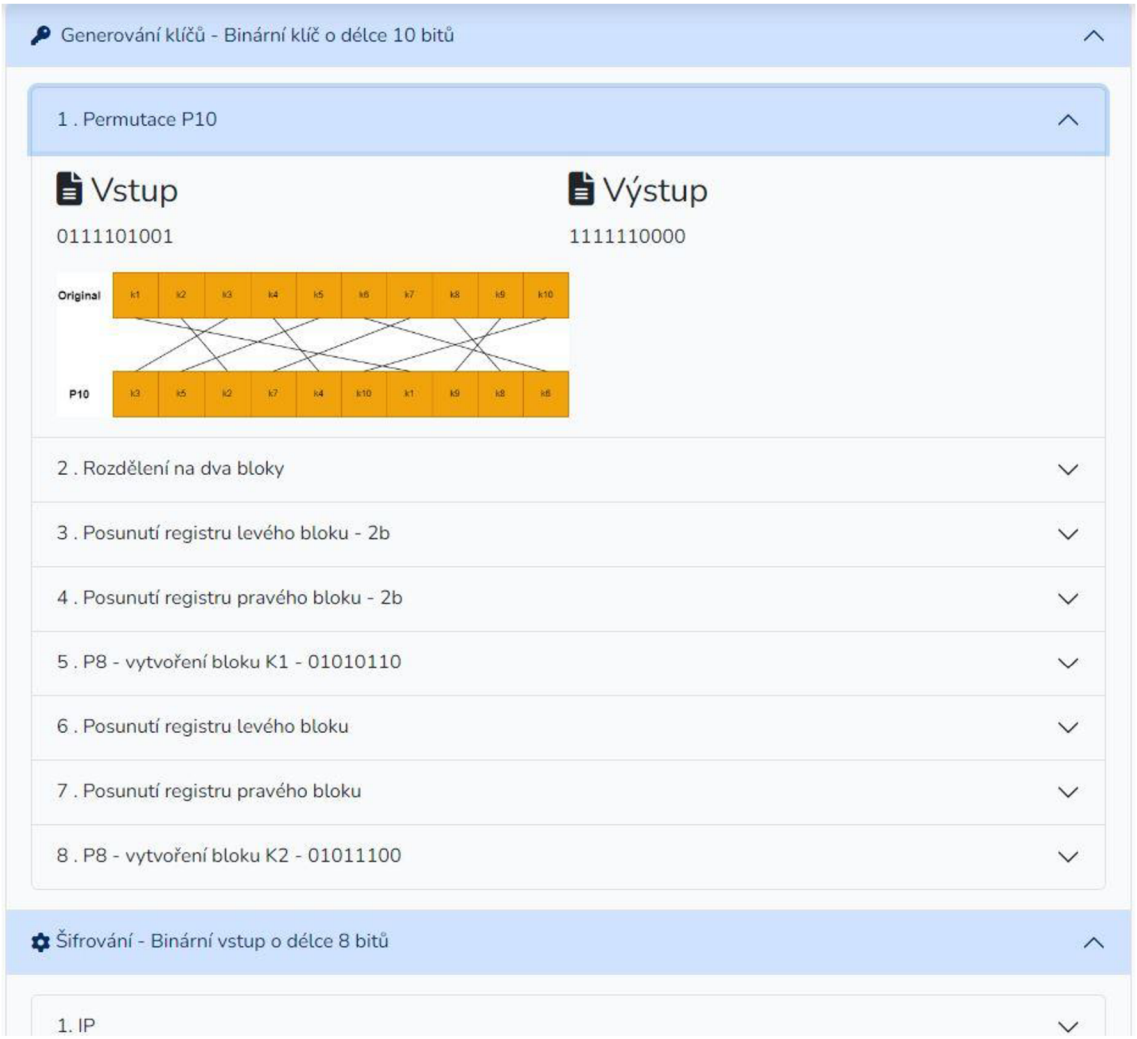
5.4.2 Blokové šifry

Při implementaci blokových šifer byla vytvořena obecná třída *BlockCipher*, která obsahuje např. definici S-boxů pro šifru Blowfish. Funkce v této třídě mohou být v případě dalších rozšíření použity znova. Většina blokových šifer má vlastní výstupní třídu, která obsahuje specifické informace pro daný algoritmus. Aby byly koncepty lépe pochopitelné pro studenty, byly zpracovány a implementovány zjednodušené verze algoritmů DES, Triple DES a AES.

S-DES

Samotná implementace S-DES je rozdělena na dvě fáze. První fází je generování rundovních klíčů, druhou pak samotná operace nad daty (tzn. šifrování/dešifrování).

Vyplněním a odesláním formuláře na stránce obsahující S-DES se vygeneruje blok s přehledem vstupních a výstupních dat. Tento přehled je doplněn o dvě výše popsané fáze. Zmíněné části lze rozkliknout a zobrazit tím detailní výpis kroků. Pokud je v daném kroku provedená komplexnější operace, pak se u kroku nachází pomocné schéma či ilustrace (viz. Obr. 32).



Obr. 32 – Implementace S-DES. Zdroj: vlastní zpracování

Jedním z klíčových výzev při implementaci bylo zpracování permutačních operací. Pro řešení této problematiky byla vytvořena tzv. "permutační pole", která určují, které bity z původních dat se mají zahrnout do výstupu (v jakém pořadí). Tento proces je realizován prostřednictvím cyklu, který projde permutační pole a zapisuje vybrané bity do nové proměnné.

Substituce pomocí S-boxů (uvedeny na Obr. 33) je prováděna pomocí dvourozměrných polí (S0 a S1). Vstupní řetězec o délce 4 bitů je rozložen do jednotlivých bitů, přičemž kombinace prvního a posledního bitu určuje řádek a kombinace druhého a třetího bitu určuje sloupec v příslušném S-boxu. Tyto bity jsou poté převedeny na decimální hodnotu, která slouží k vyhledání substituované hodnoty v S-boxové matici. Výsledkem je decimální hodnota, která je následně převedena zpět do binární podoby.

S0

1	0	3	2
3	2	1	0
0	2	1	3
3	1	3	2

S1

0	1	2	3
2	0	1	3
3	0	1	0
2	1	0	3

Obr. 33 – Substituční S-boxy S-DES. Zdroj: vlastní zpracování

Triple S-DES

Triple DES využívá trojnásobného zopakování algoritmu DES, přičemž v každém průchodu se využívá jiný klíč. Ve webové aplikaci byl Triple S-DES složen pomocí sekvence šifrování, dešifrování a šifrování (EDE). Implementace využívá již připraveného a zjednodušeného algoritmu S-DES. Zjednodušená šifra byla zvolena především za účelem zobrazení přehledného postupu algoritmem.

Klíčový management v aplikaci byl zvolen tak, aby odpovídal nejvyššímu zabezpečení, tzn. aby všechny tři klíče byly nezávislé a bylo nutné je ve formuláři

zadat. Pro tento algoritmus byla vytvořena nová třída obsahující seznam kroků jednotlivých průchodů DES.

Blowfish

Při vytvoření třídy Blowfish je pomocí konstruktoru volána inicializační metoda *modifySubkeys*, která upravuje potřebné S-boxy (naplněné hodnotou π) a pole s předdefinovanými klíči. V první části se permutují klíče z pole se vstupním klíčem. Následně se provádí expanze subklíčů pomocí rundovní funkce. V poslední řadě je pak provedena expanze pomocí S-boxů.

Vstupní text je při inicializaci zakódován/dekódován pomocí base64 a rozdělen do 64bitových bloků, které jsou následně zašifrovány pomocí metody *encryptChunk*. Tato metoda provádí šifrovací cyklus, který se skládá z 16 rund.

Ve Feistelově funkci je vstupní blok dat rozdělen na čtyři osmibitové části, což odpovídá rozdělení na jednotlivé bajty. Tyto čtyři části jsou označeny jako *$a, $b, $c* a *$d*. Hodnoty v *$a* a *$b* jsou použity jako indexy do prvních dvou S-boxů (1 a 2). Výstupy těchto S-boxů jsou sečteny a tato suma představuje první část výsledku Feistelovy funkce. Následně je provedena exkluzivní disjunkce (XOR) výsledku s hodnotou získanou z třetího S-boxu (3) podle indexu *$c*. Výsledek z předchozího kroku je opět sečten s hodnotou z posledního S-boxu (4) podle indexu *$d*. Provedení lze vidět v následující ukázce kódu (Obr. 34).

```
1.  protected function feistelFunction($inputBlock)
2.  {
3.      //create 4 8-bit chunks
4.      $d = $inputBlock & 0xFF;
5.      $c = ($inputBlock >> 8) & 0xFF;
6.      $b = ($inputBlock >> 16) & 0xFF;
7.      $a = ($inputBlock >> 24) & 0xFF;
8.
9.      $feistelOutput = $this->sboxes[0][$a] + $this->sboxes[1][$b];
10.     $feistelOutput = $feistelOutput ^ $this->sboxes[2][$c];
11.
12.     return $feistelOutput + $this->sboxes[3][$d];
13. }
```

Obr. 34 - Feistelova funkce. Zdroj: vlastní zpracování

Zásadní problematikou byla implementace klíčového managementu. Výpočet výchozích hodnot podklíčů a S-boxů by byl příliš náročný, proto bylo využito již předpřipravených polí, která jsou dostupná online [49]. V těchto polích se nachází předem vygenerované hodnoty π, které jsou použity při transformaci podklíčů. Pro účely lepší čitelnosti výsledků jsou jednotlivé výstupy kódovány pomocí base64. Jedná se o plnou verzi algoritmu, proto je potřeba využít kódování, aby byl výstup čitelný a lépe strukturovaný.

S-AES

Při inicializaci S-AES se provádí prvotní kontrola délky klíče a textu. Pokud jsou vstupní data kratší či delší, než je požadováno, pak jsou prodloužena či zkrácena. Dalším krokem je generování rundovních klíčů, což znamená rozdělení vstupního klíče, provedení nutných XOR operací a substituování pomocí S-boxu. Výstupem jsou tři podklíče, které algoritmus následně využívá pro šifrování či dešifrování.

Nejzajímavější částí implementace je transformace pomocí funkce „*MixColumns*“. Tato funkce jako parametr přijímá stavovou matici, kterou následně vynásobí nadefinovaným polynomem. V rámci tohoto promíchání dat je vytvořena funkce, která provádí operaci násobení v Galoisově tělese $GF(2^4)$. Tato funkce se vykonává pomocí cyklu přes jednotlivé bity druhého operandu.

Při násobení v $GF(2^4)$ se v každé iteraci cyklu nejprve zkontroluje, zda nejméně významný bit druhého operandu (b) je 1. Pokud ano, provede se bitový XOR výsledku s prvním operandem (a), což přidá hodnotu prvního operandu (a) k výsledku. Poté se zkontroluje, zda je nejvíce významný bit prvního operandu (a) nastaven na 1. Pokud ano, provede se operace redukce modulo $x^4 – x + 1$, která zajistí, že hodnota prvního operandu (a) zůstane v $GF(2^4)$. Nakonec se celý výsledek redukuje modulo *x^4 + x + 1*, aby se zajistilo, že zůstane v rámci konečného pole $GF(2^4)$. Tímto způsobem je zajištěno, že výsledky zůstanou v binární soustavě a

v rozmezí Galoisova konečného tělesa. Provedení násobení lze vidět níže (viz. Obr. 35).

```php
private function gfMultiply(int $a, int $b): int
{
    // Initialize the result to 0.
    $result = 0;

    // Perform multiplication in the Galois Field GF(16) using the standard algorithm.
    for ($i = 0; $i < 4; $i++) {
        // If the least significant bit of $b is 1, then XOR the result with $a.
        if (($b & 1) != 0) {
            $result ^= $a;
        }

        // Check if the most significant bit of $a is set.
        $msbSet = ($a & 0x8) != 0;

        // Left shift $a by 1.
        $a <<= 1;

        // If the most significant bit of $a was set, XOR $a with 0x3 to reduce modulo x^4 + x + 1.
        if ($msbSet) {
            $a ^= 0x3; // Reduce modulo x^4 + x + 1
        }

        // Right shift $b by 1.
        $b >>= 1;
    }

    // Reduce the result modulo x^4 + x + 1 and return it.
    return $result & 0xF; // Reduce modulo x^4 + x + 1
}
```

Obr. 35 - Implementace násobení v Galoisově tělese. Zdroj: vlastní zpracování

Odesláním testovacího formuláře uživatel obdrží komplexní výpis všech provedených akcí. Tyto akce jsou rozděleny do dvou fází – generování klíčů a provedené operace.

5.4.3 Proudová šifra A5/1

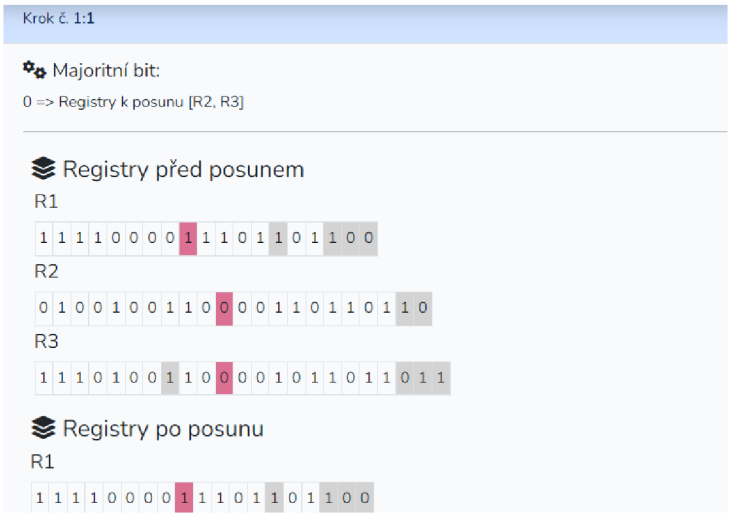
Šifra A5/1 využívá pro své operace tří stavových registrů. Tyto registry jsou hned po provolání konstruktoru inicializovány metodou „*initializeRegisters()*“. Tato funkce na svém začátku všechny tři registry naplní nulou a následně provádí několik cyklů. V prvním cyklu se do registrů propisuje vliv klíče, druhým cyklem se zahrnuje i inicializační vektor. Uvnitř cyklů se provádí posun všech registrů a do každého registru je na první pozici XORován bit z klíče či vektoru. Inicializace je završena sto iteracemi, které provádějí posun registrů na základě výběru registru dle majoritního

bitu. Tato část není vidět v uživatelském rozhraní, jelikož by se velmi rychle stala nepřehlednou a matoucí. Inicializace registrů je popsána ukázkou (viz. Obr. 36).

```php
public function initializeRegisters()
    {
        $this->R1 = array_fill(0, 19, 0);
        $this->R2 = array_fill(0, 22, 0);
        $this->R3 = array_fill(0, 23, 0);

        for ($i = 0; $i < 64; $i++) {
            $this->clockAllRegisters(irregular: false);
            $keyBit = ($this->key[63 - $i]);
            $this->R1[0] ^= $keyBit;
            $this->R2[0] ^= $keyBit;
            $this->R3[0] ^= $keyBit;
        }

        for ($i = 0; $i < 22; $i++) {
            $this->clockAllRegisters(irregular: false);
            $frameBit = ($this->key[21 - $i]);
            $this->R1[0] ^= $frameBit;
            $this->R2[0] ^= $frameBit;
            $this->R3[0] ^= $frameBit;
        }

        for ($i = 0; $i < 100; $i++) {
            $this->clockAllRegisters();
        }
    }
```

Obr. 36 - A5/1 - Vnitřní implementace inicializace registrů. Zdroj: vlastní zpracování

Samotné šifrování a dešifrování spočívá ve vygenerování tzv. klíčového streamu a XORování se vstupními daty (ukázka viz. Obr. 37). Toho je docíleno pomocí jednoduchého for cyklu. Výsledkem odeslaného formuláře je seznam kroků obsahující XOR dat s klíčovým streamem a ukázku registrů při vytvoření klíčového bitu. Podstatné části registrů jsou zde barevně označeny. Červené obarvení ukazuje pozici, ze které se volí bit k vyhodnocení majoritního bitu, šedě pak bit, který je zvolen k operaci XOR za účelem vytvoření klíčového bitu.



Obr. 37 - A5/1 - Ukázka výstupu. Zdroj: vlastní zpracování

5.4.4 Asymetrické šifry

V rámci tvorby aplikace byla implementována šifra RSA a algoritmus pro výměnu klíčů Diffie-Hellman. Implementace tohoto typu šifer je náročná nejen z hlediska výpočetní složitosti, ale i z hlediska vizuálního představení uživatelům. Jelikož asymetrické algoritmy slouží k digitálním podpisům a výměnám klíčů, je poměrně složité vytvořit uživatelsky přívětivou ukázku s krokováním.

RSA

Prvním asymetrickým algoritmem, který je v aplikaci implementován, je RSA. Tento algoritmus vytváří veřejný a soukromý klíč pomocí několika matematických operací, přičemž jednou z nich je násobení vysokých prvočísel (viz. kapitola 4.4.1 RSA). Násobek těchto prvočísel může být natolik velký, že by přetekl maximální hodnotu číselného datového typu, a proto je nutné tento fakt zohlednit i v implementaci. Pro práci s velkými čísly byla použita nativní knihovna GMP, která umožňuje prakticky nekonečnou přesnost číselné hodnoty v závislosti na maximální paměti zařízení, na kterém aplikace běží. Ve svém vnitřním fungování se knihovna k vloženému číslu chová jako k textovému řetězci a umožňuje nad nimi provádět optimalizované operace, jako je například mocnění, násobení či modulo.

Kód k šifrování pomocí RSA a využití knihovny GMP lze vidět v ukázce (viz. Obr. 38). Obrázek představuje inicializaci velkých čísel, jejich násobení a mocnění. Po průchodu jsou jednotlivé výstupy algoritmu ukládány pomocí modelu *RSAStep* a zobrazovány v UI.

```php
$p = gmp_init($this->p);
$q = gmp_init($this->q);

// Encryption key generation -> calculate n from prime numbers
$n = gmp_mul($p, $q);

$e = 65537;
// declaring phi
$phi = gmp_mul(gmp_sub($p, 1), gmp_sub($q, 1));
// searching for co-prime number
while ($e < gmp_intval($phi)) {
    /*
     * e must be co-prime to phi and
     * smaller than phi.
     */
    if (gmp_intval(gmp_gcd($e, $phi)) === 1) {
        break;
    } else {
        $e++;
    }
}
$d = gmp_invert(gmp_init($e), $phi);
$d = gmp_intval($d);

// Encryption process
$result = '';
$steps = [];
foreach (explode(' ', $this->text) as $char) {
    $c = gmp_powm(gmp_init($char), $e, $n); // uses binary exponentiation (CORM09)
    $steps[] = new RSAStep(inputChar: (int)$char, outputChar: gmp_intval($c));
    $result .= $c . ' ';
}
```

Obr. 38 – Algoritmus RSA. Zdroj: vlastní zpracování

Diffie-Hellman

Diffie-Hellmanův algoritmus pro výměnu klíčů pracuje stejně jako RSA s velkými čísly, která by mohla překročit limit datového typu. Pro implementaci bylo nutné využít knihovny GMP. Vizualizace výměny klíčů neumožnila využití jednotného designu pro formulář, proto se některé části oproti ostatním algoritmům

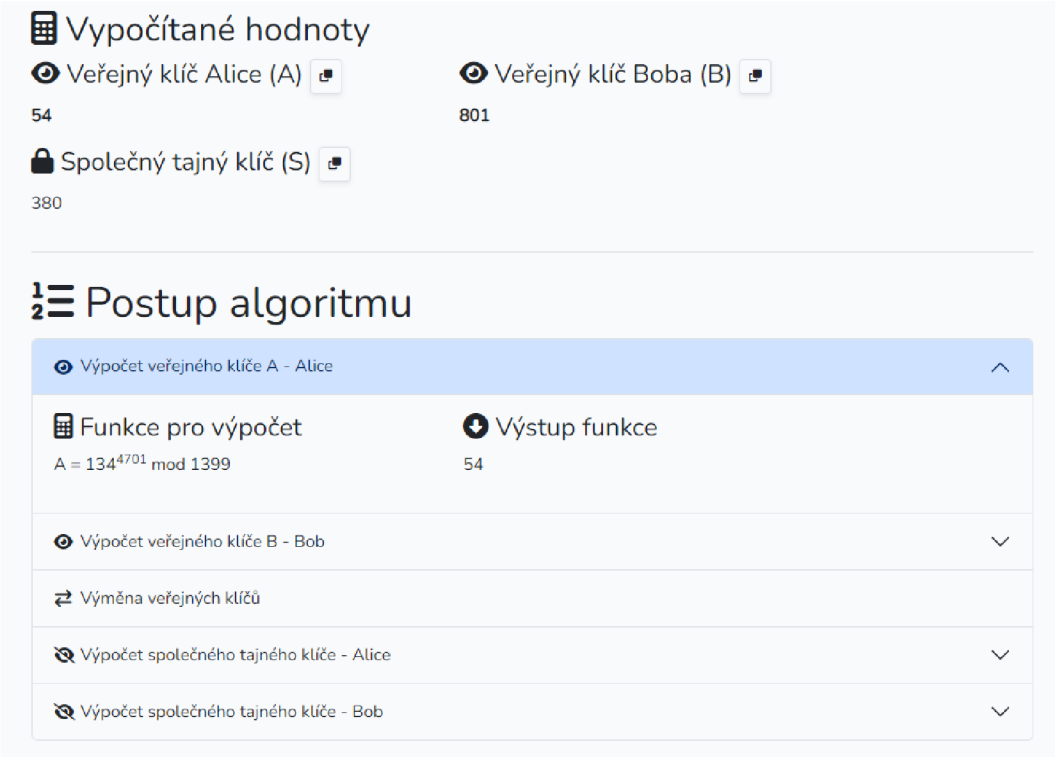
liší. Konkrétním rozdílem je například chybějící výběr akce šifrování a dešifrování (viz. Obr. 39).

Obr. 39 – Formulář Diffie-Hellman. Zdroj: vlastní zpracování

Při implementaci vznikla nová validace dat v podobě kontroly, zda je zadaná hodnota primitivním kořenem k zadanému modulu. Tato validace probíhá na klientské straně pomocí Javascriptu a nativních validačních funkcí (zobrazení chyby ve formuláři) v kombinaci s vlastním řešením validace. Validaci bylo nutné převést i na backend aplikace pro případ, že by se do controlleru dostala data z jiného zdroje, než je formulář. Též byl vytvořen nový generátor náhodných dat, který tuto validaci bere v potaz.

Samotná validace vstupní hodnoty *g* probíhá pomocí několika funkcí. První funkcí je „*isPrimitiveRoot (g, n)*“, která přijímá parametry v podobě primitivního kořene (g) a prvočísla (n). Tato funkce kontroluje, zda je číslo g primitivním kořenem modulo n za pomoci funkce „*modExponentiation (g, exponent, n)*“, která využívá algoritmus *Exponentiation by squaring* pro modulární umocňování. Další částí je funkce „*primeFactors(num)*“, která používá *Trial Division* pro nalezení prvočíselných faktorů daného čísla. Principy těchto algoritmů jsou blíže popsané v [37].

Výstup tohoto algoritmu se příliš neliší od již zavedeného designu. Změnou je pouze interpretace informací, aby byly čtenářem lépe pochopitelné. Tento výstup je vyobrazen na obrázku (viz. Obr. 40).



Obr. 40 – Výstup algoritmu Diffie-Hellman. Zdroj: vlastní zpracování

5.5 Nasazení do produkčního prostředí

Pro rychlé a snadné zprovoznění aplikace byl vytvořen linuxový skript *build.sh*, který se spouští nad běžícím kontejnerem s projektem. Soubor značně ulehčuje nasazení aplikace do produkčního prostředí, jelikož provádí všechny potřebné akce pro korektní spuštění aplikace. V případě změn ve zdrojovém kódu je tento skript nutno spustit, protože provede instalaci vyžadovaných závislostí a pročistí dočasné soubory, které obsahují předkreslené obsahy stránek.

6 Testování aplikace

Obsah byl postupně otestován za účelem dosažení korektnosti a bezchybnosti jednotlivých implementovaných algoritmů pomocí dostupných nástrojů a příkladů. V druhé fázi byla aplikace otestována uživateli, kteří obdrželi krátký dotazník se sadou dotazů zaměřených na získání zpětné vazby. Cílem dotazníkového šetření bylo získání názorů a návrhů na zlepšení funkcionalit aplikace. Dotazy byly zaměřeny na design, kvalitu obsahu, technickou funkčnost aplikace a úroveň nápomocnosti v otázce podpory výuky.

Tento dotazník byl vytvořen za pomoci nástroje ChatGPT 3.5, který pomohl s formulací dotazů. Formulář s dotazníkem je zpracován v online formě a je dostupný na adrese https://forms.gle/y2sw8usD6dFv9BMw8.

6.1 Otestování správnosti implementace algoritmů

Otestování správnosti implementace kryptografických primitiv ve výukové aplikaci má klíčový význam pro poskytnutí kvalitního vzdělání v oblasti kryptografie. Zajišťuje, že studenti mohou spoléhat na to, že techniky, které se učí, jsou implementovány správně a fungují přesně tak, jak bylo autory algoritmů zamýšleno. Testování proběhlo pomocí online nástrojů a příkladů z literatury. Výsledky jsou následně srovnány s výsledky ve webové aplikaci.

Caesarova a Vigenèrova šifra

Pro otestování klasických šifer byl využit nástroj s názvem Cryptool [50]. Vstupním textem pro Caesarovu a Vigenèrovu šifru bylo zvoleno slovo „*slepice*“. Pro Caesarovu šifru byl zvolen posun o 13 znaků, přičemž výsledkem z Cryptool se stal řetězec „fyrCvpr“, který odpovídá i výstupu z implementované aplikace. Pro Vigenèrovu šifru byl zvolen šifrovací klíč „*vejce*“. Šifrováním stejného vstupního textu bylo dosaženo výsledku „*npnrmxi*“. Ve webové aplikaci byl výsledek identický.

Vernamova šifra

Z oblasti klasických šifer byla otestována i poslední Vernamova šifra. Korektnost této šifry bylo možné vyzkoušet pomocí příkladu uvedeném v [51]. Jako vstup posloužila posloupnost „*10110010111001*“. Šifrovacím klíčem pak byl řetězec „*11010001010100*“. Po provedení operace XOR je implementací vygenerován výsledek „*01100011101101*“, který odpovídá i uvedenému výsledku z příkladu.

S-DES

Pro otestování správnosti S-DES byl využit příklad uvedený v [52]. První částí k testování je generování rundovních klíčů. Při vstupu „*1101001101*“ byly v obou případech vytvořeny podklíče *K1 = „11011010“* a *K2 = „10001101“*. Zašifrováním vstupního řetězce „*10101010*“ vznikl šifrovaný text „*00111001*“ se stejným postupem a výsledkem jako v uvedeném dokumentu.

Triple S-DES

K Triple S-DES neexistuje žádný dostupný nástroj ani dokument s příkladem, jelikož se jedná o unikátní implementaci. Přesto však lze správné fungování ověřit pomocí trojího využití S-DES, jehož správnost je ověřena z předešlého kroku. Vstupní parametry byly zvoleny následovně. Vstupní binární řetězec „*11011001*“, K1 = „*1010000101*“, K2 = „*1010100100*“ a K3 = „*1001100100*“. Po manuálním provedení EDE (Encrypt, Decrypt, Encrypt) na S-DES byl vytvořen výsledek „*11010110*“. Tento výsledek byl totožný s výsledkem Triple S-DES.

Blowfish

Pro algoritmus Blowfish je k dispozici omezený počet nástrojů a dokumentace obsahující příklady. Většina těchto nástrojů je navržena s důrazem na poskytnutí nejvyšší úrovně zabezpečení, pracuje s různými režimy expanze klíčů či využívá metody zvětšování výstupních bloků. Chování algoritmu je také snadné ovlivnit jinou metodou zpracování vstupních dat či mírným odchýlením od původního algoritmu. Z těchto důvodů nebylo možné ověřit správnost algoritmu za využití vzorového příkladu z dokumentace či nástroje. Místo testu korektnosti byla provedena kontrola kódu proti pseudo kódu [18] uvedenému autorem šifry.

Implementovaný algoritmus postupuje dle sepsaných instrukcí, odlišnost je pouze ve zpracování dat před vstupem do algoritmu a ve finální formě zobrazení výstupních dat.

S-AES

Vyzkoušení korektnosti S-AES bylo otestováno skrze příklad uvedený v [42]. Jako vstup posloužil řetězec „*0110 1111 0110 1011*“, což v převodu do ASCII znamená „*OK*“. Jako klíč byl využit řetězec „*1010 0111 0011 1011*“. Po provedení šifrování byl vytvořen šifrovaný text „*0000 0111 0011 1000*“. Výsledek je po porovnání totožný jako v příkladu.

A5/1

A5/1 byl testován vůči naprogramovanému python skriptu, který je dostupný ve veřejném repozitáři na platformě Github [53]. Po zadání dvou stejných vstupů do implementované varianty a skriptu vznikl stejný šifrovaný text. Data zde nejsou uvedena, jelikož jsou vstupní binární řetězce příliš dlouhé.

RSA

K testu správnosti RSA byla využita veřejně dostupná RSA kalkulačka [54]. Po zadání vstupního textu „*Ahoj*“ proběhlo převedení do ASCII „*97 104 111 106*“. Tento text byl použit k šifrování s klíči „*p = 13*“ a „*q = 23*“ a v obou případech byl vytvořen šifrovaný text „*106 117 63 97*“.

Diffie-Hellman

Diffie-Hellmanův algoritmus byl ověřen pomocí online nástroje [55] pro výpočet společného klíče. Nastaveno bylo prvočíslo „*p = 13*“ k němu primitivní kořen „*g = 7*“, soukromý klíč Alice „*a = 5*“ a soukromý klíč Boba „*b = 7*“. Po odeslání formuláře se v obou případech vytvořil společný šifrovací klíč „*2*“.

Závěr testování korektnosti algoritmů

Testováním za pomoci online nástrojů a dostupných příkladů v literatuře bylo zjištěno, že všechny implementované algoritmy pracují nad vstupními daty korektně a zobrazují odpovídající výsledky.

6.2 Vyhodnocení dotazníkového šetření

Vzorek respondentů se skládal celkem z 36 uživatelů, z toho 29 (80,56 %) bylo studujících a 7 (19,44 %) nestudujících. Ze vzorku studujících byla většina na vysoké škole (68,97 %), menšinu tvořili středoškoláci (31,03 %). V průzkumu převažovaly technicky založené obory (79,31 %), přičemž dotazník byl vyplněn i studenty z oborů bez návaznosti na technologii (20,69 %). Vzorek uživatelů je vyobrazen na následujícím obrázku (viz. Obr. 41).

Studující / nestudující	Počet	Procentuální podíl
Studenti	29	80,56 %
Nestudující	7	19,44 %
Celkem	36	100,00 %

Vysoká škola / Střední škola	Počet	Procentuální podíl
Vysoká škola	20	68,97 %
Střední škola	9	31,03 %
Celkem	29	100,00 %

Obory respondentů	Počet	Procentuální podíl
Aplikovaná informatika	16	55,17 %
Jiné netechnické obory	6	20,69 %
Jiné technické obory	4	13,79 %
Informační management	3	10,34 %

Technicky založené obory	23	79,31 %
Netechnické obory	6	20,69 %

Nestudující - dosažené vzdělání	Počet	Procentuální podíl
Vysokoškolské vzdělání	4	57,14 %
Středoškolské vzdělání	3	42,86 %
Celkem	7	100,00 %

Obr. 41 – Vzorek testujících uživatelů. Zdroj: vlastní zpracování.

Z celkového počtu respondentů bylo 47,22 % velmi spokojeno s webovou aplikací, druhou nejvíce volenou odpovědí byla „*spokojen(a)*“ se zastoupením v 41,6 %. Uživatelské rozhraní bylo až na výjimky hodnoceno kladně jako „*intuitivní*“ (47,22 %) a „*intuitivní a snadné*“ (47,22 %). Ve dvou případech došlo k průměrnému

a podprůměrnému hodnocení *(průměrné* a *obtížné na používání*). Z hlediska funkčnosti převažovala odpověď, že testování proběhlo bez technických problémů (88,89 %). Webová aplikace byla také hodnocena z hlediska informativnosti obsahu. V tomto případě se 69,44 % uživatelů shodlo na tom, že je obsah velmi informativní a dobře strukturován. Obsah byl také shledán 30,56 % uživatelů jako informativní, ale s možností zlepšení. Na následujícím obrázku jsou vyobrazena zpracovaná získaná data (viz. Obr. 42). Z dat byly odebrány hodnoty, které nebyly ve výsledcích dotazníku zastoupeny.

Celkem odpovědí	36	
Obecný dojem	Počet	Procentuální podíl
Velmi spokojen(a)	17	47,22 %
Spokojen(a)	15	41,67 %
Neutrální	4	11,11 %
Uživatelské rozhraní	Počet	Procentuální podíl
Velmi intuitivní a snadné	17	47,22 %
Intuitivní	17	47,22 %
Průměrné	1	2,78 %
Obtížné na používání	1	2,78 %
Funkčnost	Počet	Procentuální podíl
Bez problémů	32	88,89 %
Občasné problémy	3	8,33 %
Průměrná funkčnost	1	2,78 %
Informativnost	Počet	Procentuální podíl
Obsah je velmi informativní a dobře strukturovaný	25	69,44 %
Obsah je informativní, ale mohl by být lépe organizován	11	30,56 %

Obr. 42 – Hodnocení webové aplikace. Zdroj: vlastní zpracování

Uživatelům se nejlépe pracovalo s klasickými a asymetrickými šiframi. Z klasických šifer byla zvolena Caesarova šifra jako nejvíce srozumitelná a uživatelům se s ní pracovalo nejlépe. V oblasti asymetrických šifer se uživatelům nejlépe pracovalo s algoritmem RSA. Z blokových šifer se uživatelům nejlépe pracovalo s šifrou Blowfish a nejvíce srozumitelným se stal Triple-SDES. Preference klasických šifer oproti ostatním může být odůvodněna nižší obtížností pro pochopení algoritmů. Zhodnocením všech získaných dat lze konstatovat, že jsou uživatelé s aplikací spokojeni. Uživatelské rozhraní je intuitivní a obsahově

dostatečně informativní. Aplikace je považována jako převážně bezproblémová a případné chyby byly opraveny.

6.3 Zpětná vazba od uživatelů

V průběhu testování bylo díky návrhům od uživatelů provedeno několik vylepšení aplikace a drobných oprav nedostatků. Mezi tato vylepšení patří lepší způsob otevírání odkazů, zlepšení přístupnosti v rámci odeslaného formuláře a drobné úpravy zobrazení pro menší zařízení. Testování také umožnilo nalezení a následnou opravu drobné chyby při generování vstupních dat do formuláře.

7 Závěr

Tato diplomová práce se zabývala vývojem kryptografické webové aplikace v PHP frameworku a měla za cíl podpořit výuku principů kryptografických primitiv. Výsledkem této práce je literární rešerše kryptografie v praxi a uplatnění vybraných kryptografických primitiv ve výukové aplikaci. Další částí je rozbor fungování jednotlivých algoritmů, návrh nového systému a teoretický pohled na implementaci jednotlivých implementačních nástrojů. V rámci tohoto rozboru byla popsána problematika škálování webových aplikací za využití principu kontejnerizace, ale i nasazování aplikace na produkční prostředí.

Implementací zmíněných návrhů vznikla nová verze webové aplikace na adrese https://educrypter.uhk.cz, která bude sloužit k podpoře výuky kryptografie na fakultě informatiky a managementu na Univerzitě Hradec Králové. Tato aplikace byla důkladně otestována z pohledu korektnosti jednotlivých kryptografických primitiv, ale i použitím v reálném prostředí prostřednictvím uživatelů se zájmem o kryptografii. Aplikace byla shledána uživateli jako užitečná a dostatečně informativní pro pochopení komplexnějších šifrovacích metod.

Možným rozšířením vzniklé aplikace je implementace dalších vhodných kryptografických algoritmů. Aplikace by mohla být zlepšena například přidáním algoritmů na principu eliptických křivek, krokované tvorby digitálních podpisů, ukázkou hashovacích algoritmů či komplexním rozborem náhodných generátorů klíčů.

8 Seznam použité literatury

[1] *Laravel - The PHP Framework For Web Artisans* [online]. [vid. 2024-03-06]. Dostupné z: https://laravel.com/

[2] *The GNU MP Bignum Library* [online]. [vid. 2024-04-18]. Dostupné z: https://gmplib.org/

[3] Application Performance Monitoring & Error Tracking Software. *Sentry* [online]. [vid. 2024-03-06]. Dostupné z: https://sentry.io/welcome/

[4] *Psalm - a static analysis tool for PHP* [online]. [vid. 2024-03-06]. Dostupné z: https://psalm.dev/

[5] *Docker: Accelerated Container Application Development* [online]. 10. květen 2022 [vid. 2024-03-06]. Dostupné z: https://www.docker.com/

[6] *Welcome! - The Apache HTTP Server Project* [online]. [vid. 2024-04-18]. Dostupné z: https://httpd.apache.org/

[7] HEJZLAR, Zdeněk. *Zabezpečení TLS*. Hradec Králové, 2023. Bakalářská práce. Univerzita Hradec Králové.

[8] BANOTH, R. a R. REGAR. *Classical and Modern Cryptography for Beginners*. 2023. Classical and Modern Cryptography for Beginners. ISBN 978-3-031-32959-3.

[9] NÁRODNÍ ÚŘAD PRO KYBERNETICKOU A INFORMAČNÍ BEZPEČNOST. *Doporučení v oblasti kryptografických prostředků* [online]. V3.0. B.m.: Národní úřad pro kybernetickou a informační bezpečnost. 2023 [vid. 2024-04-13]. Dostupné z: https://nukib.gov.cz/download/uredni_deska/Minimalni%20pozadavky%20na%20kryptograficke%20algoritmy.pdf

[10] MICROSOFT CORPORATION. *BitLocker™ Drive Encryption Security Policy for FIPS 140-2 Validation* [online]. Bezpečnostní pravidla. v 1.1. B.m.: Microsoft Corporation. 2011 [vid. 2024-04-13]. Dostupné z: https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp1054.pdf

[11] SIDHU, Armaan. Analyzing Modern Cryptography Techniques and Reviewing their Timeline (2023) [online]. 2023 [vid. 2024-04-13]. Dostupné z: https://www.researchgate.net/publication/369013746_Analyzing_Modern_Cryptography_Techniques_and_Reviewing_their_Timeline_2023

[12] ABDALLAH, Eslam G., Yu Rang KUANG a Changcheng HUANG. Advanced Encryption Standard New Instructions (AES-NI) Analysis: Security, Performance, and Power Consumption. In: *ICCAE 2020: 2020 12th International Conference on Computer and Automation Engineering*: *Proceedings of the 2020*

12th International Conference on Computer and Automation Engineering [online]. Sydney NSW Australia: ACM, 2020, s. 167–172 [vid. 2024-04-17]. ISBN 978-1-4503-7678-5. Dostupné z: doi:10.1145/3384613.3384648

[13] NIPPON TELEGRAPH AND TELEPHONE CORPORATION a MITSUBISHI ELECTRIC CORPORATION. *Camellia Encryption Algorithm Selected for New e-Government Recommended Ciphers List* [online]. No.2750. Tokyo: Nippon Telegraph and Telephone Corporation Mitsubishi Electric Corporation. 2013 [vid. 2024-04-13]. Dostupné z: https://www.mitsubishielectric.com/news/2013/pdf/0326-b.pdf

[14] SCHAEFER, Edward. A simplified data encryption standard algorithm. *CRYPTOLOGIA*. 1996, **20**(1), s. 77-84. ISSN 0161-1194. Dostupné z: doi:10.1080/0161-119691884799

[15] KUMAR, Sandeep, Christof PAAR, Jan PELZL, Gerd PFEIFFER, Andy RUPP a Manfred SCHIMMLER. How to Break DES for BC 8,980. *SHARCS '06–Special-purpose Hardware for Attacking Cryptographic Systems*. 2006, **2006**(17–35).

[16] *NVD - CVE-2016-2183* [online]. [vid. 2024-03-26]. Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2016-2183

[17] BARKER, Elaine a Allen ROGINSKY. *Transitioning the use of cryptographic algorithms and key lengths*. NIST SP 800-131Ar2. Gaithersburg, MD: National Institute of Standards and Technology. 2019. Dostupné z: doi:10.6028/NIST.SP.800-131Ar2

[18] SCHNEIER, Bruce. Description of a new variable-length key, 64-bit block cipher (Blowfish). In: *International Workshop on Fast Software Encryption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, s. 191–204. ISBN 978-3-540-58108-6. Dostupné z: doi:10.1007/3-540-58108-1_24

[19] PATIL, Priyadarshini, Prashant NARAYANKAR, NARAYAN D.G., a MEENA S.M. A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science*. 2016, **78**, 1st International Conference on Information Security & Privacy 2015, s. 617-624. ISSN 1877-0509. Dostupné z: doi:10.1016/j.procs.2016.02.108

[20] SCHNEIER, Bruce, Doug WHITING, David WAGNER, Chris HALL, Niels FERGUSON a Bruce KELSEY. Twofish: A 128Bit Block Cipher. *NIST AES Proposal*. 1998, **1998**(15.1), s. 23-91.

[21] *Bruce Almighty: Schneier preaches security to Linux faithful - Computerworld* [online]. 2016 [vid. 2024-03-27]. Dostupné z: https://web.archive.org/web/20161202063854/https://www.computerworld.com.au/article/46254/bruce_almighty_schneier_preaches_security_linux_faithful/?pp=3

[22] ALEXANDROV NIKOLOV, Petar. *Analysis and Design of a Stream Cipher* [online]. B.m., 2019 [vid. 2024-03-27]. b.n. Dostupné z: http://rua.ua.es/dspace/handle/10045/93272

[23] TABBANE, Sami. 4G and 5G networks security techniques and algorithms [online]. 2019 [vid. 2024-03-24]. Dostupné z: https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2019/ITUPITA2018/ITU-ASP-CoE-Training-on-/4G%20and%205G%20network%20security%20techniques%20and%20algorithms.pdf

[24] NIR, Yoav. *ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec* [online]. Request for Comments. RFC 7634. 2015. Dostupné z: doi:10.17487/RFC7634

[25] 3GPP. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2* [online]. Technická specifikace. v2.1. 2009 [vid. 2024-04-13]. Dostupné z: https://www.gsma.com/aboutus/wp-content/uploads/2014/12/uea2uia2d1v21.pdf

[26] CANTEAUT, Anne. Stream Cipher. In: Henk C. A. VAN TILBORG, ed. *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2005, s. 596–597. ISBN 978-0-387-23483-0. Dostupné z: doi:10.1007/0-387-23483-7_412

[27] RAMADAN, Mohammed, Guohong DU, Fagen LI a Chun XU. EEE-GSM: End-to-end encryption scheme over GSM system. *International Journal of Security and Its Applications*. 2016, **10**, 229–240. Dostupné z: doi:10.14257/ijsia.2016.10.6.22

[28] HUMADI, Ahmed. *Symmetric and Asymmetric Encryption*. 2020 [vid. 2023-04-18]. Dostupné z: doi:10.13140/RG.2.2.21500.56962

[29] Aryan, Chaithanya KUMAR a Durai VINCENT P M. Enhanced diffie-hellman algorithm for reliable key exchange. *IOP Conference Series: Materials Science and Engineering*. 2017, **263**, 042015. Dostupné z: doi:10.1088/1757-899X/263/4/042015

[30] AIKINS-BEKOE, Stephen a James HAYFRON-ACQUAH. Elliptic Curve Diffie-Hellman (ECDH) Analogy for Secured Wireless Sensor Networks. *International Journal of Computer Applications*. 2020, **176**, s. 1-8. Dostupné z: doi:10.5120/ijca2020920015

[31] ANDRESS, Jason. Cryptography. In: *The Basics of Information Security*. B.m.: Elsevier, 2014, s. 69–88. ISBN 978-0-12-800744-0. Dostupné z: doi:10.1016/B978-0-12-800744-0.00005-1

[32] STALLINGS, William. *Cryptography and network security: principles and practice*. Seventh edition, global edition. Boston: Pearson, 2017. ISBN 978-1-292-15858-7.

[33] *The Vigenère Cipher Encryption and Decryption* [online]. [vid. 2024-03-11]. Dostupné z: https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Base.html

[34] LUGRIN, Thomas. One-Time Pad. In: *Trends in Data Protection and Encryption Technologies*. 2023, s. s. 3-6. ISBN 978-3-031-33385-9. Dostupné z: doi:10.1007/978-3-031-33386-6_1

[35] HOANG, Viet Tung a Phillip ROGAWAY. On generalized Feistel networks. In: *Annual Cryptology Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. s. 613-630. ISBN 978-3-642-14622-0. Dostupné z: doi:10.1007/978-3-642-14623-7_33

[36] KENEKAYORO, Patrick. The data encryption standard thirty four years later: An overview. *African Journal of Mathematics and Computer Science Research*. 2010, **3**(10), s. 267-269. ISSN 2006-9731.

[37] MENEZES, A. J., Paul C. VAN OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1997. CRC Press series on discrete mathematics and its applications. ISBN 978-0-8493-8523-0.

[38] ANTONIOS, Fiolitakis, Nikolaos PETRAKIS, Panagiotis MARGARONIS a Emmanuel ANTONIDAKIS. Hardware Implementation of Triple-DES Encryption/Decryption Algorithm [online]. 2006 [vid. 2023-04-18]. Dostupné z: https://www.researchgate.net/publication/228400978_Hardware_Implementation_of_Triple-DES_EncryptionDecryption_Algorithm

[39] VUPPALA, Akshitha, R Sai ROSHAN, Shaik NAWAZ a JVR RAVINDRA. An Efficient Optimization and Secured Triple Data Encryption Standard Using Enhanced Key Scheduling Algorithm. *Procedia Computer Science*. 2020, **171**, Third International Conference on Computing and Network Communications (CoCoNet'19), s. 1054-1063. ISSN 1877-0509. Dostupné z: doi:10.1016/j.procs.2020.04.113

[40] SOUSI, Ahmad-Loay, Dalia YEHYA a Mohamad JOUDI. *AES Encryption: Study & Evaluation* [online]. 2020 [vid. 2024-04-13]. Dostupné z: https://www.researchgate.net/profile/Mohamad-Joudi-2/publication/346446212_AES_Encryption_Study_Evaluation/links/5fd14d81299bf188d406c8d3/AES-Encryption-Study-Evaluation.pdf

[41] DEY, Sankhanil a Ranjan GHOSH. *Multiplication over Extended Galois Field: A New Approach to Find Monic Irreducible Polynomials over Galois Field GF(p^q).* 2017. Dostupné z: doi:10.7287/peerj.preprints.3258

[42] HOLDEN, Joshua. *A Simplified AES Algorithm* [online]. Institute of Technology. 2010. Dostupné z: https://www.rose-hulman.edu/class/ma/holden/Archived_Courses/Math479-0304/lectures/s-aes.pdf

[43] MCANDREW, A. *Introduction to cryptography with open-source software*. 2016. Introduction to Cryptography with Open-Source Software. ISBN 978-1-4398-2571-6.

[44] URBANOVÁ, Pavla. *Euclidean algorithm*. Plzeň, 2010. Diplomová práce. Západočeská Univerzita v Plzni.

[45] WANG, Zixing. Methods of Primality Testing [online]. 2021 [vid. 2024-04-13]. Dostupné z: https://www.researchgate.net/publication/348899360_Methods_of_Primality_Testing

[46] GAMBOA, Ruben a Woodrow GAMBOA. All Prime Numbers Have Primitive Roots. *Electronic Proceedings in Theoretical Computer Science*. 2022, **359**, 9–18. Dostupné z: doi:10.4204/EPTCS.359.3

[47] FAWAZ, Paraiso, Stephanie CHALLITA, Yahya AL-DHURAIBI a Philippe MERLE. Model-Driven Management of Docker Containers. In: *IEEE 9th International Conference on Cloud Computing (CLOUD)*. San Francisco: IEEE, 2016, s. s. 718-725. Dostupné z: doi:10.1109/CLOUD.2016.0100

[48] GUNTUPALLI, Ravi Chandra Chaitanya. *User Interface Design – Methods and Qualities of a Good User Interface Design*. Trollhättan, 2008. Diplomová práce. University West.

[49] Blowfish Algorithm with Examples. *GeeksforGeeks* [online]. 14. říjen 2019 [vid. 2024-03-01]. Dostupné z: https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/

[50] CrypTool Portal. *CrypTool Portal* [online]. [vid. 2024-03-28]. Dostupné z: https://www.cryptool.org/en/cto/

[51] YOUNG BILL. *Foundations of Computer Security - Lecture 42: A Perfect Cipher* [online]. University of Texas at Austin. [vid. 2024-04-13]. Dostupné z: https://www.cs.utexas.edu/~byoung/cs361/lecture42.pdf

[52] WYSS-GALLIFENT, Justin. DES and the Simplified DES Algorithm. *Department of Mathematics - University of Maryland* [online]. 2022 [vid. 2024-04-13]. Dostupné z: https://www.math.umd.edu/~immortal/ClassNotes/simplifieddes.pdf

[53] *A Python implementation of the stream cipher A5/1 algorithm* [online]. [vid. 2024-04-01]. Dostupné z: https://github.com/dixitaayush8/A5-1/tree/master

[54] *RSA Calculator* [online]. [vid. 2024-04-01]. Dostupné z: https://www.cs.drexel.edu/~popyack/IntroCS/HW/RSAWorksheet.html

[55] *Diffie Hellman Key Exchange Online Secret Calculator* [online]. [vid. 2024-04-01]. Dostupné z: https://www.dcode.fr/diffie-hellman-key-exchange

9 Přílohy

1) Kód aplikace: https://github.com/hejzlzd1/EducrypterUHK
2) Implementace aplikace: https://educrypter.uhk.cz
3) Formulář z fáze testování: https://forms.gle/s4WVh4rjetFtG7Cu6

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Zdeněk Hejzlar**
Studium:	I2200600
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Aplikační podpora výuky kryptografie s využitím PHP frameworku**
Název diplomové práce AJ:	Application support for teaching cryptography using the PHP framework

Cíl, metody, literatura, předpoklady:

Cílem práce je navrhnout webovou aplikaci pro podporu výuky kryptografie.

V teoretické části student stručně popíše základní principy fungování vybraných kryptografických primitivů a vypracuje literární rešerši jejich využití v praxi. V praktické části navrhne a implementuje webovou aplikaci, kde bude uživatel mít možnost vyzkoušet si vybrané kryptografické primitivy s různými vstupními parametry a zjistit, jak probíhají jednotlivé fáze jejich výpočtů.

Doporučená osnova:

1) Úvod

2) Popis vybraných kryptografických primitivů (včetně odůvodnění, proč byly vybrány tyto primitivy)

3) Využití kryptografických primitivů v praxi (literární rešerše)

4) Výběr a popis PHP frameworku a dalších nástrojů použitých při implementaci

5) Návrh aplikace a její implementace

6) Testování aplikace

7) Závěr (včetně diskuze nad možnostmi využití aplikace v praxi)

Bruce Schneier: Applied cryptography: protocols, algorithms and source code in C. Wiley Publishing, New York, 1996.

K. Burda, Kryptografie okolo nás. 2019. (https://knihy.nic.cz/files/edice/Kryptografie_okolo_nas.pdf)

A. J. Menezes, J. Alfred, P. C. Van Oorschot, S. A. Vanstone. Handbook of applied cryptography. CRC press, 2018. (http://labit501.upct.es/ fburrull/docencia/SeguridadEnRedes/old/teoria/bibliography/HandbookOfAppliedCryptography_AMenezes.pdf)

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Jana Medková, Ph.D.
Datum zadání závěrečné práce:	26.1.2021