VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Martin Tatar

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

WEBOVÁ APLIKACE PRO ŠIFROVÁNÍ SOUBORŮ
WEB APPLICATION FOR FILE ENCRYPTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Martin Tatar

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Václav Zeman, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Martin Tatar ***ID:*** 211815

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Webová aplikace pro šifrování souborů

POKYNY PRO VYPRACOVÁNÍ:

Práce je zaměřena na analýzu moderních symetrických šifrovacích algoritmů. V teoretické části práce bude provedeno srovnání vlastností standardizovaných i nově používaných symetrických šifrovacích algoritmů včetně rozboru operačních módů a popisu možností implementace. Na základě uvedeného rozboru bude navržena a realizována webová aplikace pro šifrování souborů.

DOPORUČENÁ LITERATURA:

[1] Ferguson, Niels, Schneier, Bruce. Practical Cryptography. Wiley, 2003.

[2] Menezes, Alfred J., Oorschot, Paul C. van, Vanstone, Scott A.. Handbook of Applied Cryptography. 2001.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá vytvořením webové aplikace pro šifrování souborů. V teoretické části jsou symetrické šifrovací algoritmy rozděleny na blokové a proudové. Vybrané šifry jsou popsány a jejich vlastnosti srovnány. Dále jsou rozebrány operační módy využívané blokovými šiframi. Vytvořená aplikace šifruje jak soubory, tak vložený text pomocí několika volitelných symetrických šifrovacích algoritmů a umí operovat v různých módech. Kromě této praktické funkcionality je aplikace doplněna i popisky využitých šifer a operačních módů.

KLÍČOVÁ SLOVA

symetrické šifrovací algoritmy, operační módy, bloková šifra, proudová šifra, šifrování, dešifrování, otevřený text, šifrový text

ABSTRACT

The bachelor thesis is focused on developing a web application for file encryption. In the theoretical part symmetric encryption algorithms are divided into block ciphers and stream ciphers. Selected ciphers are described and their properties are compared. Then the modes of operation for block ciphers are described. The developed application encrypts both files and text inputs by the selected algorithm and can operate in various modes of operation. In addition to this functionality the application is supplemented with descriptions of available ciphers and modes of operation.

KEYWORDS

symmetric encryption algorithms, modes of operation, block cipher, stream cipher, encryption, decryption, plaintext, ciphertext

TATAR, Martin. *Webová aplikace pro šifrování souborů*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 54 s. Bakalářská práce. Vedoucí práce: doc. Ing. Václav Zeman, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Martin Tatar
VUT ID autora:	211815
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Webová aplikace pro šifrování souborů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Václavu Zemanovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Kryptografie obecně se začala využívat už před několika tisíci lety a lidstvo provází nadále, avšak prošla velkými změnami a v dnešní době je mnohem komplexnější a propracovanější.

Jako první známější šifra se dá uvést Caesarova šifra. Tu využíval Julius Caesar k šifrování zpráv při svých vojenských taženích. Z dnešního pohledu je to velice jednoduchá šifra, která zpravidla používá pouhý posun o tři místa v abecedě, avšak je možné použít posun o libovolný počet míst.

S postupem času se kryptografie vyvíjela a své velké uplatnění měla v první i druhé světové válce. Důležité informace se nesměly dostat k nepříteli, takže bylo nutno využívat šifrovanou komunikaci. Na druhé straně se nepřátelé snažili šifry rozluštit, aby zjistili kritické informace, které by jim pomohly zvítězit. Z tohoto období je nejznámější šifra nazvaná Enigma. Oproti Caesarově šifře se Enigma prováděla na stroji a byla o poznání složitější, ale i tak byla rozluštěna.

V pozdějších letech vznikla poptávka po nových, spolehlivých šifrovacích algoritmech. Díky tomu vznikla první standardizovaná šifra DES (Data Encryption Standard), která však byla později taky prolomena a nahrazena novější šifrou AES (Advanced Encryption Standard). Mimo tyto standardizované šifrovací algoritmy vzniklo i několik dalších, hojně užívaných algoritmů. Například asymetrický algoritmus RSA (iniciály autorů, Rivest, Shamir, Adleman), který je využíván dodnes a je vhodný jak pro šifrování, tak pro podepisování. Vlivem rozvoje informačních a telekomunikačních technologií dochází k přenosu čím dál většího množství dat. Ať už jde o využívání internetového bankovnictví, online nakupovaní a placení, nebo o komunikaci mezi uživateli, je žádoucí, aby data byla šifrována a zamezilo se jejich zneužití případnými útočníky. Společně s rozšířeným přístupem k Internetu se kryptografie stala nedílnou součástí života mnoha lidí.

Tato bakalářská práce se zabývá moderními symetrickými šifrovacími algoritmy. Zaměřuje se jak na standardizované, tak nově používané symetrické šifrovací algoritmy a jejich srovnání. Dále se věnuje operačním módům, používaným u blokových šifer. Na základě teoretické části je realizována webová aplikace pro šifrování souborů.

1 Symetrické šifrovací algoritmy

Symetrická kryptografie využívá symetrické šifrovací algoritmy k šifrám, které používají stejný tajný klíč k šifrování i dešifrování dat. Pokud je tedy třeba zašifrovat otevřenou zprávu p symetrickou šifrou, použije se klíč k k získání zašifrovaného textu c. K opětovnému získání zprávy p se opět použije stejný klíč k. Symetrické šifry jsou opakem asymetrických šifer používaných v asymetrické kryptografii, které používají pár klíčů, tvořený veřejným klíčem k šifrování, který, jak z názvu vyplývá, je veřejný a soukromého klíče k dešifrování, který je tajný. Jedna z výhod symetrických šifer oproti asymetrickým je jejich výrazně vyšší rychlost. Naopak nevýhodou symetrických šifer je distribuce tajného klíče. [1]

1.1 Blokové šifry

Bloková šifra data šifruje po blocích, které mají určenou fixní velikost n bitů. Bloková šifra má tedy jako vstup blok otevřeného textu o dané délce n bitů a výstupem je stejně velký n bitový blok textu šifrového.

1.1.1 Velikost bloku

Přestože bloková šifra může operovat s různou velikostí bloku, je vhodné se vyvarovat příliš krátkým blokům. Krátké bloky jsou náchylné k útokům. Při použití krátkého bloku je možno využít slovníkový útok.

Na druhou stranu příliš velké bloky také nejsou žádoucí. Před započetím šifrování je potřeba načíst celý blok, čímž může docházet ke zpoždění. Další nevýhodou velkých bloků je neefektivita, jelikož bloková šifra operuje pouze s celými bloky. To v praxi znamená, že se zpráva na danou velikost bloku musí doplnit výplní, tzv. „paddingem“. Nejčastěji využívané jsou bloky o velikosti násobků 8, konkrétně 64b a 128b.[2]

1.2 Proudové šifry

Na rozdíl od blokové šifry, proudová šifra otevřený text šifruje po 1 bitu, nebo 1 bytu. Využívá k tomu generátor náhodného proudu bitů, který nesmí být předvídatelný, aby nedošlo k prolomení šifry. Zároveň se stejný klíč nesmí použít dvakrát.[3] Proudová šifra nejběžněji používá operaci XOR. Pokud je tedy otevřený text potřeba zašifrovat, použije se proud pseudonáhodných bitů a v kombinaci s otevřeným textem se operace XOR provede. Tímto se získá šifrový text. Ke zpětnému dešifrování se operace XOR provede na šifrový text s klíčem.

Výhodou proudové šifry oproti blokové je její vyšší rychlost. Své uplatnění proudové šifry najdou například v zařízeních s omezeným hardwarem, kde není možné načítat větší bloky najednou.

1.2.1 Synchronní proudové šifry

Aby synchronní proudová šifra fungovala, musí být přenos synchronizován, jelikož jednotlivé znaky spolu korespondují a záleží tak na jejich pořadí. Pokud je tedy během procesu přidán, nebo odebrán nějaký bit, dojde ke ztrátě synchronizace. V tomto případě bude dešifrování neúspěšné. K napravení tohoto stavu se používají různé způsoby. Jedním z nich je značkování šifrového textu v pravidelných intervalech. Dalším způsobem je systematické zkoušení různých posunů k získání správného dešifrování. Na druhou stranu, pokud dojde u nějakého bitu k chybě, ale žádný není přidán, ani ztracen, tak se chyba nebude dál šířit a zbytek zprávy zůstane neporušen. Tato vlastnost je užitečná pro přenosy, ve kterých často dochází k chybám. Vede to ovšem i ke slabině, které může útočník využít, pokud se mu podaří měnit konkrétní bity v šifrovém textu a pozorovat, jak se projeví změnou v otevřeném textu.[4]

1.2.2 Asynchronní proudové šifry

Dalším typem proudových šifer jsou asynchronní proudové šifry. Asynchronní proudová šifra využívá fixní délku n předchozích bitů šifrového textu ke spočítání proudu bitů používaného jako klíč. Tento typ proudové šifry má výhodu v tom, že po obdržení nutné délky šifrového textu se sám synchronizuje s generátorem proudového klíče. To usnadňuje nápravu při ztrátě, nebo přidání bitů. Pokud k tomu totiž dojde, bude ovlivněna pouze část zprávy o maximální délce n.[4]

2 Srovnání šifrovacích algoritmů

V této kapitole budou popsány a srovnány standardizované i nově používané symetrické šifrovací algoritmy.

2.1 DES

Data Encryption Standard je symetrická, bloková šifra, založená na Feistelově síti, využívající bloky o velikosti 64 bitů a 64 bitový klíč, ze kterého je ovšem 8 bitů paritních, což zanechává efektivní délku klíče na 56 bitech. Blok p, který má být zašifrován nejprve projde počáteční permutací IP. Poté je blok rozdělen na 2 stejně velké části L_0 a R_0, obě o velikosti 32 bitů. Pravá část R_0 je poté vedena do funkce f, kde je provedena expanzní funkce a operace XOR s rundovním klíčem k_i. Výstup z funkce f je pomocí substituce opět zredukován na 32 bitů a jde do operace XOR s levou částí L_0. Výstup z toho se stává příští pravou částí R_1 a původní část R_0 se stává novou levou částí L_1.

Takto DES probíhá v šestnácti rundách a na konci dochází ke konečné permutaci IP^{-1}, která je inverzní k permutaci počáteční.[5] Schéma DES je zobrazeno na obrázku 2.1.

Dešifrování probíhá stejně, ale rundovní klíče jsou použity v obráceném pořadí.

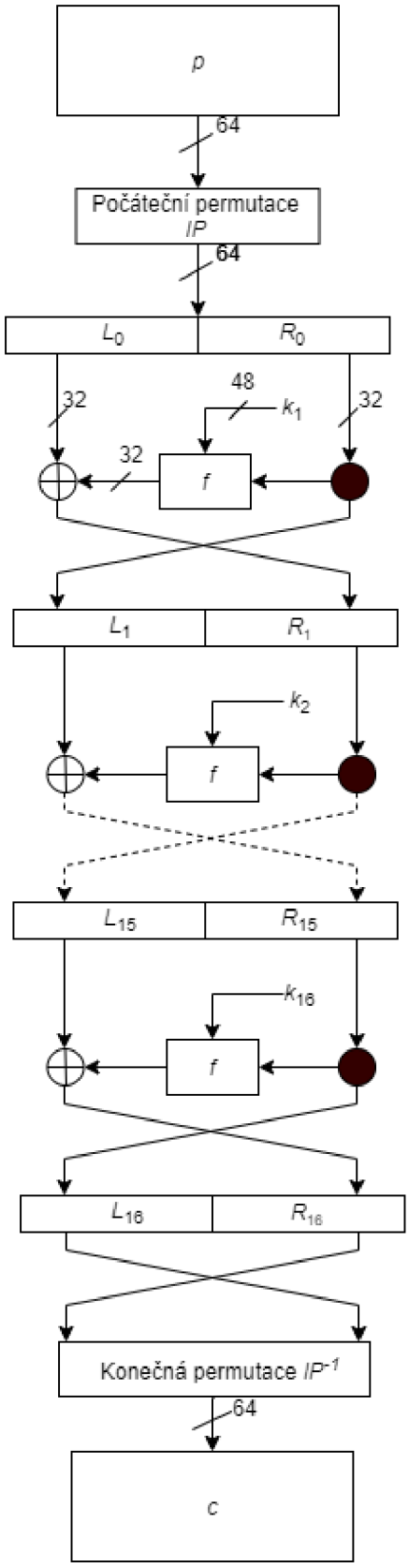
2.1.1 Využití DES

V dnešní době je DES již zastaralý a kvůli krátkému klíči nebezpečný. Již v roce 1998 byl sestrojen „DES-cracker“, pomocí kterého byl klíč hrubou silou rozluštěn za přibližně dva dny.

Od té doby je prolomení DES ještě snazší záležitostí díky vyššímu výkonu dnešních počítačů. To znamená, že DES je prolomitelný za pár hodin.[6]

2.1.2 3DES

Jelikož byl DES prolomen kvůli svému krátkému klíči, byl vymyšlen způsob jak se proti tomuto bránit, aniž by byl algoritmus změněn. 3DES je tedy uměle zesílený DES, tím že DES proběhne třikrát. Tím se získá efektivní délka klíče 168 bitů. Nevýhoda 3DES je, že už DES je ve srovnání s novějšími algoritmy považován za pomalý. Tím pádem 3DES trvá třikrát déle.



Obr. 2.1: Schéma DES

2.2 AES

Po prolomení šifry DES vyhlásil NIST (National Institute of Standards and Technology) konkurz na jejího nástupce. Později byla vybrána šifra *Rijndael* jako nový Advanced Encryption Standard. AES je symetrická, bloková šifra, která má velikost bloku danou jako 128 bitů a délku klíče volitelnou 128, 192, nebo 256 bitů. V závislosti na délce klíče operuje v 10, 12, nebo 14 rundách. AES je bytově orientovaný a pracuje s maticemi 4x4, tedy 16 byty. Používány jsou operace nad Galoisovým tělesem $GF(2^8)$.

2.2.1 Šifrování

Pomocí AES *key schedule* jsou ustanoveny rundovní klíče. Poté na začátku dochází k *AddRoundKey* - XOR operace vstupu s klíčem.

Dále v 9, 11, nebo 13 rundách (podle délky klíče) dochází k těmto krokům:

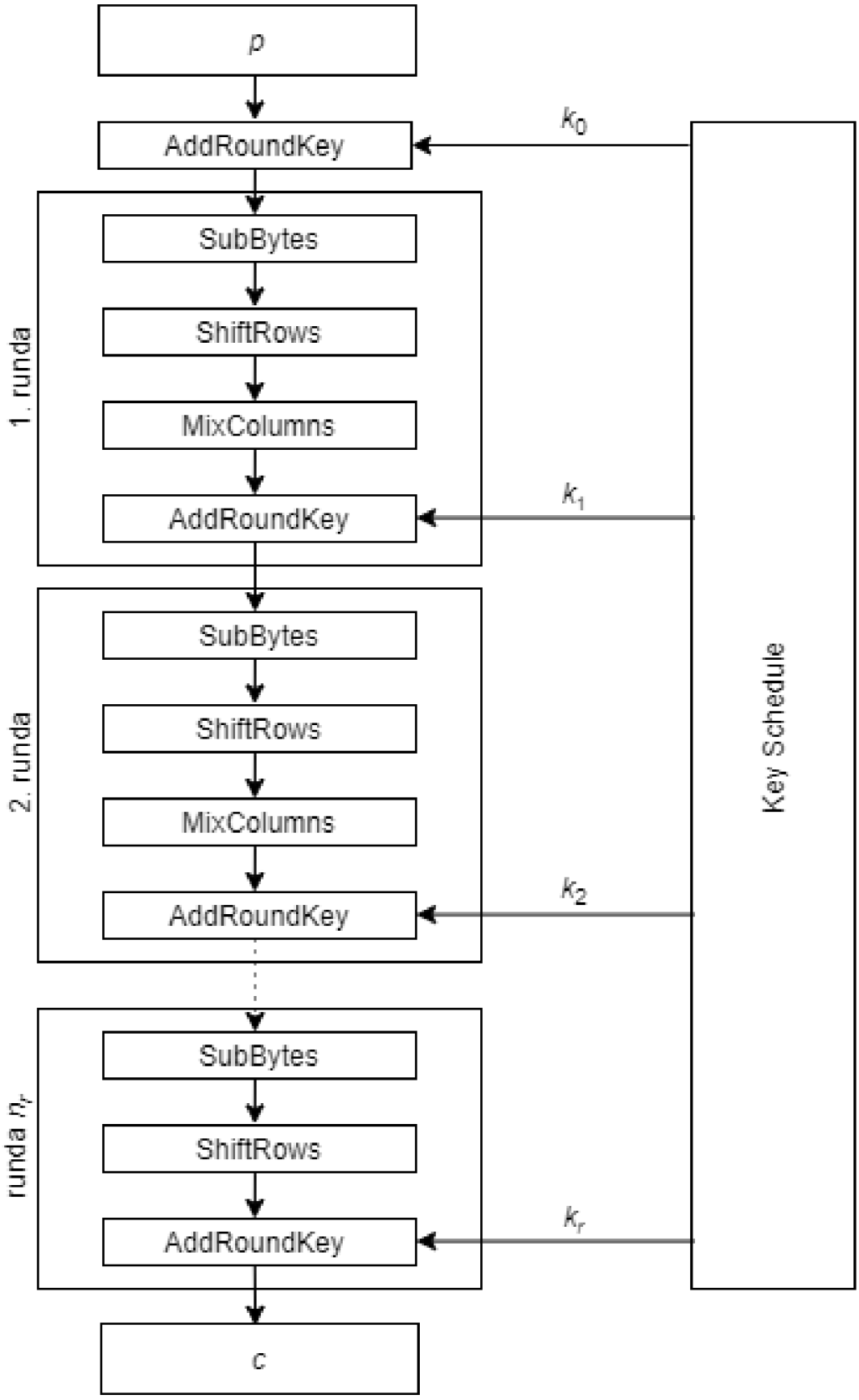
a) *SubBytes* - nelineární substituce, kde je každý byte zaměněn s příslušným bytem z vyhledávací tabulky.
b) *ShiftRows* - rotace řádků v matici. První řádek je ponechán, druhý rotuje o jednu pozici, třetí o dvě pozice a čtvrtý o tři pozice, vždy vlevo.
c) *MixColumns* - násobení maticí modulo polynom $m(x) = x^8+x^4+x^3+x+1$.
d) *AddRoundKey*

Poslední runda (desátá, dvanáctá, nebo čtrnáctá) probíhá stejně, ale vynechává se *MixColumns*. Průběh je zobrazen na obrázku 2.2.

2.2.2 Dešifrování

Na rozdíl od DES nelze AES dešifrovat pouhým prohozením pořadí rundovních klíčů, avšak je to také aplikováno, konkrétně při *AddRoundKey*. Ke každému kroku dešifrování tedy potřebuje inverzní funkci. Ty jsou:

a) *Inv SubBytes* - k tomu slouží inverzní tabulka
b) *Inv ShiftRows* - první řádek opět bez rotace, druhý rotuje o jednu pozici, třetí o dvě pozice a čtvrtý o tři pozice, tentokrát však doprava.
c) *Inv MixColumns* - násobení inverzní maticí.
d) *AddRoundKey* - díky aplikaci operace XOR je sama sobě inverzní funkcí.[7] [8]



Obr. 2.2: Schéma AES

2.3 Present

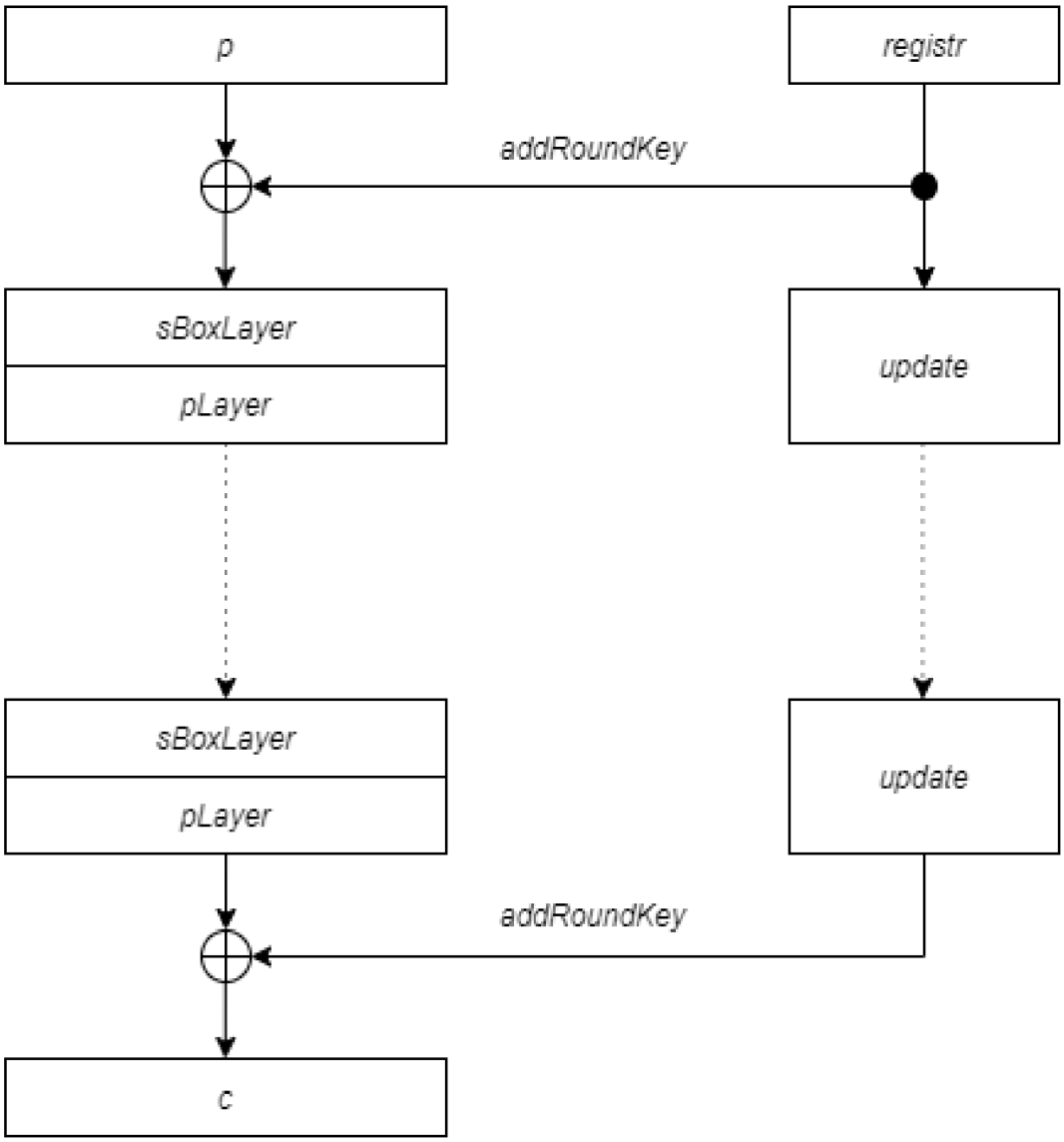
Symetrická, bloková šifra Present byla navržena pro zařízení s omezeným výpočetním výkonem. Díky své optimalizované hardwarové implementaci se může rovnat i proudovým šifrám, které často bývají využívané právě na takových zařízeních, kde je potřeba šetřit dostupné zdroje. Present je založena na substitučně permutační síti. Používá bloky o velikosti 64 bitů, délka klíče je volitelná buď 80 bitů, nebo 128 bitů a probíhá ve 31 rundách. Jelikož jsou rundovní klíče využity jak na začátku, tak na konci, je jich potřeba 32 a jsou odvozeny od hlavního klíče daným postupem.

Dále Present využívá tyto operace:

a) *addRoundKey* - přidání příslušného rundovního klíče.
b) *sBoxLayer* - substituce 4 bitů na 4 bity podle dané tabulky.
c) *pLayer* - bitová permutace opět dle dané tabulky.
d) *Key schedule* - hlavní klíč od uživatele je uložen v registru K. Rundovní klíč k_i je stanoven jako prvních 64 bitů registru zleva. Poté co je rundovní klíč k_i využit, registr se aktualizuje a bude k dispozici následující klíč.

2.3.1 Využití

Tato šifra má velmi specifické využití. Vzhledem k rozšířenosti šifry AES se Present zaměřuje na zařízení, kde právě AES není vhodný. Tato zařízení by se měla spokojit s úrovní bezpečnosti jakou nabízí 80 bitový klíč, měla by šifru implementovat hardwarově a nepředpokládat šifrování velkých objemů dat. Šifra je zároveň vhodná do prostředí, kde je potřeba nízké spotřeby energie. Díky své jednoduchosti, 64 bitovým blokům a při využití 80 bitového klíče je pomocí Present šifrování i dešifrování jednodušší než pouhé šifrování pomocí AES.[9] [10]



Obr. 2.3: Schéma Present

2.4 IDEA

IDEA (International Data Encryption Algorithm) je symetrická, bloková šifra založená na Lai-Massey systému, který je podobný Feistelově síti. IDEA šifruje 64 bitové bloky a využívá 128 bitový klíč. Operuje v 8 rundách, po kterých dochází ještě k výstupní transformaci.

2.4.1 Šifrování

Proces šifrování zahrnuje:

a) *Key schedule* - 128 bitový klíč k je rozdělen na 8 podklíčů o délce 16 bitů. Pro vygenerování dalších 8 podklíčů se hlavní klíč k posune o 25 bitů doleva.

Takto je potřeba vygenerovat 52 podklíčů k_i, jelikož každá runda jich využívá 6 a výstupní transformace 4.

b) *Modulární součet* - součet modulo 2^{16}.

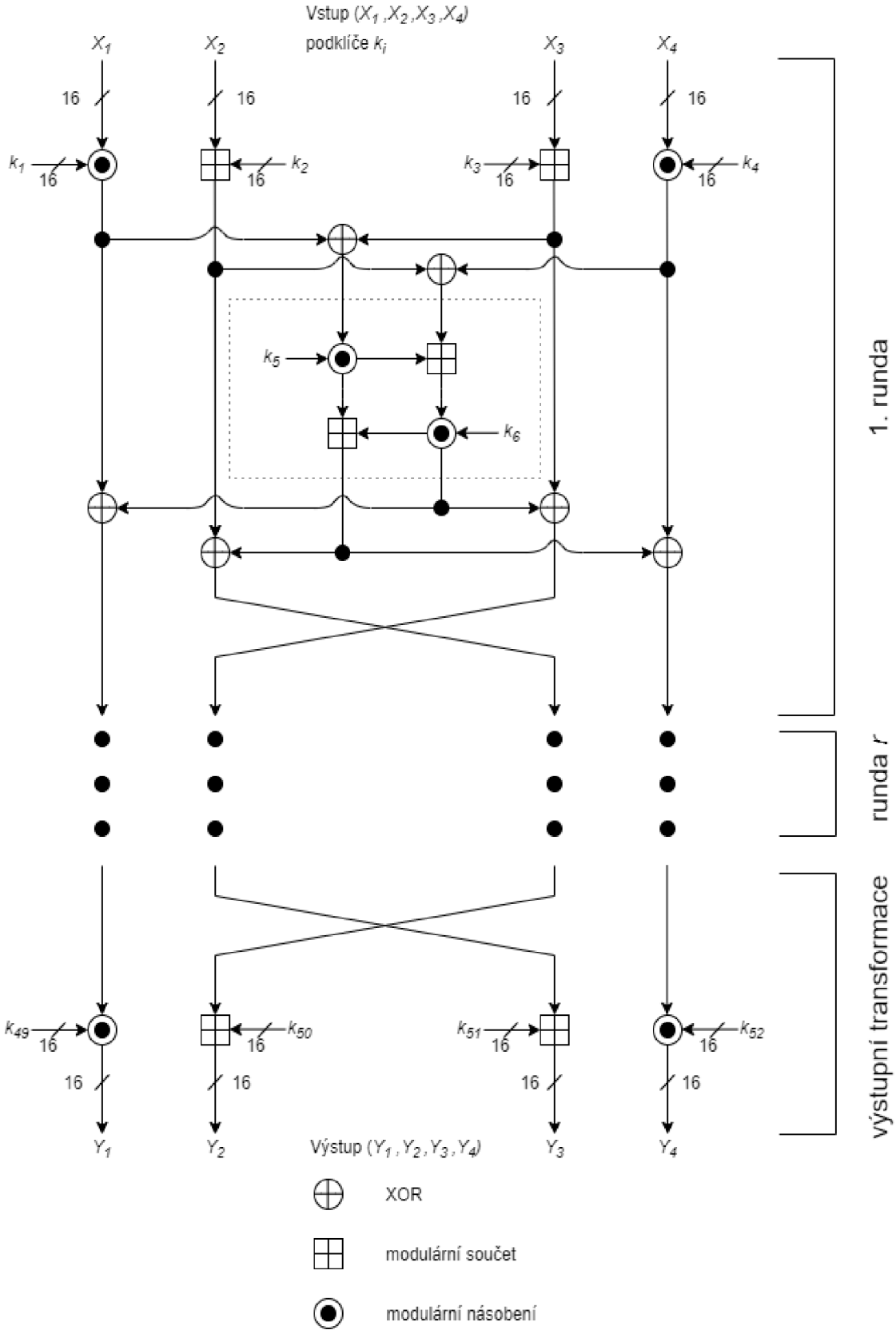
c) *Modulární násobení* - modulo $2^{16} + 1$.

Vstupní 64 bitový blok X se rozdělí na 4 podbloky (X_1, X_2, X_3, X_4), každý o délce 16 bitů. Poté dochází modulárnímu násobení X_1 s k_1, modulárnímu součtu X_2 s k_2 a X_3 s k_3, modulárnímu násobeníX_4 s k_4. Výsledky těchto operací se dál označí ve stejném pořadí jako $\underline{X_1}$, $\underline{X_2}$, $\underline{X_3}$ a $\underline{X_4}$. Dále následuje operace XOR $\underline{X_1}$ s $\underline{X_3}$ a XOR $\underline{X_2}$ s $\underline{X_4}$. Tím vznikne $\underline{X_{13}}$ a $\underline{X_{24}}$. $\underline{X_{13}}$ je poté opět modulárně násobeno s k_5, čímž vzniká $X_{13}k_5$, to je modulárně sečteno s $\underline{X_{24}}$. Z toho vycházející $X_{1234}k_5$ se modulárně násobí s k_6, z čehož je $X_{1234}k_5k_6$. $\underline{X_{24}}$ modulárně sečteno s $X_{13}k_5k_6$ se rovná $X_{1234}k_{123456}$. $X_{1234}k_{123456}$ jde poté do operace XOR s $\underline{X_2}$ a $\underline{X_4}$, zatímco $\underline{X_1}$ a $\underline{X_3}$ s $X_{1234}k_5k_6$.

Během tohoto procesu, což byla jedna runda, tedy byly promíchány všechny podbloky s šesti podklíči. Výstupem jsou opět 4 podbloky, které pokračují v další rundě s novými podklíči. Celý proces je zobrazen také na obrázku 2.4.

2.4.2 Dešifrování

Dešifrování využívá stejný postup, ale podklíče jsou užity v obráceném pořadí.[11] [12]



Obr. 2.4: Schéma IDEA

2.5 Blowfish

Blowfish je symetrická, bloková šifra, kterou v roce 1993 vydal Bruce Schneier jako volné dílo. Díky tomu byla vystavena rozsáhlému testování a dodnes nebyla nalezena žádná účinná kryptoanalytická metoda k jejímu prolomení. Vydána byla i jako náhrada ze nedostačující DES a zaujala svou rychlostí.

Blowfish využívá bloky o délce 64 bitů. Délka klíče je variabilní od 32 bitů po 448 bitů. Operuje v 16 rundách a je založena na Feistelově síti. K tomu ještě využívá S-boxy, které jsou závislé na klíči.

2.5.1 Šifrování

Před začátkem šifrování Blowfish potřebuje předpočítat podklíče k_i pomocí daného *key schedule*, které se uloží do pole P. Podobně jako DES, také Blowfish je založen na Feistelově sítí a vstupní blok je tedy rozdělen na dvě poloviny L a R o velikosti 32 bitů.

Každá runda poté obsahuje tyto 4 kroky:

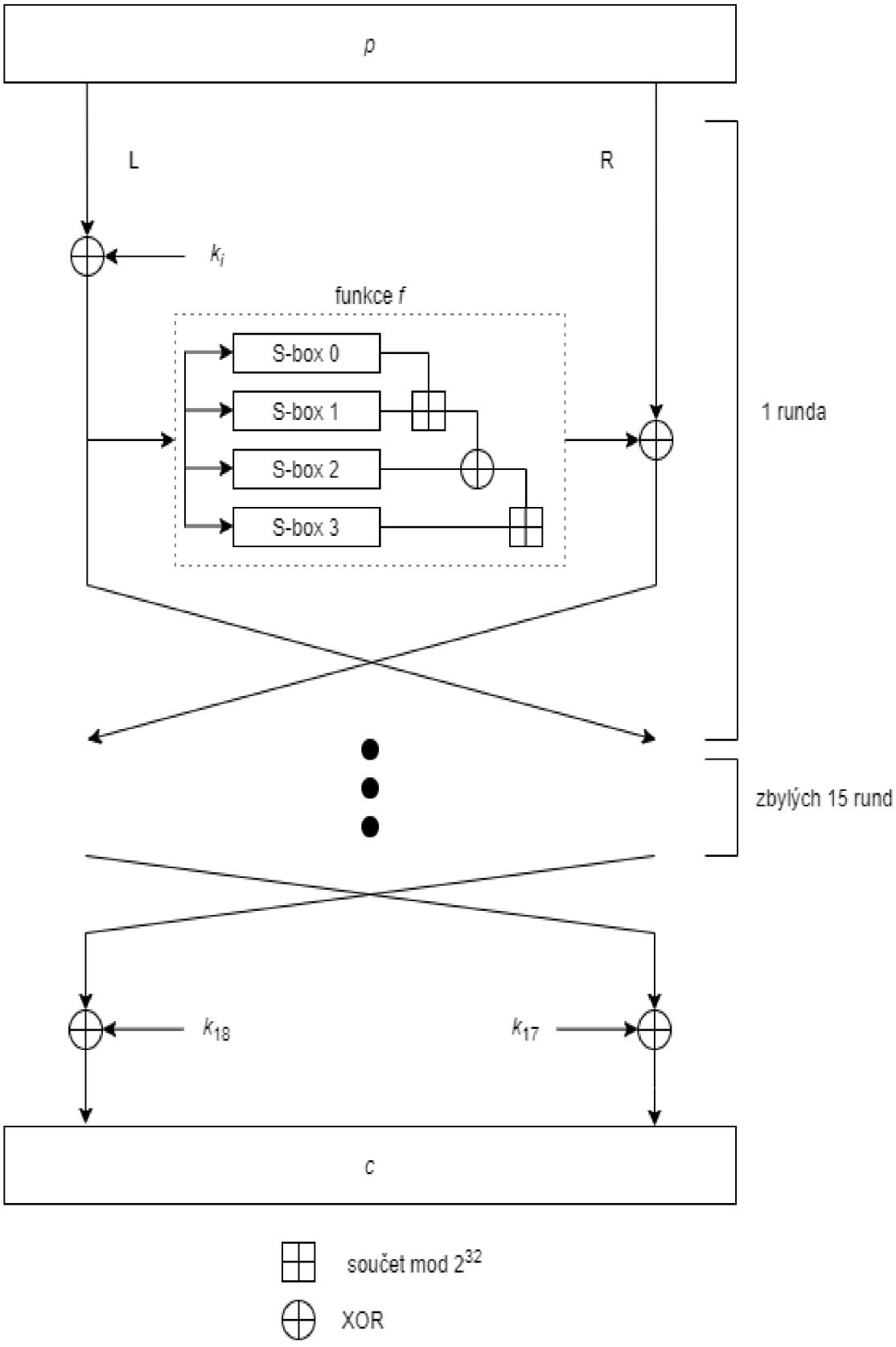
a) XOR poloviny L s k_i.
b) Výstup z předchozího kroku se použije jako vstup pro funkci f, která obsahuje S-boxy.
c) XOR výstupu funkce f s polovinou R.
d) Prohození L a R.

Takto probíhá 16 rund. Na konci jsou poloviny opět prohozeny a jsou použity jako vstupy do operace XOR s k_{17} a k_{18}. Poloviny se opět spojí a výsledkem je 64 bitů šifrového textu c.

Schéma celého procesu je zobrazeno na obrázku 2.5.

2.5.2 Dešifrování

Díky své struktuře Blowfish provádí dešifrování stejným způsob jako šifrování. Jediná změna je že se podklíče k_i použijí v opačném pořadí.[13] [14]



Obr. 2.5: Schéma Blowfish

2.6 Twofish

Šifra Twofish navazuje na Blowfish a byla jedním z kandidátů na Advanced Encryption Standard. Ten však nakonec vzešel z šifry *Rijndael*.

Twofish je symetrická, bloková šifra, která využívá bloky o délce 128 bitů a klíč má volitelnou délku 128 bitů, 192 bitů, nebo 256 bitů. Založena je na Feistelově síti a na rozdíl od AES nezávisle na délce klíče operuje 16 rund. Tato šifra je závislá na klíči, který ovlivňuje její průběh. Polovina klíče o délce n je použita jako šifrovací klíč, zatímco druhá polovina n bitů je použita k vypočítání obsahu tabulek, které se používají pro S-boxy. Toto je použito k znesnadnění analýzy S-boxů případným útočníkem. Zároveň je to důvod proč implementace Twofish předpočítávají klíče. Poté se předpočítají S-boxy a uloží se do paměti.

2.6.1 Struktura

Vstupní blok p o velikosti 128 bitů rozdělí na 4 části po 32 bitech. Ty jsou spolu s rundovními klíči použity při operaci XOR.

Každá runda poté obsahuje:

a) Rundovní funkci *f*. Ta obsahuje dvě totožné funkce g, funkci PHT a přičtení klíče.

b) Funkce g obsahuje 4 S-boxy, jejichž výstupy pokračují do MDS (Maximum Distance Separable), přičemž dochází k lineární transformaci.

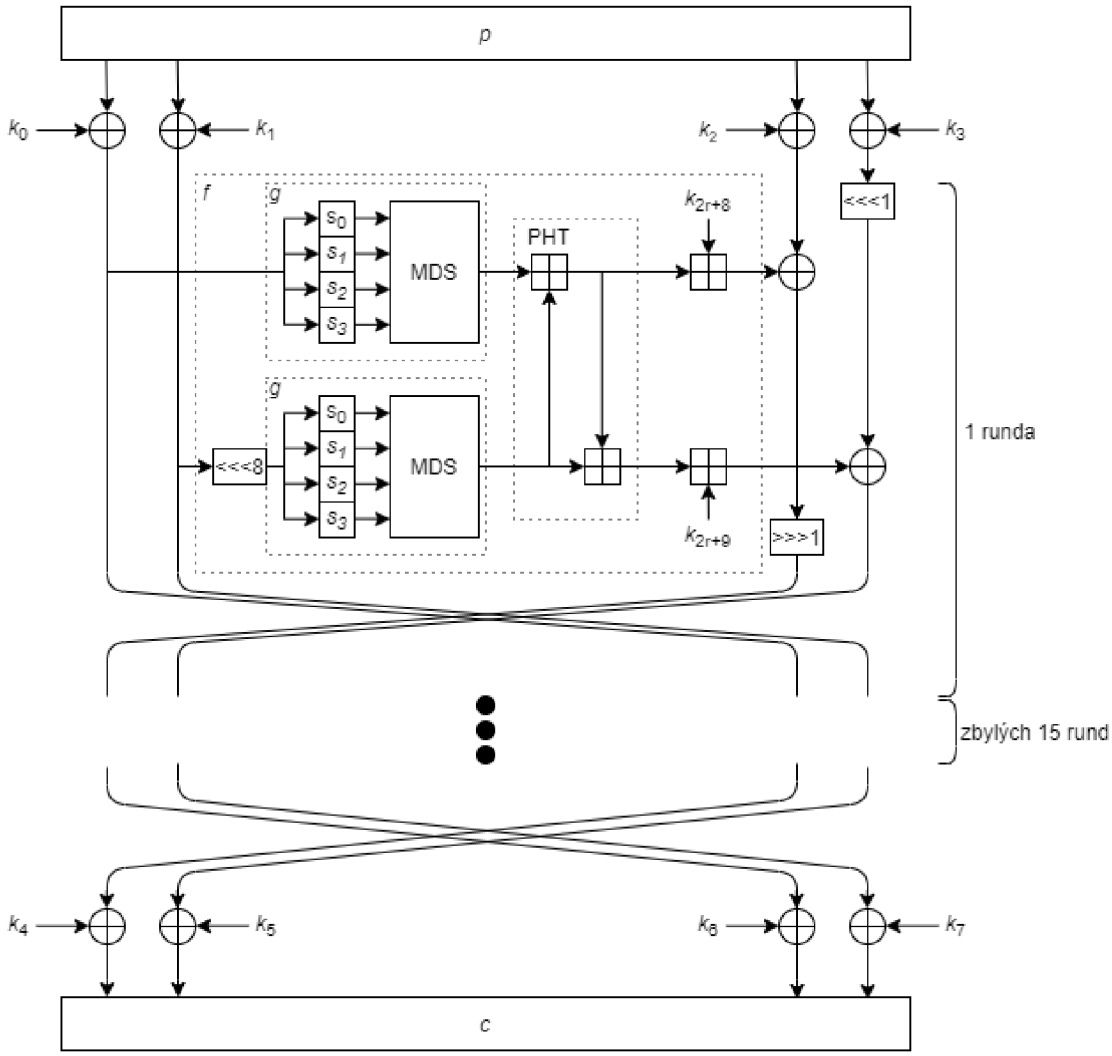
c) PHT (pseudo-Hadamard transform) - další transformace, napomáhá difusi. Byla použita také v rodině šifer *SAFER*. V šifře Twofish přijímá výstupy z obou funkcí g a provádí s nimi 32 bitovou součtovou operaci.

d) XOR

e) <<< a >>> značí rotaci 32 bitových částí, která je specifikovaná číslem značícím o kolik bitů má být rotace provedena.

Na konci je opět využita operace XOR, po níž se 4 části opět spojí a společně tvoří 128 bitový výstup c.

Struktura Twofish je také zobrazena na obrázku 2.6.[15] [16]



Obr. 2.6: Schéma Twofish

2.7 Serpent

Další finalista ve výběru nového Advanced Encryption Standard. Serpent navrhli Ross Anderson, Eli Biham a Lars Knudsen.

Při návrhu šifry Serpent byl kladen důraz hlavně na bezpečnost. Zatímco AES byl navržen s přihlédnutím k výkonnosti, Serpent si raději nechal rezervu. Příkladem tohoto je 32 rund ve kterých Serpent operuje.

Výsledkem byl kvalitní spolehlivý šifrovací algoritmus, avšak pozitiva Rijndael převážila. Robustní design Serpent vedl tomu, že dosahoval přibližně pouze třetinové rychlosti Rijndeal.

2.7.1 Struktura

Serpent je symetrická, bloková šifra. Velikost bloku je 128 bitů. Velikost klíče je volitelná mezi 128 bitů, 192 bitů, nebo 256 bitů.

Založen je na substitučně-permutační síti a operuje v celkem 32 rundách.

Vstupní blok o velikosti 128 bitů je rozdělen na 4 stejně velké části. Každá runda využívá jeden z osmi S-boxů. S-box zpracovává 4 bity na 4 bity, provádí tedy paralelně 32 operací. Další z osmi S-boxů je využit v příští rundě.

Serpent je navržen, tak aby operace probíhaly paralelně.

K implementaci je možno využít softwarový trik, který celý proces významně urychlí. Jelikož má Serpent 32 rund a v každé využívá 32 S-boxů dochází k celkem 1024 vyhledávání v tabulce. Aby to nebylo prováděno postupně, což by bylo pomalé, tak se S-boxy převedou na boolean vzorce. Výstupní 4 bity jsou přepsány jako boolean vzorec 4 vstupních bitů.

Díky tomuto potom procesor vyhodnocuje přímo pomocí instrukcí *AND, OR, XOR*. Tomuto způsobu se říká *„bitslice implementation“*.[15] [17]

2.8 RC4

Navržena roku 1987 a únikem zveřejněna je proudová šifra RC4, někdy také označována *ARCFOUR*, nebo *ARC4* kvůli již existující ochranné známce. Jejím tvůrcem je Ron Rivest, který se také podílel na navržení známého asymetrického algoritmu RSA. RC4 zaujala obzvláště svou rychlostí, v porovnání s blokovými šiframi je dvakrát, třikrát rychlejší. Šifrování i dešifrování probíhá pomocí operace XOR.

2.8.1 Popis

Pro svůj chod potřebuje RC4 *keystream* - vygenerovaný pseudonáhodný bitový proud.

K jeho vygenerování používá tajný vnitřní stav, který se skládá ze dvou částí:

a) Permutaci všech 256 bytů, značenou S.
b) Dvou 8 bitových ukazatelů i a j.

Permutace je zahájena pomocí volitelného klíče s délkou mezi 40 a 2048 bity, k čemuž využívá algoritmus KSA, který naplánuje klíč. Ten se cyklicky zapisuje do pole K.

KSA

Tento algoritmus se využívá k zahájení permutace v poli S. Délka klíče k se značí v bytech a může nabývat hodnot od 1 do 256. Nejčastěji se však využívá délka 5 až 16 bytů. Pole S je inicializováno permutací a poté proběhne 256 iterací.

Pro i od 0 do 255, j = 0, j = (j + S[i] + K[i]) mod 256. Poté se prohodí hodnoty S[i] a S[j].

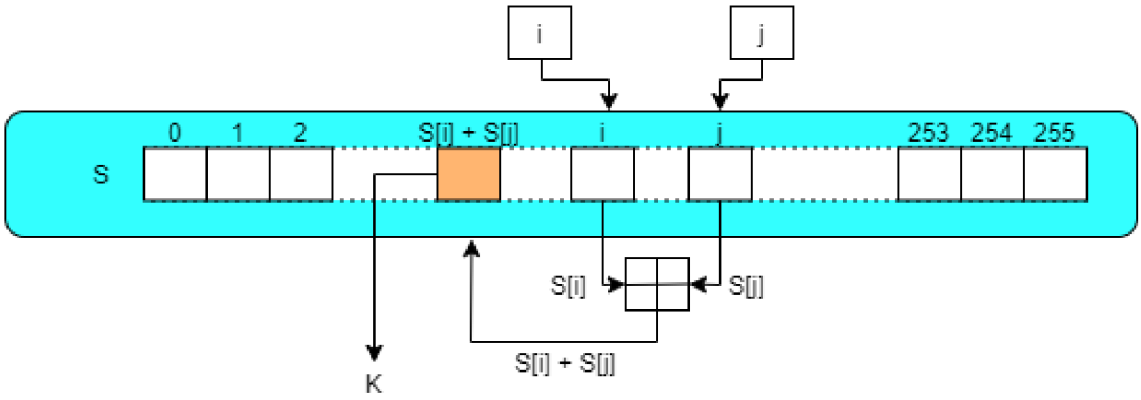
PRGA

Podle potřeby tento algoritmus provádí iterace, při kterých mění stav a produkuje byte proudu klíče.

Při každé iteraci provádí:

a) Inkrementuje i.
b) S[i] přičte k j.
c) Vymění hodnoty S[i] a S[j], poté použije S[i] + S[j] mod 256 k získání třetího prvku K z pole S.
d) XOR s dalším bytem zprávy.

Každý prvek S je vyměněn aspoň jednou za 256 iterací. PRGA je také zobrazen na obrázku 2.7.



Obr. 2.7: Nalezení K

2.8.2 Využití

RC4 našel využití v mnoha protokolech. Jmenovitě například WEP, WPA, Microsoft Office XP, volitelně v protokolech TLS, SSH, nebo Kerberos. V dnešní době se však RC4 už nedoporučuje využívat, vyzvaly k tomu společnosti Microsoft, Mozilla, či organizace IETF.[18] [19]

2.9 Srovnání

Výše popsaných 9 symetrických šifrovacích algoritmů je zobrazeno v tabulce 2.1.

Ke každé šifře je zobrazen její typ - tedy bloková, nebo proudová, struktura na které je založena, délka klíče v bitech, velikost bloku v bitech a počet rund ve kterých operuje.

Nejužívanější je Advanced Encryption Standard, tedy AES, avšak většina z nich se stálé využívá. Například Present najde své uplatnění v zařízeních s omezenou kapacitou. Twofish i Serpent jsou stále dobré šifrovací algoritmy, avšak na AES ztrácejí co se rychlosti týče.

IDEA je při správné implementaci také stále bezpečná k užití, avšak podobně jako Twofish a Serpent ztrácí na AES a tudíž k jejímu použití není mnoho důvodů.

Blowfish je předchůce Twofish a sám jeho tvůrce doporučuje raději používat právě Twofish.

Data Encryption Standard doplatil na svůj krátký klíč, který má efektivní délku pouze 56 bitů a je tedy prolomitelný útokem hrubou silou za několik hodin. Krátký klíč řeší 3DES, ten je však třikrát pomalejší.

RC4 našel ve své době uplatnění v mnoha protokolech a byl jednou z nejvyužívanějších proudových šifer. Před pár lety však několik organizací vyzvalo k jeho dalšímu neužívání.

Šifra	Typ	Struktura	Délka klíče [b]	Velikost bloku [b]	Počet rund
DES	bloková	Feistelova síť	64 (56)	64	16
3DES	bloková	Feistelova síť	168	64	48
AES	bloková	Substitučně-permutační síť	128, 192, 256	128	10, 12, 14
Present	bloková	Substitučně-permutační síť	80, 128	64	31
IDEA	bloková	Lai-Massey	128	64	8
Blowfish	bloková	Feistelova síť	32-448	64	16
Twofish	bloková	Feistelova síť	128, 192, 256	128	16
Serpent	bloková	Substitučně-permutační síť	128, 192, 256	128	32
RC4	proudová	-	40-2048	-	1

Tab. 2.1: Srovnání šifrovacích algoritmů

3 Operační módy

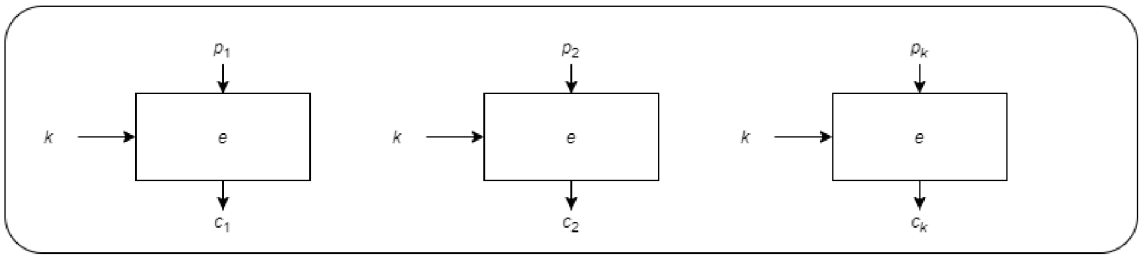
V minulé kapitole byly popsány vybrané symetrické šifrovací algoritmy, převážně blokové šifry. Jedna z hlavních vlastností, která jim dala jméno, je že pracují s bloky o fixní velikosti. Ta je někdy 64 bitů, ale převážně se dnes využívá 128 bitů.

V praxi se však málokdy stane, že velikost dat, která mají být zašifrována odpovídá násobku velikosti bloku. Pro doplnění zprávy na požadovanou délku se tedy využívá tzv. „padding“. Zároveň velice často je potřeba zašifrovat data o větší velikosti než je jeden blok. Proto je potřeba otevřený text P rozdělit na jednotlivé bloky $p_1, p_2 \ldots, p_k$

V závislosti na využitém operačním módu se otevřený text P šifruje na šifrový text C po blocích jak bylo popsáno, nebo je díky vlastnostem některých operačních módů tento proces změněn na způsob proudové šifry.

3.1 ECB

Když je otevřený text P rozdělen na bloky $p_1, p_2 \ldots, p_k$, tak by nejjednodušší způsob jak šifrování provést byl jednotlivé bloky šifrovat postupně jak jdou za sebou. A právě takto pracuje Electronic codebook neboli ECB mód, což je zobrazeno na obrázku 3.1.

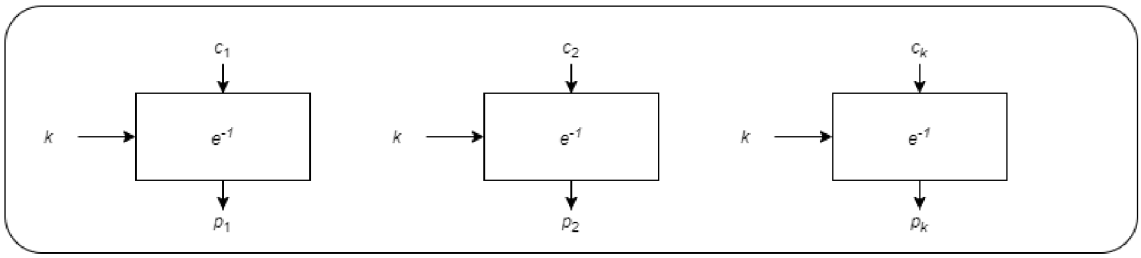


Obr. 3.1: Šifrování pomocí módu ECB

Dešifrování probíhá obdobně, pouze se zašifrované bloky $c_1, c_2 \ldots, c_k$ mění zpět na otevřené bloky $p_1, p_2 \ldots, p_k$, jak je možné vidět na obrázku 3.2, v závislosti na použitém šifrovacím algoritmu.

Šifrování i dešifrování lze provádět paralelně, což celý proces urychluje a je možné ihned načíst kteroukoliv část. Toto všechno zní jako výhody. V čem je tedy s tímto módem potíž? Postrádá difuzi. Pokud se nějaký blok bude opakovat, k čemuž v praxi dochází často, tak budou tyto bloky zašifrovány stejně. Na zjednodušeném příkladu bude vysvětleno proč je toto problém, i když samotná šifra není prolomena.

Platba z banky A do banky B obsahuje 5 informací, které jsou přeneseny v pěti blocích. První blok obsahuje označení banky ze které platba odchází, druhý obsahuje



Obr. 3.2: Dešifrování pomocí módu ECB

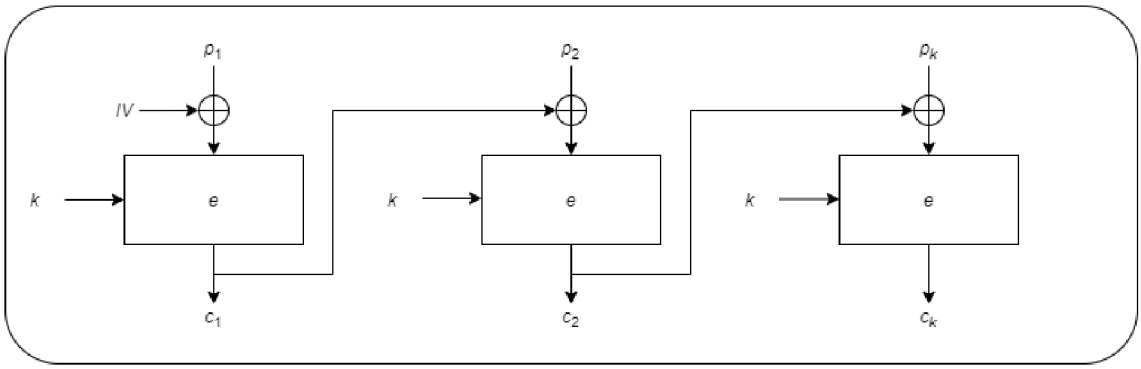
konkrétní účet odesílatele, třetí označuje banku, která platbu přijímá, čtvrtý účet příjemce a pátý blok obsahuje částku, která je poslána. Pokud si útočník založí účet u banky A i B a bude si sám sobě posílat peníze, tak může analyzovat provoz. Během toho co banky A a B používají stejný klíč, tak budou všechny útočníkovy platby zašifrovány stejně. Pokud je tedy identifikuje, tak zjistí jak jeho platby vypadají zašifrovaně. V tomto okamžiku je pro něj důležitý čtvrtý blok - ten který označuje jeho účet na který přichází peníze. Pokud tedy najde další provoz, který obsahuje stejný první a třetí blok, tedy platbu z banky A do banky B, tak změnou čtvrtého bloku za svůj přesměruje cizí platbu na svůj účet. To vše aniž by prolomil šifru, nebo získal klíč. To znamená, že útok narušil integritu zprávy, zatímco šifra zůstala nerozluštěna.

Jako další příklad se dá uvést bitmapová grafika. Pokud jsou na obrázku větší části, které mají stejnou barvu, tak budou i stejně šifrovány. To v praxi znamená, že obrázek sice bude zašifrován, avšak stejná barva bude stejně šifrovaná, což znamená, že pro lidské oko bude obrázek stále rozeznatelný, pouze barva bude vypadat jinak.

Právě kvůli těmto důvodům je ECB mód nebezpečný a jeho používání se rozhodně nedoporučuje. [20] [21]

3.2 CBC

U ECB módu byl problém s tím, že vše probíhalo vždy stejně. Proč je to špatně bylo také vysvětleno. Aby k tomu nedocházelo a stejné bloky byly šifrovány jinak, je potřeba přidat nějakou další formu náhodnosti. Cipher Block Chaining (CBC) mód, jak jeho název napovídá, využívá provázanosti bloků. Zašifrovaný blok c_i je použit pro zašifrování bloku p_{i+1} pomocí operace XOR na vstupu. Po operaci XOR bloků c_i a p_{i+1} probíhá standardně šifrování k získání dalšího bloku c_{i+1}. Ten je poté použit k šifrování p_{i+2} a tak dále. Jak se ale zašifruje první blok p_1? K tomu se využívá inicializační vektor IV. Proces šifrování Cipher Block Chaining módem je zobrazen na obrázku 3.3.



Obr. 3.3: Šifrování pomocí módu CBC

3.2.1 Inicializační vektor

Inicializační vektor je potřeba měnit a nevyužívat ho se stejným klíčem vícekrát, jelikož by to vedlo ke stejnému problému jako u ECB módu. Tedy k tomu, že stejná zpráva by byla zašifrována vícekrát stejně. Naopak, když bude stejná zpráva zašifrována dvakrát, pokaždé s jiným inicializačním vektorem, tak oba šifrové texty budou vypadat úplně jinak.

K tomuto se nejčastěji využívá „nonce“, což je zkratka pro *number used once*, neboli *číslo použité jednou*.

Další způsob může být počítaná hodnota tzv. „counter“, kterou si musí obě strany uložit a při každé komunikaci ji inkrementovat. Nebo mohou inicializační vektor odvodit od předem dohodnutých parametrů.

Obecně lze inicializační vektor generovat mnoha různými způsoby. Důležité je, aby se stejným klíčem nebyl opakován. [15] [22]

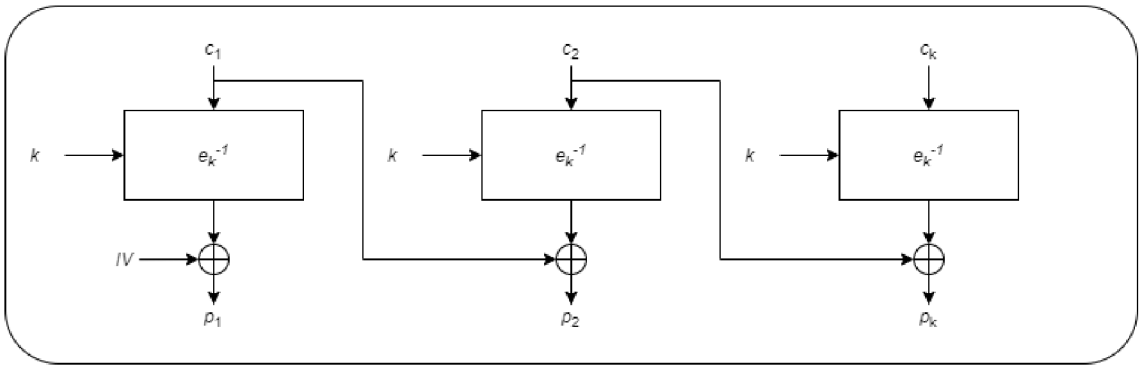
3.2.2 Dešifrování

Jelikož je proces šifrování za použití Cipher Block Chaining módu závislý na předchozích blocích, není proces dešifrování tak jednoduchý jako u ECB módu. I u tohoto zpětného procesu je potřeba využít další blok.

Pokud se tedy dešifruje blok c_i je potřeba nejprve provést dešifrování blokové šifry. To ale není vše. Ještě je potřeba zvrátit operaci XOR. K tomu stačí jednoduše udělat tuto operaci znovu. Musí se ale použít stejný blok, který byl použit k šifrování. To znamená blok c_{i-1}, jak je možné vidět na obrázku 3.4.

Procesy šifrování a dešifrování lze tedy zapsat takto:

a) Šifrování prvního bloku: $c_1 = e_k(p_1 \oplus IV)$
b) Šifrování zbylých bloků: $c_i = e_k(p_i \oplus c_i - 1)$, $i \geq 2$
c) Dešifrování prvního bloku: $p_1 = e_k^{-1}(c_1) \oplus IV$
d) Dešifrování ostatních bloků: $p_i = e_k^{-1}(c_i) \oplus c_{i-1}$, $i \geq 2$



Obr. 3.4: Dešifrování pomocí módu CBC

CBC řeší největší problém ECB módu. Avšak toto řešení má svou vlastní nevýhodu. Jelikož k šifrování bloku je potřeba zašifrovaný předchozí blok, není možné dělat šifrování paralelně. V praxi je tedy vždy nutné počkat až se předchozí blok zašifruje a je možné ho použít pro další blok. A to znamená, že tím CBC utrpí na rychlosti. Pokud je tedy potřeba zašifrovat větší množství dat může se to projevit.

Další vlastnost, kterou CBC získává svým provázáním bloků je, že pokud nastane nějaká chyba v jednom bloku, tak ta stejná chyba ovlivní i další blok. [20] [21]

3.3 OFB

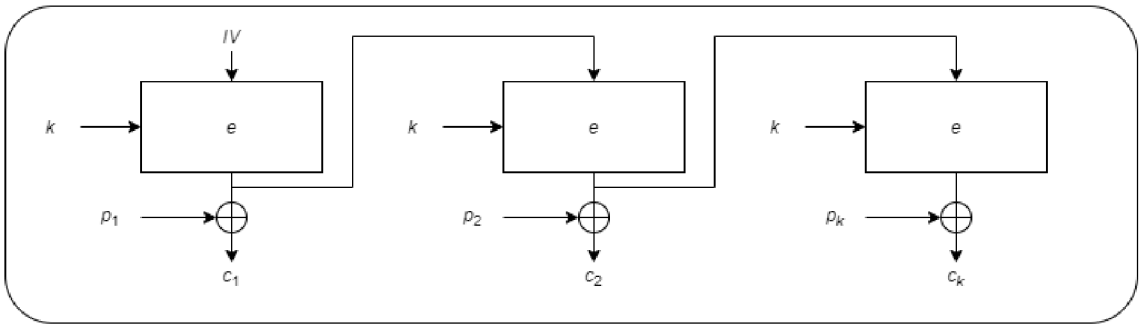
Output Feedback mód se od předchozích módů liší tím, že svou funkcí mění proces šifrování na způsob synchronní proudové šifry.

Samotná bloková šifra je stále využívána, avšak pouze ke generování bitového proudu, který je poté využit k šifrování pomocí operace XOR spolu s otevřeným textem k získání textu šifrového.

Na začátku je potřeba inicializační vektor IV, který je použit jako vstup pro blokovou šifru. V závislosti na použité blokové šifře je výstupem n bitů, což odpovídá velikosti jednoho bloku. Právě ty jsou použity jako bitový proud k šifrování. Pro získání dalšího bloku je předešlý výstup znovu zašifrován. Tento proces je opakován stále dokola v závislosti na velikosti dat, které je potřeba zašifrovat.

Jelikož je bitový proud přímo závislý na inicializačním vektoru je velmi důležité aby se neopakoval. Se stejným inicializačním vektorem vyjde stejný bitový proud k šifrování, což je nežádoucí. Samotný proces šifrování je zobrazen na obrázku 3.5.

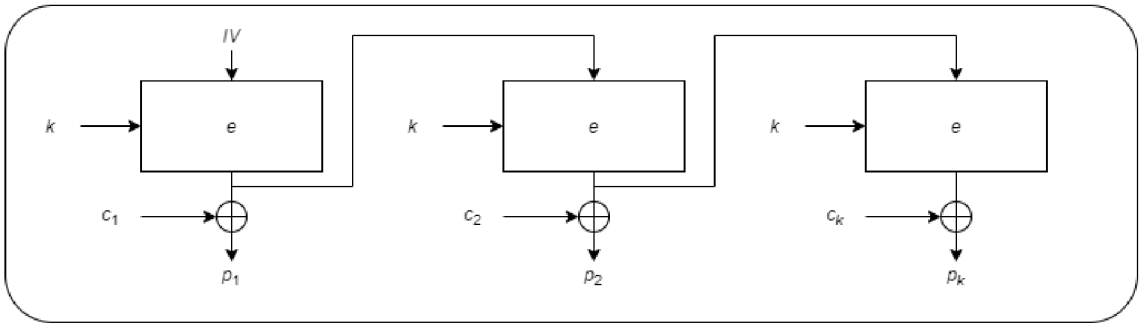
Vzhledem ke způsobu generování bitového proudu opět dochází k provázanosti stejně jako u CBC módu. To znamená, že šifrování ani dešifrování nelze provádět paralelně, jelikož musí být vždy nejprve ukončen předchozí proces. Stejně tak nelze přistupovat k náhodným částem zprávy, ale musí se k nim postupně dostat.



Obr. 3.5: Šifrování pomocí módu OFB

3.3.1 Dešifrování

Díky tomu, že je bloková šifra použita pouze ke generování bitového proudu, který otevřený text šifruje pomocí operace XOR na šifrový text je dešifrování velmi podobné šifrování. Opět je na začátku potřeba stejný inicializační vektor, který je i v tomto případě zašifrován, protože k operaci XOR je potřeba stejný vstup jako při šifrování. Z tohoto důvodu není implementace dešifrování u zvolené blokové šifry potřeba. Zašifrovaný inicializační vektor poté slouží jak pro XOR, tentokrát s prvním šifrovým blokem pro získání bloku otevřeného, tak jako vstup pro další zašifrování a získání bitového proudu k dešifrování následujícího bloku. Dešifrování v módu OFB je zobrazeno na obrázku 3.6. [15] [20] [21]



Obr. 3.6: Dešifrování pomocí módu OFB

3.4 CTR

Stejně jako u OFB módu se i u CTR využívá bloková šifra ke generování bitového proudu, který je následně použit k šifrování otevřeného textu pomocí operace XOR. Samotné šifrování tedy opět probíhá jako u proudové šifry.

V čem se tyto módy ale liší je způsob získávání vstupního bloku pro blokovou šifru ke generování bitového proudu. U OFB to byl inicializační vektor, který byl zašifrován blokovou šifrou a tento výstup byl opět zašifrován. Tento proces se opakoval tolikrát, kolik bylo potřeba zašifrovat dat.

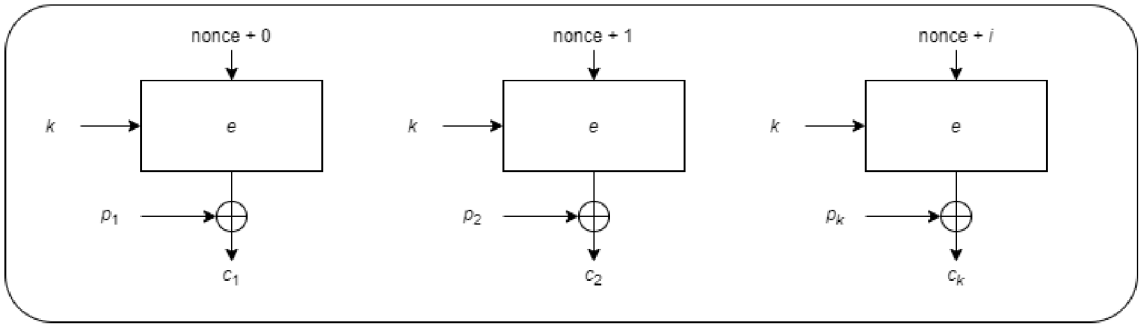
U CTR módu se však jako vstupní blok používá kombinace „nonce“ o menší velikosti než je velikost bloku a zbytek je vyhrazen pro hodnotu i.

Hodnota i slouží jako počítadlo a je inicializována na nule. S každým blokem je poté tato hodnota inkrementována, což zajišťuje, aby se neopakoval vstupní blok a tím pádem ani výstup, který se používá k šifrování.

V závislosti na tom, kolik bitů z bloku je vyhrazeno pro nonce a kolik zůstane pro hodnotu i je určena velikost dat, která se dá s jedním klíčem zašifrovat. V případě použití šifry, která využívá bloky o velikosti 128 bitů by při vyhrazení 96 bitů pro nonce a zbylých 32 bitů pro počítadlo bylo možno zašifrovat 2^{32} bloků se stejným nonce.

Stejně jako u předchozího módu je důležité nikdy nepoužít stejnou kombinaci nonce a klíče opakovaně. To by vedlo ke stejnému bitovému proudu, což může vést k úniku informací o zprávě. Další vlastnost co se opakuje jako u OFB módu je způsob šifrování. Používá se operace XOR, při šifrování bitový proud z blokové šifry spolu s otevřeným textem. Proces je zobrazen na obrázku 3.7.

Výhodou CTR módu oproti OFB módu je možnost paralelního výpočtu. To je možné díky zmiňovanému rozdílnému generování bitového proudu. Jelikož část nonce je stále stejná a mění se pouze hodnota i, tak nedochází k provázanosti jednotlivých bloků.

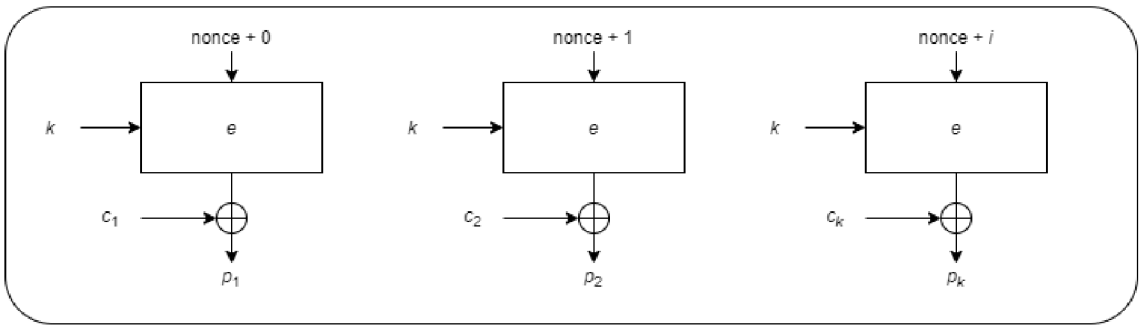


Obr. 3.7: Šifrování pomocí módu CTR

3.4.1 Dešifrování

Aby bylo možné dešifrovat je potřeba jako vstup použít stejný nonce společně s hodnotou i odpovídající části, která má být dešifrována. Bloková šifra vstup opět šifruje, aby výstupem byl stejný bitový proud, který je poté použit při operaci XOR. Platí

tedy, že u blokové šifry není implementace dešifrování potřebná. V čem se proces dešifrování u counter módu liší od šifrování je vidět na obrázku 3.8. Konkrétně jde o prohození otevřených bloků s šifrovými, právě tak, aby výstupem dešifrování byl otevřený text. [15] [20] [23]



Obr. 3.8: Dešifrování pomocí módu CTR

3.5 GCM

Galois counter mód kromě samotného šifrování počítá také MAC (message authentication code), který slouží k ověření pravosti zprávy a zajišťuje tak autentizaci. MAC funguje jako kontrolní součet. Odesílatel ho spočítá a připojí ke zprávě. Ta je odeslána a po přijetí příjemce udělá stejný výpočet. Pokud se MAC shoduje, tak si příjemce může být jistý, že zprávu skutečně odeslal ten, od koho si myslí. Stejně tak díky tomu ví, že se zprávou nebylo nijak manipulováno a nebyla změněna. Ať už chybou při přenosu, či potenciálním útočníkem. Jinými slovy tedy chrání původnost i integritu.

Důvěrnost je zajištěna šifrováním stejně jako u CTR módu. Jako vstup je potřeba inicializační vektor IV, který je složen z nonce a hodnoty i, která se s každým blokem inkrementuje. GCM je určen pro využití s blokovými šiframi, které používají bloky o velikosti 128 bitů, například AES.

3.5.1 Šifrování

Než šifrování začne, je potřeba několik vstupních hodnot. Ty jsou:

a) Otevřený text P.
b) Inicializační vektor IV.
c) AAD (additional authenticated data) - další autentizovaná data. Ta se posílají nezašifrovaná a mohou obsahovat například adresy, nebo parametry pro síťový protokol.

Na začátku je od inicializačního vektoru odvozena počáteční hodnota CTR_0. Ta je inkrementována a vznikne $CTR_1 = CTR_0 + 1$. CTR_1 je poté zašifrována zvolenou blokovou šifrou a pomocí operace XOR se s ní zašifruje první blok otevřeného textu. Pro další bloky se CTR opět inkrementuje a zašifruje k získání dalšího bitového proudu k šifrování otevřeného textu.

K autentizaci GCM používá násobení v Galoisově poli GF(2^{128}) spolu s polynomem $P(x) = x^{128} + x^7 + x^2 + x + 1$. Pro každý blok p_i se odvodí autentizační parametr g_i. Ten se spočítá jako XOR součet aktuálního šifrového bloku c_i a g_{i-1}, který je vynásoben hodnotou H. H se získá tak, že se blokovou šifrou zašifruje jeden blok plný nul, neboli $H = e_k(0)$. První autentizační parametr g_0 se spočítá jako $g_0 = AAD \times H$, což je opět násobení v Galoisově poli.

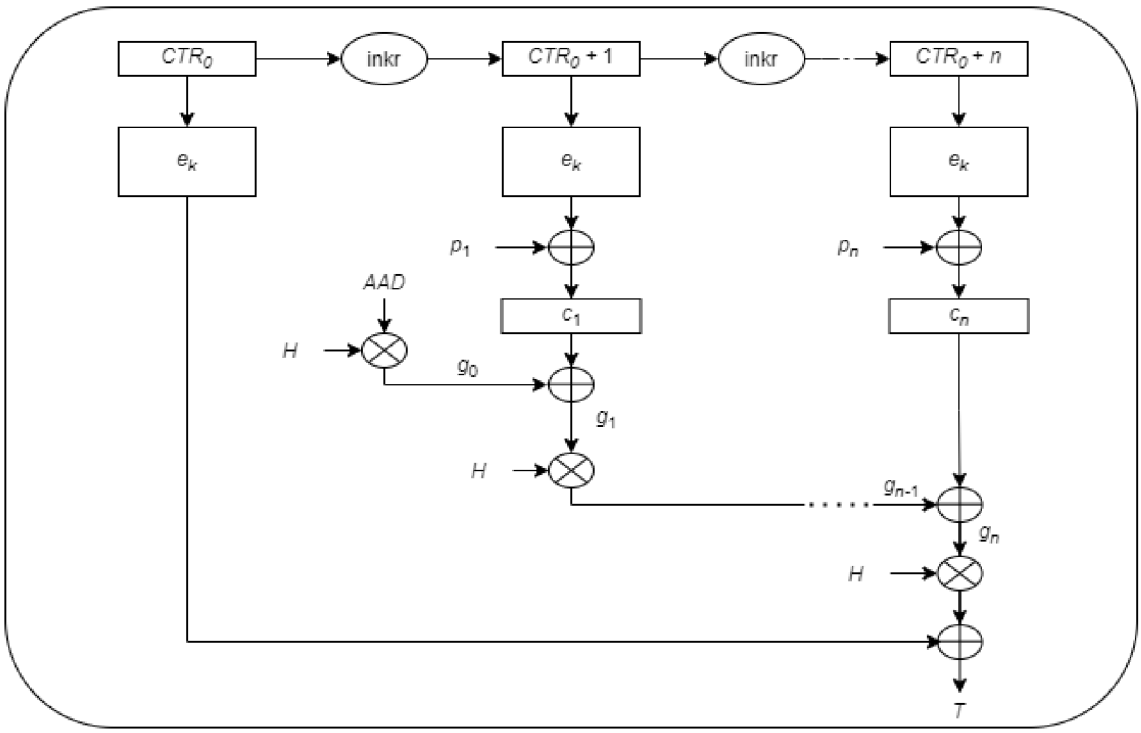
Nakonec se ještě spočítá autentizační značka T. $T = (g_n \times H) \oplus e_k(CTR_0)$. Právě ta slouží k autentizaci zprávy. Proces je zobrazen na obrázku 3.9.

Výstupy šifrování tedy jsou:

a) Šifrový text C, který má stejnou bitovou délku jako otevřený text P, který byl na vstupu.

b) Autentizační značka T.

T může nabývat bitové délky 128, 120, 112, 104, nebo 96. V určitých případech i 64, nebo 32 bitů.



Obr. 3.9: Šifrování pomocí módu GCM

3.5.2 Dešifrování

Příjemce dešifruje šifrový text C stejně jako u CTR módu. Poté, aby zkontroloval autentičnost zprávy spočítá autentizační značku T'. V tuto chvíli mohou nastat dvě možnosti. První možnost je, že $T' = T$. V tomto případě vše proběhlo v pořádku a příjemce ví, že zpráva skutečně přišla od zamýšleného odesílatele a nebyla nijak pozměněna. V případě že T' se liší od T bude jako výstup chybová hláška. [20] [24] [25]

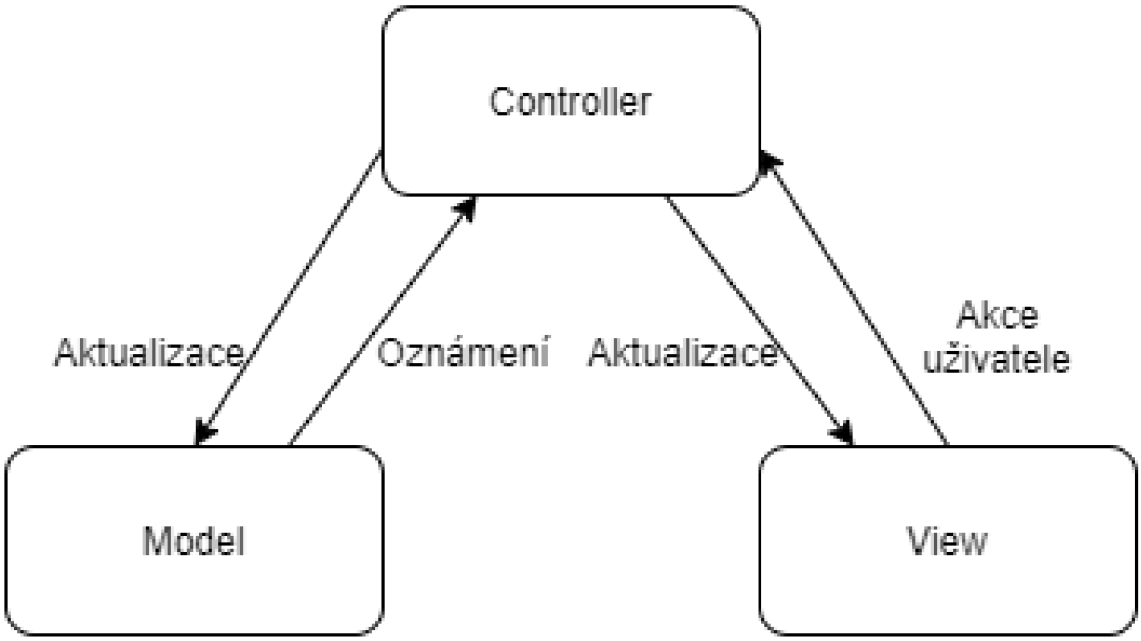
4 Praktická část

V teoretické části práce byly popsány symetrické šifrovací algoritmy, byly rozděleny na blokové a proudové, následně byly vybrané šifry popsány a srovnány a nakonec byly popsány operační módy, které se u blokových šifer používají.

Následující kapitola se věnuje praktické části, ve které je vytvořena webová aplikace sloužící k demonstraci vybraných symetrických šifrovacích algoritmů i operačních módů.

4.1 MVC

Aplikace byla navržena jako webová aplikace s využitím architektury Model-View-Controller (MVC), resp. Spring web Model-View-Controller framework. Jedná se o běžně používané schéma (možné vidět na obrázku 4.1) při vývoji webových, ale také desktop aplikací. Také se používá označení návrhový vzor MVC. Cílem je oddělit logiku aplikace a zobrazení dat. MVC aplikace je rozdělena na nezávislé komponenty. Každá má definovanou úlohu a samostatně zajišťuje předem definovanou činnost. Tento princip umožňuje přehledný vývoj a údržbu aplikací.



Obr. 4.1: MVC - schéma architektury

Model odpovídá za logiku, která souvisí s daty. Obsluhuje data přenášená mezi řadičem a pohledem. Poskytuje data pro zobrazení, provádí aktualizaci změněných dat. Může provádět např. databázový dotaz, validaci hodnot, různé výpočty apod.

Model neví nic o výstupu, neví jak se data interpretují. Model obsahuje v aplikaci objekt typu EncryptForm, který uchovává hodnoty z formuláře a je doplněn o výsledky šifrování/dešifrování zprávy nebo souboru.

View (pohled) je určen k zobrazení dat, která jsou získána z modelu. Pohled by měl obsahovat jen nezbytnou logiku potřebnou pro zobrazení dat. Zejména se jedná o příkazy pro cyklické, podmíněné zobrazení dat a validaci. Pohled nezajímá, odkud data přicházejí. Jeho úkolem je pouze data zobrazit. Pohledy jsou v aplikaci realizovány jako HTML stránky, které se renderují s podporou šablonovacího systému Thymeleaf.

Cotroller (řadič) zpracovává události vyvolané uživatelem. Provádí přípravu dat z modelu a výběr pohledu, ve kterém jsou data interpretována. Může také zapisovat přímo do streamu odpovědí a dokončit požadavek. Aplikace používá jediný Controller definovaný třídou WebController. Metody třídy WebController obsluhují příslušné HTTP požadavky aplikace. [26]

4.2 Framework Spring

Webová aplikace je tvořena pomocí frameworku Spring, což je populární open source aplikační rámec pro vývoj Java aplikací. Založení projektu je možné na stránce `https://start.spring.io/`. Na této stránce je potřeba nastavit požadované parametry dle potřeby. Pro tuto webovou aplikaci byl zvolen projekt Gradle. Gradle je nástroj pro automatizaci sestavování programu. Dále byl nastaven jazyk Java ve verzi 11, název projektu a packaging Jar. Nakonec byly přidány dependencies „Spring Web“ a „Thymeleaf“.

Po kompletním nastavení byl vygenerován projekt, který byl stažen a importován do vývojového prostředí Intellij IDEA. Spuštěnou aplikaci je možné zobrazit ve webovém prohlížeči na adrese http://localhost:8080/.

4.3 Frontend

Frontend webové aplikace je tvořen pomocí HTML kódu. Ke grafické úpravě byl využit framework Bootstrap. Ten obsahuje velké množství připravených CSS tříd, které je možné využít při formátování webových aplikací a také řadu připravených komponent například formulářové prvky, navigace, modální okna. Bootstrap rovněž umožňuje zajistit responsibilitu aplikace. Bootstrap je veřejně dostupný pod otevřenou licencí. Ve výpisu 4.1 je uveden příklad CSS tříd použitých při formátování vstupního prvku typu select pro výběr módu šifrování (form-group, form-select). Současně je patrné jak šablona pracuje s daty modelu. Pomocí Thymeleaf atributu

th je na úrovni tagu select adresován atribut mode, který uchovává hodnotu vybraného operačního módu. Ten je atributem objektu encrypt (výpis 4.2), který je stejným způsobem odkazován na úrovni definice formuláře. Touto cestou je na úrovni šablony zajištěno propojení vstupu z úrovně HTML formuláře a modelu.

Výpis 4.1: Volba operačního módu

```
<div class="form-group">
    <label for="mode">Mód:</label>
    <select class="form-select" aria-label="Vyberte␣mód
␣␣␣␣šifrování" th:field="*{mode}">
        <option th:value="ECB" th:text="ECB"></option>
        <option th:value="CBC" th:text="CBC"></option>
        <option th:value="OFB" th:text="OFB"></option>
        <option th:value="CTR" th:text="CTR"></option>
    </select>
</div>
```

Výpis 4.2: Propojení

```
<form action="#" th:action="@{/}" th:object="${encrypt}"
method="post" enctype="multipart/form-data">
```

4.3.1 Aplikace

Aplikace je zaměřena na šifrování zpráv a souborů pomocí symetrických šifrovacích algoritmů. V základní nabídce se nachází rovněž odkaz na dvě statické HTML stránky, ve kterých se nachází popis šifrovacích algoritmů a operačních módů.

Pro funkčnost aplikace je potřeba zadat několik parametrů. Na pořadí zadání nezáleží. Jedná se o:

- šifrovací algoritmus
- operační mód
- uživatelské heslo, které slouží k vygenerování klíče pro zvolený šifrovací algoritmus

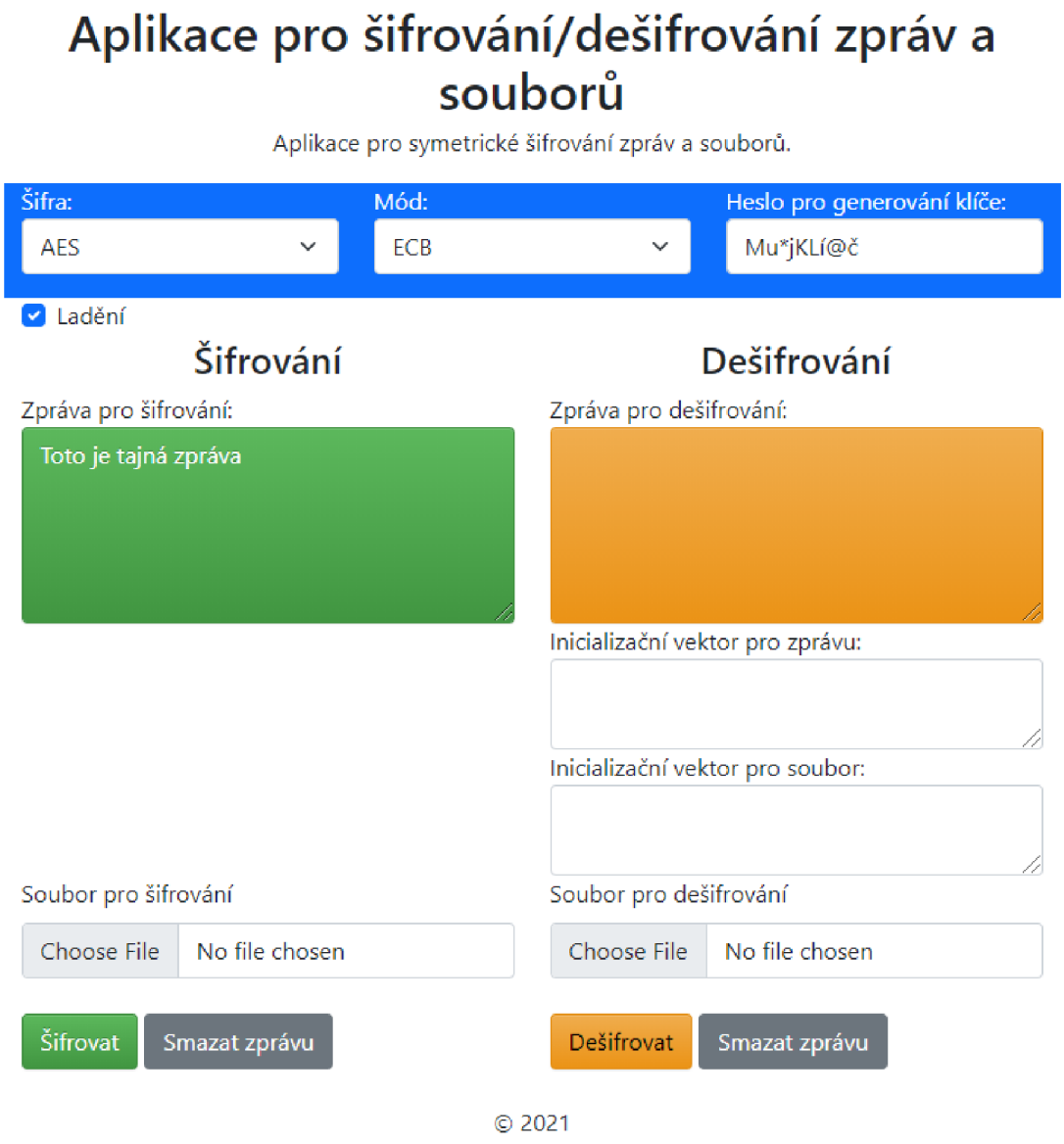
Pod řádkem s parametry jsou dvě textová pole, první v levé části je nadepsané „Šifrování“, zde je možné vložit text k zašifrování. V pravé části pod nadpisem „Dešifrování“, se zadává text k dešifrování. Pod dešifrováním jsou další 2 textová pole pro inicializační vektory.

Nad nadpisem „Šifrování“ je ještě checkbox „Ladění“. Ten je určen k tomu, aby po zašifrování otevřeného textu byl původní text smazán a zobrazen pouze šifrový text. Tato funkce funguje i zpětně, tedy vymaže šifrový text a zobrazí pouze otevřený po dešifrování.

Pro jednodušší pochopení uživatele a možnost okamžitého spuštění mají vstupy výchozí hodnoty, avšak je možné je změnit i před prvním šifrováním.

Pro případ, kdy chce uživatel šifrovat či dešifrovat soubor jsou připraveny tlačítka k nahrání daného souboru v odpovídající sekci, tedy vlevo soubor k šifrování, nebo vpravo soubor k dešifrování.

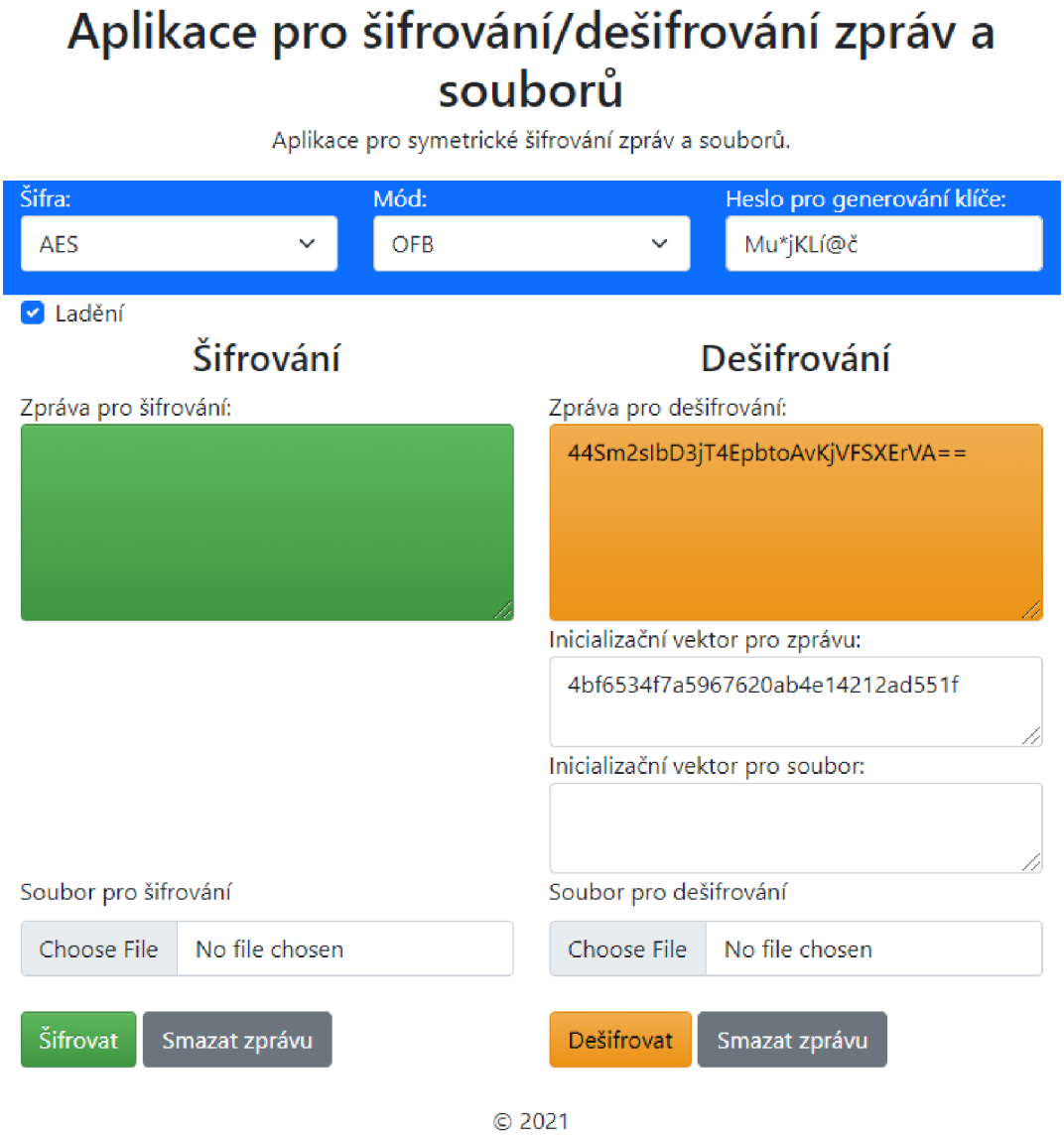
Nakonec jsou opět pod svou odpovídající částí stránky umístěna tlačítka „Šifrovat“ a „Dešifrovat“ pro potvrzení a odeslání formuláře a následné provedení zvolené akce. Tlačítka jsou ještě doplněna o vedlejší „Smazat zprávu“ pro vymazání textového pole. To obsluhuje kód v JavaScriptu. Výchozí vzhled stránky je zobrazen na obrázku 4.2.



Obr. 4.2: Výchozí vzhled stránky

Obě textová pole pro šifrování a dešifrování textu zároveň slouží i pro zobrazení výstupu. Pokud jsou tedy zvoleny parametry a je zadán text k šifrování a dojde k potvrzení, výsledek se po zpracování zobrazí v poli pro dešifrování a opačně.

Pokud si uživatel zvolí operační mód, který pracuje s inicializačním vektorem, tak se po zašifrování zobrazí i použitý inicializační vektor, který je automaticky generován. To je možné vidět na obrázku 4.3.



Obr. 4.3: Šifrový text, šifra AES v módu OFB a inicializační vektor

Obrázek zachycuje výsledek šifrování otevřeného textu „Toto je tajná zpráva“ pomocí šifry AES v operačním módu OFB. Heslo zůstalo na výchozí hodnotě.

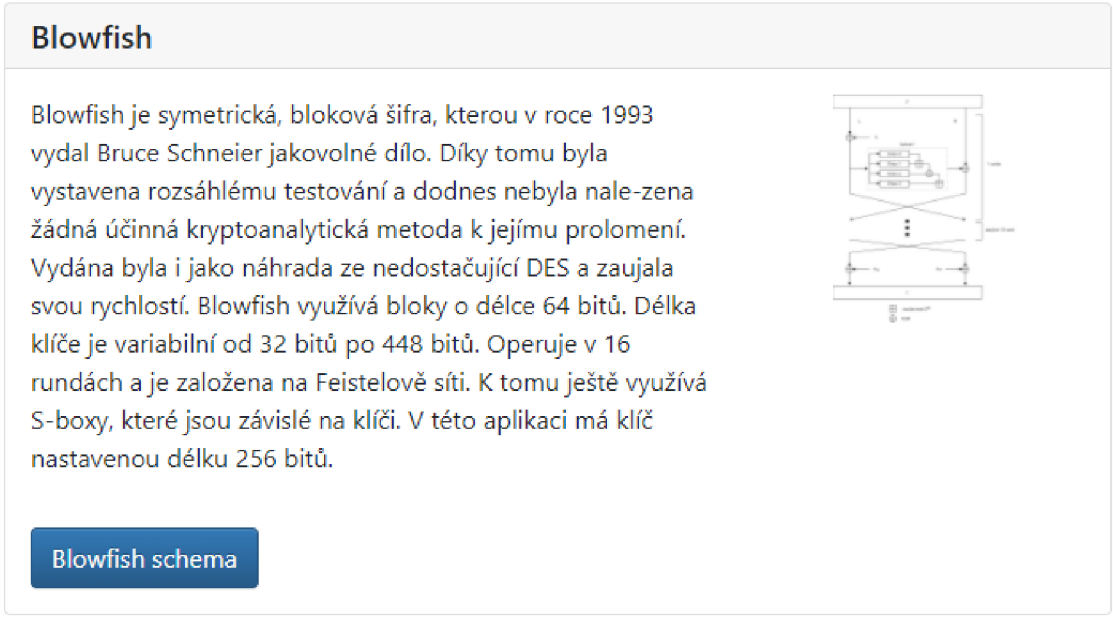
Pokud chce uživatel šifrový text ihned dešifrovat zpět, tak se inicializační vektor automaticky načte a dešifrování s ním proběhne, avšak pokud by si uživatel chtěl

zprávu uložit a dešifrovat později, je potřeba aby si uložil i inicializační vektor. Ten totiž musí být pro úspěšné dešifrování použit stejný jako při šifrování, jak už bylo popsáno v předchozí kapitole.

4.3.2 Šifrovací algoritmy a operační módy

V hlavní nabídce jsou položky „Šifrovací algoritmy“ a „Operační módy“. Odkazy uživatele přesměrují na samostatné stránky věnované dané problematice.

Pokud uživatel zvolí „Šifrovací algoritmy“, zobrazí se mu stránka, kde jsou krátce popsány symetrické šifrovací algoritmy dostupné v aplikaci. Po praktické zkoušce šifrování pomocí aplikace si tedy uživatel může krátce přečíst o tom jak jednotlivé šifry fungují. Popisky jsou doplněny náhledem na schémata samotného šifrovacího algoritmu. Ten je možno kliknutím zvětšit pro lepší zobrazení. Vzhled stránky s šifrovacími algoritmy je vidět na obrázku 4.4.



Obr. 4.4: Stránka s popisky šifrovacích algoritmů

Volbou tlačítka „Operační módy“ se uživatel zase dostane na stránku věnující se stručnému popisu operačních módů dostupných v této aplikaci. Stejně jako u šifrovacích algoritmů jsou popisky doplněny obrázky. Obrázek je dostupný jak pro šifrování, tak dešifrování a uživatel se tedy může podívat, jak se tyto procesy liší.

4.4 Thymeleaf

Pohledy jsou v aplikaci vytvořeny jako HTML stránky. Z důvodu jejich snadnějšího vytváření byl do projektu začleněn šablonovací systém Thymeleaf. Thymeleaf renderuje výsledné dokumenty na straně serveru. Pomocí atributu th lze v šablonách zobrazovat hodnoty proměnných a objektů z modelu. Thymeleaf podporuje fragmentování a definici šablon, které umožňují vytvářet pohledy bez zbytečného opakování kódu. Jako fragment se obvykle definuje část kódu, která se ve stránkách aplikace opakuje. Thymeleaf umožňuje fragmenty také parametrizovat. Pomocí Thymeleaf lze přidávat tagům logiku. Takové doplnění se označuje jako procesor a obvykle se provádí na úrovni atributu, například atributy pro formuláře - th:field, th:object. [27]

4.5 Backend

V hierarchii zpracování HTTP požadavků ve Spring Web MVC procházejí všechny přes tzv. Front controller (Dispatcherservlet). Centrální servlet požadavky zpracuje a předá dále jednotlivým registrovaným řadičům. Úkolem řadiče je připravit data v modelu a vybrat odpovídající pohled pro jejich interpretaci. [28]

Backend se stará o obsluhu následujících HTTP požadavků:

a) GET "/"- zobrazení formuláře pro zadání parametrů šifrování.
b) POST "/"- zpracování formuláře se zadanými hodnotami zpráv, souborů, algoritmů a módů a provedení operací pro šifrování a dešifrování v závislosti na parametru action.
c) GET "/getFile/id"- stažení zpracovaného souboru.
d) GET "/sifry"- zobrazení stránky s popisem šifrovacích algoritmů.
e) GET "/operacnimody"- zobrazení stránky s popisem operačních módů.

Výše uvedené požadavky jsou zpracovány jednotlivými metodami třídy *WebController*, která je opatřena anotací @Controller. Spojení požadavku s metodou třídy je provedeno pomocí příslušné anotace @GetMapping (mapování požadavku HTTP GET), nebo @PostMapping (mapování požadavku HTTP POST):

a) **@GetMapping**("/")
public ModelAndView *showForm*()
b) **@PostMapping**(
value="/",
consumes = MediaType.MULTIPART _FORM_DATA_VALUE,
produces=MediaType.MULTIPART_FORM_DATA _VALUE,
params="action=Šifrovat")
public ModelAndView *encrypt*(@ModelAttribute("encrypt")EncryptForm ef,

BindingResult result, ModelMap model, @RequestParam("file") MultipartFile file)

c) **@PostMapping**(
value="/", consumes = MediaType.MULTIPART_FORM_DATA_VALUE, produces=MediaType.MULTIPART_FORM_DATA_VALUE, params="action=Dešifrovat")
public ModelAndView *decrypt*(@ModelAttribute("encrypt")EncryptForm ef, BindingResult result, ModelMap model, @RequestParam("cryptedFile") MultipartFile cryptedFile)

d) **@GetMapping**("/operacnimody")
public ModelAndView *operacniMody*()

e) **@GetMapping**("/sifry")
public ModelAndView *pageSifry*()

f) **@GetMapping**("/getFile/file_name")
public void *getLogFile*(@PathVariable("file_name") String fileName, HttpServletResponse response)

Všechny uvedené metody jsou ukončeny spojením pohledu a modelu prostřednictvím objektu třídy ModelAndView.

Metody pro šifrování a dešifrování jsou součástí třídy *EncryptDecrypt*. Třída obsahuje tyto hlavní metody:

- encrypt (String strToEncrypt, String secret, String algorithm, String mode) - šifrování zpráv.
- decrypt (String strToEncrypt, String secret, String algorithm, String mode) - dešifrování zpráv
- encryptFile (File inputFile, File outputFile, String secret, String algorithm, String mode) - šifrování souborů
- decryptFile (File inputFile, File outputFile, String secret, String algorithm, String mode) - dešifrování souborů

EncryptDecrypt také obsahuje metodu, která ze vstupů od uživatele sestaví String *cipherAndMode* na základě kterého rozhoduje, zda operace potřebuje inicializační vektor, či nikoliv. To je rozhodnuto pomocí příkazu case. V případě, že je inicializační vektor potřeba je pomocí podmínky if rozhodnuto, jakou velikost má inicializační vektor mít. Toto je možné vidět na následujícím výpisu.

Výpis 4.3: Switch rozhodující o inicializačním vektoru

```
1 switch(cipherAndMode) {
2             case "AES/ECB/PKCS5PADDING":
3             case "DES/ECB/PKCS5PADDING":
4             case "DESede/ECB/PKCS5PADDING":
5             case "Blowfish/ECB/PKCS5PADDING":
```

```
6               case "RC4":
7                   initVector = null;
8                   break;
9               default:
10                  if (initVectorArray == null) {
11                  initVectorArray = Generate.generateIv(
12                  (ALGORITHM.equals("AES"))
13                  ? 16: 8);}
14                  initVector = new IvParameterSpec(
15                  initVectorArray);
16          }
```

Pokud je jako šifrovací algoritmus nastaven RC4, tak se žádný operační mód neuvažuje, jelikož se jedná o proudovou šifru.

Dále se podobným způsobem na základě zvoleného šifrovacího algoritmu nastavuje délka klíče. Samotný klíč je vytvořen pomocí vstupu od uživatele - hesla. K tomu je využita třída SecretKeyFactory a funkce PBKDF2. Pro jednodušší ovládání uživatelem je sůl konstantní.

Zbylé metody už slouží k samotnému šifrování a dešifrování. V těch je využita třída Cipher. Ta nejprve pomocí metody *getInstance* přijímá argument *cipherAndMode*, který je získán ze vstupů od uživatele. Poté je nastavena operace šifrování, nebo dešifrování a přidány argumenty v podobě klíče a inicializačního vektoru. Na následujícím výpisu je zobrazena metoda *encrypt*, která šifruje textový vstup.

Výpis 4.4: Metoda encrypt

```
1  public static String encrypt(String strToEncrypt,
2  String secret, String algorithm, String mode) {
3          try {
4               initVectorArray = null;
5               setCipherAndMode(algorithm, mode);
6               setKey(secret);
7               Cipher cipher= Cipher.getInstance(cipherAndMode);
8               cipher.init(Cipher.ENCRYPT_MODE, secretKey,
9               initVector);
10
11              return Base64.getEncoder().encodeToString(cipher.
12              doFinal(strToEncrypt.getBytes("UTF-8")));
13          } catch (Exception ex) {
14              ex.printStackTrace();
15          }
16          return null;
```

```
17      }
```

Pomocná třída *Generate* slouží ke generování náhodného pole bytů potřebného k vytvoření inicializačního vektoru. Počet generovaných bytů je potřeba nastavit přes vstupní argument. Generování potřebných bytů obstarává třída SecureRandom. Dále třída *Generate* obsahuje ještě 2 podpůrné metody na změnu datových typů pro práci s inicializačním vektorem a jeho zobrazením uživateli.

Pro ukládání vstupních parametrů z formuláře slouží objekt ze třídy *EncryptForm*. Třída obsahuje atributy, konstruktor s výchozími hodnotami a odpovídající gettery a settery. Atributy této třídy jsou vidět na následujícím výpisu.

Výpis 4.5: Proměnné třídy EncryptFrom

```
1     private String msg;
2     private String algorithm;
3     private String secret;
4     private String cryptedMsg;
5     private String mode;
6     private String action;
7     private Boolean debug;
8     private String iva;
9     private String ivaFile;
10    private String cryptedFile;
11    private String decryptedFile;
```

Závěr

Bakalářská práce s názvem Webová aplikace pro šifrování souborů pojednává o symetrických šifrovacích algoritmech a operačních módech.

V úvodu práce bylo popsáno co je to symetrický šifrovací algoritmus, jak funguje a v čem se liší od asymetrických šifer. Dále bylo popsáno dělení symetrických šifrovacích algoritmů na blokové a proudové a v čem se liší.

Vybrané šifry byly detailně popsány a doplněny obrázkovým schématem. Algoritmy byly srovnány a v tabulce byly uvedeny jejich rozdílné parametry. Jednalo se zejména o typ šifry, strukturu, délku klíče, velikost bloku a počet rund ve kterých operují.

Další část práce se věnovala různým operačním módům, ve kterých moderní symetrické blokové šifry operují. Operační módy byly popsány a zobrazeny na obrázcích. Uvedeno bylo také jak ovlivňují proces šifrování. Zvláště u vybraných módů jejich vlastnost, která mění způsob šifrování na proudovou šifru.

V praktické části byla na základě teoretické části vytvořena webová aplikace. Ta umožňuje praktické vyzkoušení šifrování jak otevřeného textu, tak souborů. Na výběr má uživatel několik šifrovacích algoritmů i operačních módů. Praktická funkcionalita je doplněna informativní částí, kde si uživatel může přečíst stručné informace o šifrovacích algoritmech i operačních módech.

Aplikace byla sestavena pro účel výuky a nebyl kladen důraz na bezpečnost jejího provozování. Mezi slabiny aplikace patří ukládání souborů při procesu šifrování a dešifrování na straně serveru. Ty jsou před vlastní kryptografickou operací vždy uloženy do složky na straně serveru. Z tohoto umístění se provádí jejich šifrování nebo dešifrování a výsledek je uložen do složky downloads. V reálném nasazení by bylo vhodné aplikaci upravit tak, aby k ukládání zdrojových souborů nedocházelo, zejména aby nebyl soubor v otevřené podobě anonymně dostupný. Šifrování by mohlo probíhat na úrovni objektu http request. Dále by mělo být aktivováno zabezpečení protokolu HTTP vrstvou SSL/TLS, aby byl zabezpečen přenos hesel použitých v aplikaci.

Literatura

[1] *Symmetric Ciphers* [online]. [cit. 2020-11-07]. Dostupné z: <https://brilliant.org/wiki/symmetric-ciphers/>.

[2] *Block Cipher* [online]. [cit. 2020-11-08]. Dostupné z: <https://www.tutorialspoint.com/cryptography/block_cipher.htm>.

[3] *Stream Ciphers vs. Block Ciphers* [online]. 2015 [cit. 2020-11-08]. Dostupné z: <https://www.jscape.com/blog/stream-cipher-vs-block-cipher>.

[4] *Types of stream ciphers* [online]. [cit. 2020-11-08]. Dostupné z: <https://en.wikipedia.org/wiki/Stream_cipher#Types>.

[5] *FIPS 46-3, Data Encryption Standard* [online]. 1999 [cit. 2020-11-17]. Dostupné z: <https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf>.

[6] *Data Encryption Standard* [online]. [cit. 2020-11-17]. Dostupné z: <https://en.wikipedia.org/wiki/Data_Encryption_Standard>.

[7] *FIPS 197, Advanced Encryption Standard (AES)* [online]. 2001 [cit. 2020-12-02]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.

[8] Advanced Encryption Standard. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-27]. Dostupné z: <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>.

[9] *PRESENT: An Ultra-Lightweight Block Cipher* [online]. Bochum, 2007 [cit. 2020-12-05]. Dostupné z: <http://www.lightweightcrypto.org/present/present_ches2007.pdf>.

[10] PRESENT. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-27]. Dostupné z: <https://en.wikipedia.org/wiki/PRESENT>.

[11] MENEZES, Alfred J., Paul C. van OORSCHOT a Scott A. VANSTONE. *Handbook of Applied Cryptography*. CRC Press, 1997. ISBN 978-0-84-938523-0.

[12] International Data Encryption Algorithm. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-27]. Dostupné z: <https://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm>.

[13] *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)* [online]. [cit. 2020-12-08]. Dostupné z: <https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html>.

[14] Blowfish (cipher). *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-12-8]. Dostupné z: <https://en.wikipedia.org/wiki/Blowfish_(cipher)>.

[15] FERGUSON, Niels a Bruce SCHNEIER. *Practical Cryptography*. Indianapolis: Wiley, 2003. ISBN 978-0-471-22357-3.

[16] Twofish. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-5-27]. Dostupné z: <https://en.wikipedia.org/wiki/Twofish>.

[17] *Serpent (cipher)* [online]. San Francisco [cit. 2020-12-10]. Dostupné z: <https://en.wikipedia.org/wiki/Serpent_(cipher)>.

[18] *RC4* [online]. San Francisco [cit. 2020-12-10]. Dostupné z: <https://en.wikipedia.org/wiki/RC4>.

[19] RC4 Encryption Algorithm. *GeeksforGeeks* [online]. 2019 [cit. 2020-12-10]. Dostupné z: <https://www.geeksforgeeks.org/rc4-encryption-algorithm/>.

[20] PAAR, Christof a Jan PELZL. *Understanding cryptography: a textbook for students and practitioners*. 2nd corrected printing 2010. Heidelberg: Springer, c2010. ISBN 978-3-642-04100-6.

[21] Block cipher mode of operation. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 19.4. 2021n. l.]. Dostupné z: <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>.

[22] Initialization vector. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-04-19]. Dostupné z: <https://en.wikipedia.org/wiki/Initialization_vector>.

[23] *Recommendation for Block Cipher Modes of Operation* [online]. Gaithersburg, 2001 [cit. 2021-4-25]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>.

[24] *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC* [online]. Gaithersburg, 2007 [cit. 2021-4-30]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>.

[25] Galois/Counter Mode. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-4-30]. Dostupné z: <https://en.wikipedia.org/wiki/Galois/Counter_Mode>.

[26] MVC Framework - Introduction. *Tutorialspoint* [online]. [cit. 2021-5-30]. Dostupné z: <https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm>.

[27] Thymeleaf. *Thymeleaf* [online]. [cit. 2021-5-30]. Dostupné z: <www.thymeleaf.org>.

[28] Web MVC Framework. *Spring* [online]. [cit. 2021-5-30]. Dostupné z: <https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>.

Seznam symbolů a zkratek

DES	Data Encryption Standard
IP	Initial permutation
3DES	Triple DES
AES	Advanced Encryption Standard
NIST	National Institute of Standards and Technology
GF	Galois field
IDEA	International Data Encryption Algorithm
PHT	Pseudo-Hammard Transform
MDS	Maximum Distance Separable
RSA	Rivest-Shamir-Adleman, asymetrický šifrovací algoritmus
KSA	Key Scheduling Algorithm
PRGA	Pseudo-Random Generation Algorithm
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
TLS	Transport Layer Security
SSH	Secure Shell Protocol
ECB	Electronic Codebook
CBC	Cipher Block Chaining
IV	Inicializační Vektor
Nonce	number used once
OFB	Output Feedback
CTR	Counter
GCM	Galois Counter Mode
MAC	Message Authencication Code

HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
PBKDF2	Password-Based Key Derivation Function 2
SSL	Secure Sockets Layer
TLS	Transport Layer Security