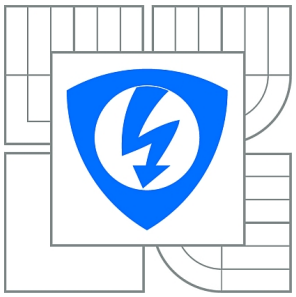
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

PROGRAM (.NET/C#) PRO VÝUKU A VYSVĚTLENÍ FUNKCE ŠIFRY AES

.NET/C# PROGRAM FOR TEACHING AND EXPLAINING THE FUNCTION OF AES CIPHER

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

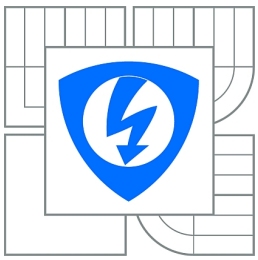
AUTOR PRÁCE — Bc. MARTIN ONDREJECH

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL BURDA, CSc.

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Martin Ondrejech	***ID:***	70243
Ročník:	2	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Program (.NET/C#) pro výuku a vysvětlení funkce šifry AES

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je vytvořit program, který pomocí animací a vhodných grafických prvků názorně vysvětlí funkci šifry AES ve všech krocích a iteracích počínaje upravením vstupní zprávy až po odeslání výsledného kryptogramu. Program by měl znázornit všechny mezivýsledky v průběhu šifrování a také tok dat při různých módech šifry AES. Cílem není vytvořit šifrovací program, ale program pro znázornění funkce šifry AES.

DOPORUČENÁ LITERATURA:

[1] DAEMEN Joan. The Design of RijndaeL: AES - The Advanced Encryption Standard. Springer, 2002. 255s. ISBN: 978-3540425809
[2] TROELSEN Andrew. Pro C# 5.0 and the .NET 4.5 Framework. Apress, 2012. 1560s. ISBN: 978-1430242338

Termín zadání:	11.2.2013	***Termín odevzdání:***	29.5.2013

Vedoucí práce: doc. Ing. Karel Burda, CSc.
Konzultanti diplomové práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Klíčová slova

AES, Advanced Encryption Standard, .NET, WPF, Windows Presentation Foundation, šifrování, C#, Visual Studio

Keywords

AES, Advanced Encryption Standard, .NET, WPF, Windows Presentation Foundation, cipher, C#, Visual Studio

Abstrakt

Tato práce pojednává o šifrovacím algoritmu AES, způsobu jeho výběru a popisu jednotlivých bloků této symetrické blokové šifry, možnostech vývojového prostředí .NET framework a současné implementace algoritmu AES v programovacím jazyce C#. Součástí práce je aplikace pro výuku a vysvětlení funkce šifry AES.

Abstract

This work is focused on cipher algorithm AES, his selection method and description of cipher blocks, possibilities of .NET framework and current implementation of AES algorithm in C#. The book also contain complete application for teaching and explanation of AES cipher.

Bibliografická citace

ONDREJECH, M. Program (.NET/C#) pro výuku a vysvětlení funkce šifry AES. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2013. 47 s. Vedoucí diplomové práce doc. Ing. Karel Burda, CSc..

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Program (.NET/C#) pro výuku a vysvětlení funkce šifry AES“ jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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 následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (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.

V Brně dne 29.05.2013 .....

podpis autora

Poděkování

Děkuji vedoucímu práce doc. Ing. Karlovi Burdovi, CSc. za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne 29.05.2013 .....

podpis autora

Obsah

Úvod

Tato práce pojednává o šifrovacím algoritmu Advanced Encryption Standard a o možnostech naprogramování aplikace pro výuku a vysvětlení funkce této šifry na platformě .NET.

V první části práce je popsán samotný standard AES, jeho historie, funkce, a jsou zde detailněji popsány jednotlivé bloky šifry.

V druhé části je stručně popsána platforma .NET framework a její možnosti, popis programovacího jazyka C# a jeho nejdůležitější funkce.

Samostatná kapitola je věnována již existujícím implementacím algoritmu AES v C#, jejich popisu a možnostech použití.

V třetí části diplomové práce jsou popsány jednotlivé části programu, jejich funkce a význam.

Na přiloženém DVD jsou umístěny zdrojové kódy aplikace pro výuku a vysvětlení funkce šifry AES, návod na používání aplikace a elektronická verze diplomové práce.

1 Advanced Encryption Standard

1.1 Historie

Na začátku roku 1997 oznámil NIST začátek vývoje nového šifrovacího standardu nazvaného Advanced Encryption Standard. AES mělo podle plánu postupně nahradit dosavadně používané standardy DES a triple-DES a stát se součástí Federal Information Processing Standard. Označení AES v tuto chvíli není označení pro konkrétní šifrovací algoritmus, ale pouze standardu, pro který bude konkrétní algoritmus teprve vybrán [1].

V procesu vybírání vhodného algoritmu pro AES bylo použito zajímavé novinky. Na rozdíl od DES, SHA-1 nebo DSA bylo přistoupeno k plně otevřenému výběru [1]. To znamená, že kdokoliv může navrhnout šifru, která splňuje minimální požadavky NISTu, jako kandidáta na to, stát se oficiálně AES.

Přestože uznání AES jakožto FIPS standardu samo o sobě nic neznamená, protože FIPS je závazný pouze pro administrativu Spojených států amerických, jeho schválení jakožto standardu v USA mu dodalo punc kvality a následně bylo AES schváleno jako standard také organizacemi ISO, IETF a IEEE, což umožňuje jeho celosvětové použití.

1.1.1 Požadavky na kandidáty AES

V polovině roku 1997 zveřejnil NIST konečné požadavky, které musí kandidáti splňovat, aby byly zahrnuti do výběru. Šifra musí podporovat délku datových bloků 128 bitů a délku šifrovacího klíče 128, 192 nebo 256 bitů. Každá šifra musí být také minimálně stejně bezpečná jako už existující triple-DES, ale musí být zároveň mnohem efektivnější, aby se dala jednoduše implementovat na různých systémech od čteček karet s omezenou výpočetní silou po výkonné procesory typu x86. Každý ze zájemců o to stát se součástí AES musel souhlasit s tím, že pokud bude vybrán, poskytne svůj šifrovací algoritmus k dispozici pro použití bez licenčních poplatků [1].

Další nezbytné podmínky pro to dostat se do výběru byly:

- kompletní specifikaci blokové šifry ve formě algoritmu
- referenční implementaci algoritmu v jazyce ANSI C a matematicky optimalizovanou implementaci v jazycích ANSI C a JAVA
- výsledky Known Answer Tests a Monte Carlo Tests
- prohlášení o předpokládané hardwarové a softwarové výpočetní účinnosti, předpokládané síle proti kryptoanalytickým útokům a výhody a nevýhody konkrétní blokové šifry v různých prostředích
- analýza jak je šifra silná proti známým kryptoanalytickým útokům [1]

1.1.2 Výběr finálního algoritmu

Požadavky stanovené NISTem nakonec splnilo 15 šifer, které byly uznány jako oficiální kandidáti na AES. Byly to CAST-256, Crypton, DEAL, DFC, E2, Frog, HPC, LOKI97, Magenta, Mars, RC6, Rijndael, SAFER+, Serpent, Twofish.

Vybírání vítěze bylo rozděleno do dvou kol a přes tři AES konference. První kolo začalo na The First Advanced Encryption Standard Candidate conference, konané od 20. do 22. srpna 1998 ve Ventuře v Kalifornii. Zde byly představeni všichni kandidáti a kryptografická komunita byla požádána o zhodnocení a okomentování všech kandidátů v průběhu příštích měsíců.

Hodnocení jednotlivých šifer bylo rozděleno do tří hlavních kategorií: bezpečnost, náklady a charakteristika algoritmu a jeho implementace.

Bezpečnost byla jedna z nejdůležitějších kategorií pro hodnocení jednotlivých šifer, ale zároveň jedna z nejobtížněji hodnotitelných, protože většina šifer se ukázala patřit do kategorie bez známého slabého místa. Pouze FROG, Magenta, LOKI97 a DEAL ukázaly nedostačující bezpečnostní požadavky, co se týče kryptografických útoků a u šifry HPC byla demonstrována slabina v listu zaslaném komunitou do NISTu [1].

Další důležitou kategorií v hodnocení kandidátů byly náklady spojené s implementací a provozem jednotlivých šifer. Kritéria pro hodnocení nákladů byly výpočetní náročnost, velikost výsledného programu, náročnost na paměť a velikost čipu pro hardwarovou implementaci. Součástí této kategorie bylo také potvrzení od každého předkladatele, že uvolní svou šifru k volnému použití v případě, když bude vybrána jako AES, a zároveň si nebudou nárokovat vlastnictví nebo patenty na nápady použité v konkurenčních šifrách, které mohou být vybrány do AES [1].

Poslední hlavní kategorií byla charakteristika algoritmu a jeho implementace zaměřená na:

- **Univerzálnost**: označuje, jak účinně dokáže být šifra implementována na různých platformách od smart card adaptérů s omezenou datovou pamětí a malou RAM, přes různorodé procesory používané v PC, tabletech, smartphone (x86, ARM atd.) až po speciální hardware který musí provádět šifrování a dešifrování za běhu v řádech gigabitů za sekundu [1].
- **Key agility**: vyjadřuje, jak rychle dokáže algoritmus ustanovit klíč potřebný k šifrování nebo dešifrování. Tato vlastnost je velmi důležitá v aplikacích, kde dochází k častému měnění klíče, například protokol šifrování paketů na protokolu IP v IPSEC [1].
- **Jednoduchost**: definuje, jak složitý je popis jednotlivých šifer, kolik různých operací je potřeba použít, zda je šifra symetrická nebo naopak nesymetrická a jak jednoduše může být algoritmus použitý v šifře porozuměn [1].

Konec prvního kola výběru se přiblížil na druhé AES konferenci konané v Římě v březnu roku 1999. Na této konferenci byly diskutovány různé kryptografické útoky, analýzy a různé další různé srovnání jednotlivých kandidátů. Výsledkem této konference bylo oznámení pěti kandidátů postupujících do druhého kola v srpnu roku 1999 a byly jimi: MARS, RC6, Rijndael, Serpent a Twofish [2].

Po vyhlášení pěti finálních kandidátů na to stát se oficiálním standardem AES začalo druhé kolo volby. Všichni postupující kandidáti prošli dalším zhodnocením, zaměřeným tentokráte hlavně na intelektuální vlastnictví jednotlivých šifer, jejich výkon a velikost potřebných čipů v dedikovaném hardwaru. V otázce hardwarového výkonu hodně pomohla agentura NSA, která provedla nezávislé simulace pro všechny kandidáty [1]. Výsledky hodnocení a došlých příspěvků byly prezentovány a komunikovány na třetí AES konferenci v New Yorku v dubnu roku 2000.

Na konferenci byly diskutovány výsledky důkladnějších kryptografických útoků na jednotlivé kandidáty, ale žádný z nich neprokázal slabinu v kompletním algoritmu. Většina z prezentovaných útoků byla pouze na verze algoritmů s redukovanými počty rund a měly pouze akademický význam, protože pro použití v praxi vykazovaly velkou časovou a výpočetní náročnost [1].

V oblasti implementace na dedikovaném hardwaru bylo zaměření na použití v ASICs a FPGAs, kde nejlepších výsledků dosahovala šifra Serpent, následována Rijndael a Twofish. Náklady na RC6 už vyšly vyšší, protože tahle šifra používá 32 bitové násobení. Šifra MARS se, co se týče nákladů pro dedikovaný hardware, ukázala být jako nejhorší ze všech [1].

Na konci třetí AES konference proběhlo veřejné hlasování všech účastníků konference, jakou šifru preferují a nejvíce hlasů získal Rijndael. 2. října 2000 potom NIST oficiálně potvrdil, že standardem AES se stane šifra Rijndael, bez jakýchkoliv dalších modifikací se zdůvodněním:

Rijndael vypadá trvale výkonný jak v hardwarové, tak v softwarové implementaci napříč širokou škálou výpočetních prostředí. Čas potřebný k ustanovení šifrovacího klíče je excelentní a key agility dobrý. Požadavky na pamět jsou velmi nízké, a i přesto Rijndael dokáže dosáhnout excelentních výkonů v prostředích s omezenou pamětí. Operace použité v algoritmu Rijndael poskytují jednoduchou obranu před proudovými a časovými útoky, navíc se zdá, že dodatečná obrana proti takovýmto útokům může být přidána bez výraznějších dopadů na Rijndaelův výkon [3].

1.2 Šifrovací klíč

Standard AES definuje délku klíče K 128, 192 nebo 256 bitů definovanou jako $N_k = 4$, 6 nebo 8, kde číslo reprezentuje počet 32 bitových slov v klíči. Na délce klíče N_k

je v algoritmu AES přímo závislý počet rund N_r nutných k vykonání celého procesu. Závislost mezi N_k a N_r viz Tab. 1.1.

Tab. 1.1: Závilost počtu rund N_r na délce klíče N_k

	Délka klíče N_k	**Počet rund N_r**
AES-128	4	10
AES-192	6	12
AES-256	8	14

1.2.1 Expanze šifrovacího klíče

Každá z rund v průběhu šifrování vyžaduje přidání rundovního klíče o velikosti N_k pro zašifrování výstupu každé rundy. Protože použití jednoho společného klíče pro všechny rundy by z bezpečnostního hlediska postrádalo smysl, je potřeba z původního klíče K vygenerovat dalších $N_b * (N_r + 1)$ slov, které jsou potom rozděleny po 4 bytech a uloženy do pole w_i, kde i je v rozsahu $0 \leq i < N_b * (N_r + 1)$. Schéma expanze šifrovacího klíče pro 128 a 192 bitový klíč je znázorněna na obr. 1.1.

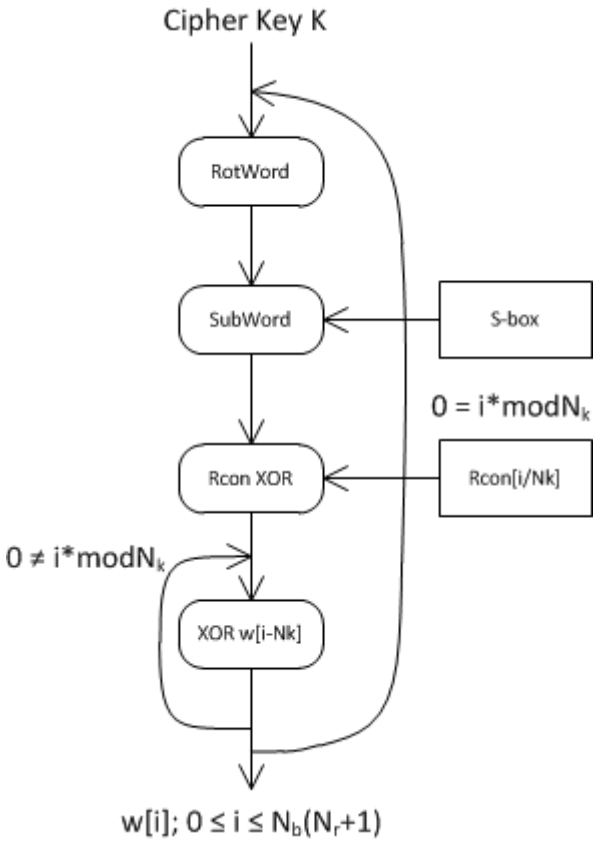
Proces pro expanzi 256 bitového klíče je trochu odlišný. V každé rundě, pro kterou platí, že $i - 4$ je násobek N_k, je aplikována funkce SubWord na slovo $w[i-1]$ před operací XOR s $w[i-N_k]$ [4].

RotWord

Funkce RotWord vezme na vstupu slovo $w[i-1] = [a_0,a_1,a_2,a_3]$ a po provedení cyklické permutace vrátí na výstupu čtyř bytové slovo $[a_1,a_2,a_3,a_0]$ [4].

SubWord

Funkce SubWord vezme 4 bytový výstup z funkce RotWord a aplikuje substituci podle substitučního boxu definovaného pro AES.



Obr. 1.1: Expanze šifrovacího klíče pro AES 128 a 192

Rcon

Rundovní konstanta Rcon je pole sestávající z hodnot $[x^{i-1},\{00\},\{00\},\{00\}]$, kde x^{i-1} je mocnina x = {02} v konečném poli GF(2^8) v polynomiální formě (2 = 0x7 + 0x6 + 0x5 + 0x4 + 0x3 + 0x2 + 1x + 0 = x). Index i musí splňovat podmínku $1 \leq i$. Potom

$$Rcon[i] = x^{i-1} mod x^8 + x^4 + x^3 + x + 1 \text{ [4]}. \quad (1.1)$$

Seznam všech přípustných hodnot pro standard AES viz tab. 1.2.

Tab. 1.2: Hodnoty možných rundovních konstant pro standard AES

i	Rcon[i]
1	01000000
2	02000000
3	04000000
4	08000000
5	10000000
6	20000000
7	40000000
8	80000000
9	1b000000
10	36000000

Výstupní slovo w[i] je vždy tvořeno jako výsledek funkce XOR předchozího slova w[i−1] a slova w[i−N_k]. Pouze pokud je runda i násobkem N_k, aplikuje se na slovo w[i−1] před funkcí XOR transformace sestávající z prohození bytů RotWord, nahrazení bytů SubWord a přidání rundovní konstanty Rcon. Na výsledné slovo je potom teprve použita funkce XOR se slovem w[i−N_k].

Příklad výpočtu expanze 128 bitového klíče pro počáteční šifrovací klíč K = 16a3905c81fc43d7247bc66de7da35bf je v tab. 1.3.

Tab. 1.3: Výpočet expanze 128 bitového klíče

i		RotWord	SubWord	Rcon	XOR Rcon	w[i]
0						16a3905c
1						81fc43d7
2						247bc66d
3						e7da35bf
4	e7da35bf	da35bfe7	57960894	01000000	56960894	403598c8
5	403598c8					c1c9db1f
6	c1c9db1f					e5b21d72
7	e5b21d72					026828cd
8	026828cd	6828cd02	4534bd77	02000000	4734bd77	070125bf
9	070125bf					c6c8fea0
10	c6c8fea0					237ae3d2
11	237ae3d2					2112cb1f
12	2112cb1f	12cb1f21	c91fc0fd	04000000	cd1fc0fd	ca1ee542

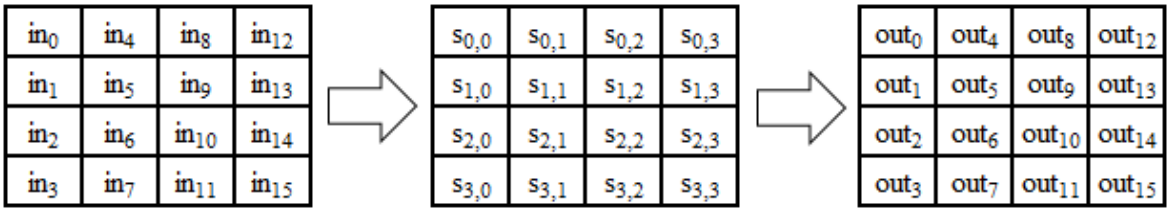
13	ca1ee542					0cd61be2
14	0cd61be2					2facf830
15	2facf830					0ebe332f
16	0ebe332f	be332f0e	aec315ab	08000000	a6c315ab	6cddf0e9
17	6cddf0e9					600beb0b
18	600beb0b					4fa7133b
19	4fa7133b					41192014
20	41192014	19201441	d4b7fa83	10000000	c4b7fa83	a86a0a6a
21	a86a0a6a					c861e161
22	c861e161					87c6f25a
23	87c6f25a					c6dfd24e
24	c6dfd24e	dfd24ec6	9eb52fb4	20000000	beb52fb4	16df25de
25	16df25de					debec4bf
26	debec4bf					597836e5
27	597836e5					9fa7e4ab
28	9fa7e4ab	a7e4ab9f	5c6962db	40000000	1c6962db	0ab64705
29	0ab64705					d40883ba
30	d40883ba					8d70b55f
31	8d70b55f					12d751f4
32	12d751f4	d751f412	0ed1bfc9	80000000	8ed1bfc9	8467f8cc
33	8467f8cc					506f7b76
34	506f7b76					dd1fce29
35	dd1fce29					cfc89fdd
36	cfc89fdd	c89fddcf	e8dbc18a	1b000000	f3dbc18a	77bc3946
37	77bc3946					27d34230
38	27d34230					facc8c19
39	facc8c19					350413c4
40	350413c4	0413c435	f27d1c96	36000000	c47d1c96	b3c125d0
41	b3c125d0					941267e0
42	941267e0					6edeebf9
43	6edeebf9					5bdaf83d

1.3 Šifrování

Na začátku každého šifrování je vždy vstupní text rozdělen na bloky o velikosti Nb = 4, kde číslo 4 znamená počet 32 bitových slov, což je dohromady blok o délce 128 bitů. Tento vstupní blok potom odpovídá vstupním bytům in_i pro $0 \leq i < (N_b * 4)$. Vstupní byty jsou poté zkopírovány do pole State, což je dvourozměrné pole bytů sestávající ze čtyř řádků r a N_b sloupců c. Délka bloku N_b je pro AES pevně stanovena jako 4 přímo standardem, tudíž výsledná velikost pole $s_{r,c}$ je 4x4. Vstupní byty jsou kopírovány do pole State podle vzorce:

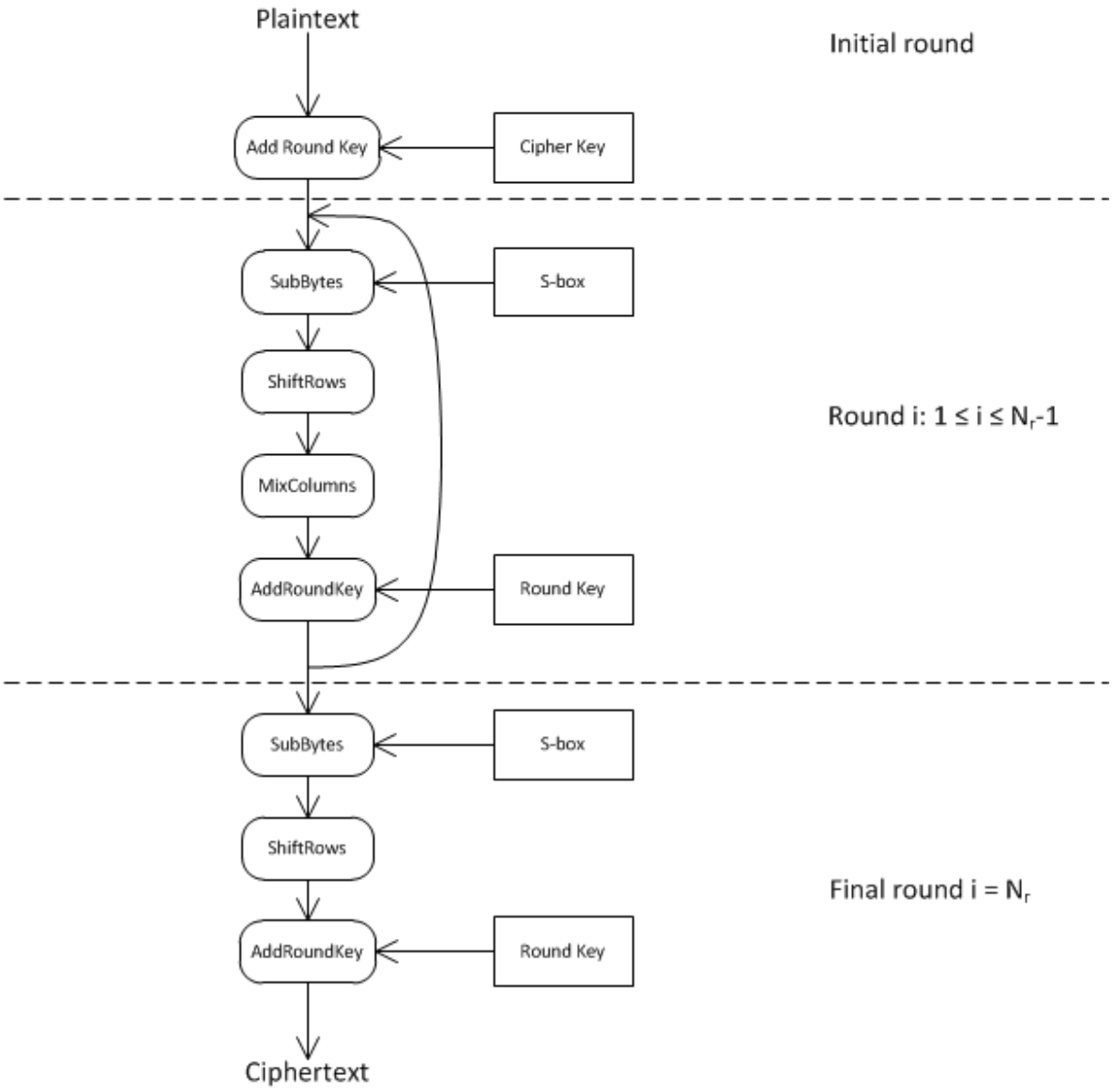
$$s_{r,c} = in[r + 4c] \text{ pro } 0 \leq r < 4 \text{ a } 0 \leq c < N_b \quad (1.2)$$

K tomuto poli je následně přidán šifrovací klíč K a jsou na něm provedeny transformace jednotlivých rund. Výsledné pole po provedení N_r rund je poté zkopírováno jako výstup out_i pro $0 \leq i < (N_b * 4)$ jak je znázorněno na obr. 1.2.



Obr. 1.2: Proces kopírpvání vstupních bytů in na pole State a na výstupní byty out

V každé rundě je na pole State uplatněn soubor transformací, jak je znázorněno na obrázku 3. Poslední runda N_r je mírně odlišná od předchozích N_{r-1} rund. V posledním kole se už neprovádí operace MixColumns, protože rozptyl který tato funkce přidává, by zde již neměl žádný vliv na bezpečnost, protože by se již nepředal do další rundy, která by ho mohla využít a pouze by průběh šifrování zpomalil.



Obr. 1.3: Schéma algoritmu AES

1.3.1 SubBytes

Funkce SubByte je nelineární bytová substituce prováděná nad každým bytem nezávisle. Pro provedení substituce se využívá záznamů v substituční tabulce, což je dvourozměrné pole zapisované zpravidla v hexadecimálním tvaru jako v tabulce 4. Samotná transformace probíhá tak, že z pole State vezmeme jedno pole v hexa tvaru $s_{r,c}$ = {XY} kde X označuje číslo řádku v substituční tabulce a Y číslo sloupce. V průsečíku těchto dvou hodnot se nachází nová hodnota, která bude zkopírována do původního State. Například pokud se v původním poli nachází hodnota {6b}, pak vezmeme průsečík šestého řádku se sloupcem b a nová hodnota podle substituční tabulky bude {7f}, viz tab. 1.4.

Tab. 1.4: S-box hodnoty v hexadecimálním tvaru

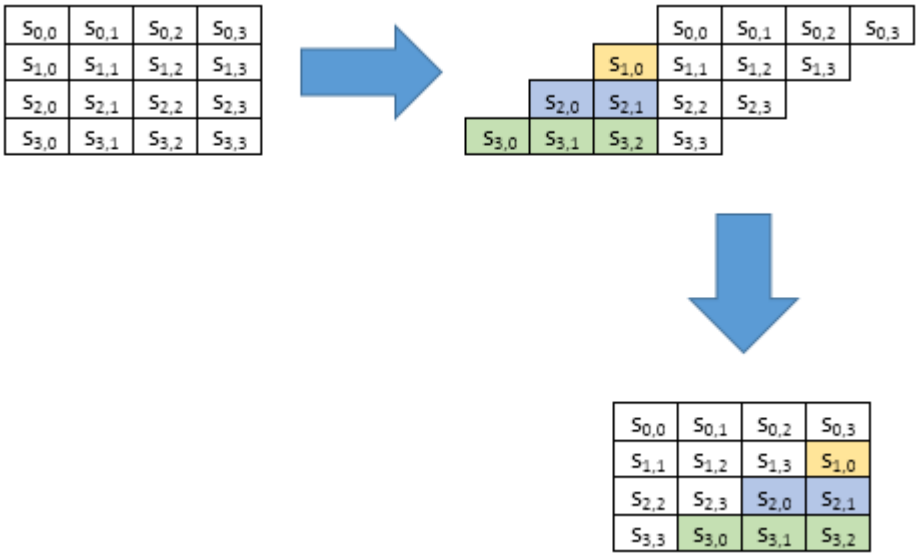
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

1.3.2 ShiftRows

Funkce ShiftRows zařídí posunutí bytů v posledních třech řádcích o různý ofset. Počet sloupců o který je byte $s_{r,c}$ posunut doleva závisí na indexu řádku r. Pokud r = 0 tak nedochází k žádnému posunu, při r = 1 je jeden byt posunut o jedno místo doleva, až po r = 3, kdy je posun o tři byty, viz obr. 1.4. Matematicky vyjádřeno v rovnici (1.3).

$$s'_{r,c} = s_{r,(c+r) mod N_b} \text{ pro } 0 < r < 4 \text{ a } 0 \leq c < N_b \qquad (1.3)$$

Aplikací této transformace dosáhneme přesunutí bytů z vyšších pozic na nižší, zatímco u bytů na nejnižších pozicích dojde k zalomení na pozice vyšší.

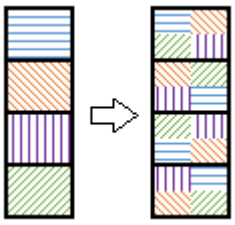


Obr. 1.4: Průběh funkce ShiftRows

1.3.3 MixColumns

Operace MixColumns se provádí na každém sloupci v poli State a slouží k rozptylu bitů z jednoho bytu mezi všechny ostatní byty daného sloupce, jak je znázorněno na obr. 1.5. Pro potřeby funkce MixColumns, považujeme každý sloupec pole State jako polynom GF(2^8). Každý sloupec je potom vynásoben polynomem a(x) podle rovnice (1.4)

$$s'(x) = a(x) \otimes s(x), \tag{1.4}$$



Obr. 1.5: Princip přidání rozptylu bitů funkcí MixColumns

kde $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Po vynásobení polynomem a(x) musí být výsledek ještě zredukován operací mod($x^4 + 1$), aby bylo zaručeno, že výsledný polynom bude menšího stupně než 4 a bude tudíž patřit do konečného pole GF(2^8).

1.3.4 AddRoundKey

Transformace AddRoundKey přidává k poli State pomocí operace XOR rundovní klíč, který byl vygenerován z šifrovacího klíče K pomocí expanze šifrovacího klíče. Každý rundovní klíč w sestává z N_b slov, které jsou přičítany k jednotlivým sloupcům podle (1.5).

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{i*N_b+c}]$$
$$\text{pro } 0 \leq c < N_b, 0 \leq i \leq N_c \tag{1.5}$$

1.3.5 Dešifrování

Každý zašifrovaný text může být dešifrován pomocí inverzní šifry AES algoritmu, využívající lehce odlišné funkce InvShiftRows, InvSubBytes, InvMixColumns od šifrovacího algoritmu a společnou funkci AddRoundKey.

Jako první je na vstupní šifrovaný text, označovaný jako State, aplikována transformace AddRoundKey, která má stejnou funkci a pracuje úplně stejně jako při šifrování. Jako druhý parametr tato funkce přebírá expandovaný šifrovací klíč, který ale narozdíl od šifrování přičítá pomocí operace XOR od konce w[N_r] po začátek w[0].

Druhou transformací v pořadí je funkce ShiftRows, kdy byty v posledních třech řádcích pole State jsou posunuty o různý offset podle rovnice (1.6).

$$s'_{r,(c+r)modN_b} = s_{r,c} \text{ pro } 0 < r < 4 \text{ a } 0 \leq c < 4 \tag{1.6}$$

Třetí transformací pro dešifrování je funkce InvSubBytes, která má stejnou funkcionalitu jako transformace SubBytes z šifrování popsaná v kapitole 1.3.1. Jediný rozdíl je, že na rozdíl od SubBytes používá jako substituční tabulku inverzní S-box, viz tab. 1.5.

Tab. 1.5: Inverzní substituční tabulka

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Poslední transformací používanou v dešifrovacím algoritmu je InvMixColumns. Tato funkce postupně násobí všechny sloupce pole State s fixním polynomem $a^{-1}(x)$ podle vztahu (1.7).

$$s'(x) = a^{-1}(x) \otimes s(x) \tag{1.7}$$

Polynom $a^{-1}(x)$ je pevně definován jako:

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.$$

Pro správný výsledek je ještě potřeba na výsledném polynomu $s'(x)$ provést operaci modulo $x^4 + 1$.

2 .NET Framework

.NET Framework je vývojové prostředí primárně určené pro programování aplikací v operačním systému Microsoft Windows. Jeho první verze 1.0 byla uvedena v roce 2002 spolu s prvními vývojovými nástroji (Visual Studio .NET a .NET Framework SDK) [10]. V současné době je poslední verzí .NET Framework 4.5 vydaný v roce 2012 spolu s nejnovější verzí IDE Visual Studio 2012.

Primárním účelem vývoje celé nové platformy .NET bylo odstranění různých problémů a nepříjemností spojených se staršími možnostmi programování pro Windows, jako bylo používání COM a Windows API, kdy hlavní nevýhodou byla vzájemná nekompatibilita různých verzí softwaru a jejích dynamických knihoven, taktéž známá jako DLL hell.

Ve starších verzích platformy .NET představovala pouhou nádstavbu operačního systému Windows, nicméně od verze 3.0 a operačního systému Windows Vista jsou již plně součástí operačních systémů [10]. Jednotlivé součástí platformy .NET například jsou:

- **Společný běhový systém CLR** je základ platformy .NET, který definuje např. společný typový systém nebo automatickou správu paměti (garbage collector). Program v CLR běží v takzvaném mezijazyku IL a obsahuje JIT překadač, který vždy převede kód z IL do nativního kódu procesoru na kterém se zrovna program spouští.
- **Základní knihovna tříd BCL** definuje základní datové typy, jako je System.String, rozhraní pro práci se soubory, interakci s databázemi nebo práci s grafikou.
- **ADO.NET** je knihovna sloužící pro prácí s databázemi.
- **WPF**
- **LINQ** je dotazovací jazyk přidaný do .NET ve verzi 3.5 a umožňuje vyhledávat výsledky z jakéhokoliv objektu třídy, která implementuje rozhraní IEnumerable. Struktura dotazu LINQ je podobná zápisu v jazyku SQL.
- **Programovací jazyky.** Platforma .NET umožňuje vyvíjet programy v různých programovacích jazycích. Jsou to například C#, VB.NET, F#, C++/CLI a jiné.

2.1 C#

C# je objektový programovací jazyk určený pro vývoj na platformě .NET. Tento jazyk je vyvíjen společností Microsoft dohromady se samotnou platformou .NET a tudíž disponuje největšími možnostmi využití všech výhod .NET [10]. Syntaxe jazyka

je odvozena od programovacího jazyka C++ a některá funkčnost jazyka byla inspirována programovacím jazykem Java (např. garbage collector).

2.1.1 Nejdůležitější změny v průběhu vývoje jazyka C#

V průběhu vývoje jednotlivých verzí jazyka C# došlo k významným vylepšením některých funkcí a chování C#. Níže jsou vypsány nejdůležitější z nich podle verze, ve které byly přidány.

Genericita

Genericita byla přidána do jazyka C# od verze 2.0 a umožňuje definovat kontejnery také s abstraktními datovými typy <T>. U takto definovaného kontejneru se můžeme až při deklaraci rozhodnout jaký datový typ bude zrovna použit.

Parciální třídy

Parciální třídy přibyly ve verzi 2.0 a umožňují rozdělit zápis třídy do více souborů uvedením klíčového slova *partial* při deklaraci třídy.

Statické třídy

Statické třídy jsou v C# přítomny od verze 2.0 použitím klíčového slova *static* při deklaraci třídy. Statická třída smí obsahovat pouze statické metody a nemůže mít žádné potomky.

Implicitně typované lokální proměnné

Od verze 3.0 přibyla možnost nechat překladač odvodit z typu inicializátoru, jakého typu bude lokálně deklarovaná proměnná. Použitím klíčového slova var místo datového typu sdělíme překladači, aby si odvodil, jakého typu bude výsledná proměnná podle hodnoty, kterou do ní přiřazujeme. Například oři zápisu *var cislo = 6* bude proměnná *cislo* typu integer.

Automatická implementace vlastností

Automatická implementace vlastností, přidaná ve verzi 3.0, umožňuje zkrácený zápis přístupových metod *set* a *get*, kdy stačí pouze zkáceně uvést která z těchto metod má být implementována, případně obě.

```
public class Cipher{
string Key { get; set;}
}
```

Lambda výrazy

Od verze C# 3.0 jsou přítomny lambda výrazy, které můžeme využít, když potřebujeme zavolat nějakou jednoduchou metodu. Lambda výraz je funkcí jedné nebo více proměnných a je volán tak, že za identifikátorem proměnné použijeme operátor => a výraz, který má být výsledkem lambda výrazu.

```
i => i+i
```

Nepovinné parametry

Od verze C# 4.0 můžeme v deklaraci metody uvádět implicitní hodnoty parametrů, které se tak stanou nepovinnými a můžeme je při volání metody vynechat. Překladač si automaticky dosadí implicitně definovanou metodu.

2.1.2 AES v .NET C#

Od verze .NET Framework 3.5 je přímo v jazyce C# přítomna třída pro práci s algoritmem AES System.Security.Cryptography.Aes. Tato samotná třída je pouze abstraktní a může být pouze děděna jednotlivými implementacemi.

V současné době jsou v .NET 4.5 odvezeny dvě třídy z třídy Aes, které lze použít šifrování a dešifrování pomocí algoritmu AES, třídy AesManaged a AesCryptoServiceProvider. Obě dvě třídy umožňují z uživatelského a programátorského hlediska stejné použití, zásadní rozdíl je v tom, že třída AesCryptoServiceProvider má certifikaci FIPS 140-1 [5], že používá výhradně zdroje splňující minimální bezpečnostní požadavky definované NISTem v dokumentu FIPS 140-1. Na některých systémech může být administrátorem vynuceno používání pouze těchto certifikovaných tříd.

Nejdůležitější součásti třídy AES jsou vlastnost Key, která slouží k nastavení šifrovacího klíče,

public virtual byte[] Key { get; set; } [7],

a metody CreateEncryptor() a CreateDecryptor(). Metoda CreateEncryptor() vytvoří šifrovací objekt typu ICryptoTransform využívající aktuální šifrovací klíč nastavený ve vlastnosti Key [8]. Metoda CreateDecryptor() naopak vytváří dešifrovací objekt typu ICryptoTransform, taktéž s aktuální hodnotou šifrovacího klíče Key [9]. Samotné šifrování pak probíhá zápisem vstupních dat do streamu, odkud můžeme rovnou číst zašifrované bloky.

3 Programování aplikace

3.1 Složka bloků

V této složce se nachází všechny třídy podílející se na samotné šifře AES. Každá z tříd odpovídá jednomu bloku šifry.

3.1.1 Key.cs

Třída obsluhující práci se šifrovacím klíčem. Zde jsou definovány všechny metody potřebné pro práci s šifrovacím či rundovním klíčem.

```
public int[,] cipherKey = new int[4, 4];
```

Multidimenzionální pole celých čísel o čtyřech řádcích a čtyřech sloupcích. Toto pole slouží k uložení počátečního šifrovacího klíče o délce 128 bitů.

```
public int[,] roundKey = new int[44, 4];
```

Multidimenzionální pole celých čísel sloužící k uchování rundovního klíče expandovaného z šifrovacího klíče.

```
private int keyLength = 4;
```

Délka šifrovacího klíče N_k v bytech potřebná pro správné nastavení cyklu expanze šifrovacího klíče.

```
private int rcon = 1;
```

Počáteční hodnota rundovní konstanty R_{con}.Rundovní konstanta je po každém použití inkrementována v mezích $GF(2^8)$.

```
public void generateRandomKey()
```

Metoda pro vygenerování náhodného šifrovacího klíče o délce $N_k = 4$. Metoda Random() ze jmenného prostoru System vygeneruje máhodně 128 bitové slovo, které je potom použito jako šifrovací klíč.

```
public void ExpandCipherKey(sbox SBoxTable)
```

Metoda pro expanzi šifrovacího klíče sloužící k vygenerování celého rundovního klíče, používaného v jednotlivých rundách, viz 1.2.1.

Metoda potřebuje jako vstupní parametr instanci třídy sbox pro substituci bytů a nemá žádný výstup. Veškeré výsledky jsou ukládány přímo do pole roundKey definovaného na začátku třídy.

```
private int[] rotWord(int[] arr)
```

Soukromá metoda pro provedení cyklické permutace RotWord, viz 1.2.1. Metoda na vstupu očekává vstupní slovo o délce N_k bytů a na výstupu vrací pole o stejné délce po provedení permutace. Metoda rotWord() může být volána pouze v rámci třídy Key.

```
private int[] subWord(int[] arr, sbox SBoxTable)
```

Soukromá metoda nahrazující byty ze vstupního pole arr podle substituční tabulky SBoxTable. Metoda na výstupu vrací pole o délce N_k bytů a může být volána pouze v rámci třídy Key.

```
private int[] rConXor(int[] arr)
```

Soukromá metoda přičítající ke vstupnímu poli rundovní konstantu rcon. Po každém zavolání metody je rundovní konstanta rcon zvětšována podle rovnice (1.1). Metoda může být volána pouze v rámci třídy Key.

```
private int[] wordXor(int[] arr, int[,] rKey, int i)
```

Soukromá metoda provádějící exkluzivní disjunkci vstupního slova arr s již vypočítanou hodnotou rundovního klíče rKey[i - N_k], viz 1.2.1. Metoda wirdXor může být volána pouze v rámci třídy Key.

3.1.2 SubBytes.cs

Třída odpovídající bloku SubBytes nahrazující všechny byty vstupního slova novými hodnotami podle substituční tabulky. Princip fungování bloku SubBytes je popsán v kapitole 1.3.1.

3.1.3 ShiftRows.cs

Třída odpovídající bloku ShiftRows z kapitoly 1.3.2. Slouží k přesenutí bytů vstupního bloku na vybraných řádcích z vyšších pozic na nižší. Princip fungování bloku ShiftRows je popsán v kapitole 1.3.2.

3.1.4 MixColumns.cs

Třída odpovídající bloku MixColumns z kapitoly 1.3.3 slouží k promíchání hodnot v jednotlivých sloupcích vstupního bloku. Princip fungování bloku MixColumns je popsán v kapitole 1.3.3.

3.1.5 RoundKey.cs

Třída odpovídající bloku AddRoundKey z kapitoly 1.3.4 slouží k přičtení rundovního klíče k hodnotám vstupního bloku. Princip fungování bloku AddRoundKey je popsán v kapitole 1.3.4.

3.2 Složka logů

Ve složce logů se nachází pohromadě všechny třídy mající spojitost s ukládáním historie šifrování, sloužící následně k přehlednému výpisu jednotlivých kroků šifry.

3.2.1 bLog.cs

Je pouze rodičovská třída, ve které jsou definovány všechny společné proměnné, které jsou potom následně děděny odvozenými třídami. Třída samotná sestává z dvou proměnných typu integer:

```
protected int input;
protected int output;
```

které mají sloužit pro uchovávání jedné vstupní hodnoty do bloku a tomu odpovídající hodnotě výstupní. Proměnné jsou definovány s klíčovým slovem protected, což znamená, že k daným proměnným lze přistupovat pouze z aktuální třídy bLog a z tříd, které jsou odvozené z této třídy. K chráněným proměnným nelze přistupovat zvenku, pro takový přístup, ať už čtení nebo zápis, je potřeba napsat speciální metodu.

Všechny možné typy přístupů jsou uvedeny v tabulce 3.1. [13]

Tab. 3.1: Přehled možných přístupů k členům v C#

Klíčové slovo	Definice
internal	Člen deklarovaný jako internal je přístupný pouze v souborech patřících do jednoho sestavení (assembly).
private	Nejvíce omezující nastavení přístupu. Členy deklarované s tímto klíčovým slovem jsou přístupny pouze v rámci třídy, nebo struktury, ve kterých jsou deklarovány.
protected	K členu deklarovanému jako chráněnému se může přistupovat odkudkoliv z třídy, ve které byl deklarován a z každé třídy od ní odvozené.
public	Členy deklarované jako veřejné jsou přístupny odkukoliv bez omezení. Nejnižší možné nastavení přístupových práv.

Tato třída měla být rodičovská pro všechny další logovací třídy, ale v průběhu vývoje bylo zjištěno, že některé třídy vyžadují práci s celými poli čísel a ukládání pouze jednoho čísla je nedostatečné. To je případ tříd MixColumnLog a ShiftRowsLog, které pro smysluplné a rozumné zobrazení potřebují mít k dispozici celý řádek v případě ShiftRowsLog, případně celý sloupec pro MixColumnLog.

Třídy SByteLog a RoundKeyLog jsou potom odvozeny z této rodičovské třídy.

3.2.2 SByteLog.cs

Třída odvozená od rodičovské třídy bLog sloužící k uložení vstupní hodnoty před bytovou substitucí a adekvátní výstupní hodnoty po substituci.

```
public SByteLog(int inp, int outp)
{
    Input = inp;
    Output = outp;
}
```

Třída samotná sestává pouze z metod umožňujících přístup k rodičovským proměnným input a output deklarovaných jako protected, takzvaných getters a setters, a konstruktoru, který naplní obě proměnné při vytváření třídy.

Konstruktor má 2 parametry typu integer, čímž je zajištěno, že objekt vytvořený z téhle třídy bude mít vždy nastavenou vstupní a výstupní hodnotu. Jakýkoliv jiný pokus o vytvoření instance třídy povede k chybě ve vývojovém prostředí Visual Studia, např.:

Error 1 'aesApp.log.SByteLog' does not contain a constructor that takes 0 arguments

3.2.3 RoundKeyLog.cs

Třída taktéž odvozená od rodičovské třídy bLog používaná k ukládání historie z bloku AddRoundKey pro vypsání průběhu přidávání rundovního klíče při šifrování. Oproti základním, zděděným proměnným input a output dále deklaruje vlastní proměnnou key.

Proměnná key je typu integer a deklarovaná jako private. Slouží k uchování hodnoty rundovního klíče, použitého pro exkluzivní disjunkci se vstupním číslem.

Dále třída obsahuje veřejné getters a setters pro nastavení a čtení všech chráněných proměnných, a konstruktor se třemi parametry typu integer:

```
public RoundKeyLog(int inp, int outp, int k)
{
    Input = inp;
    Output = outp;
    Key = k;
}
```

3.2.4 ShiftRowsLog.cs

Samostatná třída určená k uchovávání historie průběhu bloku ShiftRows. Tato třída není odvozená od třídy bLog, protože blok prohození řádku ShiftRows nepracuje jenom

se samostatnými čísly, ale s celým řádkem pole naráz. Proto potřebujeme také ukládat pro přehledný výpis celý tento řádek naráz.

V třídě jsou deklarované dvě pole typu integer, pojmenované input a output pro zachování konzistentnosti s ostatními třídami:

```
private int[] inputRow = new int[4];
private int[] outputRow = new int[4];
```

Obě pole jsou deklarované jako private a mají nastavenou pevnou velikost 4. Velikost bloku N_b je v AES pevně definovaná na 4 byte ve standardu. Dále jsou zde přítomny getters a setters pro nastavení/čtení soukromých proměnných a konstruktor používaný při vytváření nové instance třídy:

```
public ShiftRowsLog(int[] inp, int[] outp)
{
    Buffer.BlockCopy(inp, 0, inputRow, 0, 16);
    Buffer.BlockCopy(outp, 0, outputRow, 0, 16);
}
```

Konstruktor přebírá jako parametr dvě pole typu integer a ukládá je do příslušných proměnných. Protože však nyní pracujeme s poli, nemůžeme provést obyčejné přiřazení pomocí operátoru =. To by v tomhle případě předalo pouze ukazatel na původní pole, jehož hodnoty už ale nemusí být později k dispozici a mohou být přepsány naprosto jinými. Proto musíme použít tvrdé zkopírování pomocí metody Buffer.BlockCopy ze jmenného prostoru System, který má definici: [14]

```
public static void BlockCopy(
   Array src,
   int srcOffset,
   Array dst,
   int dstOffset,
   int count
)
```

Vysvětlení jednotlivých parametrů je v tabulce 3.2.

Tab. 3.2: Parametry metody Buffer.BlockCopy

Array src	Zdrojové pole, ze kterého se má provádět kopírování
int srcOffset	Vzdálenost od začátku pole v Bytech, od které se má začít kopírovat
Array dst	Cílové pole, do kterého se mají nakopírovat hodnoty
int dstOffset	Vzdálenost v Bytech od začátku cílového pole, od které se mají začít vkládat hodnoty
int count	Počet Bytů kolik se má zkopírovat ze zdrojového pole od hodnoty srcOffset

Metoda BlockCopy vyžaduje, aby cílové pole, do kterého se mají kopírovat hodnoty ze zdrojového pole, bylo dopředu definované a tím pádem mělo dopředu v paměti už alokované místo. To se provedlo pomocí new int[4] při deklaraci proměnných na začátku třídy. BlockCopy je definovaná jako statická metoda, což znamená, že je možno k této metodě přistupovat, aniž by se musela vytvářet nová instance třídy Buffer.

3.2.5 MixColumnLog.cs

Taktéž samostatná třída, používaná pro uchovávání historie průběhu bloku MixColumns v jednotlivých kolech. Třída nedědí z třídy bLog, protože také, stejně jako ShiftRowsLog, pracuje s celým polem hodnot a ne pouze s jednou. Na rozdíl od ShiftRowsLog ale nepracuje s řádkem pole state, ale se sloupcem.

V těle třídy jsou definovány soukromé proměnné:

```
private int[] input = new int[4];
private int output;
private int[] aPoly = new int[4];
```

Pole input, typu integer, obsahuje vždy jeden sloupec z pole state, na kterém jsou prováděna operace bloku MixColumns.

Pole aPoly, taktéž typu integer, obsahuje hodnoty polynomu a(x), viz kapitola 1.3.3.

Proměnná output potom obsahuje výstupní hodnotu vypočtenou z polí input a aPoly.

Dále třída obsahuje veřejné getters a setters, pro nastavení a čtení soukromých proměnných, a konstruktor pro vytvoření instance třídy:

```
public MixColumnLog(int[] inp, int outp, int[] apol)
{
    Buffer.BlockCopy(inp, 0, this.input, 0, 16);
    Buffer.BlockCopy(apol, 0, this.aPoly, 0, 16);
    this.output = outp;
}
```

Pro správné nastavení hodnot polí je potřeba použít překopírování pomocí metody BlockCopy stejně jako v případě třídy ShiftRowsLog.

3.2.6 blockLog.cs

Třída blockLog je určená pro sesbírání a udržování historie všech bloků a všech rund pohromadě.

```
private int[,] inputBlock = new int[4, 4];
```

Vícerozměrné pole uchovávající vstupní hodnotu zadanou uživatelem před začátkem šifrování.

```
private int[,] outputBlock = new int[4, 4];
```

Vícerozměrné pole uchovávající zašifrovanou hodnotu po projití všech N_r rund.

```
private List<SByteLog> sBhistory;
```

Seznam objektů typu SByteLog, ve kterých je uložena historie bloku SubBytes všech N_r rund.

```
private List<RoundKeyLog> rKHistory;
```

Seznam objektů typu RoundKeyLog, ve kterých je uložena historie bloku AddRoundKey všech N_r rund.

```
private List<ShiftRowsLog> sRHistory;
```

Seznam objektů typu ShiftRowsLog, ve kterých je uložena historie bloku ShiftRows všech N_r rund.

```
private List<MixColumnLog> mCHistory;
```

Seznam objektů typu MixColumnLog, ve kterých je uložena historie bloku MixColumns všech N_r rund.

Dále jsou ve třídě definovány getters, pro přístup pro čtení jednotlivých seznamů, a veřejné přístupové metody pro nastavení jednotlivých seznamů:

```
public void AddHistory(List<SByteLog> hist)

public void AddRKHistory(List<RoundKeyLog> hist)

public void AddSRHistory(List<ShiftRowsLog> hist)

public void AddMCHistory(List<MixColumnLog> hist)
```

3.3 Složka support

3.3.1 sbox.cs

Pomocná třída, ve které je definovaná substituční tabulka používaná v blocích SubWord při expanzi šifrovacího klíče a SubBytes při šifrování.

Všechny hodnoty jsou napevno definované v proměnné sBox, která je typu Hashtable.

```
private Hashtable sBox;
```

Třída Hashtable z jmenného prostoru System.Collections ukládá data jako hodnoty key => value [15].

Vkládání probíhá pomocí metody Add, jejíž definice je následující:

```
public virtual void Add(
        Object key,
        Object value
)
```

Jako dvojice key => value může figurovat jakýkoliv object, v našem případě je použita dvojice integer – integer.

4 Závěr

V rámci diplomové práce jsem provedl teoretický rozbor blokové šifry AES, nastudoval a popsal funkčnost jednotlivých bloků důležitých k pochopení funkce samotné šifry, rozebral možnosti platformy .NET framework a navrhl rozhraní aplikace pro výuku a vysvětlení funkce šifry AES.

Aplikace pro výuku a vysvětlení funkce šifry AES byla naprogramována ve vývojovém prostředí Microsoft Visual Studio 2012 a úspěšně odzkoušena v operačním systému Microsoft Windows 8 64bit. Jako cílová verze .NET byl nastaven .NET Framework 4.5. Na této konfiguraci byl úspěšně odzkoušen algoritmus AES-128 pro vstupní slovo 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34 a šifrovací klíč 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c.

Správnost výsledného kryptogramu, jakožto i všech mezivýsledků byla poté ověřena ve standardu AES v příkladu šifry přiloženém k této publikaci [4]. Tento příklad ukazuje výstupy ze všech bloků, pro všechny rundy, takže je jednodušše ověřitelné, že naprogramovaná aplikace funguje správně.

Seznam použitých zdrojů

[1] DAEMEN, Joan. *The design of Rijndael: AES - the Advanced Encryption Standard*. Berlin: Springer, 2002, 238 s. ISBN 35-404-2580-2.

[2] NECHVATAL, James et al. Status report on the first round of the development of the Advanced Encryption Standard. Journal of Research of NIST [online]. 1999, Issue 5 [cit. 2012-12-12]. Dostupné z: http://nvlpubs.nist.gov/nistpubs/jres/104/5/j45nec.pdf

[3] NECHVATAL, James et al. Report on the Development of the Advanced Encryption Standard (AES). In: NATIONAL INSTUTE OF STANDARDS AND TECHNOLOGY. NIST's AES Report [online]. 2000 [cit. 2012-12-12]. Dostupné z: http://csrc.nist.gov/archive/aes/round2/r2report.pdf

[4] FIPS PUB 197. *Announcing the ADVANCED ENCRYPTION STANDARD (AES)*. Gaithersburg, 2001. Dostupné z: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[5] FIPS PUB 140-1. *SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES*. Gaithersburg, 1994. Dostupné z: http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf

[6] Aes Class. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2012 [cit. 2012-12-12]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.security.cryptography.aes.aspx

[7] SymmetricAlgorithm.Key Property. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2012 [cit. 2012-12-12]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.security.cryptography.symmetricalgorithm.key.aspx

[8] SymmetricAlgorithm.CreateEncryptor Method. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2012 [cit. 2012-12-12]. Dostupné z: http://msdn.microsoft.com/en-us/library/09d0kyb3.aspx

[9] SymmetricAlgorithm.CreateDecryptor Method. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2012 [cit. 2012-12-12]. Dostupné z: http://msdn.microsoft.com/en-us/library/79w421xb.aspx

[10] VIRIUS, Miroslav. *C# 2010: hotová řešení*. 1. vyd. Brno: Computer Press, 2012, 424 s. K okamžitému použití (Computer Press). ISBN 978-80-251-3730-7

[11] PETZOLD, Charles. *Mistrovství ve Windows Presentation Foundation*. Vyd. 1. Brno: Computer Press, 2008, 928 s. ISBN 978-80-251-2141-2.

[12] Random Class. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2013 [cit. 2013-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.random.aspx

[13] Access Modifiers (C# Reference). MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2013 [cit. 2013-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/library/wxh6fsc7.aspx

[14] Buffer.BlockCopy Method. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2013 [cit. 2013-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.buffer.blockcopy.aspx

[15] Hashtable Class. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2013 [cit. 2013-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.collections.hashtable.aspx

[16] Hashtable.Add Method. MICROSOFT CORPORATION. *Microsoft Developer Network* [online]. © 2013 [cit. 2013-05-20]. Dostupné z: http://msdn.microsoft.com/en-us/library/system.collections.hashtable.add.aspx

Seznam použitých zkratek a symbolů

AES – Advanced Encryption Standard

API – Application Programming Interface

ASICs – Application-Specific Integrated Circuits

BCL – Basic Class Library

CLR – Common Language Runtime

COM – Component Object Model

DES – Data Encryption Standard

DLL – Dynamic Link Library

DSA – Digital Signature Algorithm

FIPS – Federal Information Processing Standard

FPGAs – Field Programmable Gate Arrays

GF – Galois Field

IDE – Integrated Development Environment

IEEE – Institute of Electrical and Electronics Engineers

IETF – Internet Engineering Task Force

IL – Intermediate Language

IP – Internet Protocol

IPSEC – Internet Protocol Security

ISO – International Organization for Standardization

JIT – Just In Time compilation

K – šifrovací klíč (Key)

LINQ – Language Integrated Query

N_b – počet 32-bitových vstupních bloků

NIST – US National Institute of Technology

N_k – délka šifrovacího klíče v 32-bitových slovech

N_r – počet rund potřebných k provedení algoritmu

NSA – National Security Agency

SDK – Software Development Kit

SHA – Secure Hash Algorithm

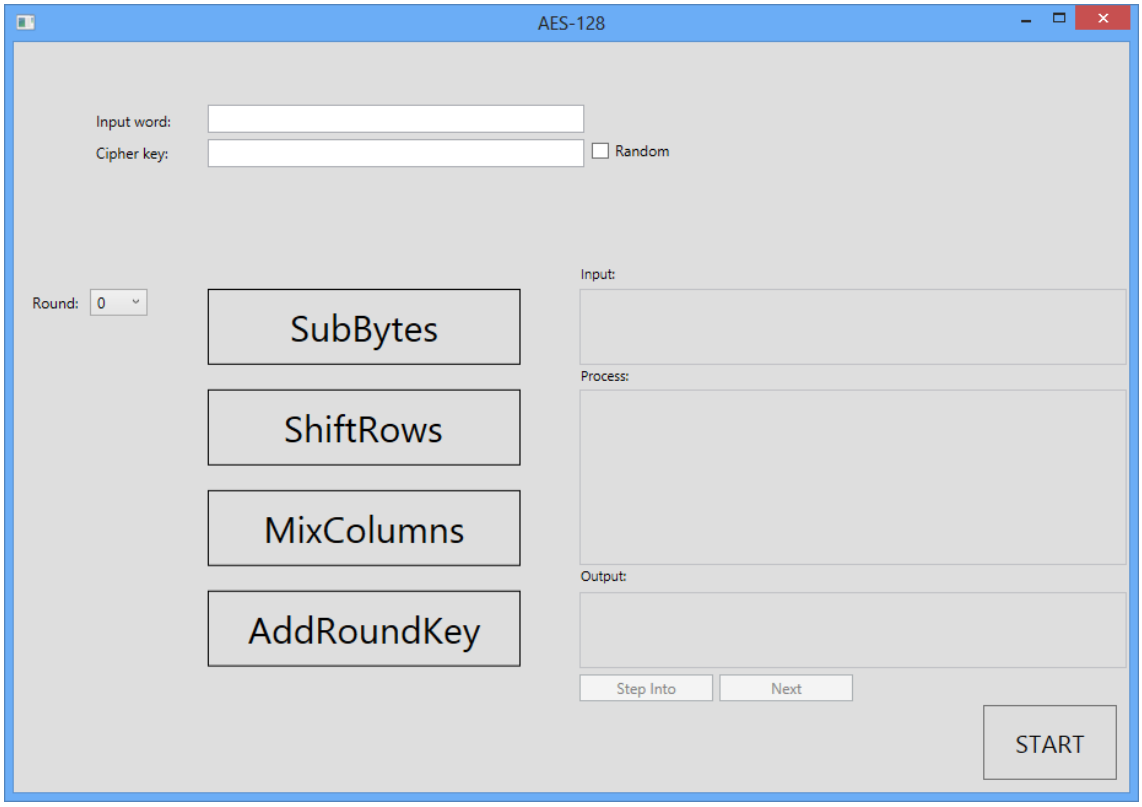
WCF – Windows Community Foundation

WPF – Windows Presentation Foundation

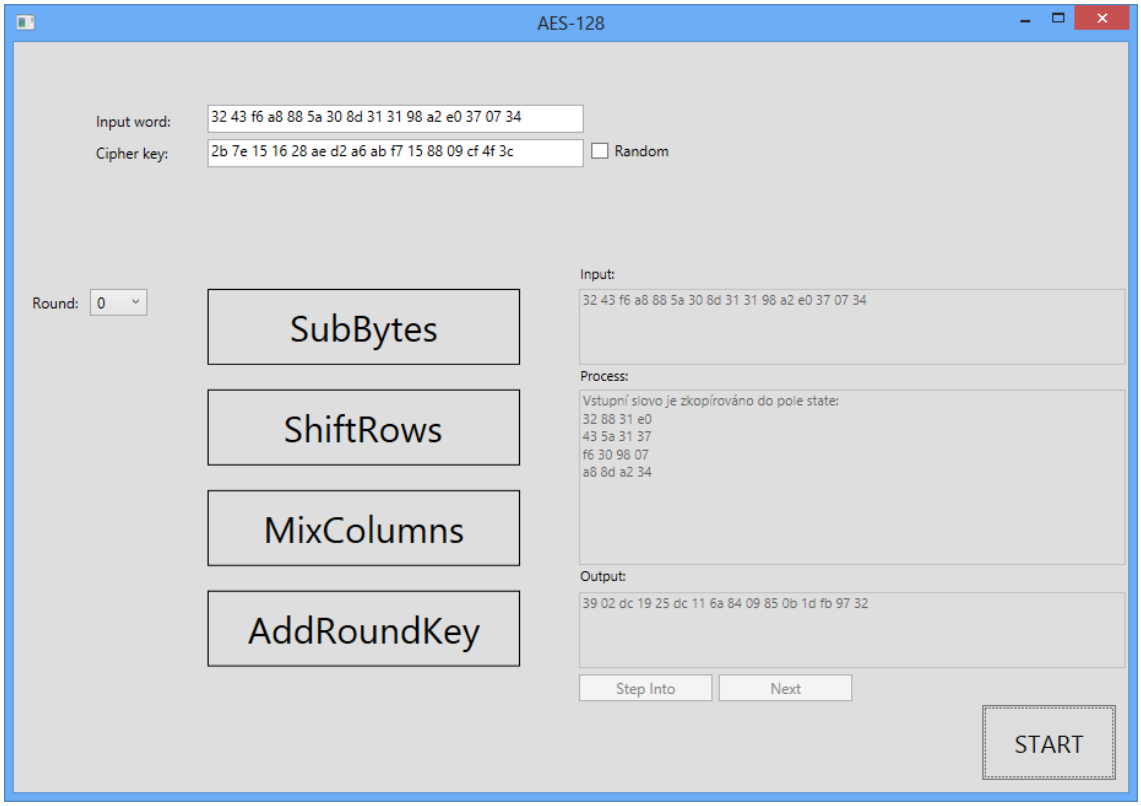
Seznam tabulek

Seznam obrázků

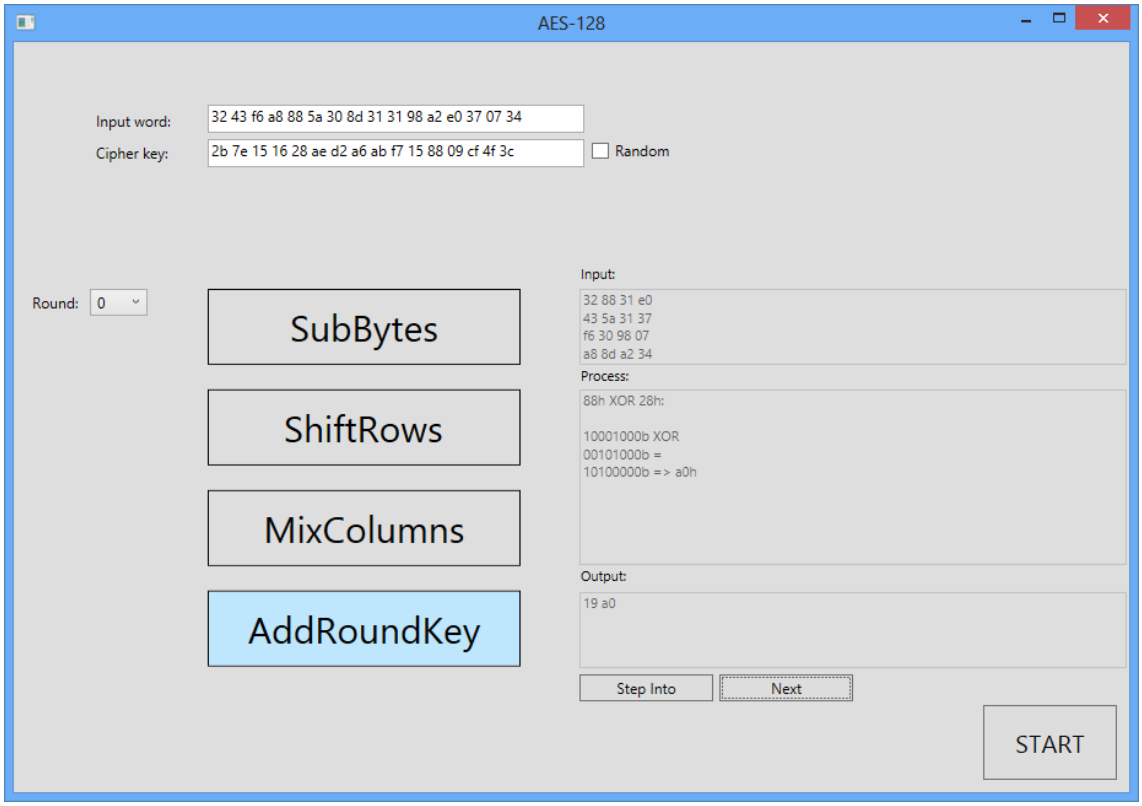
A Rozhraní aplikace pro výuku šifry AES



Obr. A.1: Úvodní obrazovka programu



Obr. A.2: Výpis úpravy vstupní zprávy



Obr. A.3: Výpis výpočtu v bloku AddRoundKey

B Vybrané zdrojové kódy programu

blocks/SubBytes.cs

```
namespace aesApp.blocks
{
    public class SubBytes
    {
        private int[,] inputWord;
        private int[,] outputWord = new int[4,4];
        private List<SByteLog> history = new List<SByteLog>();

        public List<SByteLog> History
        {
            get { return history; }
        }

        public SubBytes(int[,] input)
        {
            this.inputWord = input;
        }

        public int[,] Substitute(sbox SBox)
        {
            for (int i = 0; i < 4; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    this.outputWord[i,j] = SBox.getSubValue(this.inputWord[i,j]);

                    // Save history to the log
                    history.Add(new SByteLog(this.inputWord[i, j],
this.outputWord[i, j]));
                }
            }

            return this.outputWord;
        }
    }
}
```

blocks/MixColumns.cs

```
namespace aesApp.blocks
{
    public class MixColumns
    {
        private int[,] inputWord;
        private int[,] outputWord = new int[4,4];
        private int[] aPoly = new int[4]{2, 3, 1, 1};
        private List<MixColumnLog> history = new List<MixColumnLog>();

        public List<MixColumnLog> History
        {
            get { return history; }
        }

        public int[,] OutputWord
        {
            get { return outputWord; }
            set { outputWord = value; }
        }

        public MixColumns(int[,] state)
        {
            this.inputWord = state;
        }

        public int[,] Mix()
        {
            int[] pomArray = new int[4];

            for (int i = 0; i < 4; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    pomArray[j] = this.inputWord[j, i];
                }

                pomArray = multiply(pomArray);

                for (int j = 0; j < 4; j++)
                {
                    this.outputWord[j, i] = pomArray[j];
                }
            }
            return this.outputWord;
        }

        private int[] multiply(int[] column)
        {
            int[] result = new int[4] {0, 0, 0, 0};

            for (int k = 0; k < 4; k++)
            {
                for (int l = 0; l < 4; l++)
                {
                    if (this.aPoly[l] == 1)
                    {
                        result[k] = result[k] ^ column[l];
                    }
```

```
                    else if (this.aPoly[l] == 2)
                    {
                        if (column[l] >= 128)
                        {
                            result[k] = result[k] ^ (((column[l] << 1) - 256) ^
27);
                        }
                        else
                        {
                            result[k] = result[k] ^ (column[l] << 1);
                        }
                    }
                    else
                    {
                        if (column[l] >= 128)
                        {
                            result[k] = result[k] ^ ((((column[l] << 1) - 256) ^
27) ^ column[l]);
                        }
                        else
                        {
                            result[k] = result[k] ^ ((column[l] << 1) ^
column[l]);
                        }
                    }
                }
                // Save the history
                this.history.Add(new MixColumnLog(column, result[k], aPoly));

                //zmenit polynom o jedno doprava
                int pom = this.aPoly[3];
                Buffer.BlockCopy(this.aPoly, 0, this.aPoly, 4, 12);
                this.aPoly[0] = pom;
            }

            return result;
        }

    }
}
```

blocks/ShiftRows.cs

```
namespace aesApp.blocks
{
    public class ShiftRows
    {
        private int[,] inputWord;
        private int[,] outputWord = new int[4,4];
        private List<ShiftRowsLog> history = new List<ShiftRowsLog>();

        public List<ShiftRowsLog> History
        {
            get { return history; }
        }

        public int[,] OutputWord
        {
            get { return outputWord; }
            set { outputWord = value; }
        }

        public ShiftRows(int[,] input)
        {
            this.inputWord = input;
        }

        public int[,] Shift()
        {
            int[] pomArray = new int[4];
            int[] oldArray = new int[4]; // for loging purposes

            for (int i = 0; i < 4; i++)
            {
                Buffer.BlockCopy(this.inputWord, i * 4 * 4, pomArray, 0, 16);
                Buffer.BlockCopy(pomArray, 0, oldArray,0, 16);

                if(i > 0)
                {
                    pomArray = moveBytes(pomArray, i);
                }

                // save history
                this.history.Add(new ShiftRowsLog(oldArray, pomArray));

                Buffer.BlockCopy(pomArray, 0, this.outputWord, i * 4 * 4, 16);
            }

            return this.outputWord;
        }

        private int[] moveBytes(int[] pomArray, int i)
        {
            int pom;

            for (int j = 0; j < i; j++)
            {
                pom = pomArray[0];
                Buffer.BlockCopy(pomArray, 4, pomArray, 0, 12);
                pomArray[3] = pom;
            }
```

```
            return pomArray;
        }
    }
}
```

C DVD

/zdrojove_kody – Zdrojové kódy aplikace ve formátu řešení .sln pro Visual Studio 2012

/aplikace – Spustitelný soubor aplikace

/dokumenty – Text práce ve formátu PDF