UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Martin Bárta

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Vizualizace šifrování AES v Javě FX

Martin Bárta

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Bárta**
Osobní číslo:	**I13095**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vizualizace šifrování AES v Javě FX**
Zadávající katedra:	**Katedra informačních technologií**

Z á s a d y p r o v y p r a c o v á n í :

Cílem práce je v jazyce Java FX vizualizovat klíčové kroky šifry AES pro její názornou demonstraci.

V teoretické části bude popsána historie šifrování, zavedeny důležité pojmy a provedeno základní rozdělení šifer. Hlavní důraz bude v popisu symetrické šifry AES.

Praktická část bude obsahovat popis implementace 128 bitové verze AES v módu ECB.
Práce bude zpracovaná v jazyce Java SE s nadstavbou JavaFX a bude po vložení textu a klíče graficky zobrazovat jednotlivé kroky, které jsou prováděny během šifrování a dešifrování.
Dále bude možno zobrazit animaci expanze klíče.

Rozsah grafických prací:

Rozsah pracovní zprávy: **cca 35 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

STALLINGS, William. Cryptography and network security: principles and practice. 5th ed. Boston: Prentice Hall, c2011. ISBN 0136097049.

EBBERS, Hendrik. Mastering JavaFX 8 Controls. New York: McGraw Hill Education, 2014. ISBN 0071833773.

PECINOVSKÝ, Rudolf. Java 8: úvod do objektové architektury pro mírně pokročilé. První vydání. Praha: Grada Publishing, 2014. Knihovna programátora (Grada). ISBN 978-80-247-4638-8.

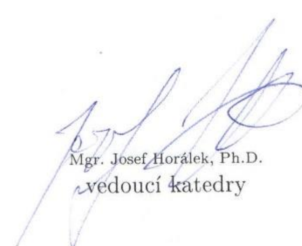
Vedoucí bakalářské práce: **Ing. Zdeněk Šilar, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.



Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 09. 05. 2017

Martin Bárta

PODĚKOVÁNÍ

Mé poděkování směřuje k Ing. Ph.D. Zdeňku Šilarovi za užitečné rady při odborných konzultacích v průběhu řešení bakalářské práce.

ANOTACE

Cílem této práce je vysvětlení problematiky šifrování a jeho důležitosti při komunikaci přes internet. Práce je pak zaměřena především na šifrovací algoritmus AES, který spadá do kategorie symetrických blokových šifer. Součástí práce je JavaFX program, který krok po kroku ukáže průběh šifrování a dešifrování reálných uživatelem zadaných dat za pomoci AES se 128bitovým klíčem.

KLÍČOVÁ SLOVA

AES, šifrování, dešifrování, klíč, symetrická šifra, bloková šifra, Java, JavaFX

AES VISUALIZATION IN JAVA FX

ANNOTATION

The purpose of this bachelor thesis is to explain encryption and its importance during the communication over the Internet. The main part of this thesis is focused on AES which is the symmetric block cipher. The work itself includes the JavaFX computer program for the step by step visualization of AES encryption and decryption using the key with 128 bits' length.

KEYWORDS

AES, encryption, decryption, symmetric cipher, block cipher, Java, JavaFX

OBSAH

1 Úvod....12
2 Základní pojmy z kryptografie 13
2.1 Kryptografie 13
2.2 Šifra 13
2.2.1 Dělení podle použití klíče 13
2.2.2 Dělení podle zpracování dat 15
2.2.3 Příklady nejznámějších šifer různých druhů....15
2.3 Kryptosystém 15
3 Symetrické blokové šifry 16
3.1 Feistelova šifra 16
3.2 Kryptosystém DES....17
3.3 Operační módy blokových šifer 17
4 Kryptosystém AES 18
4.1 Porovnání DES s AES....18
4.2 Struktura AES 19
4.2.1 Galoisova tělesa 19
4.3 Operace prováděné při šifrování a dešifrování 20
4.3.1 Operace přidání podklíče 21
4.3.2 Operace záměna bajtů....21
4.3.3 Operace prohození řádků 23
4.3.4 Operace kombinování sloupců....24
4.4 Expanze klíče 26
4.5 Bezpečnost AES....27
5 Implementace vizualizačního programu 28
5.1 Struktura projektu....28
5.1.1 Stručný popis jednotlivých souborů projektu 29

5.2 Koncept FXML souboru a jeho kontroléru 30
5.3 Layouty JavaFX 33
5.3.1 Svazování vlastností komponent 34
5.4 Sázecí systém LaTeX 36
5.5 Animace v JavaFX 37
5.5.1 Třída `Transition` pro vytváření přechodových animací 37
5.5.2 Třida `Timeline` pro vytváření animací 38
5.5.3 Ovládání `Timeline` animace pomocí tlačítek 40
5.6 Uživatelské rozhraní pro ovládání aplikace 41
5.7 Distribuce aplikace 45
6 ZÁVĚR 48
7 Použitá literatura 49

SEZNAM ILUSTRACÍ A TABULEK

Obrázek 1 – Symetrický šifrovací algoritmus 13

Obrázek 2 – Asymetrický šifrovací algoritmus 14

Obrázek 3 – Feistelova šifra (šifrování a dešifrování) [1] 16

Obrázek 4 – Převod bloku do stavové matice 19

Obrázek 5 – Průběh šifrování a dešifrování v AES [1] 20

Obrázek 6 – Přepočet indexů řádku a sloupce pro substituci 22

Obrázek 7 – Konstrukce normálního a inverzního SBoxu [1] 22

Obrázek 8 – SBox (vlevo) a inverzní SBox (vpravo) 23

Obrázek 9 – Funkce používaná při expanzi klíče [1] 26

Obrázek 10 – Struktura projektu v prostředí NetBeans 28

Obrázek 11 – Grafické prostředí z FXML souboru před a po kliknutí na tlačítko 32

Obrázek 12 – `StateArrayControl` bez a se zobrazenou tabulkou 33

Obrázek 13 – Vlastní JavaFX komponenta s různou šířkou 36

Obrázek 14 – Soustava rovnic operace kombinování sloupců 37

Obrázek 15 – Část průběhu `Timeline` animace prohození řádků 40

Obrázek 16 – Ovládací panel pro ovládání animace 40

Obrázek 17 – Úvodní obrazovka aplikace 42

Obrázek 18 – Zadávání vlastního klíče 42

Obrázek 19 – Hlavní okno aplikace s průběhem šifrování 43

Obrázek 20 – Detail ovládacích tlačítek posouvání operací a bloků dat 44

Obrázek 21 – Konečný stav vizualizace pro konkrétní blok dat 45

Obrázek 22 – Přidávání jar souborů v aplikaci JarSplice 46

Obrázek 23 – Definování spustitelné `main` metody v aplikaci JarSplice 46

Tabulka 1 – Nejznámější šifry 15

Tabulka 2 – Porovnání rychlosti zpracování dat dle kryptosystému [2] 18

Tabulka 3 – Verze kryptosystému AES [1] 19

Tabulka 4 – Rundové konstanty pro jednotlivá čísla rund [1] 27

SEZNAM ZKRATEK A ZNAČEK

AES	Advanced Encryption Standard
RSA	Rivest – Shamir – Adleman
DES	Data Encryption Standard
USA	The United States of America
ECB	Electronic Codebook
CBC	Cipher Block Chaining
CTR	Counter
XOR	Exclusive or
EFF	Electronic Frontier Foundation
NIST	National Institute of Standards and Technology
3DES	Triple Data Encryption Standard
EDE	Encrypt Decrypt Encrypt
MB	Megabajt
TB	Terabajt
MB/s	Megabajt za sekundu
GF	Galois Field
FXML	FX Markup Language
UML	Unified Modeling Language
CD	Compact Disc

1 ÚVOD

Tato bakalářská práce vznikla za účelem vysvětlení základních pojmů z oblasti kryptografie a podrobného seznámení se s AES. Práce je rozdělena na dvě části, praktickou a teoretickou.

V úvodu teoretické části je obsažen úvod do kryptografie společně se základním rozdělením šifer. Dále se pak teoretická část soustředí na detailní popis symetrických blokových šifer. K dispozici je detailní popis fází algoritmu AES včetně konstrukce SBox substituční tabulky a algoritmu pro rozšíření vstupního klíče. Dále je k dispozici výčet módů, ve kterých mohou blokové šifry operovat. Tyto módy popisují způsob, jakým se mezi sebou provazují jednotlivé zašifrované bloky, popřípadě jakou akci provést s předchozím zašifrovaným blokem předtím, než je možné zašifrovat blok následující.

V praktické části je implementována 128bitová verze AES. Tato implementace je využita pro zašifrování textu zadaného uživatelem. Veškerá data se uloží do vytvořené datové struktury tak, aby se všechny operace mohly graficky zobrazit. O grafickou prezentaci se starají komponenty z knihovny JavaFX. Ke každému kroku je dále zobrazen popis operace, a to včetně matematických vzorců, které byly vykresleny jako obrázek s pomocí knihovny JLaTeXMath. Dále má každý krok svoji animaci zobrazující průběh korespondující operace, která je prováděna se zadanými daty ve stavové matici. Tyto animace byly vytvořeny s pomocí třídy z knihovny JavaFx, konkrétně pak *`Timeline`*. Výslednou animaci je možné s pomocí jednoduchého ovládacího panelu spustit, pozastavit nebo úplně vypnout a vrátit se tak na začátek. Animaci nelze posouvat v čase dopředu a ani zpět.

K výrobě programu bylo použito vývojové prostředí NetBeans ve verzi 8.1, tento program byl pro psaní vizualizačního programu naprosto dostačující. Vývojové prostředí podporuje práci s formátem FXML použitým pro tvorbu grafického prostředí aplikace.

Pro finální distribuci byl dále použit program JarSplice, který umožňuje vytvářet jeden jar soubor z více jar souborů, v tomto případě byla přidána do projektu knihovna JLaTeXMath.

Tento program má výukový charakter a je doporučen uživatelům, kteří si chtějí rozšířit znalosti z kryptografie, popřípadě implementovat svoji vlastní verzi šifrovacího algoritmu. Z písemné dokumentace si je možné osvojit základní znalosti pro vytváření animací a tvorbu nových layoutů přizpůsobených pro specifický účel. To vše za pomocí knihovny JavaFX.

2 ZÁKLADNÍ POJMY Z KRYPTOGRAFIE

Znalosti potřebné k sepsání teoretické části této práce byly získány a nastudovány z knihy [1].

2.1 Kryptografie

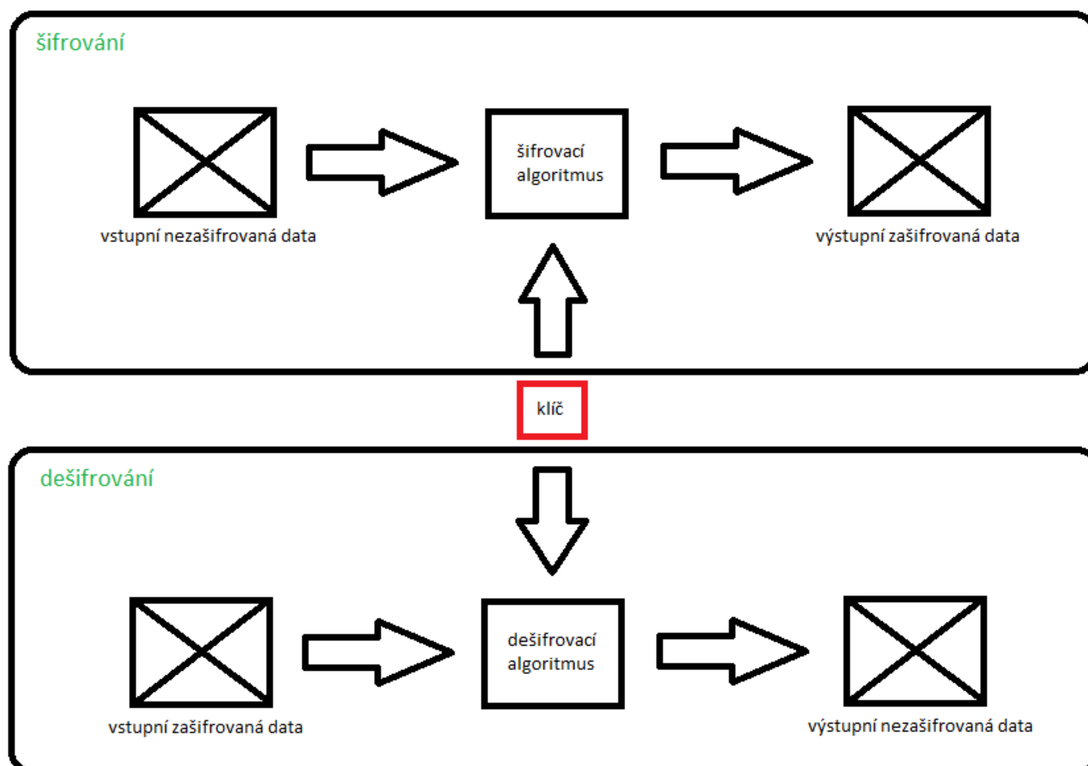
Pod pojmem kryptografie rozumíme vědu zabývající utajováním zpráv, přesněji převodem vstupních dat, popř. textu do podoby, která je čitelná jenom s dodatečnou znalostí, a pro ostatní tak nesrozumitelná. Slovo kryptografie pochází z řečtiny a je sloučeninou slov kryptós (skrytý) a gráphein (psát).

2.2 Šifra

Šifra je v kryptografii chápána jako algoritmus, který vykonává šifrování a následné dešifrování textu nebo jakýchkoliv jiných vstupních dat. Šifry se dělí do několika kategorií, např. podle použití klíče nebo podle způsobu zpracování dat.

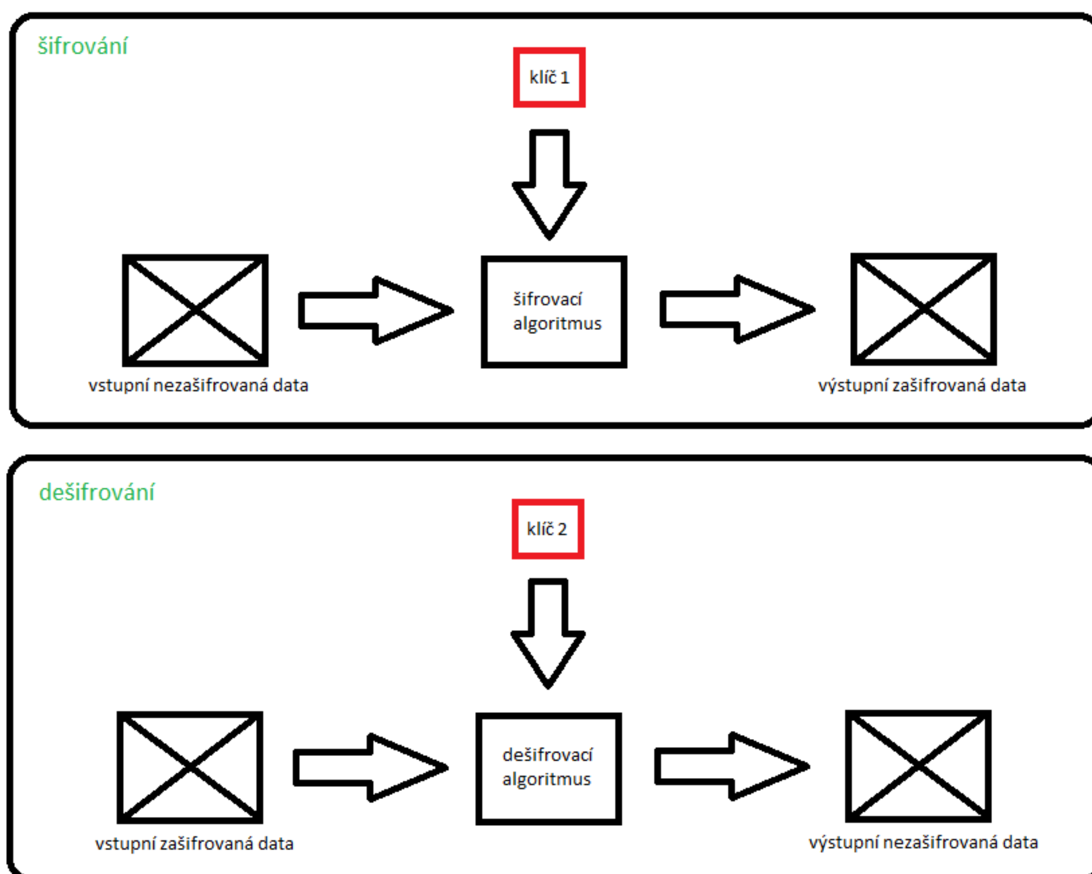
2.2.1 Dělení podle použití klíče

- Symetrická šifra – využívá stejný klíč jak pro šifrování, tak pro dešifrování. Symetrické šifry bývají označovány jako šifry tajného klíče. Průběh šifrování a dešifrování symetrické šifry je zobrazen na Obrázku 1.



Obrázek 1 – Symetrický šifrovací algoritmus

- Asymetrická šifra – využívá dva různé klíče, jeden pro šifrování a druhý pro dešifrování. Asymetrické šifry jsou označovány jako šifry veřejného klíče a to proto, že jeden klíč zpravidla bývá veřejný. Průběh šifrování a dešifrování asymetrické šifry je zobrazen na Obrázku 2.



Obrázek 2 – Asymetrický šifrovací algoritmus

- V současné době nelze jak symetrické, tak asymetrické šifry označit jako zastaralé. Oba typy mají své uplatnění a své místo. Bývá pravidlem, že symetrické šifry šifrují data rychleji, neboť jsou méně výpočetně náročné, proto se obecně používají k šifrování větších objemů dat. Asymetrické šifry jsou pomalejší kvůli náročnějším výpočtům, nehodí se tak k šifrování velkého objemu dat. Jejich uplatnění spočívá především v bezpečné distribuci klíčů symetrických šifer přes internet. Po navázání bezpečného spojení se zpravidla přechází komunikaci, kterou zajišťují symetrické šifry, jako je například AES. Opakované použití stejného klíče se u asymetrických šifer nedoporučuje, jedná se o bezpečnostní hrozbu.

2.2.2 Dělení podle zpracování dat

- Proudová šifra – typicky zpracovává vstup vždy jen po jednom bajtu. Zvláštním případem proudové šifry pak může být šifra pracující vždy jen na jednom bitu. Pro generování klíče se používá generátor pseudonáhodných čísel, klíč dosahuje stejné velikosti jako šifrovaná data.
- Bloková šifra – zpracovává vstup dat po blocích. Tyto bloky jsou šifrovány a dešifrovány jako celek a výstupem je blok o stejné velikosti. V dnešní době šifry typicky používají bloky o velikosti 64 nebo 128 bitů.

2.2.3 Příklady nejznámějších šifer různých druhů

V následující Tabulce 1 nalezneme několik nejznámějších šifer, v tabulce jsou k nalezení také skupiny šifer, pod které daná šifra spadá.

Tabulka 1 – Nejznámější šifry

Šifra	Symetrická šifra	Asymetrická šifra	Proudová šifra	Bloková šifra
AES	✓	✗	✗	✓
DES	✓	✗	✗	✓
Vigenèrova šifra	✓	✗	✓	✗
Vernamova šifra	✓	✗	✓	✗
RSA	✗	✓	✗	✓

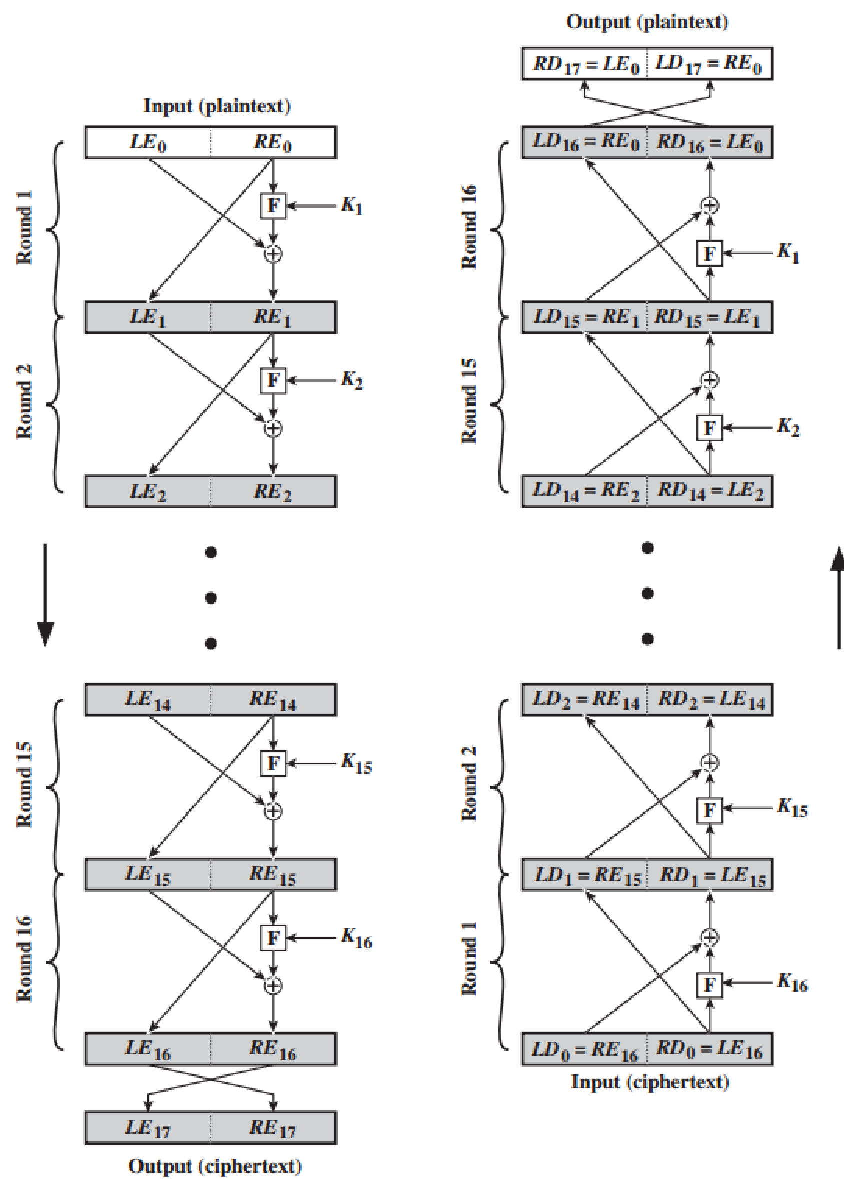
2.3 Kryptosystém

Kryptosystém je v kryptografii chápán jako skupina algoritmů. Většinou jsou tyto algoritmy tři. První algoritmus se stará o generování klíče, druhý o šifrování a třetí o dešifrování dat. Například za zkratkou AES se tak neskrývá pouze šifra, ale celý kryptosytém, obsahující zmíněné algoritmy, které spolu navzájem souvisí.

3 SYMETRICKÉ BLOKOVÉ ŠIFRY

3.1 Feistelova šifra

- Dříve byla pro blokové šifry typická struktura Feistelovy šifry (Obrázek 3). Feistelova struktura se skládá z několika stejných rund. V každé rundě je prováděna substituce na polovině zpracovávaných dat, za kterou následuje permutace na obou polovinách dat. Originální klíč je rozšířen tak, aby byl v každé rundě použit klíč unikátní.
- Nejznámější šifrou, odpovídající Feistelově struktuře je DES, ten byl dříve vůbec nejpoužívanější kryptosystémem, postupně ovšem zastaral a byl nahrazen AES. AES už však Feistelově struktuře neodpovídá.



Obrázek 3 – Feistelova šifra (šifrování a dešifrování) [1]

3.2 Kryptosystém DES

- Data Encryption Standard byl poprvé veřejnosti představen v 70. letech 20. století. Šifra operuje na blocích dat o velikosti 64 bitů. Klíč má velikost 56 bitů a jeho velikost byla zvolena podle požadavku implementace pro jeden specifický čip a tím omezenému výkonu hardwaru tehdejší doby.
- V roce 1977 byl DES v USA schválen jako federální standart pro všechna neutajovaná data.
- O dvacet později v roce 1998 byl DES definitivně prokázán jako nebezpečný. EFF prohlásila prolomení DES na speciálním jednoúčelovém počítači za méně než 3 dny.
- Jako odpověď na prolomení DES se začal používat 3DES, jedná se o stejný algoritmus, jen se provádí nad stejnými bloky dat třikrát pomocí tří různých 56bitových klíčů. 3DES je dodnes považován za bezpečný, nicméně je výpočetně třikrát pomalejší než už i tak pomalý obyčejný DES.
- Krytosysém DES je jedním z nejznámějších kryptosystémů a měl velký vliv na pozdější vývoj celého odvětví počítačové bezpečnosti.

3.3 Operační módy blokových šifer

V případě používání stejného klíče pro šifrování více bloků narůstá bezpečnostní riziko. V zašifrovaných datech se mohou projevovat například určité pravidelnosti typické pro daný jazyk. Tyto anomálie poté mohou usnadnit práci útočníkům. Pro odstranění problému se znovu užíváním stejného klíče, aniž by se klíč musel měnit, nám mohou pomoci některé operační módy, ve kterých samotná bloková šifra operuje. Následující výčet představuje nejběžnější módy společně s jejich základní charakteristikou:

- ECB (electronic codebook) – každý blok vstupních dat je zašifrován zvlášť s použitím stejného klíče pro všechny bloky. Typicky se používá pro jednorázový přenos samostatných hodnot např. klíče.
- CBC (cipher block chaining) – vstupní blok pro šifrování je logickou operací XOR z následujícího bloku nezašifrovaných dat a předchozího bloku zašifrovaných dat. Mód vhodný pro použití k dlouhodobé komunikaci za použití stejného klíče.
- CTR (counter) – Na každý blok vstupních dat je použita logická operace XOR se zašifrovaným čítačem. Po každém zpracovaném bloku se hodnota čítače inkrementuje. Mód vhodný pro použití k dlouhodobé komunikaci za použití stejného klíče. Doporučeno používat při zvýšených požadavcích na rychlost implementace.

4 KRYPTOSYSTÉM AES

- Advanced Encryption Standard je v současnosti jedním z nejpoužívanějších kryptosystémů.
- AES byl publikován organizací NIST v roce v 2001 jako náhrada za DES.
- Veškeré rovnice použité v této kapitole jsou převzaté z [1].

4.1 Porovnání DES s AES

- Oproti 64 bitovým DES blokům používá AES bloky o velikosti 128 bitů. To umožňuje používat stejný klíč mnohonásobně déle. Při použití bloků o velikosti 64 bitů bychom měli změnit klíč maximálně po 2^{32} blocích, což odpovídá 32 GB dat. Při použití bloku o velikosti 128 bitů je toto číslo 2^{64} bloků. To je více než 250 miliónů TB, tak velký objem dat nám zaručí za předpokladu neuniklého klíče, že se klíč nemusí měnit nikdy.
- AES existuje ve třech verzích. Ty se od sebe liší velikostí klíče a počtem rund, které se provádí při šifrování a dešifrování. Nejmenší možný klíč má velikost 128 bitů, další pak 192 a 256 bitů. To je v porovnání s velikostí 56 bitového klíče DES minimálně více jak dvojnásobek. To zaručuje zvýšenou odolnost proti útokům typu brute-force.
- Jelikož se AES implementuje pomocí jednoduchých bitových operací, je výsledné šifrování a dešifrování mnohem rychlejší viz tabulka 2, AES tak lze bez problémů implementovat na jednoduchých 8bitových procesorech. Jednotlivé implementace v Tabulce 2 byly měřeny na počítači s procesorem Pentium 4. Implementace samotných algoritmů jsou z knihovny Crypto++ jazyka C++, knihovna je k dispozici zdarma[1].

Tabulka 2 – Porovnání rychlosti zpracování dat dle kryptosystému [2]

	MB zpracováno	čas [s]	Rychlost [MB/s]
AES-128	256	4,196	61,010
AES-192	256	4,817	53,145
AES-256	256	5,308	48,229
DES	128	5,998	21,340
3DES-EDE3	64	6,499	9,848

[1] Knihovna ke stažení z: https://www.cryptopp.com/

- Díky všem těmto aspektům neexistuje reálný důvod, proč si vybrat DES namísto AES.

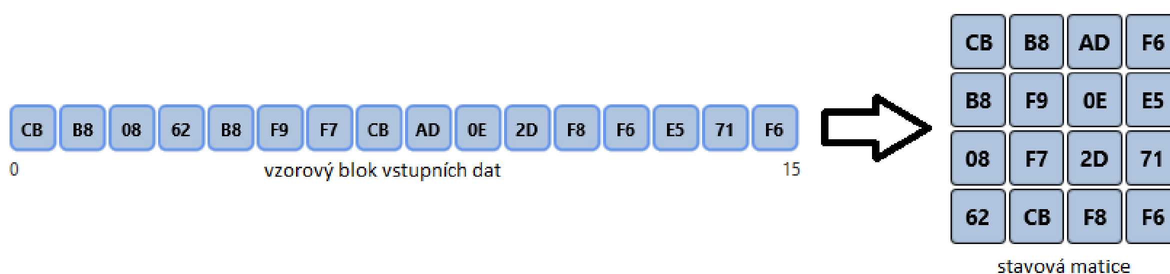
4.2 Struktura AES

AES si zachovává některé vlastnosti Feistelovy šifry. Šifrování a dešifrování je prováděno v rundách. Počet rund se liší podle velikosti klíče. V každé rundě jsou prováděny čtyři operace. Výjimkou je pak první (nultá) a poslední runda. Přehled verzí AES je dostupný v Tabulce 3.

Tabulka 3 – Verze kryptosystému AES [1]

	AES-128	AES-192	AES-256
Velikost klíče [bitů]	128	192	256
Velikost bloku [bitů]	128	128	128
Počet rund	10	12	14
Velikost expandovaného klíče [bajtů]	176	208	240

Veškeré operace jsou prováděny v tzv. stavové matici, což je matice o velikost 4x4 bajtů. Do této matice jsou převedeny vstupní data a po provedení všech rund a jejich operací jsou z ní přečteny data výstupní. Převod bloku na stavovou matici je zobrazen na Obrázku 4. Stejným postupem se provádí převod ze stavové matice do výstupních dat.



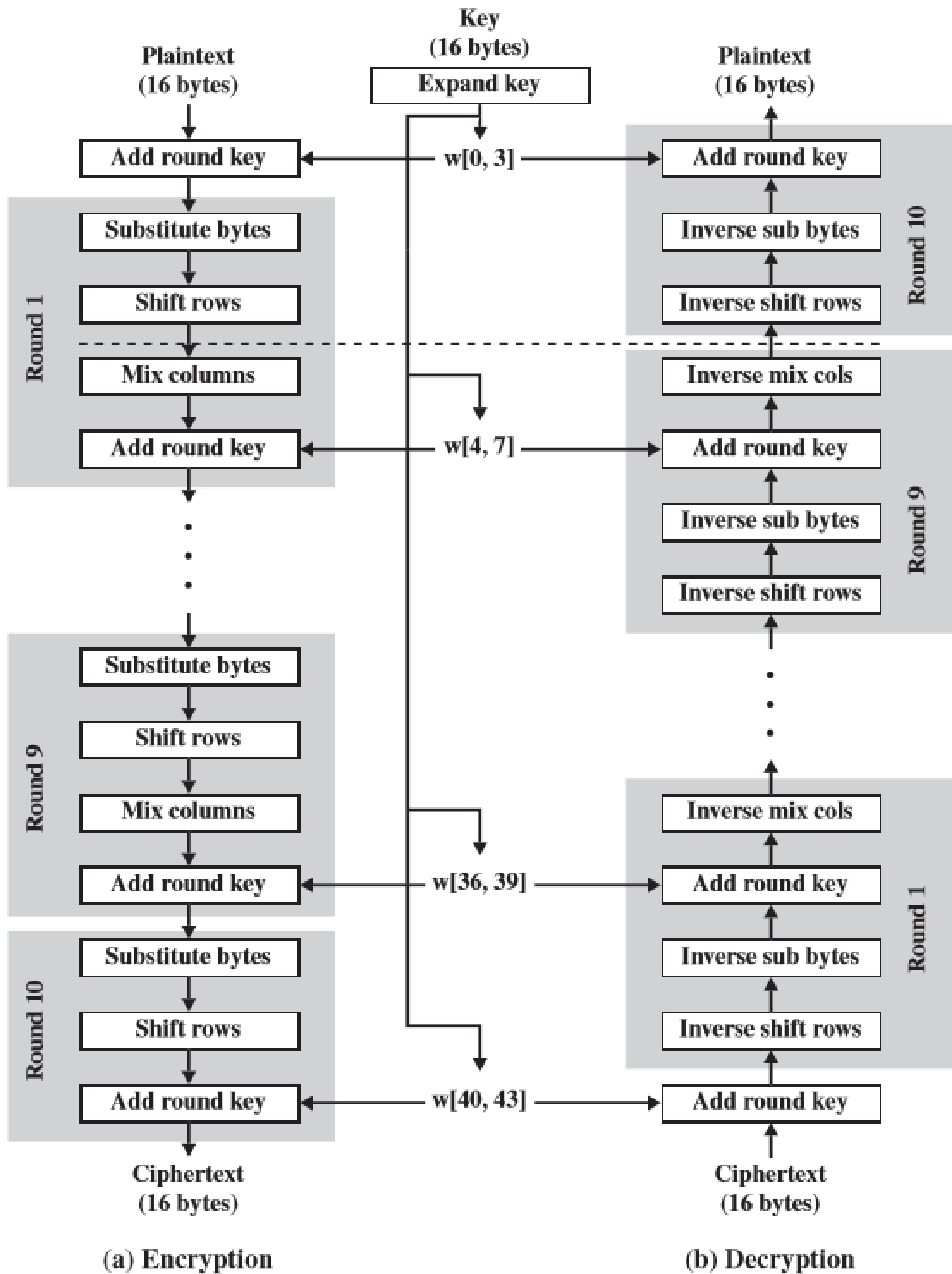
Obrázek 4 – Převod bloku do stavové matice

4.2.1 Galoisova tělesa

Všechny operace v AES jsou prováděny na celých bajtech. Aritmetické operace jako sčítání násobení a dělení jsou prováděny pomocí konečných těles v GF(2^8). V jednoduchosti lze říci, že nám konečná tělesa umožňují provádět aritmetické operace, aniž by výsledek opustil množinu konečného pole, ve kterém je výpočet prováděn. Znalosti potřebné k pochopení konečných těles lze nastudovat a získat v [1].

4.3 Operace prováděné při šifrování a dešifrování

Na Obrázku 5 je znázorněn celý průběh šifrování a dešifrování jednoho bloku v AES. V tomto konkrétním případě se jedná o verzi se 128bitovým klíčem.



Obrázek 5 – Průběh šifrování a dešifrování v AES [1]

Z Obrázku 5 je patrné, že dešifrovací algoritmus je totožný s šifrovacím, pouze je prováděn v opačném pořadí a s inverzními operacemi.

Jednotlivé operace mají v českém překladu názvy:

- Přidání podklíče,
- Záměna bajtů,
- Prohození řádků,
- Kombinování sloupců.

4.3.1 Operace přidání podklíče

V této operaci dochází k použití logické operace XOR 128bitové stavové matice se 128bitovým klíčem dané rundy. Každá runda má svůj unikátní klíč, to zajišťuje algoritmus expandování klíče. Celá operace je vyjádřena v Rovnici 1.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,0} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} \quad (1)$$

kde,

S – je vstupní bajt stavové matice,

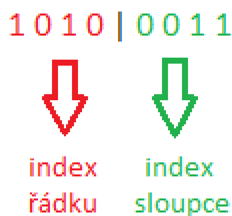
w – je bajt slova rundového klíče,

S' – je výstupní bajt stavové matice na konci operace.

Inverzní operace k operaci přidání klíče je totožná, nicméně klíče jednotlivých rund jsou při dešifrování používány v opačném pořadí, ostatně to je patrné i z obrázku 5.

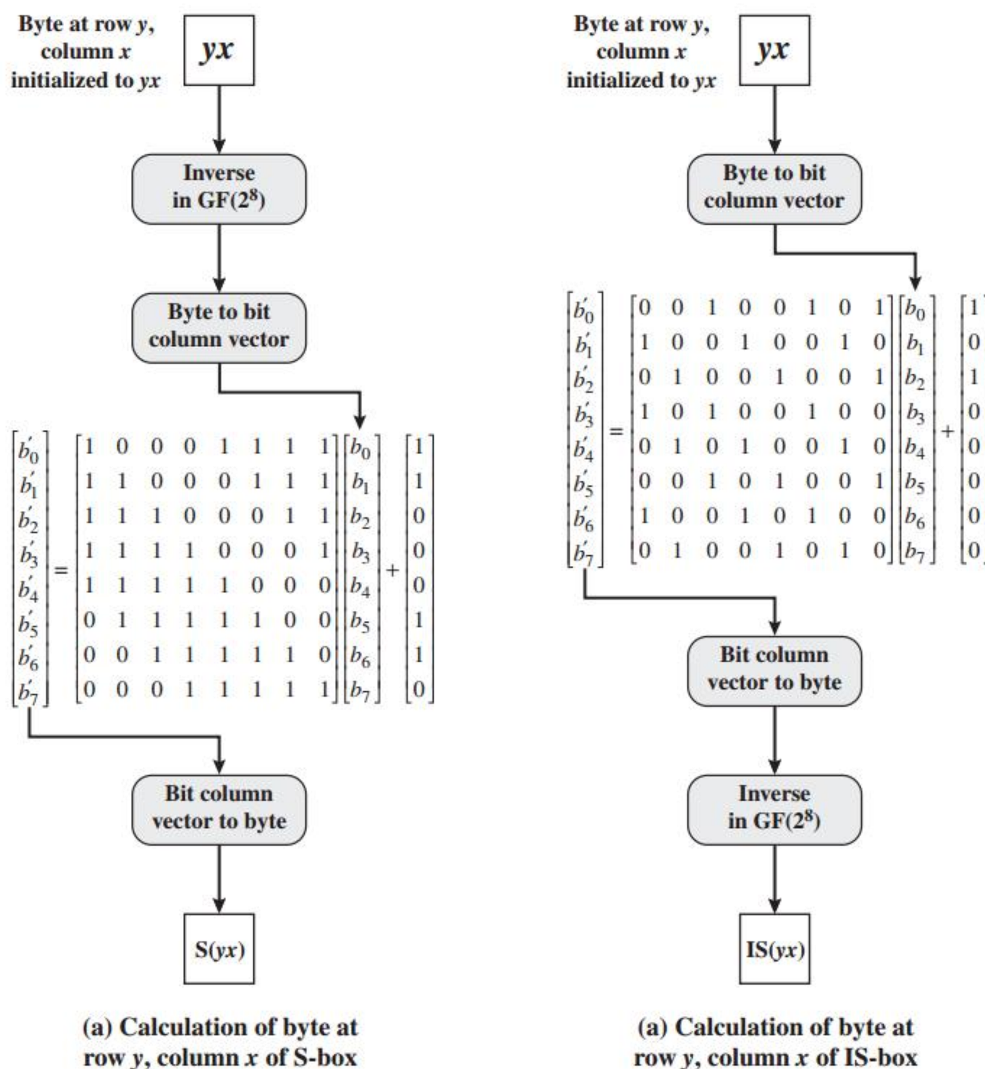
4.3.2 Operace záměna bajtů

Operace záměna bajtů nahrazuje vstupní bajty stavové matice bajty ze substituční tabulky SBox popř. její inverzní verze. Abychom věděli, který bajt ze substituční tabulky bude použit, je třeba si odvodit jeho index řádku a sloupce. To se provádí velice jednoduše. Vstupní bajt se rozdělí na dvě poloviny. Čtyři bity nejvíce nalevo jsou použity jako index řádku substituční tabulky, zbylé čtyři bity nejvíce napravo pak reprezentují index sloupce. Příklad přepočtu indexů pro substituci je patrný na Obrázku 6.



Obrázek 6 – Přepočet indexů řádku a sloupce pro substituci

SBox je tabulka o velikosti 16x16 bajtů, přičemž každá hodnota od 0 do 255 se zde nachází pouze a právě jednou. SBox a inverzní SBox je pro všechny rundy stejný. Jeho konstrukce je znázorněna na Obrázku 7.



Obrázek 7 – Konstrukce normálního a inverzního SBoxu [1]

Z obrázku 7 je patrné, že konstrukce substituční tabulky není úplně triviální. Především pak počítání převrácených hodnot v GF(2^8) může být výpočetně relativně náročné. Z tohoto důvodu se v praxi implementuje SBox jako pole již vypočtených konstant viz Obrázek 8. Pro ověření hodnot je možné si převrácené hodnoty v GF(2^8) spočítat třeba s pomocí rozšířeného Euklidova algoritmu a poté dopočítat celý SBox popř. jeho inverzní podobu.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
10	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
11	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
12	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
13	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
14	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
15	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
10	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
11	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
12	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
13	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
14	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
15	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Obrázek 8 – SBox (vlevo) a inverzní SBox (vpravo)

4.3.3 Operace prohození řádků

Prohození řádků je jednoduchou operací. Jednotlivé bajty na jednotlivých řádcích stavové matice se kruhově posouvají o daný počet míst. Pokud se jedná o inverzní prohození řádků, tak se bajty posouvají doprava, v normální verzi je proveden posuv doleva.

V následujícím seznamu je uvedena velikost posuvu bajtů na jednotlivých řádcích stavové matice:

- První řádek zůstává nedotčen.
- Na druhém řádku je proveden kruhový posuv bajtů o jedno místo doleva, popř. doprava v inverzní verzi.
- Na třetím řádku je proveden kruhový posuv bajtů o dvě místa doleva, popř. doprava v inverzní verzi.
- Na čtvrtém řádku je proveden kruhový posuv bajtů o tři místa doleva, popř. doprava v inverzní verzi.

Pro úplnou názornost je celá operace vyjádřena následující Rovnicí 2.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,0} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,1} & S_{1,2} & S_{1,3} & S_{1,0} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,3} & S_{3,0} & S_{3,1} & S_{3,2} \end{bmatrix} \tag{2}$$

kde,

S – je v první matici vstupní a ve druhé výstupní bajt stavové matice.

Inverzní verze operace je pak patrná z následující Rovnice 3.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,0} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,3} & S_{1,0} & S_{1,1} & S_{1,2} \\ S_{2,2} & S_{2,3} & S_{2,0} & S_{2,1} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,0} \end{bmatrix} \tag{3}$$

kde,

S – je v první matici vstupní a ve druhé výstupní bajt stavové matice.

4.3.4 Operace kombinování sloupců

Operace kombinování sloupců je operací nejnáročnější na pochopení. Operace operuje zvlášť pro každý sloupec stavové matice. Každý výsledný bajt nového sloupce je funkcí všech čtyřech bajtů původního sloupce. Operace kombinování sloupců je definována jako maticové násobení v Rovnici 4.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,0} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} \tag{4}$$

kde,

S – je vstupní bajt stavové matice,

S'– je výstupní bajt stavové matice.

Je třeba mít na paměti, že sčítaní je v $GF(2^8)$ prováděno jako exkluzivní logický bitový součet. Násobení v $GF(2^8)$ se řídí pravidly násobení konečných těles.

Z Rovnice 4 lze vyjádřit vztahy pro jednotlivé bajty daného sloupce, tyto vztahy najdeme v Rovnicích 5, 6, 7 a 8.

$$S'_{0,j} = (2 \cdot S_{0,j}) \oplus (3 \cdot S_{1,j}) \oplus S_{2,j} \oplus S_{3,j} \tag{5}$$

$$S'_{1,j} = S_{0,j} \oplus (2 \cdot S_{1,j}) \oplus (3 \cdot S_{2,j}) \oplus S_{3,j} \tag{6}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus (2 \cdot S_{2,j}) \oplus (3 \cdot S_{3,j}) \tag{7}$$

$$S'_{3,j} = (3 \cdot S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \cdot S_{3,j}) \tag{8}$$

kde,

S' – je výstupní bajt stavové matice,

j – je index sloupce stavové matice,

$\cdot$ – je násobení v GF(2^8).

V případě inverzní operace kombinování sloupců je použita inverzní rovnice k Rovnici 4. Že se jedná o inverzní rovnici, není hned patrné, ověření si je možné přečíst v odborné literatuře [1]. Inverzní rovnice odpovídá Rovnici 10.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,0} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} \tag{9}$$

Analogicky jako v případě normálního kombinování sloupců lze pro inverzní verzi odvodit vztahy pro jednotlivé bajty daného sloupce. Jedná se o Rovnice 10, 11, 12 a 13.

$$S'_{0,j} = (0E \cdot S_{0,j}) \oplus (0B \cdot S_{1,j}) \oplus (0D \cdot S_{2,j}) \oplus (09 \cdot S_{3j}) \tag{10}$$

$$S'_{1,j} = (09 \cdot S_{0,j}) \oplus (0E \cdot S_{1,j}) \oplus (0B \cdot S_{2,j}) \oplus (0D \cdot S_{3j}) \tag{11}$$

$$S'_{2,j} = (0D \cdot S_{0,j}) \oplus (09 \cdot S_{1,j}) \oplus (0E \cdot S_{2,j}) \oplus (0B \cdot S_{3j}) \tag{12}$$

$$S'_{3,j} = (0B \cdot S_{0,j}) \oplus (0D \cdot S_{1,j}) \oplus (09 \cdot S_{2,j}) \oplus (0E \cdot S_{3j}) \tag{13}$$

kde,

S' – je výstupní bajt stavové matice,

j – je index sloupce stavové matice,

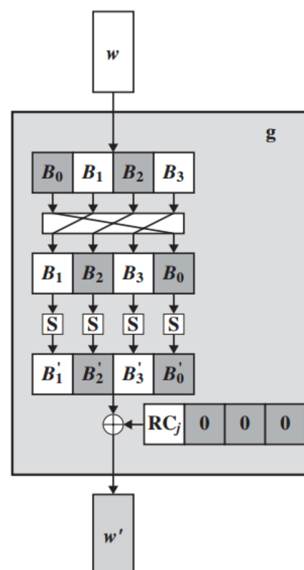
$\cdot$ – je násobení v GF(2^8).

4.4 Expanze klíče

Aby bylo možné používat pro každou rundu jiný 128bitový klíč, je nutné původní klíč rozšířit. Přesně k tomuto účelu slouží algoritmus expanze klíče. V případě tohoto algoritmu se pracuje s tzv. slovy. Jedno slovo je sekvence čtyř po sobě jdoucích bajtů. Původní 128bitový klíč tak obsahuje čtyři slova. Původní klíč je použit jen v první rundě.

Další slova jsou přidávána a závisí vždy na předchozím slovu a slovu o čtyři pozice zpět. Pokud není index nového slova v poli slov násobek čtyř, je nové slovo vypočítáno jako XOR předchozího slova se slovem o čtyři pozice zpět, jestliže však je index nového slova dělitelný čtyřmi, je použita komplexnější funkce.

Tato funkce nejprve vezme slovo o jednu pozici zpět, než je index nového slova. V tomto slovu je proveden kruhový posuv bajtů o jednu pozici doleva. Dále je každý bajt v tomto slovu nahrazen za bajty z SBox tabulky podle stejných pravidel jako při operaci záměna bajtů. Nakonec je na tomto slovu proveden XOR s tzv. rundovou konstantou viz Tabulka 4. Důležité je si uvědomit, že, rundová konstanta má velikost jeden bajt. XOR rundové konstanty s novým slovem tak ovlivní pouze jeden bajt nového slova. Při operaci XOR se za zbylé tři bajty v konstantě dosazuje bajt s hodnotou nula. Bajt, který je touto XOR operací změněn, je tak bajtem úplně prvním. Pro přehlednost je celá tato funkce zobrazena na Obrázku 9. Po provedení této funkce se se slovem na jejím výsledku provede logická operace XOR se slovem o čtyři pozice zpět. Toto nové slovo konečné.



Obrázek 9 – Funkce používaná při expanzi klíče [1]

Tabulka 4 – Rundové konstanty pro jednotlivá čísla rund [1]

Číslo rundy	1	2	3	4	5	6	7	8	9	10
Rundová konstatnta	01	02	04	08	10	20	40	80	1B	36

4.5 Bezpečnost AES

V současnosti jsou považovány všechny verze AES za bezpečné. V roce 2003 byl AES oficiálně uznán vládou USA natolik bezpečný, že všechny jeho verze jsou vhodné pro ochranu utajovaných informaci až do kategorie tajné [3]. Pro přísně tajné informace je vyžadováno AES s klíčem o velikosti alespoň 192 bitů.

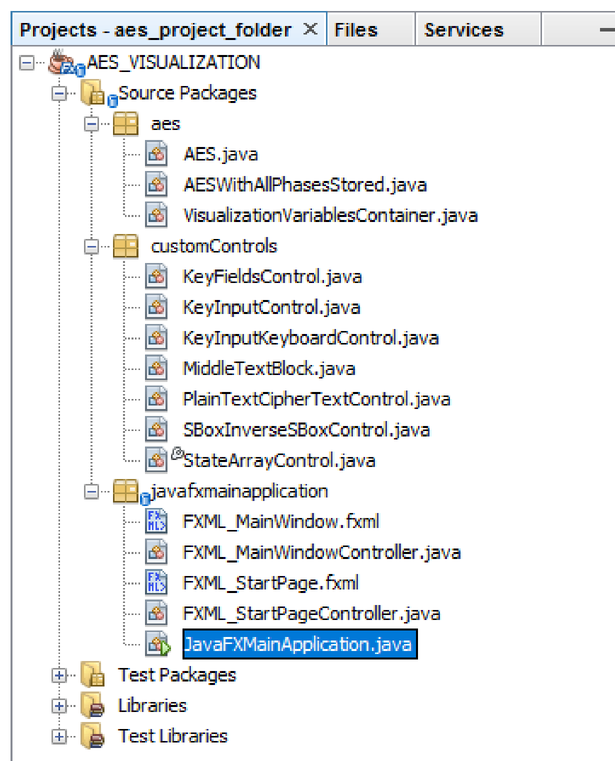
Chyby byly doposud objeveny pouze v nesprávných implementacích a útok typu brute-force se současnou úrovní techniky stále nedává z časového hlediska smysl.

5 IMPLEMENTACE VIZUALIZAČNÍHO PROGRAMU

Vizualizační program je napsaný v jazyce Java, grafické prostředí je vytvořeno pomocí knihoven z balíčku JavaFX. Pro vizualizaci samotného šifrovacího programu bylo nutno implementovat AES, implementovaná je jen 128bitová verze. Z této implementace jsou získávána reálná data průběhu šifrování a dešifrování. Aplikace dále využívá soubory ve formátu FXML pro vytváření jednotlivých scén uživatelského prostředí. Ovládací prvky spolu logicky související, jsou slučovány do jednotlivých tříd. Tyto třídy vždy dědí z vhodného existujícího layoutu, který je již k dispozici v knihovně JavaFX. Tento postup umožnuje zvýšit přehlednost kódu starajícího se jak o vytvoření takto nově vzniklé komponenty, tak i o provádění akcí, které tyto komponenty umožnují. Další výhodou tohoto postupu je fakt, že díky tomu, že tyto komponenty dědí z layoutů JavaFX, se dají později použít v FXML souboru jako jeden tag.

5.1 Struktura projektu

Projekt vizualizační aplikace je rozdělen do tří logických částí. Tyto logické části pak odpovídají v prostředí NetBeans jednotlivým balíčkům. První balíček nese název aes, druhý customControls a poslední javafxmainapplication. Celá struktura projektu je zobrazena na Obrázku 10, který pochází rovněž z vývojového prostředí NetBeans.



Obrázek 10 – Struktura projektu v prostředí NetBeans

První část je zodpovědná za samotný proces šifrování. Jeho součástí je taky vlastní datová struktura, která uchovává všechny stavy stavové matice, kterými projde AES během šifrování či dešifrování, a to i pro více bloků. Druhá část obsahuje všechny použité grafické layouty, z kterých se skládá grafické prostředí. Třetí část obsahuje FXML soubory definující grafické rozhraní, jejich kontroléry a v neposlední řadě také třídu, která obsahuje *main* metodu pro spuštění celé aplikace.

5.1.1 Stručný popis jednotlivých souborů projektu

- *AES.java* – tato třída obsahuje implantaci kryptosystému AES ve 128bitové verzi v operačním módu ECB. Kryptosystém při šifrování na vstupu očekává datový typ String a jeho výstupem je pole bajtů. Pro dešifrování je třeba na vstupu pole bajtů, výsledek dešifrování je dále předán v datovém typu *String*. Klíč je v obou případech zadáván jako pole bajtů.
- *AESWithAllPhasesStored.java* – také obsahuje kryptosystém AES jako v případě *AES.java*, liší se ale výstupním typem proměnné šifrování a dešifrování. Výstup je typu *VisualizationVariablesContainer*, obsahující potřebná data pro vizualizaci.
- *VisualizationVariablesContainer.java* – jedná se o datovou strukturu, do níž je možné ukládat stavy stavové matice, a to i pro více bloků.
- *KeyFieldsControl.java* – grafický layout umožňující zobrazit všech 176 bajtů expandovaného klíče. Komponenta dědí Pane.
- *KeyInputControl.java* – grafický layout sloužící pro zadávání klíče a zobrazení vstupního 16 bajtu velikého klíče. Komponenta dále umí klíč generovat ze jednobajtových zobrazitelných UTF-8 znaků.
- *KeyInputKeyBoardControl.java* – grafická komponenta klávesnice, kterou využívá třída *KeyInputControl* pro zadání klíče. Umožňuje zadat jen jednobajtové zobrazitelné UTF-8 znaky.
- *MiddleTextBlock.java* – komponenta sloužící k popisu právě prováděné operace a zobrazení potřebných matematických vzorců.
- *PlainTextCipherTextControl.java* – komponenta zobrazující bloky dat před šifrováním a po šifrování. Umožňuje přepínání mezi jednotlivými bloky a určuje tak, který blok dat je vizualizován.

- *SBoxInverseSBoxControl.java* – tato komponenta má na starosti zobrazení tabulky SBox nebo její inverzní verzi. Umožňuje přepínání mezi normální a inverzní verzí za běhu programu.
- *StateArrayControl.java* – centrální prvek celého programu, okolo kterého se točí vše ostatní. Komponenta zobrazuje stavovou matici a její buňky s reálnými daty. Jednotlivé stavy matic korespondující s operacemi, které lze přepínat. Dále je možné v této komponentě spustit animace pro jednotlivé operace.
- FXML_MainWindow.fxml – soubor typu FXML, ve kterém je uloženo grafické prostředí hlavního okna starajícího se o vizualizaci průběhu šifrovaní a dešifrování uživatelem zadaných vstupních dat.
- *FXML_MainWindowController.java* – třída sloužící jako kontrolér pro soubor FXML_MainWindow.fxml, stará se zejména o akce, které nastanou po kliknutí na tlačítka definovaná v kontrolovaném FXML souboru. Umožňuje zobrazit novou scénu pro opětovné zadání vstupních dat.
- FXML_StartPage.fxml – soubor, v kterém je uloženo grafické prostředí úvodního okna aplikace, které umožňuje uživateli zadat vstupní text a klíč s kterým se má text šifrovat.
- *FXML_StartPageController.java* – třída sloužící jako kontrolér pro soubor FXML_StartPage.fxml. Konkrétně se stará o akce po kliknutí na tlačítka a zároveň umožnuje vytvořit novou scénu s hlavním oknem vizualizační aplikace.
- *JavaFXMainApplication.java* – jedná se o třídu obsahující metodu *main*. Tato třída spouští celou vizualizační aplikaci.

5.2 Koncept FXML souboru a jeho kontroléru

JavaFX umožnuje využívat speciálního XML souboru s názvem FXML. Tento soubor se používá pro definování uživatelského rozhraní. K tomuto souboru je většinou vázán kontrolér, který se stará o akce jednotlivých komponent definovaných v FXML souboru. Výhodou tohoto řešení je jeho přehlednost a snadná editace. Bohužel v případě vytváření vlastních komponent nelze vše zpravovat jenom z kontroléru, a tak se značná část kódu starajícího se právě o vlastní komponenty nachází i ve třídách samotných komponent. Další výhodou FXML souboru je eliminování potřeby inicializovat jednotlivé komponenty v něm použité. Pro přístup k těmto komponentám stačí použít anotaci v kontroléru, konkrétně pak *@FXML*. Stejným způsobem je nutno označit metody, které mají být zavolány při nějaké akci na komponentách definovaných v FXML souboru. Teorie potřebná pro vytvoření aplikace byla nastudována z knihy [4] a [5].

V následující ukázce je zobrazen obsah vzorového FXML souboru. Jedná se o jednoduchý *VBox* layout, ve kterém je použita vlastní komponenta *SBoxInverseSBoxControl*. Důležité je, že i u této vlastní komponenty lze používat atributy, a to podle toho, z kterého layoutu či komponenty je vlastní komponenta odvozená. Všechny komponenty, které se ovládají pomocí kontroléru musí mít vyplněný unikátní atribut *fx:id*. Atribut *onAction* slouží k volání metody, která je definovaná v kontroléru a která se zavolá dojde-li k příslušné události. Součástí ukázkového souboru je tlačítko, po jehož stisknutí dochází k provedení kódu obsaženého v metodě *handleSwitchingBetweenEncrDecr()*.

```
<VBox fillWidth="true" prefWidth="350" prefHeight="420"
alignment="TOP_RIGHT"
xmlns="http://javafx.com/javafx/8.0.102-ea"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="javafxmainapplication.FXML_MainWindowController">
    <children>
        <VBox VBox.vgrow="ALWAYS">
            <children>
                <Label fx:id="sBoxLabel" text="SBox" textFill="#17bac3"
                       VBox.vgrow="NEVER">
                    <font>
                        <Font size="20.0" />
                    </font>
                </Label>
            <SBoxInverseSBoxControl VBox.vgrow="ALWAYS"  fx:id="sbox"/>
            </children>
            <VBox.margin>
                <Insets right="10" bottom="0" left="10" top="10"/>
            </VBox.margin>
        </VBox>
        <Button fx:id="buttonSwitch"
                onAction="#handleSwitchingBetweenEncrDecr"
                alignment="TOP_RIGHT" text="show decryption">
            <VBox.margin>
                <Insets top="0"  right="10" bottom="10"/>
            </VBox.margin>
        </Button>
    </children>
</VBox>
```

V atributu obalového *VBox* layoutu je uvedena cesta ke kontroléru. Obsah tohoto kontroléru je na ukázce níže. Výsledný vzhled grafického prostředí z FXML souboru je zobrazen na Obrázku 11. Nalevo je prostředí zachyceno před kliknutím a napravo pak po kliknutí na tlačítko s *fx:id* buttonSwitch.

```
public class FXML_MainWindowController implements Initializable {

    @FXML
    private SBoxInverseSBoxControl sbox;

    @FXML
    private Button buttonSwitch;
```

```
    @FXML
    private Label sBoxLabel;

    private boolean encryption;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        encryption = true;
    }
    @FXML
    private void handleSwitchingBetweenEncrDecr(ActionEvent event) throws
    Exception {
        if (encryption) {
            encryption = false;
            sbox.setToInverseSBox();
            buttonSwitch.setText("show encryption");
            sBoxLabel.setText("Inverse SBox");
        } else {
            encryption = true;
            sbox.setToSBox();
            buttonSwitch.setText("show decryption");
            sBoxLabel.setText("SBox");
        }
    }
}
```

SBox

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
10	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
11	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
12	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
13	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
14	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
15	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Inverse SBox

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
10	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
11	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
12	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
13	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
14	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
15	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Obrázek 11 – Grafické prostředí z FXML souboru před a po kliknutí na tlačítko

V poslední ukázce níže je zobrazen postup načítání FXML dokumentu do scény, která je posléze zobrazena v oknu aplikace. Tento postup je vykonáván při startu aplikace.

```
public class JavaFXMainApplication extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root =FXMLLoader.load(getClass().
        getResource("FXML_MainWindow.fxml"));
```

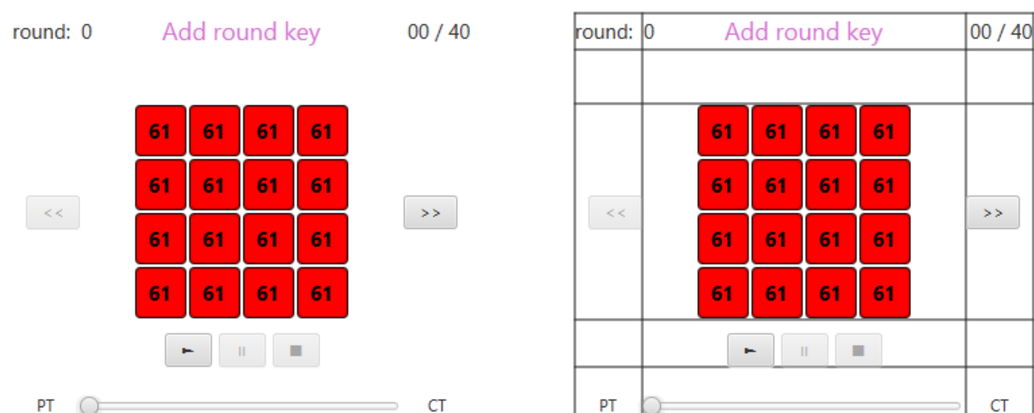
```
        stage.setScene(new Scene(root));
        stage.setTitle("AES Visualization");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

5.3 Layouty JavaFX

JavaFX nabízí poměrně širokou škálu layoutů, které jsou připraveny pro okamžité použití. Následující výčet uvádí jen layouty skutečně použité v aplikaci.

- `VBox` – umožňuje vkládat komponenty pod sebe.
- `HBox` – slouží pro vkládání komponent vedle sebe.
- `GridPane` – tabulkový layout, jehož výhodou je pozicování komponent v buňkách na určitém řádku a sloupci.
- `StackPane` – jednoduchý layout umožňující vkládat jednotlivé komponenty na sebe. Výsledkem je set komponent, které se překrývají. Nahoře je pak vždy naposled přidaná komponenta.
- `Pane` – layout, ve kterém je dobré používat absolutní pozicování.
- `AnchorPane` – layout ve kterém se prvky umisťují v offsetech od jeho hran.

Za pomoci těchto základních layoutů je možné vytvořit komplexnější layouty, ve kterých se mohou nacházet komponenty na definovaných pozicích, které spolu logicky souvisí. Příklad této komponenty je zobrazen na Obrázku 12.



Obrázek 12 – `StateArrayControl` bez a se zobrazenou tabulkou

Layout *StateArrayControl* dědí třídu *GridPane*, to nám umožňuje umísťovat komponenty na potřebná místa. Uprostřed této komponenty na třetím řádku v prvním sloupci se nachází další layout. Jedná se o jednoduchý Pane layout, který obsahuje jednotlivé *StackPane* layouty. V těchto layoutech je vždy *Text* a *Rectangle*. Každý z těchto *StackPane* layoutů musí být dále v kódů dostupný, neboť se jednotlivé popisky a barva samotného obdélníčku v průběhu běhu programu mění. Z tohoto důvodu jsou tyto *StackPane* layouty umístěny ve dvourozměrném poli.

5.3.1 Svazování vlastností komponent

Samotný *GridPane* layout podporuje zvětšování a zmenšování podle dostupného místa pro daný layout. Nicméně v případě *StateArrayControl* je celá situace o něco složitější. Cílem je, aby jednotlivé *StackPane*, reprezentující buňky ve stavové matici, vždy byly čtvercové, aby se při zvětšování pouze šířky zvětšoval i celý řádek se stavovou maticí tak, aby nedocházelo k překrývání jednotlivých komponent a v neposlední řadě také to, aby si buňky mezi sebou držely dané rozestupy. Toho všeho lze v JavaFX dosáhnout. Jednotlivé vlastnosti layoutů se dají svázat s vlastnostmi jiných layoutů a při změně velikosti jisté vlastnosti třeba velikosti obalového layoutu lze vlastnosti dalších layoutů uvnitř podle toho přepočítat. Když je vše správně nastaveno, dochází ke změnám automaticky a s požadovaným výsledkem.

Svázat vlastnost (property) je jednoduché, po získání požadované vlastnosti je nad ní zavolána metoda *bind()*. Pro ilustraci jsou některé případy svázání zobrazeny níže.

```
rectangle.widthProperty().bind(stateArrayCellsContainer.widthProperty().
divide(6).subtract(3));

rectangle.heightProperty().bind(stateArrayCellsContainer.widthProperty().
divide(6).subtract(3));

for (int i = 0; i < 4; i++) {
	for (int j = 0; j < 4; j++) {
		StateArrayCell cell = new StateArrayCell();

		cell.layoutXProperty().bind(stateArrayCellsContainer.
		widthProperty().divide(6).multiply(i + 1));

		cell.layoutYProperty().bind(stateArrayCellsContainer.
		widthProperty().divide(6).multiply(j));

		stateArrayCellsContainer.getChildren().add(cell);
	}
}
```

V prvních dvou případech upravujeme velikost obalového čtverce, ve kterém je umístěn text bajtu stavové matice. Na Obrázku 12 mají tyto buňky červenou barvu. Jelikož je

StateArrayCellContainer (*Pane*) součástí *GridPane* buňky na řádku 3 ve druhém sloupci, je nutné ošetřit taky mezery po stranách. Tyto mezery mají šířku 1/6 šířky samotné *StateArrayCellContainer Pane*. Proto se při svazování šířky a výšky používá právě jedna šestina obalové Pane. Mezery mezi jednotlivými buňkami stavové matice jsou zaručeny metodou *substract()*, která ze šířky a výšky odečte počet pixelů zadaných jako parametr.

Ve vnořeném cyklu se nastavují *LayoutXProperty* a *LayoutYProperty*, které slouží k nastavení pozic jednotlivých buněk. U souřadnice x je pak přičtena jedna šestina právě kvůli zachování mezery z levé strany. Metoda *getChildren.add(cell)* už poté jenom přidá samotnou buňku obalové Pane tak, aby se mohla vykreslit.

Jako poslední je potřeba ošetřit, aby řádek třetí řádek *GridPane* měl vždy výšku přesně takovou, aby se do něho vešly všechny buňky stavové matice. K tomu lze využít třídu *RowConstraints*, která se stará o nastavování pravidel na řádku *GridPane*. Postup je zobrazen níže.

```
RowConstraints rc2 = new RowConstraints();
rc2.setVgrow(Priority.ALWAYS);

rc2.minHeightProperty().bind(((Region)stateArrayCellsContainer.
getChildren().get(0)).widthProperty().multiply(4));

rc2.maxHeightProperty().bind(((Region)stateArrayCellsContainer.
getChildren().get(0)).widthProperty().multiply(4));
```

Předchozí ukázka kódu zobrazuje vytvoření *RowConstraints*. Metoda *rc2.setVgrow()* s atributem *Priority.ALWAYS* donutí měnit výšku řádku podle dostupného místa. V dalších krocích už se jen sváže minimální a maximální výška řádku jako čtyřnásobek šířky jedné buňky stavové matice. Tuto instanci třídy *RowConstraints* je pak třeba ještě nastavit *GridPane* layoutu, který chceme takto ovlivňovat.

Na Obrázku 13 je zobrazena komponenta pro vizualizaci vstupních a výstupních bajtů. Komponenta je ošetřena pomocí svazování vlastností tak, aby vyplňovala maximální možný prostor. Na Obrázku je tato komponenta třikrát, poprvé se šířkou 33 %, podruhé se šířkou 66 %, a v posledním případě se šířkou 100 %. Takto ošetřená komponenta umožňuje použití pro různé velikosti oken bez potřeby znát zdrojový kód skrývající se za touto komponentou.

Obrázek 13 – Vlastní JavaFX komponenta s různou šířkou

5.4 Sázecí systém LaTeX

Jazyk LaTeX obecně slouží k formátování textových souborů. LaTeX umožňuje oddělit vzhled od obsahu. Autorovi tak umožňuje soustředit se více na obsah než na samotnou výslednou vizuální stránku.

Pro zobrazení matematických vzorců je v aplikaci využita veřejná knihovna JLaTeXMath. Tato knihovna je k dispozici ke stažení zdarma[2]. Bez této knihovny nelze za použití knihovny JavaFX jakkoliv standardně zobrazovat matematické výrazy, dokonce nelze ani jakkoliv zobrazit horní a dolní indexy u obyčejného textu. Za pomocí této knihovny jsou matematické zápisy převedeny na obrázek, který je posléze zobrazen v samotné aplikaci.

Důležité je si uvědomit, že tato knihovna nedává k dispozici celý jazyk, jedná se spíše o emulátor, který využívá syntaxi jazyka LaTeX. Tato knihovna neumí například importovat balíčky a další věci. Její funkce je tak omezena skutečně pouze na vytváření matematických zápisů a jejich jednoduché stylování, což ostatně napovídá i název knihovny.

Příklad rovnice v LaTeX

Pro ilustraci schopností této knihovny je znázorněn příklad. V následující části zdrojového kódu je uveden datový typ `String`, ze kterého je posléze zkonstruován obrázek s matematickými rovnicemi.

```
private static String mixColumnsEquations = ("\\begin{align}\n"
      + "S`_{0,j} &= (2 \\cdot S_{0,j}) \\oplus (3 \\cdot S_{1,j}) \\oplus
      S_{2,j} \\oplus S_{3,j}\\\\\n "
      + "S`_{1,j} &= S_{0,j} \\oplus (2 \\cdot S_{1,j}) \\oplus (3 \\cdot
      S_{2,j}) \\oplus S_{3,j}\\\\\n "
      + "S`_{2,j} &= S_{0,j} \\oplus S_{1,j} \\oplus (2 \\cdot S_{2,j})
      \\oplus (3 \\cdot S_{3,j})\\\\\n "
      + "S`_{3,j} &= (3 \\cdot S_{0,j}) \\oplus S_{1,j} \\oplus S_{2,j}
      \\oplus (2 \\cdot S_{3,j})\\\\\n "
      + "\\end{align}\\\\\n"0);
```

[2] https://forge.scilab.org/index.php/p/jlatexmath/

Jmenný prostor align značí ohraničení, ve kterém je umožněno zarovnání jednotlivých rovnic v tomto prostoru se nacházejících. Zarovnání je podle znaku =, před kterým je nutno uvést znak &. Oplus a cdot jsou matematické operátory a za znakem _ se nachází dolní indexy. Každou rovnici je třeba ukončit ukončovacím znakem \, pro zápis v datovém typu `String` v jazyce Java je nutno uvést zpětná lomítka dvě, neboť jedno je bráno jako řídící znak. Další dvojice lomítek je zde kvůli jmennému prostoru align a znaky \n indikují konec řádku.

Pro takto zapsaný `String` použijeme knihovnu JLaTeXMath a dostaneme obrázek, jemuž odpovídá Obrázek 14.

$$\begin{aligned} S'_{0,j} &= (2 \cdot S_{0,j}) \oplus (3 \cdot S_{1,j}) \oplus S_{2,j} \oplus S_{3,j} \\ S'_{1,j} &= S_{0,j} \oplus (2 \cdot S_{1,j}) \oplus (3 \cdot S_{2,j}) \oplus S_{3,j} \\ S'_{2,j} &= S_{0,j} \oplus S_{1,j} \oplus (2 \cdot S_{2,j}) \oplus (3 \cdot S_{3,j}) \\ S'_{3,j} &= (3 \cdot S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \cdot S_{3,j}) \end{aligned}$$

Obrázek 14 – Soustava rovnic operace kombinování sloupců

5.5 Animace v JavaFX

Existují dva způsoby, jak přistupovat k animacím v JavaFX. První využívá třídy `Transition`, druhý způsob pak třídy `Timeline` Každý z těchto přístupů se hodí pro různé typy použití, obě třídy jsou podtřídy třídy *`javafx.animation.Animation`*.

5.5.1 Třída *`Transition`* pro vytváření přechodových animací

Všechny instance tříd, které dědí ze třídy `Transition` (třída `Transition` je abstraktní), mají interní časovou osu. S touto osou ovšem nelze nijak manipulovat. Animaci nelze zastavit, poté jí po uživatelově interakci znovu pustit a podobně. V podstatě jediné, co má tato časová osa na starost, je zaznamenání doby od začátku do konce animace. Tyto jednoduché animace se vždy starají jen o jednu vlastnost dané komponenty. Abychom mohli použít více těchto animací najednou, lze je umístit do `ParallelTransition` popř. do `SequentialTransition`. Tyto animace se pak budou spouštět buď paralelně, nebo tak jak byly přidány postupně za sebou.

Příklady již předpřipravených přechodových animací v JavaFX:

- `FadeTransition` – tento přechod se hodí mezi animováním stavu kdy je něco viditelné a posléze zmizí popř. naopak.

- *PathTransition* – animace, která umožňuje postupný pohyb komponenty, např. nějakého geometrického tvaru, po trajektorii. Trajektorie se může skládat jak z jednotlivých bodů, tak z přímek nebo třeba různých druhů křivek.
- *FillTransition* – animace, jež mění výplň tvarů v čase. Toho je dosáhnuto postupnou aktualizací proměnné *fill* u animovaného tvaru.
- *PauseTransition* – umožňuje pozastavit animaci po požadovanou dobu, a poté spustit následující akci (např. další animaci).
- *RotateTransition* – přináší animaci rotace. Rotace je vypočítána tak, aby probíhala plynule za pomoci aktualizace rotate proměnné u jakékoliv *Node* (uzlu). Hodnota o kolik se má uzel otočit je nastavována ve stupních.
- *ScaleTransition* – slouží pro animování zvětšování a zmenšování uzlu. Toho je docíleno aktualizací proměnných *scaleX*, *scaleY*, *scaleZ* v průběhu trvání animace.
- *StrokeTransition* – stará se o postupnou změnu barvy rámečku u tvarů. Změna probíhá aktualizací stroke proměnné daného tvaru v průběhu běhu animace.
- *TranslateTransition* – animace umožňující pohyb jakéhokoliv uzlu po jeho osách x, y, z. Animace v průběhu aktualizuje proměnné *translateX*, *translateY* a *translateZ* animovaného uzlu.

Všechny typy animací jdou sice kombinovat pomocí instancí tříd *ParallelTransition* a *SequentialTransition*, nicméně pokud je vyžadováno animovat více objektů po různé časové intervaly atd., stává se toto řešení značně nepřehledné a nepraktické. Odpovědí je třída *Timeline*.

5.5.2 Třida *Timeline* pro vytváření animací

Třída *Timeline* umožnuje vytvořit objekt časové osy. Za pomocí instance této třídy pak můžeme spouštět, pozastavovat nebo úplně zastavit přehrávání animace. Animace je tvořena klíčovými snímky (*KeyFrames*), tyto klíčové snímky udávají hodnoty jednotlivých proměnných animací dotčených objektů v konkrétním čase. Hodnoty animovaných proměnných jsou pak mezi jednotlivými klíčovými snímky aktualizovány.

Pro pochopení je vše znázorněno na příkladu z reálné implementace vizualizačního programu, který byl pro názornost zjednodušen.

```
Timeline anim = new Timeline();
SimpleDoubleProperty cell10height = new
SimpleDoubleProperty(cells[1][0].layoutYProperty().get());
SimpleDoubleProperty cell10width = new
SimpleDoubleProperty(cells[1][0].layoutXProperty().get());
```

Nejdříve si vytvoříme samotnou instanci třídy `Timeline`. Poté si vytvoříme instance třídy `SimpleDoubleProperty` pro vlastnosti buněk, které chceme animovat. Pro zjednodušení je uvedena jen jedna buňka na druhém řádku v prvním sloupci stavové matice. V Následující ukázce zdrojového kódu jsou zobrazeny klíčové snímky, které se starají o animaci této buňky.

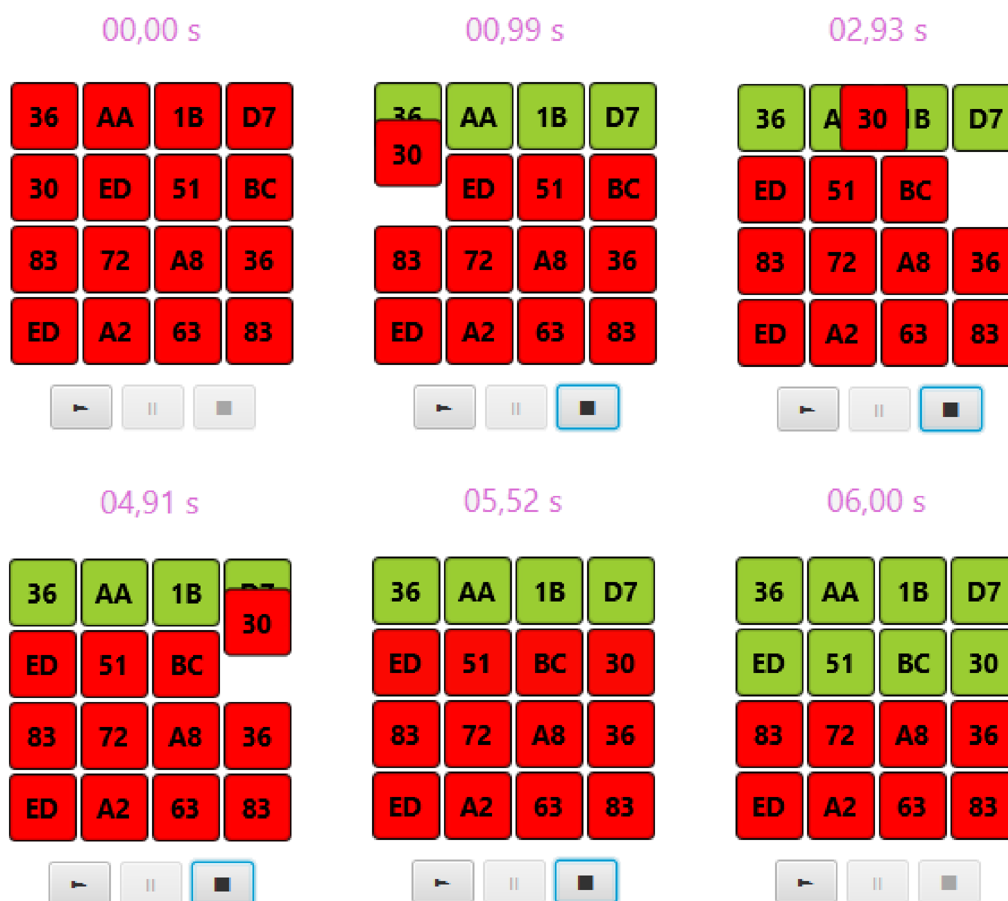
```
KeyFrame kf0 = new KeyFrame(Duration.millis(500),
      new KeyValue(cell10height, cell10height.get()));

KeyFrame kf1 = new KeyFrame(Duration.millis(1500),
      new KeyValue(cell10width, cell10width.get()),
      new KeyValue(cell10height,stateArrayCellsContainer
      .widthProperty().divide(6).multiply(0).get()));

KeyFrame kf3 = new KeyFrame(Duration.millis(4500),
      new KeyValue(cell10height, stateArrayCellsContainer.widthProperty()
      .divide(6).multiply(0).get()),
      new KeyValue(cell10width,stateArrayCellsContainer.widthProperty()
      .divide(6).multiply(4).get()));

KeyFrame kf4 = new KeyFrame(Duration.millis(5500),
      new KeyValue(cell10height,stateArrayCellsContainer.widthProperty()
      .divide(6).multiply(1).get()));

KeyFrame kf5 = new KeyFrame(Duration.millis(5500),
      new KeyValue(cells[1][0].rectangle.fillProperty(), Color.RED),
      new KeyValue(cell10height, cell10height.get()));

KeyFrame kf6 = new KeyFrame(Duration.millis(6000),
      new KeyValue(cells[1][0].rectangle.fillProperty(),
      Color.YELLOWGREEN));
```

Z ukázky zdrojového kódu výše je vidět, že každý klíčový snímek má jasně definovaný čas. V těchto okamžicích jsou velikosti proměnných daných objektů v klíčových hodnotách přesně takové, jak je udávají klíčové hodnoty. Mezi jednotlivými klíčovými hodnotami jsou korespondující proměnné přepočítávány a automaticky aktualizovány. Tyto změny se graficky okamžitě projevují a vzniká tak efekt běžící animace.

```
cells[1][0].layoutYProperty().bind(cell10height);
cells[1][0].toFront();
cells[1][0].layoutXProperty().bind(cell10width);
anim.getKeyFrames().addAll(kf0, kf1, kf2, kf3, kf4, kf5, kf6);
anim.play();
```

Po nastavení proměnných v klíčových hodnotách už stačí jen jednotlivé snímky přiřadit instanci třídy `Timeline`, animované vlastnosti přiřadit jednotlivým buňkám a animaci spustit. Vše je

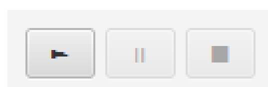
zobrazeno výše. Průběh výsledné animace je zobrazen na obrázku 11. V rámci příkladu je relevantní jen buňka s hodnotou 30, přesto jsou na obrázku zachyceny i pohyby ostatních buněk, které jsou pro příklad irelevantní. Zobrazeny jsou z důvodu použití animace v reálné implementaci programu. Metoda `toFront()` je použita z důvodu potřeby mít pohybující se buňku zobrazenou nad ostatními buňkami.



Obrázek 15 – Část průběhu `Timeline` animace prohození řádků

5.5.3 Ovládání `Timeline` animace pomocí tlačítek

Aby se animace dala ovládat, je třeba jí přiřadit ovládací prvky. K tomu nejlépe poslouží tlačítka. Ovládací panel použitý v reálné aplikaci je zobrazen na Obrázku 16.



Obrázek 16 – Ovládací panel pro ovládání animace

Ovládací panel je složen ze tří tlačítek. První tlačítko má na starost spuštění animace. Druhé umožňuje pozastavit běžící animaci a třetí tlačítko slouží k zastavení běžící či pozastavené animace. Aby byla k dispozici vždy jen tlačítka, která lze v daný okamžik použít, je třeba je

svázat s instancí *Timeline* animace, kterou mají tlačítka ovládat. Jak na to je zobrazeno v ukázce zdrojového kódu níže.

```
buttonStart.disableProperty().bind(Bindings.or(animation.statusProperty()
      .isEqualTo(Animation.Status.RUNNING), lastRoundProperty));

buttonPause.disableProperty().bind(animation.statusProperty()
      .isNotEqualTo(Animation.Status.RUNNING));

buttonStop.disableProperty().bind(animation.statusProperty()
      .isEqualTo(Animation.Status.STOPPED));
```

V posledním kole není k dispozici animace, proto se u tlačítka start bere v potaz ještě instance třídy *SimpleBooleanProperty* s názvem *lastRoundProperty*, která jen říká, zda se jedná o poslední kolo nebo ne.

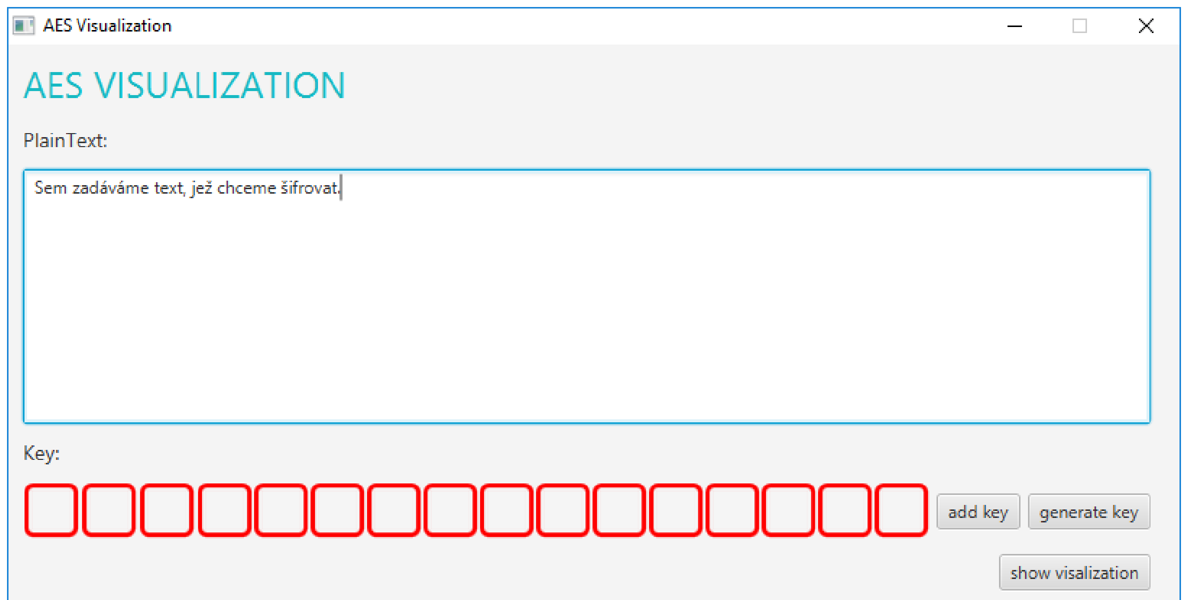
5.6 Uživatelské rozhraní pro ovládání aplikace

Po spuštění aplikace se uživateli zobrazí úvodní okno aplikace. V tomto okně se nachází textové pole sloužící k zadání nezašifrovaného vstupního textu, který chce uživatel zašifrovat. Tento text může být libovolně dlouhý a je označen anglickým slovem PlainText. Dále je zde komponenta starající se o zadávání klíče. Pro zobrazení jednotlivých znaků klíče jsou zde k dispozici čtverečky orámované červenou barvou, vedle těchto čtverečků se nacházejí dvě tlačítka, první slouží pro přidání uživatelem definovaného klíče (add key), druhé pak ke generování (generate key). Klíč je správně zadán, pokud jsou všechny rámečky jednotlivých znaků klíče zobrazeny zelenou barvou. Úvodní obrazovka aplikace je zobrazena na Obrázku 17.

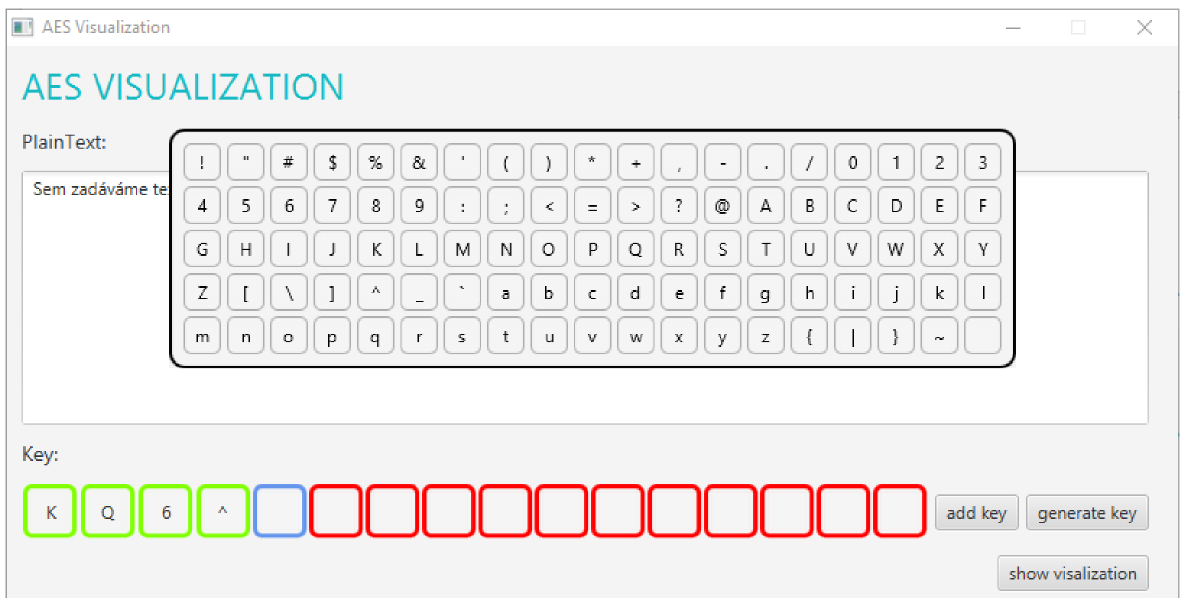
Pokud se uživatel rozhodne zadat vlastní klíč, zobrazí se klávesnice všech podporovaných znaků. Pomocí myši pak lze naklikat jednotlivé znaky klíče, klávesnice s posledním zadaným znakem automaticky zmizí. Během zadávání klíče nelze klávesnici zavřít. Z praktických důvodů jsou k dispozici jen znaky, které lze zobrazit, a které v systému kódování UTF-8 zabírají pouze jeden bajt. Ve skutečnosti lze jako klíč pro AES zvolit jakýkoliv bajt, v grafickém prostředí aplikace nikoliv. Zadávání znaků pomocí klávesnice je zobrazeno na Obrázku 18.

Pokud je vstupní text společně s klíčem v pořádku zadán, lze kliknutím na tlačítko v pravém dolním rohu (show visualization) zobrazit průběh šifrování a dešifrování zadaného textu s použitím zadaného klíče. Okno s průběhem šifrování je zobrazeno na Obrázku 19. V opačném

případě se zobrazí chybová hláška, nabádající uživatele k vyplnění všech potřebných vstupních dat.



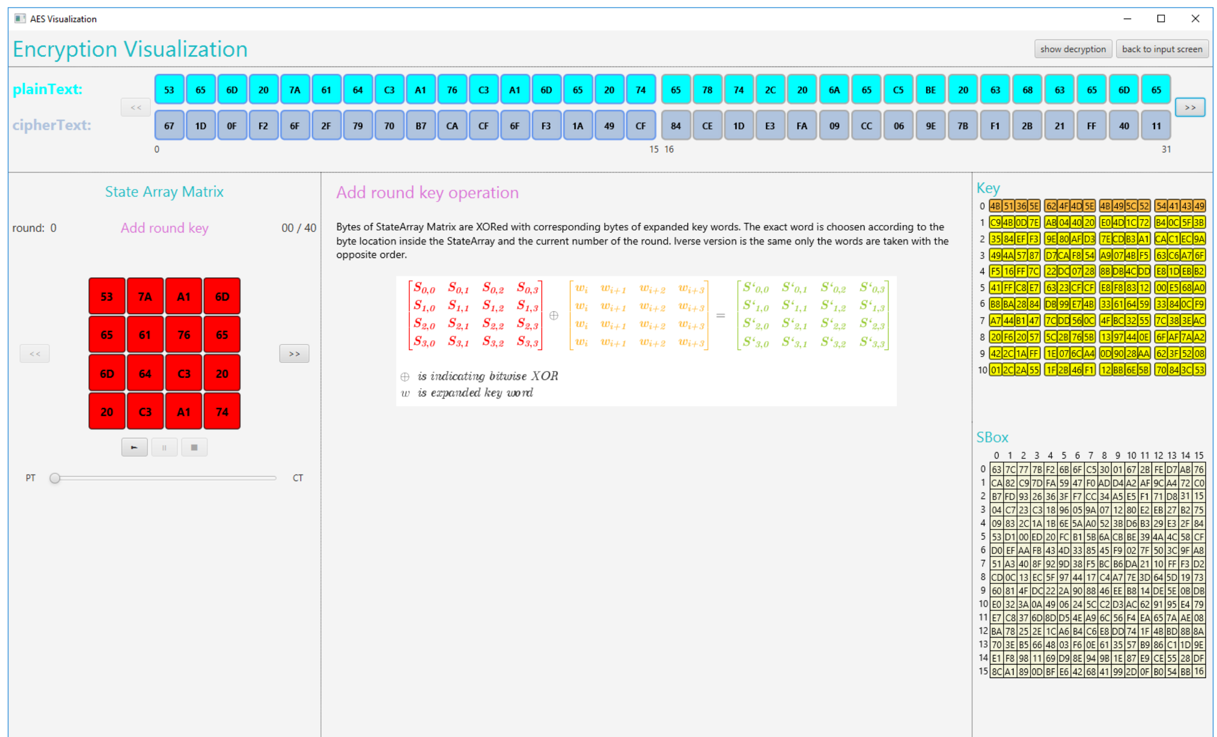
Obrázek 17 – Úvodní obrazovka aplikace



Obrázek 18 – Zadávání vlastního klíče

Obrázek 19 je sice kvůli své velikosti hůře čitelný, ale lze si z něho udělat představu o rozložení okna, ve kterém je zobrazen průběh šifrovaní či dešifrování. Větší verzi Obrázku 19 lze najít v Příloze A. Je patrné, že toto okno je rozděleno na 3 řádky, na každém řádku se nachází tři sloupce.

UML diagram tříd hlavního okna je zobrazen v Příloze B, a je k dispozici i v digitální formě na přiloženém CD.

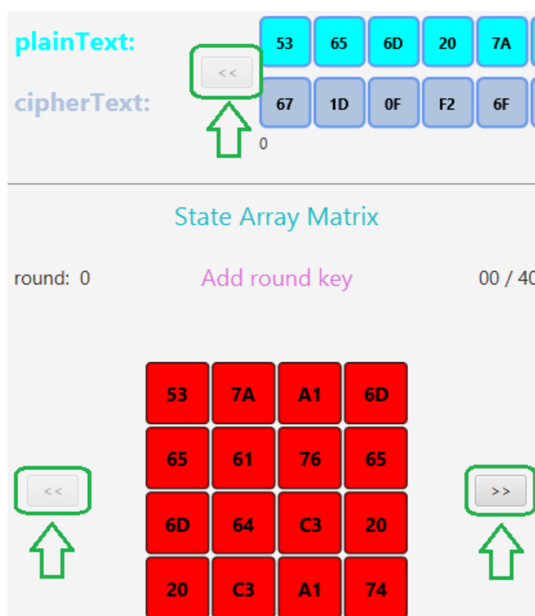


Obrázek 19 – Hlavní okno aplikace s průběhem šifrování

V prvním řádku na pravé straně nás zajímají dvě tlačítka, první zleva označené popiskem show decryption / encryption, a druhé zleva s popiskem back to input screen. První tlačítko po stisknutí přepíná mezi zobrazením průběhu šifrování nebo dešifrování. Pro dešifrování je použit stejný klíč jako pro šifrování a vstupní data jsou výsledkem předchozího šifrovacího algoritmu. Druhé tlačítko jednoduše navrací na úvodní obrazovku, kde je možné zadat nový text pro zašifrování a rovněž nový klíč.

Celý druhý řádek obsahuje jednu komponentu. Tato komponenta se stará o zobrazení jednotlivých bajtů vstupního nezašifrovaného textu a výstupního zašifrovaného textu. Pro vstupní text jsou horní šestnáctice čtverečků, zatímco pro výstupní text dolní. K dispozici jsou dva bloky o velikosti 16 bajtů. V případě vstupního textu o velikosti do 16 bajtů je pravá polovina čtverečků prázdná. V dolní části komponenty jsou zobrazeny indexy pozic bajtů ve vstupním a výstupním textu. Mezi jednotlivými bloky lze přepínat dopředu a dozadu pomocí tlačítek označených popisky << a >>, která obklopují čtverečky pro zobrazení hodnoty bajtu. Detail tlačítka předchozí blok společně s tlačítky pro přepínání operací prováděných na stavové matici je zobrazen na Obrázku 20. Maximální počet bloků vstupního textu není programově nijak omezen tzn., že délka uživatelem zadaného vstupního textu není nikterak omezena. Bajty

aktivního bloku jsou ohraničeny modrou barvou. Po přepnutí mezi šifrováním a dešifrováním se horní řádek s dolním prohodí, tak aby první řádek reprezentoval vždy vstupní data do šifrovacího nebo dešifrovacího algoritmu.



Obrázek 20 – Detail ovládacích tlačítek posouvání operací a bloků dat

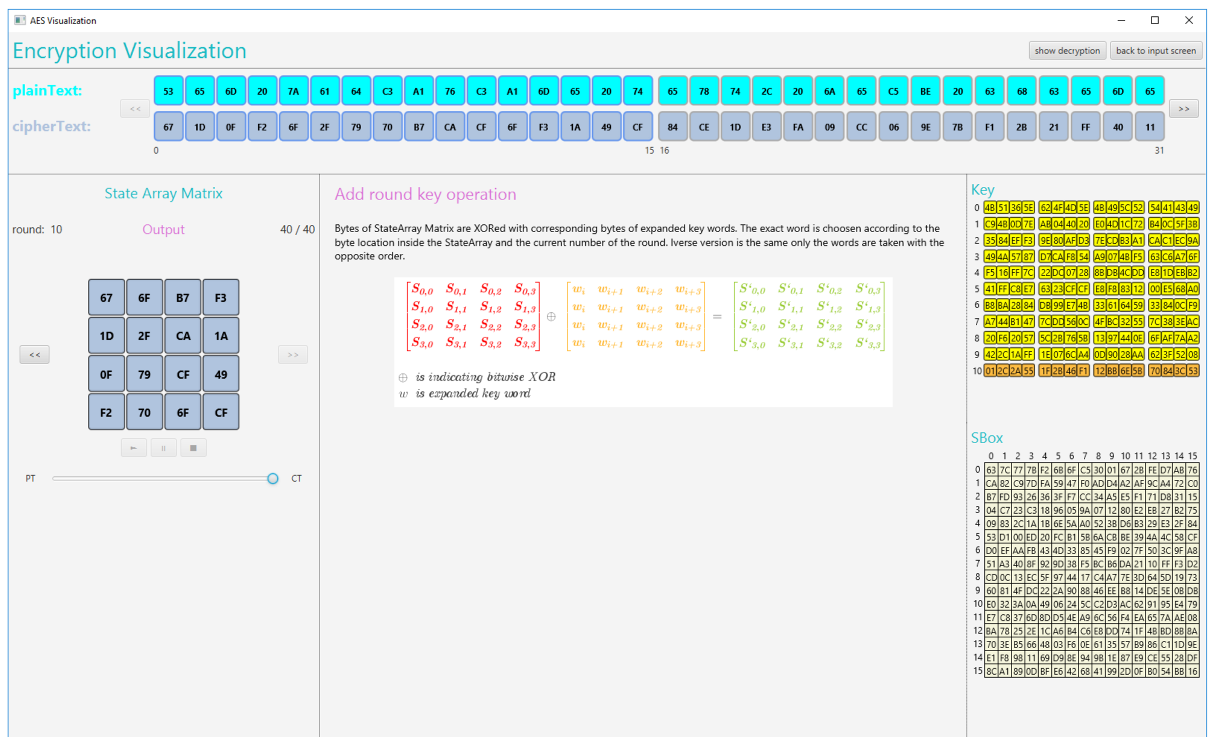
Třetí řádek okna je graficky rozdělen do tří sloupců. V prvním sloupci je zobrazena stavová matice. Celá komponenta použitá v tomto sloupci je detailně zobrazena na Obrázku 12. Tato komponenta je pro aplikaci klíčová. Nalezneme v ní název operace, pro níž jsou zobrazena data. Mezi operacemi lze přepínat tlačítky s popisky << a >>, obklopující buňky stavové matice z leva a zprava viz Obrázek 20. Pro rychlejší přepínaní lze rovněž využít jezdce ve spodní části komponenty. Tento jezdec rovněž indikuje stav průběhu šifrování či dešifrování od vstupních po výstupní data. Pro každou operaci lze v tomto bloku spustit grafickou animaci zobrazující celý průběh operace. Pro všechny animace platí jednotný formát, na vstupu jsou bajty ohraničeny v červených čtverečcích. Po vykonání operace jsou výstupní bajty dané operace zobrazeny v barvě zelené.

Pro teoretický popis právě zobrazené operace slouží prostřední sloupec třetího řádku. V tomto popisu je teoretický popis operace, a to včetně matematických vzorců potřebných pro pochopení operace.

V posledním sloupci třetího sloupce se v horní části nachází komponenta zobrazující expandovaný klíč. Klíč používaný v současné rundě je zvýrazněn oranžovou barvou. Klíč je graficky strukturovaný tak, aby přehledně odděloval jednotlivá slova klíče. Komponenta je

využita při animaci operace přidání podklíče. V dolní části třetího sloupce je zobrazena komponenta se substituční tabulkou (SBox nebo Inverse SBox). Tato komponenta obsahuje jiná data v závislosti na zobrazovaném algoritmu (šifrování nebo dešifrování), a využívá se při běhu animace operace záměna bajtů.

V konečném stavu (poté co se uživatel prokliká všemi stavy stavové matice) odpovídají hodnoty bajtů stavové matice hodnotám výstupu šifrovacího či dešifrovacího algoritmu. Tyto bajty jsou stejné jako bajty ve spodním řádku komponenty v horní části okna, která zobrazuje vstup a výstup pro šifrovací či dešifrovací algoritmus. Zároveň se barva bajtů stavové matice změní na barvu totožnou s barvou bajtů výstupu šifrovacího či dešifrovacího algoritmu. Konečný stav vizualizace je zobrazen na Obrázku 21.



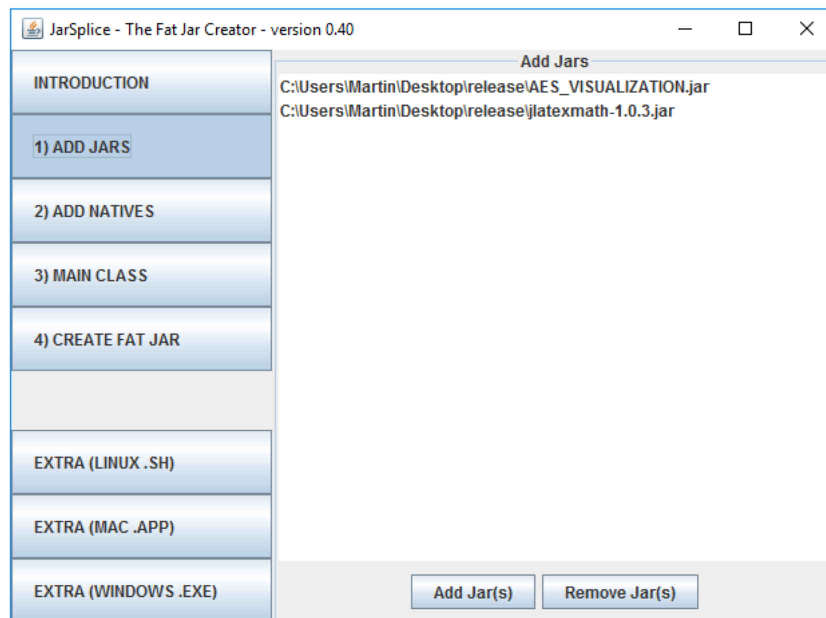
Obrázek 21 – Konečný stav vizualizace pro konkrétní blok dat

5.7 Distribuce aplikace

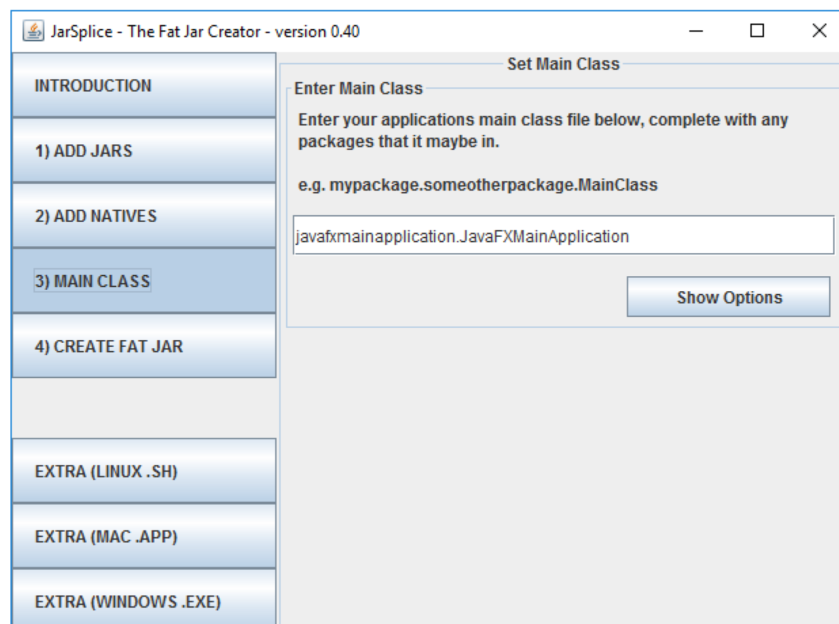
Jelikož byla při implementaci aplikace použita externí knihovna, je třeba vyřešit sloučení aplikace a této knihovny do jednoho spustitelného souboru. Tento proces obstarává např. aplikace s názvem JarSplice. Tato aplikace je k dispozici ke stažení zdarma[3]. Práce s touto aplikací je velice jednoduchá, stačí jen vložit jar soubor aplikace a jar soubor knihovny, poté zbývá už jen nastavit cestu ke spustitelné `main` metodě, kterou chceme spustit při startu

[3] Aplikace ke stažení z: http://ninjacave.com/jarsplice

aplikace. Výsledkem je nový jar soubor obsahující jak zdrojový kód aplikace, tak externí knihovnu.



Obrázek 22 – Přidávání jar souborů v aplikaci JarSplice



Obrázek 23 – Definování spustitelné `main` metody v aplikaci JarSplice

Na Obrázku 22 je zobrazeno okno aplikace JarSplice a v něm přidány dvě knihovny, které budou posléze sloučeny. Přidat je možno libovolný počet knihoven. Následuje zadání cesty ke spustitelné `main` metodě viz Obrázek 23. Je nutné uvést celou cestu včetně balíčku, ve kterém se třída s metodou `main` nachází. Poté už zbývá jen kliknout na čtvrtou záložku s názvem CREATE FAT JAR, po kliknutí a zadání umístění budoucího souboru se vytvoří nový

spustitelný jar soubor, který bude obsahovat všechny vložené jar soubory. Spustitelný jar soubor s finální verzí vizualizační aplikace je k dispozici na přiloženém CD.

V případě potřeby lze použít také možnost vytvoření spustitelného exe souboru. Výsledný soubor se tak bude tvářit jako aplikace pro operační systém Windows, nicméně výsledná aplikace nebude multiplatformní. Soubor s příponou exe vytvoříme po kliknutí na položku EXTRA (WINDOWS.EXE).

Aplikace byla testována na operačních systémech Windows 8.1, Windows 10 a Ubuntu 16.04. Díky použití programovacího jazyka Java se jedná o multiplatformní aplikaci, pro jejíž spuštění je třeba mít nainstalovanou sadu programů Java Virtual Machine.

6 ZÁVĚR

Během řešení bakalářské bylo nejtěžší nastudovat AES natolik, aby bylo možno provést vlastní implementaci. Pro pochopení AES byla naprosto klíčová kniha [1]. Poté byla tato implementace skutečně vytvořena, nicméně to byl jen začátek. Z hlediska časových nároků toto zabralo možná 5 %, možná méně. Následovala otázka, jak uložit všechna data z průběhu šifrování a dešifrování, a tak vznikla vlastní datová struktura založená na dvourozměrných polích bajtů odpovídajících stavové matici implementace AES, aby docházelo při šifrování a dešifrování zároveň k ukládání průběhu dat z těchto algoritmů. Poté bylo teprve možno přistoupit k tvorbě grafické podoby aplikace.

Samotná tvorba uživatelského rozhraní, komponent pro zobrazení vstupních a výstupních bajtů a rovněž stavové matice byla nejtěžší a časově nejnáročnější část celé práce. Bylo přímo vyžadováno podrobně nastudovat možnosti knihovny JavaFX. Tato knihovna pohodlně postačila pro samotnou realizaci. Umožňuje relativně pohodlně pracovat s jednotlivými layouty, popřípadě vytvářet vlastní. Problematika vytváření vlastních layoutů se později ukázala náročná hlavně v případě responzivní aplikace. Aby bylo možno vlastní komponenty zvětšovat či zmenšovat dle potřeby a velikosti okna aplikace, bylo potřeba tyto komponenty ošetřit, a to i na několika místech, nicméně nakonec bylo dosaženo požadovaného chování, a tak je vizualizační část aplikace responzivní a okno lze libovolně zvětšovat nad rozlišení 1366x768 obrazových bodů, což je minimální podporované rozlišení aplikace. Při menším rozlišení se části aplikace stávají nečitelné.

Vytváření animací bylo zdlouhavé, ale poměrně zábavné. Nejtěžší část při vytváření animací byla spojena s responzivním charakterem aplikace. Bylo požadováno, aby bylo možno zvětšovat okno aplikace i za běhu animace, což se nakonec povedlo, a řešení je plně funkční.

Jediná limitace spojená s knihovnou JavaFX, která se projevila v průběhu implementace aplikace, byla nemožnost psát v textu horní a dolní indexy. Z tohoto důvodu musela být využita externí knihovna, která byla posléze rovněž použita pro zobrazení matematických vzorců.

Řešení bakalářské práce bylo značně obohacující, a to jak v oblasti kryptografie a počítačové bezpečnosti, tak i z hlediska práce s knihovnou JavaFX. Do budoucna by se aplikace mohla rozšířit o detailnější animaci expandování klíče, stejně tak by bylo dobré implementovat vysvětlení matematiky konečných těles, popřípadě ještě 192 a 256bitovou verzi AES.

7 POUŽITÁ LITERATURA

[1] STALLINGS, William. *Cryptography and network security: principles and practice.*, 5th ed. Boston: Prentice Hall, c2011. ISBN 0136097049.

[2] AL TAMIMI, Abdel-Karim. *Performance Analysis of Data Encryption Algorithms* [online]. Washington University, 2008 [cit. 2017-05-05]. Dostupné z: http://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf

[3] *National Institute of Standards and Technology: Information Technology Laboratory* [online]. U.S. Department of Commerce, 1996 [cit. 2017-05-05]. Dostupné z: http://csrc.nist.gov

[4] EBBERS, Hendrik. *Mastering JavaFX 8 Controls*. New York: McGraw Hill Education, 2014. ISBN 0071833773.

[5] PECINOVSKÝ, Rudolf. *Java 8: úvod do objektové architektury pro mírně pokročilé. První vydání. Praha: Grada Publishing,* 2014. Knihovna programátora (Grada). ISBN 978-80-247-4638-8.

Příloha A – *Obrázek hlavního okna vizualizační aplikace*

AES Visualization

Encryption Visualization

show decryption | back to input screen

	0															15	16															31
plainText:	53	65	6D	20	7A	61	64	C3	A1	76	C3	A1	6D	65	20	74	65	78	74	2C	20	6A	65	C5	BE	20	63	68	63	65	6D	65
cipherText:	67	1D	0F	F2	6F	2F	79	70	B7	CA	CF	6F	F3	1A	49	CF	84	CE	1D	E3	FA	09	CC	06	9E	7B	F1	2B	21	FF	40	11

<< >>

State Array Matrix

round: 0 Add round key 00 / 40

53	7A	A1	6D
65	61	76	65
6D	64	C3	20
20	C3	A1	74

<< >>

PT CT

Add round key operation

Bytes of StateArray Matrix are XORed with corresponding bytes of expanded key words. The exact word is choosen according to the byte location inside the StateArray and the current number of the round. Iverse version is the same only the words are taken with the opposite order.

$$\begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix}$$

$\oplus$ *is indicating bitwise XOR*
w *is expanded key word*

Key

0	4B	51	36	5E	62	4F	4D	5E	4B	49	5C	52	54	41	43	49
1	C9	4B	0D	7E	AB	04	40	20	E0	4D	1C	72	B4	0C	5F	3B
2	35	84	EF	F3	9E	80	AF	D3	7E	CD	B3	A1	CA	C1	EC	9A
3	49	4A	57	87	D7	CA	F8	54	A9	07	4B	F5	63	C6	A7	6F
4	F5	16	FF	7C	22	DC	07	28	8B	DB	4C	DD	E8	1D	EB	B2
5	41	FF	C8	E7	63	23	CF	CF	E8	F8	83	12	00	E5	68	A0
6	B8	BA	28	84	DB	99	E7	4B	33	61	64	59	33	84	0C	F9
7	A7	44	B1	47	7C	DD	56	0C	4F	BC	32	55	7C	38	3E	AC
8	20	F6	20	57	5C	2B	76	5B	13	97	44	0E	6F	AF	7A	A2
9	42	2C	1A	FF	1E	07	6C	A4	0D	90	28	AA	62	3F	52	08
10	01	2C	2A	55	1F	2B	46	F1	12	BB	6E	5B	70	84	3C	53

SBox

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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
10	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
11	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
12	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
13	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
14	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
15	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Příloha B – *UML diagram hlavního okna aplikace*