ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra radioelektroniky

Modernizace systému pro generování individuálních šifrových textů

Bakalářská práce

Studijní program: Komunikace, multimédia a elektronika
Studijní obor: Multimediální technika

Vedoucí práce: Ing. Tomáš Vaněk, Ph. D.

Miroslav Smutný

Praha 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra radioelektroniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Miroslav Smutný**

Studijní program: Komunikace, multimédia a elektronika
Obor: Multimediální technika

Název tématu: **Modernizace systému pro generování individuálních šifrových textů**

Pokyny pro vypracování:

Seznamte se s existujícím nástrojem pro generování jednoduchých šifrových textů používaným pro vytváření individuálních zadání semestrálních prací.
Stávající generátor čte otevřený text z externího souboru, zašifruje jej vybranou historickou šifrou, a vygenerované zadání odešle studentům.
Cílem bakalářské práce je oprava existujících chyb (detaily sdělí vedoucí práce), celkový redesign existujícího programu a rozšíření jeho funkcionalit.
Podle pokynů vedoucího projektu v jazyce Java implementujte do stávajícího programu následující vlastnosti:
- export zadání do PDF
- odesílání zadání ve formě emailu s PDF přílohou
- implementace asymetrických kryptosytémů (např. RSA) s omezenou velikostí modulů
- implementace vybraných moderních symetrických algoritmů (např. baby Rijndael)
- u stávajících algoritmů (substituční a transpoziční šifry) přidejte možnost jejich ruční parametrizace

Seznam odborné literatury:

[1] Clifford, B.: A Description of Baby Rijndael, 2005. Dostupné na: http://orion.math.iastate.edu/cbergman/crypto/homework/babyr/babyr.pdf [on-line]
[2] RSA Education Cryptosystem, Google Archive Code, 2010. Dostupné na: https://code.google.com/archive/p/rsa-cryptosystem-education [on-line]

Vedoucí: Ing. Tomáš Vaněk, Ph.D.

Platnost zadání: do konce letního semestru 2016/2017

L.S.

doc. Mgr. Petr Páta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 2. 2016

Anotace

Tato bakalářská práce se věnuje popisu a implementaci nových funkcí a opravě chyb v aplikaci Cypher, která se využívá na Katedře telekomunikační techniky FEL ČVUT ke generování a odesílání individuálních šifrových textů studentům. V rámci řešení byly do aplikace Cypher přidány nové šifry (RSA a Baby Rijndael), dále byla implementována možnost nastavení parametrů jednotlivých šifer v GUI a v neposlední řadě také export výstupu do pdf dokumentu a jeho odesílání jako přílohy. Stejně jako stávající aplikace, i nové funkce byly implementovány v programovacím jazyce JAVA.

Klíčová slova

programování, šifry, JAVA, RSA, AES, Rijndael, Baby Rijndael, export do PDF

Summary

This bachelor thesis aims to describe and implement new functions and bug fixes in the Cypher application, which is used at the Department of telecommunication engineering FEE CTU to generate and send individual ciphertexts to students. New functions include new ciphers (RSA and Baby Rijndael), implementation of the possibility to modify parameters of each cipher algorithm in GUI and also exporting the output to pdf document and sending it as an e-mail attachment. As well as the already existing application, the new functions were implemented in the JAVA programming language.

Index terms

programming, ciphers, JAVA, RSA, AES, Rijndael, Baby Rijndael, PDF export

Poděkování

Chtěl bych poděkovat vedoucímu, panu Ing. Tomáši Vaňkovi Ph.D. za cenné rady a připomínky ohledně této bakalářské práce. Zároveň bych chtěl poděkovat rodičům za podporu a schovívavost při ponocování během programování aplikace Cypher.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze 26.5. 2016 ….....

Miroslav Smutný

Obsah

Seznam příloh

- Miroslav Smutný: Vylepšení systému pro generování šifrových textů Zpráva k Individuálnímu projektu
- CD obsahující
  - text této bakalářské práce (smutny_miroslav.pdf)
  - text zprávy k individuálnímu projektu (Zprava k projektu.pdf)
  - příklad výstupu aplikace ve formátu pdf
    - soubor pro studenty (Task.pdf)
    - soubor pro učitele (300-Miroslav-Smutny-101.pdf)
  - původní verzi aplikace Cypher (Cypher 1.6.zip)
  - finální verzi aplikace Cypher (Cypher 1.9.zip)

1 Úvod

V předkládané bakalářské práci se věnuji úpravám a implementaci nových funkcí v aplikaci Cypher. Programování probíhalo v jazyce JAVA, v kterém je celá aplikace napsaná. K programování bylo využito vývojové prostředí (IDE) NetBeans (ve verzi 8.0.2, stránky programu: [1]) a JAVA Standard Edition SDK 8u66 od společnosti Oracle.

Text má celkem deset kapitol. První z nich právě čtete. Stručný popis aplikace Cypher naleznete v následující kapitole. Ve třetí kapitole se nachází dokumentace k užitečným třídám jazyka JAVA, které jsem při řešení použil. Mimo jiné se tam věnuji problematice layout managerů.

Bakalářská práce se ale hlavně zabývá implementaci dvou moderních šifer, RSA a baby Rijndael. Teoretické informace o moderních šifrách a především o těchto dvou naleznete ve čtvrté kapitole. V kapitole následující naleznete popis jejich implementace.

Vedle těchto šifer jsem se v bakalářské práci zaobíral také přidáním možnosti nastavení parametrů jednotlivých šifer v GUI (šestá kapitola) a exportem do pdf (sedmá kapitola).

Dalším úkolem bylo upravit aplikaci dle šesté kapitoly zprávy k projektu, tedy především sjednotit složené šifry (viz [2], strana 35). To naleznete v osmé kapitole. V deváté kapitole je soupis několika dalších menších úprav, jako například úprava třídy DesEncryptor, aby nebyla závislá na interní proprietární třídě od Sun microsystems ([2], strana 36).

V desáté a poslední kapitole můžete nalézt shrnutí mojí práce. Původní (1.6) a finální (1.9) verzi aplikace Cypher naleznete na přiloženém CD.

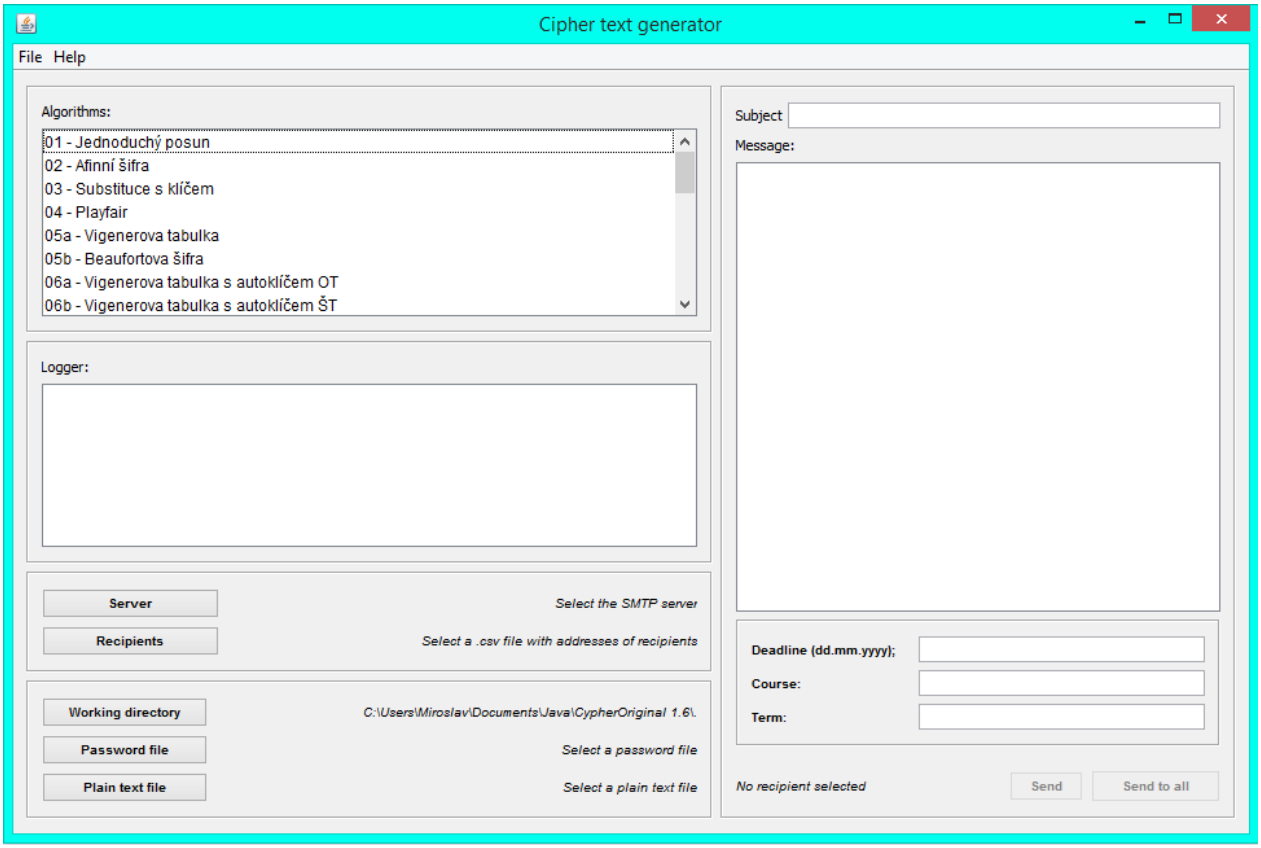
2 Popis aplikace Cypher

Aplikace Cypher slouží ke generování a odesílání šifrových textů studentům. Původními autory jsou Zdeněk Kouba, Petra Marešová a Jan Kohout. Aplikaci jsem převzal ve verzi 1.6, kterou zde ve stručnosti popíši. Více informací najdete ve zprávě k individuálnímu projektu [2], kterou najdete v příloze.

Zdrojový kód aplikace Cypher je rozdělen do čtyřech balíků začínajících globálně unikátním názvem cz.cvut.fel.*. Stejně jako ve zprávě k projektu, i zde budu tuto část názvu balíku vynechávat a balíky nazývat pouze jejich konečným názvem, tedy encoding, conf, cypher a cyphergui. Další dva balíky (konkrétně nazvané cz.cvut.fel.cyphergui.resources a cz.cvut.fel.cyphergui.resources.busyicons) tvoří soubory využívané Swing Application Frameworkem pro tvorbu GUI (více o knihovně viz [2], strana 22).

2.1 Balík cyphergui

Balík obsahující GUI. Všechna okna aplikace byla vytvořena grafickým (WYSIWYG) editorem. Balík mimo jiné obsahuje třídu CypherGUIApp, kterou se spouští celá aplikace. Vykreslení okna zajišťuje třída CypherGUIView. Obě tyto třídy využívají knihovnu Swing Application Framework.



Obrázek 1:Hlavní okno aplikace (převzato z [2])

Okno je rozděleno do dvou částí. Vlevo se nachází seznam dostupných algoritmů, logovací panel a sada tlačítek k nastavení aplikace. Vpravo najdeme několik textových polí, kam uživatel píše text zadání (e-mail), deadline úkolu, jméno předmětu a aktuální semestr. Vpravo dole jsou pak další dvě tlačítka. Jedno pro odeslání zadání aktuálnímu adresátovi, druhé pro odeslání stejného textu e-mailu a stejné sady užitých algoritmů všem adresátům v seznamu.

Vedle tohoto okna obsahuje balík třídy vytvářející několik dalších dialogových oken (například třídu PropertiesDialog zajišťující nastavení SMTP serveru).

2.2 Balík conf

Tento balík obsahuje třídy zabývající se zachováním nastavení aplikace mezi jednotlivými běhy. Nejdůležitější je třída CypherProperties, pomocí které získává aplikace data ze souboru cypher.properties. Další třída SMTPProperties spravuje informace k e-mailovým účtům, ze kterého jsou odesílány e-maily. Aby se heslo k e-mailovému účtu nenacházelo v souboru snadno čitelné, je zašifrováno šifrou DES (Data Encryption Standard). K tomu slouží třída DesEncryptor. Kód této třídy nicméně obsahuje volání interní proprietární API od Sun microsystems (konkrétně sun.misc.BASE64Decoder a sun.misc.BASE64Encoder). Tato knihovna však kvůli tomu v některé z následujících verzí Javy už nemusí být. Proto je nutné jejich funkci nahradit například třídou java.util.Base64 [3].

2.3 Balík encoding

V tomto balíku nalezneme jednotlivé šifrovací algoritmy. Všechny z nich implementují rozhraní EncodeAlgorithm. Ve verzi 1.6 to tedy znamená nutnost implementovat následující metody:

```
public String[] getName();
public String encode(String plainText,
                     String password);
public int getMinTextLength();
public int getMaxTextLength();
public int getMinPasswordLength();
public int getMaxPasswordLength();
```

Každý algoritmus je navíc singleton, což znamená, že při běhu aplikace může existovat maximálně jedna instance této třídy, což je zajištěno private konstruktorem a metodou getInstance(). To si můžeme ukázat třeba na příkladu Afinní šifry:

```
private static EncodeAlgorithm instance = null;

private AfinniSifra() {}

public static EncodeAlgorithm getInstance() {
  if (instance == null) {
    instance = new AfinniSifra();
  }
  return instance;
}
```

V tabulce na následující stránce (převzato z [2]) naleznete seznam parametrů jednoduchých algoritmů (vynechán algoritmus LowerCaseAlgorithm, jelikož ten pouze převede velká písmena na malá, tudíž není považován za šifru v pravém slova smyslu). Nuly v polích tabulky pro minimální a maximální délku hesla znamenají, že daný algoritmus nepřijímá heslo.

Vedle dvanácti jednoduchých algoritmů (plus třináctého LowerCaseAlgorithmu) se v balíku encoding nachází také několik složených šifer. Nicméně v aplikaci jsou obsaženy dva principy skládání šifer vytvořené nezávisle na sobě (od pana Kouby a od pana Kohouta), složené šifry od pana Kohouta navíc nefungují správně, tudíž je nutné princip skládání šifer sjednotit a zjednodušit (viz kapitola 8).

Encoding dále mimo jiné obsahuje statickou (static) třídu Utility, která obsahuje užitečné funkce využívané mnoha šiframi (jako například generátor náhodných čísel nebo funkci k dělení modulo).

Název	MinTextLength	MaxTextLength	MinPassLength	MaxPassLength
01 – Jednoduchý posun	60	120	0	0
02 – Afinní šifra	80	160	0	0
03 – Substituce s klíčem	150	300	5	15
04 – Playfair	100	200	5	8
05a – Vigenerova tabulka	100	350	5	10
05b – Beaufortova šifra	100	350	5	10
06a – Vigenerova tabulka s autoklíčem OT	200	400	4	10
06b – Vigenerova tabulka s autoklíčem ŠT	200	400	4	10
07 – Transpozice, pevná perioda	40	50	0	0
08 – Sloupcová transpozice	60	300	0	0
09 – Sloupcová transpozice s heslem	100	150	4	10
10 – Dvojnásobná sloupcová transpozice	60	300	0	0

Tabulka 1: Seznam jednoduchých šifrovacích algoritmů včetně jejich parametrů (převzato z [2])

2.4 Balík cypher

V balíku cypher se nachází třídy, které navzájem propojují všechny ostatní balíky. Nejdůležitější je třída Model, která obsahuje všechny informace týkající se běhu aplikace, tedy například načtené řetězce z hlavního okna aplikace, soubory s otevřeným textem a hesly nebo také seznam šifrovacích algoritmů. Pak zde nalezneme třídu Controler obsahující metody, které jsou zavolány při stisku tlačítek k odeslání zadání jak jednomu studentovi, tak všem studentům. V nich je vygenerováno zadání (pomocí třídy TaskDefinition v tomto balíku) a následně odesláno (pomocí tříd MailSender a CypherSmtpTransport). Kvůli chybě ve funkci sendToOneHandler ve třídě Controler nebylo možné odeslat individuální zadání poslednímu adresátovi v adresáři. To bylo nutné opravit.

Funkce sendMail ve třídě CypherSmtpTransport, která je užívána k odesílání e-mailu byla již připravena na odesílání přílohy, nicméně, jak se později ukázalo, nefungovala tak, jak měla, jelikož příloha přicházela poškozená, proto bylo nutné tuto funkci také opravit.

3 Stručné informace k použitým třídám

V této kapitole uvedu nejdůležitější informace o méně obvyklých třídách jazyka Java, které byly použity při implementaci úkolů v páté až desáté kapitole. Součástí této kapitoly je rovněž popis layout managerů využitých při tvorbě GUI. Jedná se pouze o stručný úvod, v případě zájmu doporučuji dokumentaci API jazyka Java [4] a velmi dobře zpracované tutoriály [5].

3.1 Properties

Třída Properties [6] (vlastnosti) (java.util.Properties) se využívá pro ukládání a opětovné získávání informací, které se snadno načtou z a nebo uloží do proudu (například z/do souboru). Snadné ukládání a získávání hodnot z Properties je možné díky tomu, že tato třída dědí ze třídy HashTable. Nevýhodou je, že informace v Properties můžou být uložené pouze jako řetězec, tudíž v případě ukládání čísel je nutné je na řetězec a z řetězce převádět. Každý řetězec je do Properties ukládán jako pár společně s klíčem (také řetězec). Ve výsledném proudu, kam se vlastnosti uloží, jsou poté oba řetězce zapsány jako klíč=hodnota (při načítání je také možné klíč:hodnota).

V konstruktoru je možné jako parametr vložit jinou instanci třídy Properties, ve které jsou hledány výchozí hodnoty v případě nenalezení. (Tedy například: V properties hledáme parametr uložený pod klíčem vyska (props.getProperty("vyska")). Klíč vyska v těchto properties ale neexistuje. Proto se prohledají vlastnosti vložené jako výchozí. Pokud se zde nachází klíč vyska, je navrácen příslušný řetězec. Pokud se tento klíč nenachází ani ve výchozích hodnotách (ani ve výchozích hodnotách výchozích hodnot a rekurzivně dále), je návrácen null.)

3.2 BigInteger

Třída (java.math.BigInteger) [7], jejíž instance reprezentuje prakticky neomezeně velké neměnné celé číslo. Nicméně omezení velikosti tu je, ale záleží na implementaci JVM (Java Virtual Machine). Minimálně se do BigIntegeru musí vejít čísla z otevřeného intervalu $(-2^{\text{Integer.MAX\_MALUE}}; +2^{\text{Integer.MAX\_VALUE}})$. Tedy dekadická čísla mající více než 646 milionů číslic.

Jelikož je instance třídy BigInteger neměnná, musí se výsledek operace ukládat do nové proměnné (přioperaci může být nutná alokace paměti pro větší číslo, takže ukládání výsledku do nové proměnné je celkem logický požadavek).

BigInteger umožňuje stejné operace jako int, nicméně pro tyto operace je nutné použít speciální funkce z této třídy (nelze použít standardní operátory +, −, <, >, ...). Dále umožňuje speciální operace (vhodné pro šifru RSA) jako generátor (pravděpodobně) prvočísel (getProbablePrime), umocňování modulo (modPow) a výpočet největšího společného dělitele (gcd).

3.3 BitSet

BitSet (java.util.BitSet) [8] je, jak název napovídá, množina bitů. Interně je uložen v poli long integerů. Teoreticky je nekonečně velký (limitace je pouze velikost operační paměti). Poskytuje užitečné funkce, kterými lze nastavit jeden určitý bit do jedničky. Bity jsou (jako standardně v Javě) indexovány od nuly. Třída samozřejmě poskytuje logické operace, jako logický součet, součin, exkluzivní součet.

Čtyři statické funkce valueOf slouží k převodu polí bytů, ByteBufferu, respektive polí long integerů a LongBufferu na BitSet. Na tyto datové typy lze samozřejmě také stávající BitSet zpět převést.

Nevýhodou (alespoň pro mé účely, jinak jde samozřejmě o vlastnost jazyka) je, že BitSet je little-endian – tedy první je nejméně významný bit i byte. Proto se pořadí řádků matic nebo bytů v implementaci šifry Baby Rijndael, kde byla tato třída využita, poněkud liší, jelikož v popisu šifry Baby Rijndael jsou čísla ve formátu big-endian.

3.4 Layout Managery

GUI lze s použitím Java Swing vytvářet buď v IDE pomocí GUI Builderu myší, nebo je možné GUI vytvořit programově [9]. Při programovém vytváření lze komponenty (panely, tlačítka, popisky) rozmístit na absolutní pozice definováním souřadnic všech komponent nebo pro jejich rozmístění v rámci kontejneru (JPanel nebo contentpane) použít layout manager. S výhodou můžeme tvořit různě složitá GUI vytvářením libovolných hiearchických struktur (komponenty rozmístíme v rámci panelu, větší množství panelů pak umístíme v dalším panelu a tak dále) v nichž každý kontejner může mít různý layout manager.

Java poskytuje celkem osm layout managerů. Pokud programátorovi kombinace předpřipravených layout managerů nestačí, může si vytvořit vlastní. V následujícím textu stručně zmíním pouze ty layout managery, které jsem v rámci bakalářské práce použil. Pro více informací o layout managerech doporučuji lekci v Java tutoriálech [9], odkud jsem čerpal následující základní informace.

3.4.1 FlowLayout

Výchozí layout manager. Naskládá komponenty za sebou zleva doprava (nicméně směr záleží na nastavení Locale, například arabské země budou mít směr opačný), pokud se komponenty nevejdou na šířku kontejneru, je přidán další řádek. Komponenty zaujímají svoji základní velikost a je mezi nimi (při použití výchozího nastavení) pětipixelová mezera.

3.4.2 BoxLayout

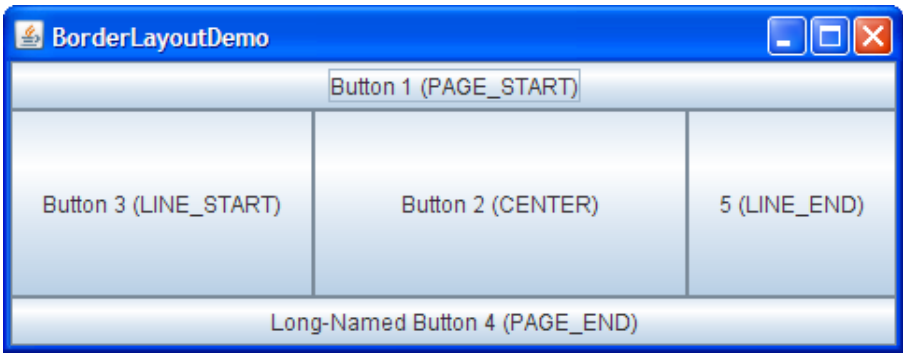
Druhý nejjednodušší layout manager po FlowLayoutu. Svou činností je mu velmi podobný, nicméně oproti FlowLayoutu má větší možnosti nastavení, například komponenty mohou za sebou následovat i ve vertikálním směru.

3.4.3 GridLayout

Komponenty rozmístí do buněk mřížky. Každá z buněk má stejnou velikost. Komponenty roztáhne, aby zabíraly celou buňku.

3.4.4 BorderLayout

Layout, který komponenty rozmístí podle světových stran (a na střed). Komponenty na hranicích mají v příslušném směru svou nejmenší velikost, zbylé místo kontejneru zabírá středová komponenta. Nejlépe se to dá vysvětlit na následujícím obrázku (převzato z [10])



Obrázek 2: Vysvětlení konstant BorderLayoutu (převzato z [10])

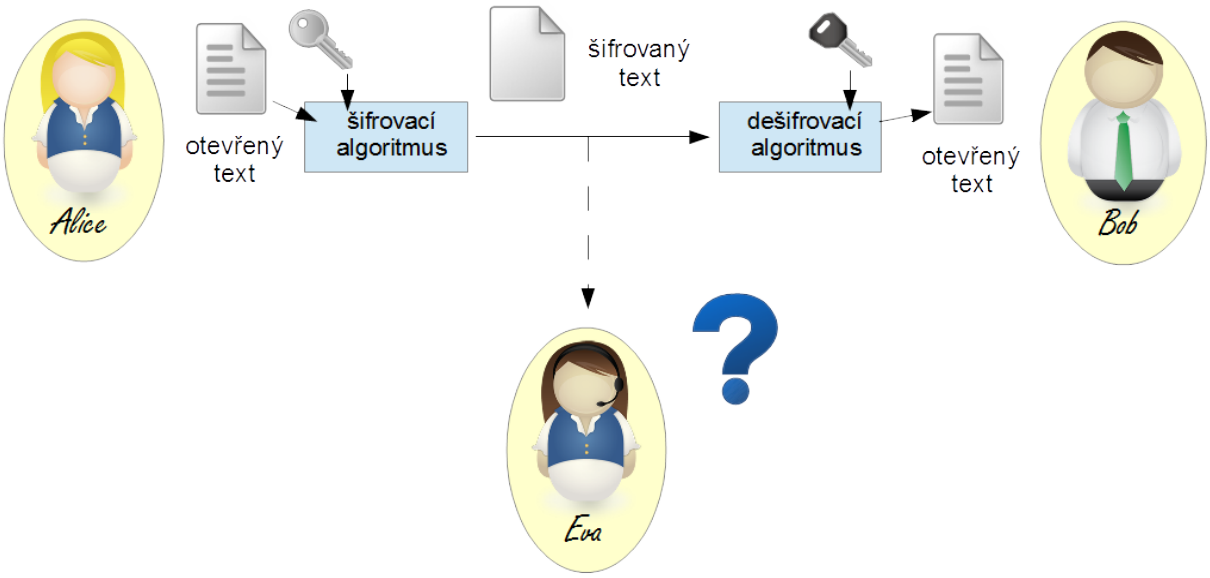
Komponenty s konstantou PAGE_START (nebo NORTH) a PAGE_END (SOUTH) zaujímají celý prostor horizontálně, ale vertikálně mají jen svou minimální velikost. Komponenty s LINE_START (WEST) a LINE_END (EAST) zaujímají celý zbylý prostor vertikálně, ale mají své minimální možné horizontální rozměry. Komponent CENTER zaujímá zbytek kontejneru.

4 Moderní šifrovací algoritmy

4.1 Model komunikace (základní pojmy kryptografie)

K utajení nebo autentizaci zprávy se používá šifrování. Šifra je algoritmus, který převede zprávu, kterou chceme odeslat, tak, aby ji bylo možné znovu dešifrovat a abychom v odposlouchávaném kanálu mohli znemožnit nebo alespoň omezit možnost vyzrazení zprávy. Oblast činnosti, která se zabývá návrhem šifer (kryptosystémů) se nazývá kryptografie. Oproti tomu snaha najít způsob, jak šifry prolomit, se nazývá kryptoanalýza. Oba tyto obory zastřešuje věda s názvem kryptologie ([11], str. 2).

Následující obrázek ukazuje model komunikace (převzato z [12]).



Obrázek 3: Model šifrované komunikace (převzato z [12])

Jako dvě komunikační strany se obvykle používají Alice (účastník A, odesílatel zprávy) a Bob (účastník B, příjemce zprávy). Kanál mezi Alicí a Bobem odposlouchává Eva (eavesdropper). Aby se Eva nedozvěděla informaci, kterou chce Alice Bobovi odeslat, dohodnou se Alice s Bobem na šifrované komunikaci a na jejích parametrech.

Šifrování proběhne tak, že Alice vezme otevřený text (plain text) a tajný šifrovací klíč (heslo, secret key), které vloží do šifrovacího algoritmu. Výsledkem je šifrovaný text (cipher text). Ten odešle komunikačním kanálem. Bob šifrovaný text přijme, s použitím dešifrovacího klíče (ten často bývá stejný jako šifrovací, ale u nesymetrických algoritmů (viz dále) jsou šifrovací a dešifrovací klíč rozdílné, proto jsou na obrázku obecně vyznačeny odlišným symbolem) přijatou zpravu dešifruje. Výsledkem je opět původní zpráva. Proto musí pro šifrovací algoritmus platit, že se jedná o jednoznačné přiřazení šifrovaného textu otevřenému textu. Pokud by výsledkem šifrování dvou a více otevřených textů byl stejný šifrovaný text, nebyl by schopný Bob zašifrovanou zprávu dešifrovat.

Pokud Eva komunikaci odposlouchává, získá pouze šifrovaný text a bez znalosti hesla a ani šifrovacího algoritmu mu nerozumí. Nicméně ví, že ke komunikaci mezi Alicí a Bobem dochází. Pokud by chtěli Alice s Bobem utajit samotný fakt, že spolu komunikují, je nutné místo kryptografie použít steganografii ([13], str. 20), tedy text zprávy ukrýt přímo všem na očích tam, kde jej nikdo nehledá (například do fotografie, kterou Alice pošle hromadným e-mailem).

4.2 Dělení šifrovacích algoritmů

Šifrovací algoritmy můžeme dělit podle doby vzniku na historické a novodobé (moderní). Dalším typem dělení je dělení podle toho, zda se pro šifrování i dešifrování používá stejné heslo nebo ne.

4.2.1 Dělení podle doby vzniku

4.2.1.1 Historické šifry

Historické šifry jsou jednoduché šifry používané v minulosti. Mezi ně řadíme šifry substituční (Caesarova – jednoduchý posun), transpoziční (sloupcová transpozice) nebo aditivní (Vigenèrova šifra). Velké množství takových šifer je implementovaných v aplikaci Cypher ([2], kapitola 3.1.1). Jejich seznam naleznete v tabulce 1 na straně 19. Pro historické šifry je typické, že obvykle pracují se zprávami jako s posloupnostmi písmen ([13], str. 47) a je možné s nimi šifrovat jen s použitím tužky a papíru. Nevýhodou historických šifer je jejich snadná prolomitelnost (i když například prolomení Enigmy nebylo zrovna snadné, i tak je považována za historickou šifru).

4.2.1.2 Novodobé šifry

V souvislosti s rozvojem výpočetní techniky bylo jasné, že historické šifry nestačí, protože i šifry relativně složité (obzvlášť pro ruční šifrování) jako je Vigenèrova, jsou s použitím počítače (a znalostí dostatečně dlouhého šifrovaného textu) snadno prolomeny (viz kapitola 6.10 o kryptoanalýze Vigenèrovy šifry v [11], str. 202). Proto bylo nutné vymyslet složitější novodobé (moderní) šifry.

Novodobé šifry jsou obvykle definovány pro otevřené texty tvořené posloupností bitů (nebo bytů) a jsou primárně určené pro šifrování počítačem. Příkladem mohou být šifry RSA, Rijndael nebo ElGamal.

4.2.2 Dělení podle symetrie klíčů

4.2.2.1 Symetrické šifry

Symetrické šifry používají jak pro šifrování, tak pro dešifrování stejný klíč. Před sedmdesátými lety dvacátého století žádné jiné kryptosystémy než symetrické neexistovaly ([13], str. 22). Takže všechny historické šifry jsou samozřejmě rovněž symetrické. U symetrických šifer existuje problém s přenosem tajného klíče (tak, aby ho Eva nezískala, tento problém řeší asymetrické kryptosystémy, viz oddíl o asymetrických šifrách).

Moderní symetrické šifry můžeme rozdělit na šifry proudové (stream ciphers) a blokové (block ciphers).

Blokové šifry šifrují najednou celý blok dat (obvykle 128, 256 nebo 512 bitů). Veškeré operace při šifrování se tak aplikují na celý blok najednou. Pokud je nutné šifrovat více bloků otevřeného textu, je nutné použít jeden z módů operace. Příkladem blokové šifry je například DES nebo AES. Více informací o blokových šifrách naleznete v oddíle 4.3

Proudové šifry oproti tomu šifrují postupně bit po bitu (resp. byte po bytu) ([11], str. 383). Jak šifrování, tak dešifrování proudových šifer obvykle spočívá pouze k provedení operace exkluzivní součet (XOR) mezi bitem otevřeného textu a bitem „klíčového proudu“ (key stream) vypočteného určitou operací z klíče. Jelikož po opětovné operaci XOR s bitem key streamu získáme opět bit otevřeného textu, je dešifrování prakticky totožné se šifrováním.

Jako příklad proudových šifer lze uvést asi nejznámější šifru RC4 [14], která se používala k přenosu internetových stránek https (SSL) nebo k zabezpečení sítě wi-fi (WPA). V současné době se již šifra RC4 v https nepoužívá, protože při současných vysokých datových tocích a tedy možného snadného zachycení dostatečně velkého množství dat je možné RC4 prolomit.

4.2.2.2 Asymetrické šifry (šifry s veřejným klíčem)

Jelikož se Alice s Bobem musí dohodnout na heslu, je nutné se dohodnout tam, kde Eva nemůže odposlouchávat, což může být problém. Pokud Eva klíč zachytí, může komunikaci dešifrovat a dozvědět se tajné informace. To vedlo v roce 1975 autory Diffieho a Hellmana [15] k vytvoření konceptu asymetrických šifer, tedy takových, kde se používají dva klíče (na každé straně jiný, proto název asymetrické). Princip asymetrických šifer spočívá v tom, že se při tvorbě klíčů používá tzv. jednosměrná funkce (one-way trapdoor function). Když si funkci představíme jako předpis $y=f(x)$, tak se jedná o jednosměrnou funkci právě tehdy, když z x můžeme snadno vypočítat y, ale z y nelze v dostatečně krátkém čase vypočítat x (tento výpočet vyžaduje velké množství operací) pokud neznáme další informaci (trapdoor, například u RSA jedno z prvočísel, ze kterých je veřejný klíč vypočítán, viz dále).

Veřejný klíč tak může být zveřejněn (pokud mi chcete něco poslat, použijte toto heslo), zašifrované zprávy pak může dešifrovat jen adresát, protože z veřejného klíče nelze snadno vypočítat soukromý klíč.

Nejznámějším zástupcem asymetrické šifry je RSA (viz oddíl 4.4.1), který je zatím jedinou implementovanou asymetrickou šifrou v aplikaci Cypher. Vedle ní existují i další asymetrické algoritmy, jako například ElGamal, který však není předmětem této práce.

4.3 Blokové šifry

4.3.1 Úvod

Blokové šifry pracují s blokem dat o pevné délce. Tato délka souvisí s prolomitelností šifry. Pokud pracuje bloková šifra s blokem o velikosti m bitů, tak k tomu, abychom mohli šifru prolomit, nám stačí zachytit 2^m bloků ([13], str. 114). Nicméně pro příliš dlouhé bloky je šifra neefektivní, protože krátké texty musí být vždy doplněny na celý násobek délky bloku. Proto jsou konkrétní implementace blokových šifer kompromisem mezi efektivitou a bezpečností.

Bloková šifra vezme blok otevřeného textu a blok klíče, provede na něm několik operací a výsledkem je blok šifrovaného textu. Pravidla pro šifrování více bloků určuje použitý režim činnosti (mode of operation) blokových šifer. Ty byly původně vyvinuty pro šifru DES, ale dají se použít pro jakoukoli blokovou šifru.

4.3.2 Režimy činnosti blokové šifry

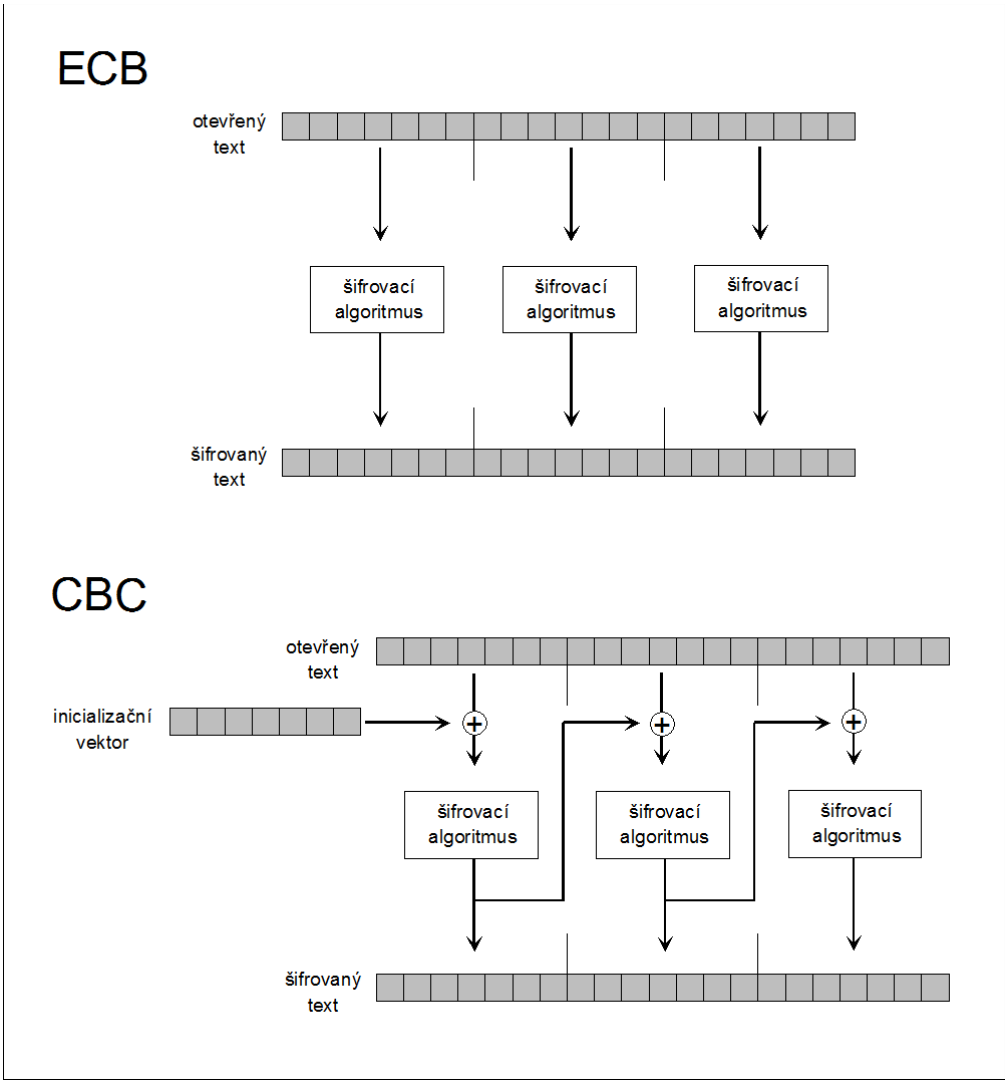
Za účelem dosažení určitých vlastností blokové šifry (obrana před copy-and-paste útoku, snadná hardwarová implementace) se používají různé režimy činnosti (módy operace) blokových šifer. Vedle dvou implementovaných režimů popsaných dále se ještě používá mnoho dalších režimů, které však nejsou předmětem této práce a proto zde nejsou popsány.

ECB (Electronic Code Book)

Nejjednodušší mód činnosti blokové šifry. Zde se otevřený text rozdělí na bloky a ty se šifrují nezávisle na bloky šifrovaného textu. Nevýhodou je, že se vlastně jedná o substituční šifru, jelikož pro stejný blok otevřeného textu dostaneme vždy stejný blok šifrovaného textu ([16], str. 12). Kódovou knihu substituční šifry tak tvoří výstupy šifrovacího algoritmu pro všechny možnosti vstupního bloku.

CBC (Cipher Block Chaining)

Režim CBC (Cipher Block Chaining) odstraňuje nedostatek módu ECB tím, že otevřený text znáhodňuje provedením operace XOR otevřeného textu s předchozím blokem šifrovaného textu (na počátku se přičítá inicializační vektor). Každý blok šifrovaného textu je tak závislý na předchozím a proto tento režim znesnadňuje použití copy-and-paste útoku, i když jej zcela neeliminuje ([16], str. 21).



Obrázek 4: Dva implementované režimy činnosti blokové šifry

4.3.3 DES

I když není tato šifra v rámci bakalářské práce implementována, šifra AES (viz dále) z ní v mnoha ohledech vychází, proto je zde stručně zmíněna.

Data Encryption Standard byla šifra založená na Feistelově šifře (více informací o ní naleznete v [13], str. 117), tedy šifrování probíhá v rundách (rounds) a v každé se používá tzv. rundovní funkce. Délky bloku otevřeného a šifrovaného textu jsou 64 bitů. Klíč je 56-bitový. Šifrování probíhá v šestnácti rundách. Šifra DES používá S-funkci (S-box function), což je substituce, která slouží k tomu, aby byla zpráva náhodně rozprostřena.

V průběhu let byl pro DES vytvořen nový režim činnosti a to trojitý DES (Triple-DES). Zde je zpráva nejprve zašifrována, poté je zase dešifrována, ale s jiným klíčem, a nakonec opět zašifrována s použitím třetího klíče. Tento mód umožnil větší bezpečnost (větší klíčový prostor) výsledné šifrované zprávy, ale neměl zvýšené požadavky na realizaci, protože bylo možné použít již hotové hardwarové bloky používané pro DES.

4.3.4 AES

Protože DES kvůli krátkému klíči náchylnému na brute-force útok již nevyhovoval, bylo v roce 1997 vyhlášeno Národním Institutem Standardů a Technologie v USA výběrové řízení na Advanced Encryption Standard (AES). Požadavkem byla velikost bloku 128 bitů, volitelná délka klíče 128, 192 a 256 bitů a minimálně stejná odolnost proti prolomení jako trojitý DES ([17], str. 222), ale menší nároky na systémové prostředky. Zvítězila šifra Rijndael pojmenovaná po svých tvůrcích, Vincentu Rijmenovi a Joanu Daemenovi, kterou si nyní popíšeme.

Stav Rijndaelu je popsán čtvercovou maticí o 128 bitech (velikost matice je tedy 4×4 byty). Do této matice se vloží otevřený text (po sloupcích zleva doprava). Poté z klíče vypočítá tzv. rundové klíče (round keys), resp. sadu matic rundových klíčů (každá matice stejně velká jako blok). První z nich přičte (operací exkluzivního součtu) k otevřenému textu. Poté Rijndael šifruje v deseti rundách (pro 128-bitový klíč). V každé rundě se stavem provádí několik operací:

- ByteSub – substituce bytů podle S-funkce (S-box)
- ShiftRow – byty ve stavové matici jsou v k-tém řádku posunuty o k pozic doleva (číslováno od nuly) Přepadávající byty se vrací zpět vpravo
- MixColumn – násobení konstantní maticí zleva (v poslední rundě se vynechává)
- AddRoundKey – přičtení (XOR) matice rundových klíčů

Nakonec je ze stavu opět po sloupcích zleva doprava vytvořen výstupní blok.

4.3.5 Baby Rijndael

Baby Rijndael je zmenšená varianta šifry Rijndael, jejíž autorem je Clifford Bergman [18]. Délka bloku a klíče je jen 16 bitů. Protože délka klíče šifry Baby Rijndael je menší, probíhá šifrování pouze ve čtyřech rundách. V každé rundě probíhají podobné operace jako u „dospělého“ Rijndaelu, jen s použitím menších struktur. Ty si nyní představíme.

Stav šifry je popsán maticí čtyř nibblů (nibble jsou čtyři bity). Ty jsou do stavové matice vloženy po sloupcích zleva doprava. Substituce tedy probíhá na úrovni nibblů (v tabulce jako hexadecimální čísla), ne bytů jako u Rijndaelu, tudíž je S-box podstatně menší (tabulka 2).

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
s(x)	a	4	3	b	8	e	2	c	5	7	6	f	0	1	9	d

Tabulka 2: S-box šifry Baby Rijndael

$$t = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Obrázek 5: Matice t v šifře Baby Rijndael

Další operace je ShiftRow. Jelikož je stavová matice velká jen 2×2 nibbly, dojde vlastně jen k výměně prvků spodního řádku. Operace MixColumn, tedy násobení konstantní bitovou maticí probíhá i zde, matici vidíte na obrázku 5 vlevo. Stav je zde považován za matici 8×2 bity.

V každé rundě (a před samotným začátkem šifrování) jsou i zde přičítány (mod 2) rundové klíče. Rundové klíče (složené ze dvou sloupců w_{2i} a w_{2i+1}, kde i=0..4) jsou definovány rekurzivně předpisy 1 až 4, kde k_n jsou nibbly klíče, S() je funkce, která substituuje nibbly podle S-boxu a funkce reverse() nibbly vymění a konečně y_i je vektor definovaný předpisem 5.

$$w_0 = \begin{pmatrix} k_0 \\ k_1 \end{pmatrix} \qquad w_1 = \begin{pmatrix} k_2 \\ k_3 \end{pmatrix} \quad (1), (2)$$

$$w_{2i} = w_{2i-2} \oplus S(\mathrm{reverse}(w_{2i-1})) \oplus y_i \quad w_{2i+1} = w_{2i-1} \oplus w_{2i} \quad (3), (4)$$

$$y_i = \begin{pmatrix} 2^{i-1} \\ 0 \end{pmatrix} \quad (5)$$

Jelikož jsou v [18] při převádění mezi hexadecimální a binární reprezentací data považována za big-endian, bylo při implementaci algoritmu v Javě, která je little-endian, nutné provést několik malých změn (viz oddíl 5.2).

4.4 RSA

RSA je algoritmus pojmenovaný po svých tvůrcích Ronaldu Rivestovi, Adim Shamirovi a Leonardovi Adlermanovi (přestože byla již předtím nezávisle objevena Cliffordem Cocksem, ale ten její objevení musel zachovat v tajnosti, protože pracoval v britské tajné službě ([11], str. 276)). Šifra RSA používá jako jednosměrnou funkci problém faktorizace čísla složeného ze dvou velkých prvočísel.

Při šifrování pomocí RSA nejprve Bob zvolí dvě různá prvočísla (v praxi se používají velká prvočísla, minimálně o 512 bitech ([13], str.161)), p a q. Poté vypočítá jejich násobek, ten označíme m, modul.

$$m = p \cdot q \tag{6}$$

Pak najde číslo e (exponent) tak, že vypočítá hodnotu Eulerovy funkce $f=\Phi(m)$ pro modul a zvolí exponent menší než m takový, který je s f nesoudělný (tedy jejich největší společný dělitel roven jedné). Eulerova funkce (Euler´s totient function) udává počet celočíselných dělitelů čísla m ([19], str. 4).

$$f = \Phi(m) = \Phi(pq) = (p-1)\cdot(q-1) \tag{7}$$

$$D(e, f) = 1 \tag{8}$$

Tím má Bob svůj veřejný klíč, tedy dvě čísla m a e, která zveřejní (pošle Alici). Dále ještě musí nalézt svůj soukromý klíč d menší než f vyhovující vztahu 9.

$$d = e^{-1} \bmod f \tag{9}$$

Ten je možné celkem snadno vypočítat s využitím Euklidova algoritmu (více informací o něm naleznete v [11], str. 281). Poté již je vše připravené na šifrování a dešifrování.

Šifra RSA pracuje s čísly. Tudíž je nutné nějak převést řetězec písmen otevřeného textu na posloupnost čísel. Problém je, že maximální číslo, které lze pomocí šifry RSA zašifrovat je rovné modulu m. Číslo m je v praxi vždy větší než 256 (velikost znaku – 8 bitů), tudíž lze v jedné várce zašifrovat víc písmen najednou. Jenže i když použijeme více písmen v jednom bloku, získáme tím vlastně jen režim ECB blokové šifry, což je substituce. Tudíž je nutné vytvořit složitější metodu, jak z otevřeného textu vytvářet čísla, která pak budeme šifrovat. Jedna z metod byla vydána v listopadu 2001 v časopise Crypto-World (Crypto #1.5, [19], str. 6), jejíž zobecněná varianta byla implementována (viz oddíl 5.1).

Pokud bude Alice Bobovi chtít poslat zprávu, použije Bobův veřejný klíč. Otevřený text (převedený dohodnutou metodou na číslo o) zašifruje pomocí vztahu 10 na šifrovaný text (reprezentovaný číslem s).

$$s = o^e \bmod m \tag{10}$$

Jelikož operace probíhá modulo m, vzniknou čísla menší než m. Ta by se těžko převáděla zpět na nějaká smysluplná písmena, tudíž se většinou ponechávají v číselném formátu. Alice tedy vzniklou posloupnost čísel odešle Bobovi.

I když Eva zachytí Bobův veřejný klíč, nemůže dešifrovat zprávu od Alice, protože nelze dostatečně rychle rozložit veřejný klíč na prvočinitele. Pokud ale bude nalezen rychlý algoritmus rozložení velkého čísla na prvočinitele, bude to znamenat konec bezpečnosti šifry RSA.

5 Implementace šifer RSA a Baby Rijndael

5.1 RSA

V balíku encoding byla vytvořena nová třída RSA. Ta samozřejmě implementuje rozhraní EncodeAlgorithm, tudíž musí obsahovat všechny funkce z tohoto rozhraní. Zde budu popisovat pouze funkci encode(). V případě zájmu naleznete celý zdrojový kód na přiloženém CD.

K implementaci šifry RSA byl využita třída BigInteger, která umožňuje modulo umocňování i počítání největšího společného dělitele. Volba náhodných prvočísel nicméně nebyla využita. Volba prvočísel probíhá s použitím funkce getRandomPrime() přidané do třídy Utility, která vybírá náhodný prvek z pole všech prvočísel menších než 500.

Formátování textu bylo definováno zobecněním pravidel #1.5 z [19]. Písmena otevřeného textu se převedou na své ASCII hodnoty. Protože šifrujeme pouze velká písmena anglické abecedy bez mezer, jsou všechny ASCII hodnoty dvouciferná čísla. Ta se naskládají za sebou do řetězce – tím si vytvoříme proud otevřeného textu. Pokud tento proud není tvořen celým násobkem „soust“ (nibblů), tedy počtu číslic, který šifrujeme najednou, je doplněn nulami.

Délka modulu vzniklého pronásobením dvou náhodně vybraných prvočísel (počet jeho dekadických cifer) omezuje počet cifer otevřeného textu. K tomu, aby stejný blok otevřeného textu nebyl vždy šifrován na stejný blok šifrovaného textu, je na konec bloku dat přidán jednociferný čítač. Pokud tedy délka modulu je k, tak vezmeme k−2 číslic z proudu otevřeného textu. Na konec této (k−2)-tice přidáme jednu číslici čítače. Tím tedy vznikne (k-1)-ciferný blok, který je samozřejmě menší než k-ciferný modul, a proto bude operace šifrování vratná. Kvůli tomu, že pro k-místné moduly se bere k−2 číslic otevřeného textu, je nutné, aby modul byl alespoň trojciferný, jinak nebude možné šifrovat. Proto je v prvním do-while testována velikost modulu (musí být větší než 100).

Vytvořené bloky jsou následne zašifrovány. Výsledkem jsou maximálně k-ciferná čísla, která jsou odeslána na výstup zarovnaná nulami na k-tice. Zde uvedená část kódu není celá funkce encode, je vynecháno načítání uživatelem definovaných parametrů, skládání řetězce s parametry šifry nebo volitelné přidávání veřejného klíče za výstup šifrování (vše naleznete na přiloženém CD)):

```
int p, q, e, bSize;
BigInteger m, f;
p=Utility.getRandomPrime();
//vybirej nove q, dokud neni rozdilne od p a
//jejich nasobek neni vetsi nez 100
do{
  q=Utility.getRandomPrime();
  m=BigInteger.valueOf(p*q);
}while((q==p)
     ||(m.compareTo(BigInteger.valueOf(100))<0));
f=BigInteger.valueOf((p-1)*(q-1));
//vybirej nove prvociselne e, dokud nejsou e a f
//nesoudelna (dokud se nejvetsi spolecny delitel
//nerovna 1)
do{
  e=Utility.getRandomPrime();
}while(!(f.gcd(BigInteger.valueOf(e)))
                      .equals(BigInteger.ONE));
bSize=(int)((m.bitLength()-1) * Math.log10(2)) + 1;
String encoded=core(oText, e, m, bSize);
```

Ve funkci core probíhá rozdělování vstupního řetězce (a doplnění nulami) a samotné šifrování.

```
private String core(String in, int e, BigInteger m,
                    int blockSize){
  BigInteger exponent=BigInteger.valueOf(e);
  StringBuilder output=new StringBuilder();

  //definice nibblu platna v tomto bloku:
  //nibble je sousto, tedy urcity pocet
//dekadickych cislic otevreneho textu, ktere
//ukousneme pri kazdem behu sifrovani
  int nibbleSize=blockSize-2;
  StringBuilder input=new StringBuilder();

  //prevedeni na retezec ASCII kodu
  for(int i=0; i<in.length(); i++)
    input.append((int) in.charAt(i));
  //doplneni nulami
  while((input.length()%nibbleSize)!=0)
    input.append('0');

  int k=0;
  for (int i=0; i<input.length(); i+=nibbleSize){
    BigInteger encodedCharSeq=BigInteger.ZERO;
    //prevedeni nibbleSize-tice na BigInteger
    for (int j=0; j<nibbleSize; j++)
      encodedCharSeq=encodedCharSeq.add(
                BigInteger.valueOf(
                     (int) (input.charAt(i+j)-'0'))
                          .multiply(
                               BigInteger.TEN
                               .pow((nibbleSize-j))));
    //pridani indexu na konec, aby dve stejne
sekvence nebyly kodovany na stejne cislo
    encodedCharSeq=encodedCharSeq.add(
          BigInteger.valueOf(k++%10));

    //sifrovani
    encodedCharSeq=encodedCharSeq.modPow(exponent, m);
    //zarovnani na blockSize-tice nulami
    output.append(String.format(
          "%0"+blockSize+"d ", encodedCharSeq));

  }
  return output.toString();
}
```

5.2 Baby Rijndael

Šifra Baby Rijndael byla implementována podle [18], s tím, že je možné volit ze dvou režimů činnosti této šifry. Jednak ECB, tak CBC. Režim CBC je výchozí. Rozhraní EncodeAlgorithm bylo upraveno tak, že každý algoritmus obsahuje instanci třídy Properties, kde jsou uloženy jeho parametry. Konkrétně u BabyRijndael to je mimo jiné zvolený režim činnosti (o této změně rozhraní EncodeAlgorithm více v oddíle 6.1).

Funkce encode pracuje tak, že otevřený text (pokud je tvořen standardním řetězcem) je rozdělen po dvojicích písmen (dvakrát osm bitů) a ty jsou postupně šifrovány (ve funkce core) podle zvoleného režimu činnosti. U této šifry je možné rovněž použít otevřený text ve formě hexadecimálních číslic (takový řetězec musí začínat předponou 0x), v takovém případě (po oddělení úvodní předpony 0x dochází k dělení po čtveřicích znaků). Díky tomu je možné mimo jiné otestovat i výsledek šifrování pro vstupní data poskytnutá autorem šifry [20].

Kvůli tomu, že Java je little-endian, bylo nutné provést několik změn oproti [18]. Zaprvé je obrácen sled řádků matice t, dále je opačné vkládání a vyčítání otevřeného textu a klíče do matice stavu a prvních dvou sloupců nultého rundovního klíče (tedy je vložen nejprve druhý a pak první znak) a stejně je pozměněn posun nibblů v operaci ShiftRow. Příslušné změny jsou na příslušném místě označeny komentářem ve zdrojovém kódu, který naleznete na CD.

Nyní popíši samotnou implementaci. Nejprve několik pomocných funkcí.

5.2.1 Pomocné funkce

Jelikož je možné volit vstup jak klasický textový, tak v podobě hexadecimálního řetězce, je nutné rozpoznat, zda uživatel nezadal neplatný vstup. Vstup od uživatele zajišťují třídy implementující rozhraní AlgPropsMediator (u třídy BabyRijndael konkrétně RijndaelMediator, více o mediatorech v následující kapitole). Mediátor musí rozhodnout, zda text zadaný uživatelem je platný. Pro klasický text je ošetření již hotové (převedení do anglické abecedy pomocí funkce convertStringToEnglish ze třídy Alphabet z balíku Cypher – ta vynechá veškeré nepřípustné znaky). Pro hexadecimální vstup poskytuje třída BabyRijndael mediátoru funkci, pomocí které o platnosti vstupu rozhodne. Jedná se o funkci isValidHexaString. Ta o platnosti rozhodne tak, že se jej pokusí převést z hexadecimálního čísla na byty funkcí parseHexaString. Pokud převod selže, vyhodí parseHexaString výjimku IllegalArgumentException a funkce isValidHexaString vrátí false.

```
public boolean isValidHexaString(String str){
  if(str.startsWith("0x")){
    try{
      parseHexaString(str.substring(2));
      return true;
    }catch(IllegalArgumentException iae){
      return false;
    }
  }
  else
    return false;
}
```

Funkce parseHexaString nejprve doplní řetězec nulami na celý počet nibblů a poté k převodu na pole bytů využije funkci parseHexBinary ze třídy DataTypeConvertor (z balíku javax.xml.bind.*). K použití této funkce jsem byl inspirován příspěvkem uživatele Vladislav Rastrusny na fóru stackoverflow.com[1]

1 Dostupné z: http://stackoverflow.com/a/5942951

```
private byte[] parseHexaString(String str)
                throws IllegalArgumentException{
  switch(str.length()%4){
//doplneni nulami na cely pocet ctveric
    case 0:
      break;
    case 1:
      str=str+"000";
      break;
    case 2:
      str=str+"00";
      break;
    case 3:
      str=str+"0";
      break;
  }
  return DatatypeConverter.parseHexBinary(str);
}
```

Další funkcí poskytovanou mediátoru je funkce getOpModes, která vrátí pole řetězců s názvy implementovaných režimů operace.

Ve třídě BabyRijndael nalezneme dále dvojici funkcí zajišťující práci s nibbly v rámci BitSetů. Jsou to funkce getNthNibble, která vrátí n-tý nibble z BitSetu `from` a setNthNibble, která nastaví příslušné čtyři bity BitSetu `to` na hodnotu nejnižších čtyřech bitů BitSetu `from`.

Protože výsledkem převodu BitSetu na pole bytů může být pole o nula prvcích (pokud jsou příslušné čtyři bity nastavené do nuly), je nutné vrácené pole dorovnat pomocí funkce Arrays.copyOf na délku jeden prvek (a funkce getNthNibble pak tento první prvek pole vrátí).

```
private byte getNthNibble(int n, BitSet from){ //n od 0
  return Arrays.copyOf(
          from.get(4*n, 4*n+4).toByteArray(), 1)[0];
}

private void setNthNibble(int n, BitSet from, BitSet to){
  for(int j=0; j<4; j++){
    to.set(4*n+j, from.get(j));
  }
}
```

V neposlední řadě je nutné zmínit funkci bs2String, která převádí BitSet na řetězec – čtveřici hexadecimálních znaků. Tato funkce je užívána při tvorbě výstupního řetězce ve funkci encode. Protože je Java little-endian, je nutné ve výpisu zaměnit pořadí bytů.

```
private String bs2String(BitSet bs){
  byte [] bts=Arrays.copyOf(bs.toByteArray(), 2);
  return String.format("%02x%02x", bts[1], bts[0]);
}
```

Nyní si popíšeme zdrojový kód jednotlivých operací.

5.2.2 Generování rundovních klíčů

Funkce generateRoundKeys vygeneruje pole rundovních klíčů podle předpisu v [18]. Rundovní klíče by bylo možné vracet jako jeden velký BitSet, jenže při každé rundě by je z něj bylo nutné vyjímat, což by znamenalo tvorbu nových instancí třídy BitSet. Efektivnější je proto mít rundovní klíče rovnou jako samostatné BitSety v poli a předávat je referencí.

```
private BitSet[]generateRoundKeys(BitSet keyBlock){
  BitSet[]roundKeys=new BitSet[numberOfRounds+1];
  BitSet col_0, col_1;
  BitSet tmp0, tmp1, tmp;
  BitSet y;
  byte digit0, digit1, swp0, swp1;
  roundKeys[0]=keyBlock;

  for(int i=1; i<numberOfRounds+1; i++){
    roundKeys[i]=new BitSet();
    col_1=roundKeys[i-1].get(0, blockSize/2);
    col_0=roundKeys[i-1].get(blockSize/2, blockSize);

    digit0=getNthNibble(0, col_1);
    digit1=getNthNibble(1, col_1);

    swp0=swap(digit0);
    swp1=swap(digit1);

    //prohozeni nibblu ve vektoru
    tmp1=BitSet.valueOf(new byte[]{swp0});
    tmp0=BitSet.valueOf(new byte[]{swp1});

    tmp=new BitSet();
    setNthNibble(0, tmp0, tmp);
    setNthNibble(1, tmp1, tmp);

    y=new BitSet();
    y.set(3+i); //vysledek je vektor [2^(i-1)  0]

    col_0.xor(tmp);
    col_0.xor(y);

    col_1.xor(col_0);

    for(int j=0; j<blockSize/2; j++){
      roundKeys[i].set(j, col_1.get(j));
      roundKeys[i].set(j+blockSize/2, col_0.get(j));
    }
  }
  return roundKeys;
}
```

5.2.3 Substituce

Funkce sOperation vymění nibbly podle S-boxu. Samotná výměna nibblů probíhá ve funkci swap(), kde je uložený S-box.

```
private BitSet sOperation (BitSet state){
  BitSet output=new BitSet();

  for(int i=0; i<blockSize/4; i++){
    byte tmp=getNthNibble(i, state);
    tmp=swap(tmp); //tablelookup

    BitSet tmpBit=BitSet.valueOf(new byte[]{tmp});
    setNthNibble(i, tmpBit, output);
  }
  return output;
}

private byte swap(byte digit){
  byte[]table={0xa, 4, 3, 0xb, 8, 0xe, 2, 0xc,
                          5, 7, 6, 0xf, 0, 1, 9, 0xd};

  return table[digit];
}
```

5.2.4 ShiftRows

Zde dojde k posunu sloupců. tedy vlastně k prohození nibbblů na druhém řádku. Protože je Java little-endian, bylo pořadí trochu upraveno (jedná se o analogii pořadí bytů při načítání do stavu – nejprve první byte (2, 1), pak nultý byte (0, 3)).

```
private BitSet shiftRow (BitSet state){
  BitSet output=new BitSet();

  int[] order={2, 1, 0, 3};
     //little Endian, takze prohozeno z 0, 3, 2, 1
  for(int i=0; i<order.length; i++){
    setNthNibble(order[i],
                    state.get(i*4, (i+1)*4), output);
  }
  return output;
}
```

5.2.5 MixColumn

Funkce pronásobí stav zleva konstantní maticí t. Proměnná blockSize je globální konstanta (16). Bity každého sloupce stavu jsou nejprve pronásobeny řádky matice (AND) a poté jsou všechny bity výsledku exkluzivně sečteny (XOR) do proměnné b, která je pak uložena na příslušné místo výsledného nového sloupce (part1, resp. part2). Oba sloupce jsou poté vloženy do proměnné output a vráceny. Protože kód pro zpracování obou sloupců je prakticky totožný, je kód funkce zkrácen. Celý kód naleznete na přiloženém CD.

```
private BitSet mixColumn(BitSet state){
  BitSet output=new BitSet(blockSize);
  BitSet part1=new BitSet(blockSize/2);
  BitSet part2=new BitSet(blockSize/2);

  BitSet tmp;
  for(int i=0; i<blockSize/2; i++){
    //prvni sloupec stavu
    tmp=state.get(0,blockSize/2);
    tmp.and(t[i]);
    boolean b=tmp.get(0);
    for(int j=1; j<blockSize/2; j++)
      b^=tmp.get(j);
    part1.set(i,b);

    //druhy sloupec stavu
    tmp=state.get(blockSize/2, blockSize);
    (...)
  }
  for(int i=0; i<blockSize/2; i++){
    output.set(i, part1.get(i));
    output.set(i+blockSize/2, part2.get(i));
  }
  return output;
}
```

Matice t je uložena jako globální proměnná. Je reprezentována polem BitSetů. Bylo by možné mít celou matici v jednom BitSetu, jenže to by znamenalo stejný problém, který je popsán u fukce generateRoundKeys.

Protože je Java little-endian, jsou řádky matice t v opačném pořadí oproti [18].

```
BitSet[] t={
     BitSet.valueOf(new byte[]{(byte)0b01110101}),
     BitSet.valueOf(new byte[]{(byte)0b10001110}),
     BitSet.valueOf(new byte[]{(byte)0b00011101}),
     BitSet.valueOf(new byte[]{(byte)0b00111010}),
     BitSet.valueOf(new byte[]{(byte)0b01010111}),
     BitSet.valueOf(new byte[]{(byte)0b11101000}),
     BitSet.valueOf(new byte[]{(byte)0b11010001}),
     BitSet.valueOf(new byte[]{(byte)0b10100011})
};
```

5.2.6 Funkce core

Funkce core reprezentuje šifrovací algoritmus zobrazený jako krabička na obrázku 4 na straně 26. Jako vstupní argumenty přijímá blok otevřeného textu a pole rundových klíčů. Výstupem je blok šifrovaného textu.

```
private BitSet core (BitSet inputBlock,
                     BitSet[] roundKeys){
  BitSet output=inputBlock; //vstup bude zmenen

  output.xor(roundKeys[0]);
  for(int i=1; i<numberOfRounds; i++){
    output=mixColumn(shiftRow(sOperation(output)));
    output.xor(roundKeys[i]);
  }
  output=shiftRow(sOperation(output));
  output.xor(roundKeys[numberOfRounds]);
  return output;
}
```

5.2.7 Funkce encode

Poslední funkcí z implementované třídy BabyRijndael, kterou si zde představíme, je funkce encode. Ta si z interní instance třídy Properties načte případné uživatelem definované parametry nebo zpracuje otevřený text a klíč předaný normální cestou jako parametry (o parametrech uložených v instanci třídy Properties viz následující kapitola). Otevřený text a klíč převede na byty podle toho, zda se jedná o standardní nebo hexadecimální řetězec. Z bytů klíče poté vygeneruje pole rundovních klíčů (byty vkládá v opačném pořadí, jelikož je Java little-endian oproti popisu šifry ve specifikaci [18]).

```
public String[] encode(String oText, String pass){
  BitSet in, out, init;
  BitSet[]roundKeys;
  String initVec=null;
  byte[] oBytes;
  byte[] pBytes;
  byte[] initBytes;
  int opType;
  StringBuilder params, sb=new StringBuilder();
(...) //nacteni uzivatelskych dat

  if(oText.startsWith("0x"))
    oBytes=parseHexaString(oText.substring(2));
  else{
    if((oText.length()%2)==1)
      oText=oText+" ";
//na konec se prida mezera pokud je lichy pocet
//pismen otevreneho textu
    oBytes=oText.getBytes();
  }
```

```
  if(pass.startsWith("0x"))
    pBytes=parseHexaString(pass.substring(2));
  else
    pBytes=pass.getBytes();
                        //BitSet je little endian
  roundKeys=generateRoundKeys(
          BitSet.valueOf(
               new byte[]{pBytes[1], pBytes[0]}));
(...)//tvorba StringBuilderu params s parametry
sifry
```

Poté probíhá samotné šifrování podle zvoleného módu. Mód je také načten z instance třídy Properties. Pro mód CBC je nutný též inicializační vektor. Ten je buď získán od uživatele (načten z Properties) nebo vygenerován náhodně pomocí funkce getRandomInteger ze třídy Utility.

```
  switch(opType){
    case 0: //ECB
      for(int i=0; i<oBytes.length/2; i++){
        in=BitSet.valueOf(new byte[]{oBytes[2*i+1],
oBytes[2*i]});
        out=core(in, roundKeys);
        sb.append(bs2String(out));
      }
      break;
    case 1: //CBC
      if(//inicializacni vektor definovan od uzivatele){
(...) //nacteni inic. vektoru od uzivatele a prevod
//na byty - totozne s dekodovanim klice - viz vyse
      }
      else
        init=BitSet.valueOf(
                new long[]{Utility
                     .getRandomInteger(0, 0xffff)});

      for(int i=0; i<oBytes.length/2; i++){
        in=BitSet.valueOf(
            new byte[]{oBytes[2*i+1], oBytes[2*i]});
        in.xor(init);
        out=core(in, roundKeys);
        init=out;
        sb.append(bs2String(out));
      }
      break;
  }
  return new String[]{sb.toString(),
                      params.toString(), oText};
}
```

6 Parametrizace algoritmů v GUI

Dalším úkolem řešeným v rámci této bakalářské práce bylo přidání možnosti nastavení algoritmů v GUI. V této kapitole popíši své řešení.

6.1 Změna rozhraní EncodeAlgorithm

Rozhraní EncodeAlgorithm bylo doplněno o trojicí funkcí,

```
public Properties getProperties();
public void setProperties (Properties props);
public void resetProperties();
```

které jsou ve všech algoritmech (kromě složených) implementovány následujícím způsobem. AlgPropsFramework je statická třída, o které se dozvíte více v oddíle 6.4. Funkce storeProps uloží instanci třídy Properties do souboru. Properties defaults, které jsou použity ve funkci resetProperties jsou výchozí hodnoty algoritmu.

```
@Override
public Properties getProperties(){
  return props;
}

@Override
public void setProperties (Properties props){
  this.props=props;
  AlgPropsFramework.storeProps(props, getName());
}

@Override
public void resetProperties(){
  props=new Properties(defaults);
  AlgPropsFramework.storeProps(props, getName());
}
```

Jelikož složené algoritmy se řídí parametry jednotlivých algoritmů, jsou ve třídě Slozena tyto funkce implementované následujícím způsobem:

```
@Override
public Properties getProperties(){
  return null;
}

@Override
public void setProperties (Properties props){}

@Override
public void resetProperties(){}
```

6.2 Úprava třídy CypherGuiView

Zde bylo pomocí GuiBuilderu přidáno menu Edit a v něm položka (jMenuItem) AlgorithmSettings. Při vybrání této položky se zobrazí okno AlgorithmPropertiesDialog (viz oddíl 6.3) s možností upravovat všechny algoritmy. Po kliknutí na tuto položku se tedy provede následující funkce:

```
private void algPropsMenuItemActionPerformed
                    (java.awt.event.ActionEvent evt) {
  algPropsDialog=new AlgorithmPropertiesDialog();
  algPropsDialog.setVisible(true);
}
```

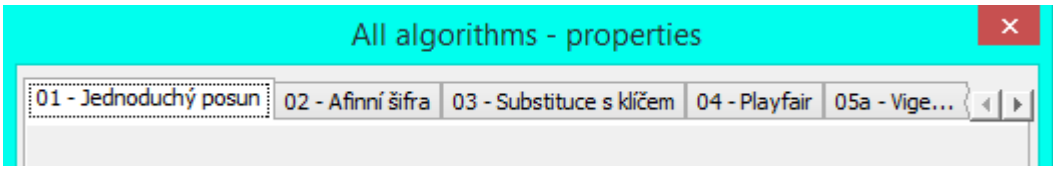
Zároveň je možné upravovat vlastnosti každého algoritmu samostatně, a to po kliknutí pravým tlačítkem myši na název příslušného algoritmu v seznamu algoritmů. Dojde opět k zobrazení AlgorithmPropertiesDialogu, tentokrát je jako parametr konstruktoru dialogu použit EncodeAlgorithm, na který bylo pravým tlačítkem kliknuto.

```
private void jListAlgorithmsMouseClicked(
                java.awt.event.MouseEvent evt) {
  if(SwingUtilities.isRightMouseButton(evt)){
    int selIndex=jListAlgorithms
          .locationToIndex(jListAlgorithms
                .getMousePosition());
    String algName=(String)algorithmListModel
          .elementAt(selIndex);
    EncodeAlgorithm rightClicked
          = model.getAlgorithmByName(algName);
    new AlgorithmPropertiesDialog(rightClicked)
          .setVisible(true);
  }
}
```

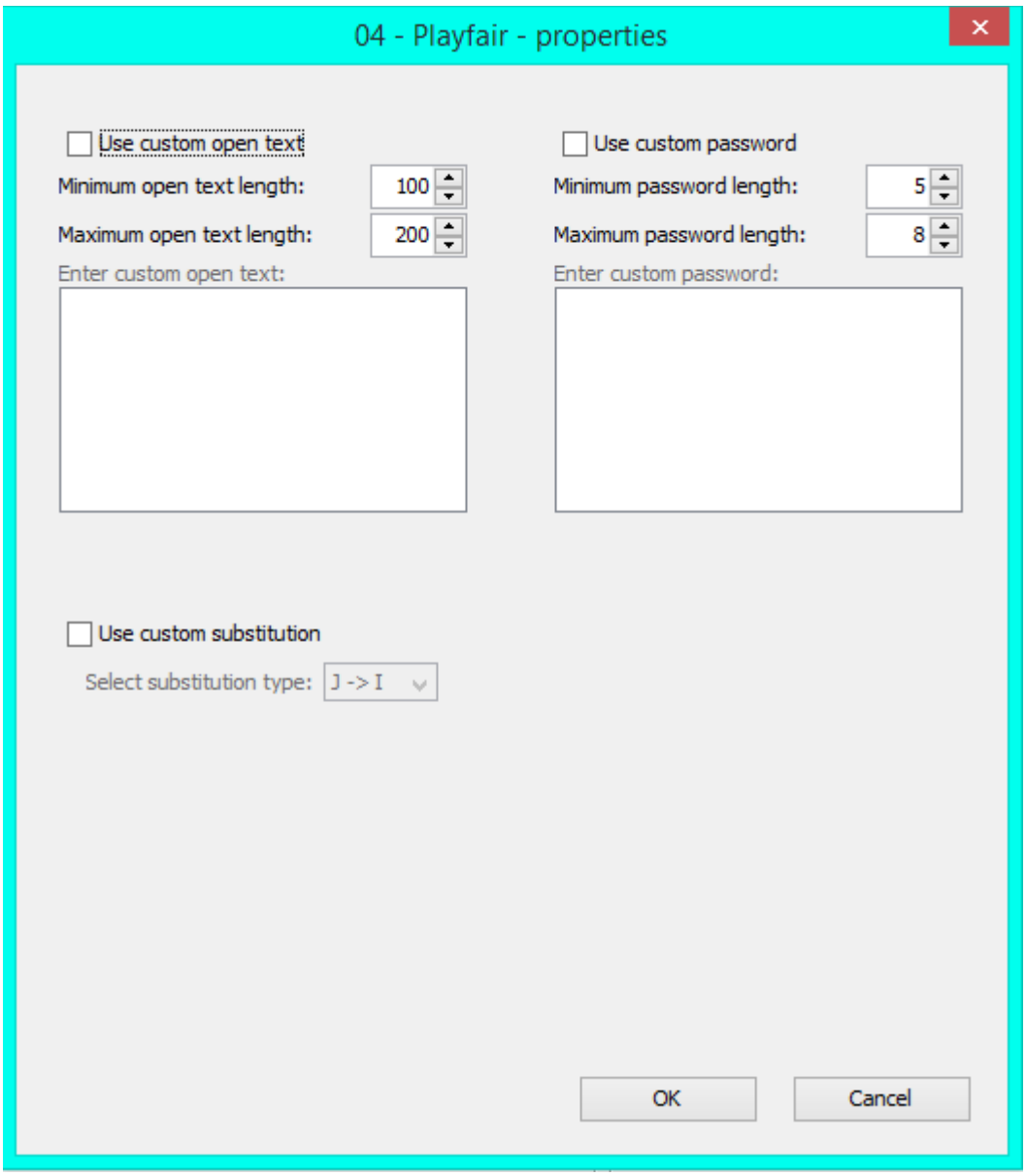
6.3 Třída AlgorithmPropertiesDialog

Třída AlgorithmPropertiesDialog je dialogové okno, ve kterém je možné upravit parametry šifrovacích algoritmů. Ve svém úvodu obsahuje zdrojový kód několik konstant, které definují rozměry dialogového okna. Dialogové okno je tvořeno ze dvou panelů – upperPanel (horní panel) a lowerPanel (dolní panel).

Horní panel je složen z 2×2 subpanelů s nastavením algoritmů. Každý subpanel má rozměry 250×250 (takže horní panel je velký 500×500 bodů). Pokud je zvoleno zobrazení všech algoritmů, jsou panely zobrazeny v záložkách (tabech) a výška horního panelu je zvětšena o 25 bodů, aby se tam záložky vešly. Spodní panel obsahuje dvě tlačítka: OK a Cancel. Jejich rozměry jsou 90×20 bodů. Jsou umístěny na střed pravé poloviny spodního panelu (pomocí GridLayout a BorderFactory). Rozměry celého spodního panelu jsou 500×50.



Obrázek 6: Výřez AlgorithmPropertiesDialogu se záložkami



Obrázek 7: AlgorithmPropertiesDialog s možnostmi nastavení šifry Playfair

AlgorithmPropertiesDialog obsahuje dva konstruktory. První z nich má jako argument vybraný EncodeAlgorithm. Poté se jedná o dialogové okno jen pro tento jeden algoritmus (jeho jméno je zobrazeno v titulku okna – viz obrázek 7 pro šifru Playfair). Dále obsahuje konstruktor nepřijímající žádné parametry. Zde je získán seznam všech algoritmů z instance třídy Model a jejich mediátory jsou vloženy do pole mediátorů (o třídě AlgPropsMediator v oddíle 6.4). Horní panel je získán zavoláním funkce getSettingsPanel z mediátoru příslušného vybraného algoritmu případně vzniká vložením získaných panelů všech algoritmů do JTabbedPane. Jako layout JTabbedPane je zvolen SCROLL_TAB_LAYOUT, aby nebyly záložky všech víc než dvaceti algoritmů v několika řadách, ale aby se jimi dalo pohodlně rolovat (viz obrázek 6).

Obsluha tlačítek probíhá funkcí actionPerformed. Každé tlačítko má přidělen „akční příkaz“, což je textový řetězec, jakýsi „podpis“, který slouží k identifikaci původce akce. Pokud uživatel stiskl tlačítko OK, je zavolána funkce propertiesEditFinished z mediátoru vybraného algoritmu (nebo všech mediátorů, pokud jsou panely v tabech). Mediátor získá data a pokud jsou v pořádku (funkce propertiesEditFinished vrátila true), je okno zavřeno. Pro zobrazení v tabech samozřejmě musí vrátit true všechny mediátory zároveň. Pokud nejsou data v pořádku alespoň v jednom tabu, zůstane dialog zobrazený. Tlačítko Cancel slouží pouze k uzavření dialogového okna bez ukládání nastavených hodnot.

```
@Override
public void actionPerformed(ActionEvent ae){
  switch(ae.getActionCommand()){
    case "OK":
      if(isSingleAlg){
        if(mediator.propertiesEditFinished())
          this.dispose();
      }
      else{
        boolean noProblem=true;
        for (AlgPropsMediator med : mediators) {
          noProblem &= med.propertiesEditFinished();
        }
        if(noProblem)
          this.dispose();
      }
      break;
    case "CN":
      this.dispose();
      break;
    default:
      break;
  }
}
```

To bylo stručné představení dialogového okna AlgPropsDialog, nyní se budeme věnovat obsahu balíku algprops.

6.4 AlgPropsFramework

AlgPropsFramework je stěžejní třída balíku algprops. Obsahuje seznam názvů parametrů algoritmů, které jsou ukládány v instancích třídy Properties jednotlivých algoritmů. Dále obsahuje funkce pro tvorbu UI primitiv, jako třeba panel se standardním textovým polem, panel se spinnerem nebo panel s checkboxem. Tyto stavební bloky jsou využívány k tvorbě subpanelů umístěných v okně AlgorithmPropertiesDialog. Ke skládání subpanelů do jednoho panelu jsou rovněž využívány funkce této třídy. Tato třída dále algoritmům poskytuje funkce k načítání a ukládání Properties do souboru.

Nyní následuje seznam parametrů šifer. Na této stránce naleznete standardní parametry, které se vyskytují u většiny šifer, na následující straně jsou speciální parametry využívané jen u jedné šifry. Název šifry, která je využívá, je uveden v komentáři vedle parametru. Z parametrů je pro úsporu místa vynecháno označení public static final a ponechán pouze datový typ String.

```
String MIN_OPEN_TEXT_LENGTH="min.otext.length";
String MAX_OPEN_TEXT_LENGTH="max.otext.length";
String MIN_PASS_LENGTH="min.pass.length";
String MAX_PASS_LENGTH="max.pass.length";
String USE_CUSTOM_OPEN_TEXT="use.custom.otext";
String USE_CUSTOM_PASS="use.custom.pass";
String CUSTOM_OPEN_TEXT="custom.otext";
String CUSTOM_PASS="custom.pass";
```

```
String USE_CUSTOM_PARAM="use.custom.param";
String CUSTOM_SHIFT="custom.shift"; //SimpleShift
String CUSTOM_A="custom.a"; //AfinniSifra
String CUSTOM_B="custom.b"; //AfinniSifra
String CUSTOM_SUBS="custom.subs.type";
                    //PlayfairCipher
String CUSTOM_COL_ORDER="custom.column.order";
                    //TranspositionFixedPeriod
String CUSTOM_P="custom.p"; //RSA
String CUSTOM_Q="custom.q"; //RSA
String CUSTOM_E="custom.e"; //RSA
String REVEAL_PUBLIC_KEY="reveal.publicKey"; //RSA
String OPERATION_MODE="operation.mode";
                    //BabyRijndael
String NUMBER_OF_ROUNDS="number.of.rounds";
                    //BabyRijndael
String USE_CUSTOM_KEY="use.custom.key";
                    //BabyRijndael
String CUSTOM_KEY="custom.key"; //BabyRijndael
String USE_CUSTOM_INIT_VEC="use.custom.init.vec";
                    //BabyRijndael
String CUSTOM_INIT_VEC="custom.init.vec";
                    //BabyRijndael
```

Ve zdrojovém kódu třídy AlgPropsFramework následují konstanty definující velikosti prvků a výchozí podkladové barvy. Ty jsou využity při přebarvení objektů, jako třeba textových polí, zpět z červené barvy, na kterou byly zbarveny, pokud uživatel zadal chybný vstup.

Nyní následuje výčet funkcí, které AlgPropsFramework obsahuje.

6.4.1 Funkce makePropsPanel

Šestice funkcí se stejným názvem vytváří panel, který se zobrazí v horní části AlgorithmPropertiesDialogu. První dvě pouze vytváří prázdný panel s textovou zprávou. K tomu, aby byla zpráva automaticky řádkována pokud se svou šířkou nevejde, je použito značek html. Konstanta PROPS_PANEL_SIZE obsahuje velikost horního panelu okna AlgorithmPropertiesDialog a COMPONENT_WIDTH je šířka subpanelu zmenšená o okraje.

```
public static JPanel makePropsPanel (){
  return makePropsPanel(
     "This algorithm does not offer any settings.");
}

public static JPanel makePropsPanel (String message){
  JPanel outPane=new JPanel();
  JLabel lbl=new JLabel("<html> <body style='width: "
     + COMPONENT_WIDTH + "px'>"+message+" </html>");
  outPane.add(lbl);
  outPane.setPreferredSize(PROPS_PANEL_SIZE);
  return outPane;
}
```

Další funkce skládají dohromady *n* subpanelů (kde *n* je přirozené číslo od jedné do čtyř) k vytvoření panelu. Tyto čtyři funkce závisí na zvoleném počtu subpanelů ve třídě AlgorithmPropertiesDialog (v současné implementaci 2×2). Při změně je nutné upravit tyto funkce na správný počet subpanelů. Název použité funkce makeEmptyPanel mluví sám za sebe (případně ji naleznete v oddíle 6.4.2).

```
public static JPanel makePropsPanel (JPanel panel1){
  return makePropsPanel(panel1, makeEmptyPanel(),
                makeEmptyPanel(), makeEmptyPanel());
}

public static JPanel makePropsPanel (
                JPanel panel1, JPanel panel2){
  return makePropsPanel(panel1, panel2,
                makeEmptyPanel(), makeEmptyPanel());
}

public static JPanel makePropsPanel (
                JPanel panel1, JPanel panel2,
                JPanel panel3){
  return makePropsPanel(panel1, panel2,
                panel3, makeEmptyPanel());
}

public static JPanel makePropsPanel (
                JPanel panel1, JPanel panel2,
                JPanel panel3, JPanel panel4){
  JPanel outPane=new JPanel();
  outPane.setPreferredSize(PROPS_PANEL_SIZE);
  outPane.setLayout(new GridLayout(2, 2));
  outPane.add(panel1);
  outPane.add(panel2);
  outPane.add(panel3);
  outPane.add(panel4);
  return outPane;
}
```

6.4.2 Funkce makeEmptyPanel

Funkce, která, jak název napovídá, vytvoří prázdný subpanel.

```
private static JPanel makeEmptyPanel(){
  JPanel outPane=new JPanel();
  outPane.setPreferredSize(
          new Dimension(PANEL_WIDTH, PANEL_HEIGHT));
  return outPane;
}
```

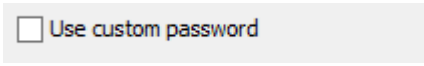
6.4.3 Funkce makePaddingPane

Funkce vytvoří prázdný panel o dané výšce (height). Je využíván v subpanelu pro transpozici s pevnou periodou pro vizuální oddělení panelu s checkboxem a panelu s textovým polem.

```
public static JPanel makePaddingPane (int height){
  JPanel outPane=new JPanel();
  outPane.setPreferredSize(new Dimension
          (COMPONENT_WIDTH, height));
  return outPane;
}
```

6.4.4 Funkce makeCheckBoxPane

Fukce vytvoří panel se zaškrtávacím políčkem (checkboxem) jako například na obrázku 8 vpravo. Parametr label určuje popisek u checkboxu, selected zda bude olíčko zaškrtnuto a checkEventListener bude informován, pokud dojde k zaškrtnutí nebo odškrtnutí políčka. Toho je využíváno v subpanelech k ovládání změn dostupnosti jednotlivých komponent při zaškrtnutí políčka (například při zaškrtnutí checkboxu, že uživatel chce použít svůj vlastní otevřený text, dojde k zpřístupnění textové oblasti, kam může uživatel svůj text napsat – viz subpanely v oddílech 6.5 a 6.6).

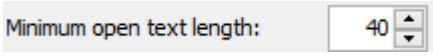


Obrázek 8: CheckboxPane

```
public static JPanel makeCheckBoxPane (
          String label, boolean selected,
          ActionListener checkEventListener){
  JPanel outPane=new JPanel();
  JCheckBox box= new JCheckBox();

  box.setSelected(selected);
  box.setText(label);
  box.addActionListener(checkEventListener);
  box.setPreferredSize(new Dimension(
          COMPONENT_WIDTH, CHECKBOXPANE_HEIGHT));

  outPane.add(box);
  return outPane;
}
```

6.4.5 Funkce makeSpinnerPane

Funkce vytvoří panel se spinnerem jako například na obrázku 9 vlevo. Parametr label označuje popisek vedle spinneru a SpinnerModel je objekt, který spinneru předává požadovanou hodnotu ze sekvence a rozhoduje, zda případná uživatelem zadaná hodnota je platná (tedy zda je součástí sekvence hodnot) [21].



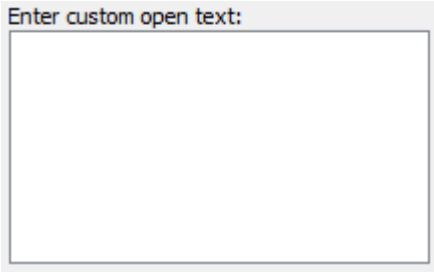
Obrázek 9: SpinnerPane

K vytvoření je použit BorderLayout. Jak popisek, tak spinner jsou tedy umístěny na hrany panelu a zaujímají minimální šířku.

```
public static JPanel makeSpinnerPane (
          String label, SpinnerModel model){
  JPanel outPane=new JPanel();
  JSpinner spinner= new JSpinner(model);
  JLabel lbl = new JLabel("<html>"+label+"</html>");
  spinner.setPreferredSize(new Dimension(
          SPINNER_WIDTH, spinner.getHeight()));
  outPane.setLayout(new BorderLayout());
  outPane.add(lbl, BorderLayout.LINE_START);
  outPane.add(spinner, BorderLayout.LINE_END);
  outPane.setPreferredSize(new Dimension(
          COMPONENT_WIDTH, SPINNERPANE_HEIGHT));
  outPane.setBorder(
          BorderFactory
               .createEmptyBorder(1, 0, 1, 0));
  return outPane;
  }
```

6.4.6 Funkce makeTextAreaPane

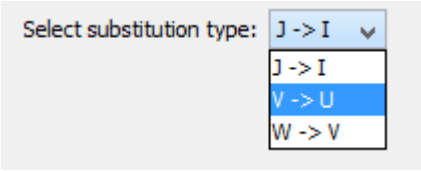
Funkce vytvoří panel s textovou oblastí jako na obrázku 10 vpravo. Do panelu je vložen objekt TextScrollPane, což je modifikovaný JScrollPane (o této třídě více viz oddíl 6.7). Pokud je text delší, je textová oblast doplněna dle potřeby posuvníkem.



Obrázek 10: TextAreaPane

```
public static JPanel
          makeTextAreaPane (
               String label, String text){
  JPanel outPane=new JPanel();
  JTextArea userText=new JTextArea();
  JLabel lbl=new JLabel("<html>"+label+"</html>");
  userText.setLineWrap(true);
  userText.setWrapStyleWord(true);
  userText.setText(text);
  TextScrollPane scrollPane
          =new TextScrollPane(userText,
               JscrollPane
                    .VERTICAL_SCROLLBAR_AS_NEEDED,
               JscrollPane
                    .HORIZONTAL_SCROLLBAR_NEVER);
  outPane.setLayout(new BorderLayout());
  outPane.add(lbl, BorderLayout.PAGE_START);
  outPane.add(scrollPane, BorderLayout.CENTER);
  outPane.setPreferredSize(
          new Dimension(COMPONENT_WIDTH,
               USERTEXTPANE_HEIGHT));
  return outPane;
}
```

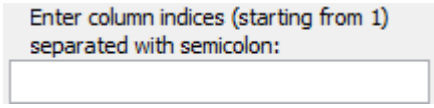
6.4.7 Funkce makeComboBoxPane



Obrázek 11: ComboBoxPane

Tato funkce vytvoří panel s ComboBoxem, tedy rozbalovacím seznamem (rozbalený ComboBox vidíte na obrázku 11 vlevo). Parametr label určuje text zobrazený u seznamu jako popisek. Prvky v comboboxu jsou definovány argumentem options. selectedIndex určuje kolikátý řetězec je vybrán. ItemListener je objekt, který bude upozorněn, pokud uživatel změní vybranou hodnotu v comboBoxu. Toho je využito v subpanelu pro šifru Baby Rijndael, kde možnost zadávat inicializační vektor závisí na vybraném režimu činnosti šifry.

```
public static JPanel makeComboBoxPane(
                String label, String[] options,
                int selectedIndex, ItemListener il){
  JPanel outPane=new JPanel();

  JLabel lbl=new JLabel("<html>"+label+"</html>");
  JComboBox cBox=new JComboBox(options);
  cBox.setSelectedIndex(selectedIndex);
  cBox.addItemListener(il);
  outPane.setPreferredSize(
                new Dimension(COMPONENT_WIDTH,
                     COMBOBOXPANE_HEIGHT));
  outPane.add(lbl);
  outPane.add(cBox);
  return outPane;
}
```



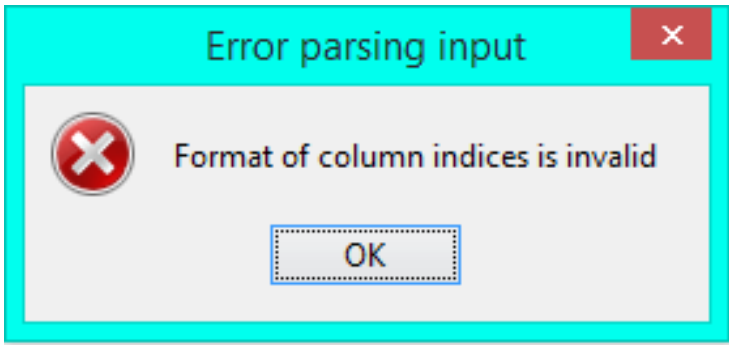
Obrázek 12: TextFieldPane

6.4.8 Funkce makeTextFieldPane

Posledním grafickým objektem je TextFieldPane jako například na obrázku 12. Funkce makeTextFieldPane vytvoří panel s popiskem a jednořádkovým textovým polem (s textem zarovnaným na střed).

```
public static JPanel makeTextFieldPane(
          String label, String text){
  JPanel outPane=new JPanel();
  JLabel lbl=new JLabel("<html>"+label+"</html>");
  JTextField tField=new JTextField(text);
  tField.setHorizontalAlignment(JTextField.CENTER);
  outPane.setPreferredSize(
          new Dimension(COMPONENT_WIDTH,
               TEXTFIELDPANE_HEIGHT));
  outPane.setLayout(
          new BoxLayout(outPane, BoxLayout.Y_AXIS));
  outPane.add(lbl);
  outPane.add(tField);
  return outPane;
}
```

6.4.9 Funkce showError



Obrázek 13: Chybové dialogové okno

Funkce využívaná mediátory k signalizaci (vedle začervenění pole), že při zpracování vstupu došlo k chybě. Funkce zobrazí dialogové chybové okno se zprávou. danou argumentem `message`.

```
public static void showError(String message){
  JOptionPane.showMessageDialog(null, message,
                "Error parsing input",
                JOptionPane.ERROR_MESSAGE);
}
```

6.4.10 Funkce pro načítání properties

AlgPropsFramework obsahuje dále trojici funkcí, které se používají pro načtení Properties ze souboru.

Funkce getPropertiesFolderName vrátí hodnotu konstanty PROPS_FOLDER_NAME (obsahuje řetězec „algprops“). Funkce getPropertiesFilename složí z názvu algoritmu název souboru s parametry.

```
public static String getPropertiesFolderName(){
  return PROPS_FOLDER_NAME;
}

public static String getPropertiesFilename(
                      String algName){
  StringBuilder sb
                =new StringBuilder(PROPS_FOLDER_NAME);
  sb.append(File.separator);
  sb.append(algName);
  sb.append(".properties");
  return sb.toString();
}
```

Funkce loadProps načte Properties ze souboru. Do vrácené instance vloží výchozí hodnoty předané parametrem defaults. Jméno souboru s nastavením algoritmu, který otevírá, je vytvořeno s pomocí funkce getPropertiesFilename, které předá druhý argument algName.

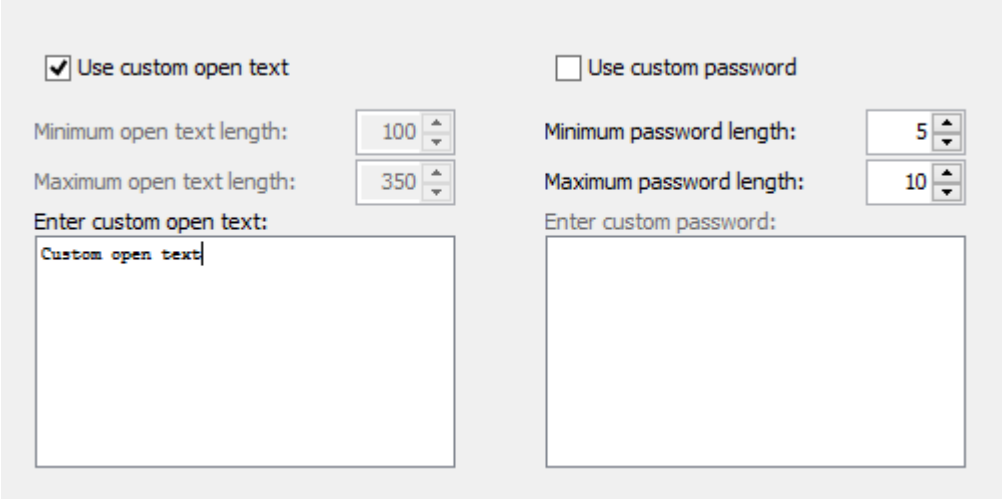
```
public static Properties loadProps(
                Properties defaults, String algName){
  Properties props=new Properties(defaults);
  FileInputStream fis;
  try{
    props.load( fis=new FileInputStream(
            AlgPropsFramework
                 .getPropertiesFilename(algName)));
  fis.close();
  }catch(IOException ex){
    //soubor nenalezen nebo chyba při čtení - pak
//jsou použity výchozí hodnoty, takže není třeba
//žádná akce
  }
  return props;
}
```

6.4.11 Funkce storeProps

Poslední funkce z třídy AlgPropsFramework slouží k ukládání instance třídy Properties do souboru. Ukládání probíhá zpravidla vždy poté, když dojde ke změně Properties. Jméno souboru, kam jsou Properties ukládány, je vytvořeno pomocí funkce getPropertiesFilename(). Pokud neexistuje složka pro ukládání souborů s parametry, je vytvořena. Na rozdíl od načítání, kde je chyba ignorována, je chyba při ukládání indikována chybovým dialogovým oknem.

```
public static void storeProps(
                Properties props, String algName){
  FileOutputStream fos;
  File f;
  String fileName=getPropertiesFilename(algName);
  try{
    if(!(f=new File(getPropertiesFolderName()))
                .exists())
      f.mkdirs();

    fos=new FileOutputStream(fileName);
    props.store(fos, null);
    fos.close();
  }
  catch(IOException ex){
    JOptionPane.showMessageDialog(null,
          "Saving of file "+fileName+" failed.",
          "Error saving properties file",
          JOptionPane.ERROR_MESSAGE);
  }
}
```

6.5 StandardSettingsSubpanel



Obrázek 14: Standard Settings Subpanel – vlevo otevřený text, vpravo heslo

Instance této třídy je jedním až ze čtyř možných subpanelů, které jsou zobrazeny v AlgorithmPropertiesDialogu. Využívá komponenty vytvořené pomocí AlgPropsFrameworku. Skládá se z CheckBoxPane, dvou SpinnerPane a TextAreaPane. StandardSettingSubpanel je možné vytvořit ve dvou téměř totožných formách – pro otevřený text a pro heslo. Obě varianty vidíte vedle sebe na obrázku 14. Pokud je zaškrtnut checkBox, zašednou spinnery a zaktivuje se textová oblast a naopak pokud není zaškrtnut, jsou aktivní spinnery a textové pole je neaktivní. Toto přepínání je řešeno přímo v této třídě, protože implementuje rozhraní ActionListener.

```
@Override
public void actionPerformed(ActionEvent ae){
  enableDisableComponents(
          ((JCheckBox) ae.getSource()).isSelected());
}

private void enableDisableComponents(boolean state){
  for(Component comp : minPane.getComponents())
    comp.setEnabled(!state);
  for(Component comp : maxPane.getComponents())
    comp.setEnabled(!state);
  for(Component comp : userTextPane.getComponents())
    comp.setEnabled(state);
}
```

Při vyčítání hodnot z panelu je nutné zkontrolovat, zda jsou data v pořádku. Většinou stačí zkontrolovat pouze spinnery, protože zadaný text je zpravidla filtrován pomocí funkce convertStringToEnglish z třídy Alphabet (pokud není zvoleno načítání surových dat – viz dále).

Ke kontrole dat ve spinnerech se využívá funkce checkSpinnerInput(). Ta zkontroluje správnost nerovností mezi minimální a maximální hodnotou. Pokud nesouhlasí, je maximální hodnota začervenêna a zobrazeno chybové dialogové okno. Začervenění spinneru probíhá na první pohled poněkud nezvykle, jedná se o doporučené obcházení již víc než deset let oznámené chyby v jazyce Java, kterou zatím nikdo nevyřešil[2].

2 http://bugs.java.com/bugdatabase/view_bug.do?bug_id=5047316

```
public void setSpinnerBackground(Color c){
  ((Jspinner.DefaultEditor)((JSpinner)maxPane
          .getComponent(1)).getEditor())
               .getTextField().setBackground(c);
}

public boolean checkSpinnerInput(){
  if(getMin()<=getMax()){
    setSpinnerBackground(AlgPropsFramework
          .SPINNER_BACKGROUND);
    return true;
  }
  else{
    setSpinnerBackground(Color.red);
    AlgPropsFramework.showError(
          "Max value must be greater "
          + "or equal than min value.");
    return false;
  }
}
```

Pokud jsou data v pořádku, jsou načtena do Properties pomocí funkce populateProps.

Pokud je paramer rawText true, je z textových polí načten text, tak, jak je zapsán, v opačném případě je převeden pomocí funkce convertStringToEnglish ze třídy Alphabet. Znění funkce naleznete ve zdrojovém kódu na přiloženém CD.

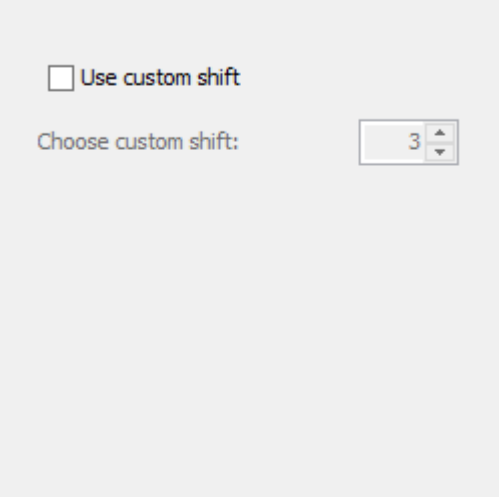
```
public Properties populateProps(Properties props){
  return populateProps(props, false);
}

public Properties populateProps(
               Properties props, boolean rawText)
```

6.6 SpecialSettingsSubpanel

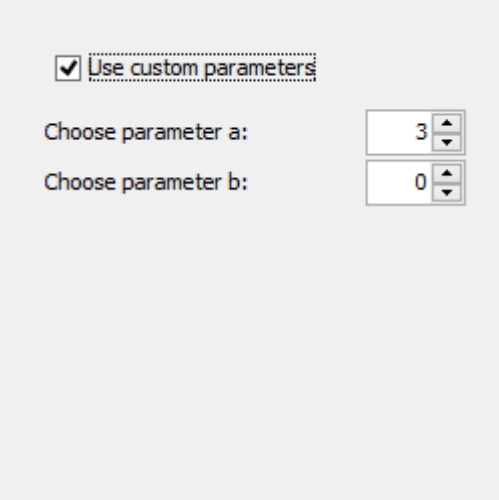
Tento panel je používán algoritmy s nestandardními parametry. V současné době jej využívá šest algoritmů. Jsou definovány jako konstanty, jež se používají jako parametr konstruktoru, kde dojde k vytvoření panelu pro příslušný algoritmus.

V úvodu zdrojového kódu třídy se nacházejí dvě vnořené třídy Check1ActionListener a Chack2ActionListener. Ty jsou využity, pokud je nutné obsloužit aktivování komponent panelu s použitím dvou checkboxů. Zatím je to využíváno pouze u šifry BabyRijndael. Pokud SpecialSettingsSubpanel obsahuje jen jeden checkBox, je jako ActionListener použita samotná třída SpecialSettingsSubpanel.

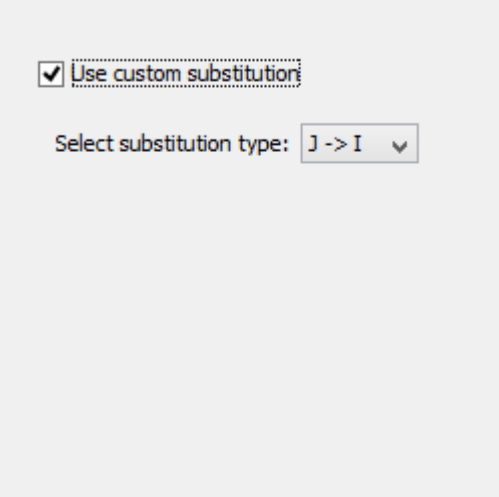
```
public static final int SIMPLE_SHIFT=0;
public static final int AFFINE=1;
public static final int PLAYFAIR=2;
public static final int TRANS_FIXED=3;
public static final int RSA=4;
public static final int RIJNDAEL=5;
```



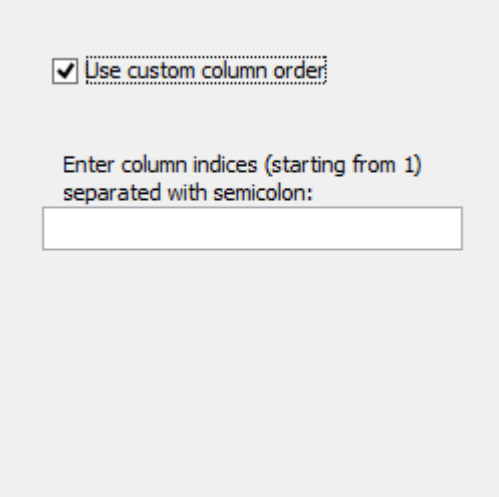
Obrázek 15: Jednoduchý posun



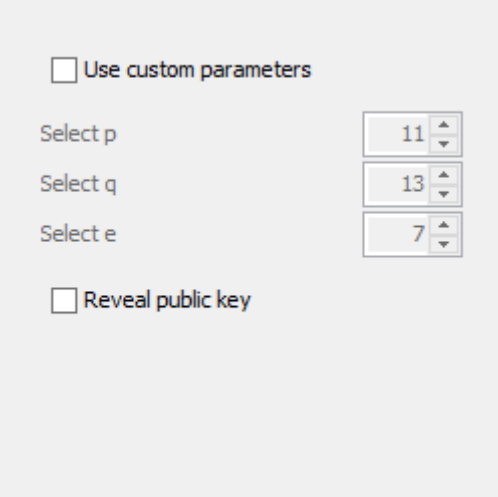
Obrázek 16: Afinní šifra



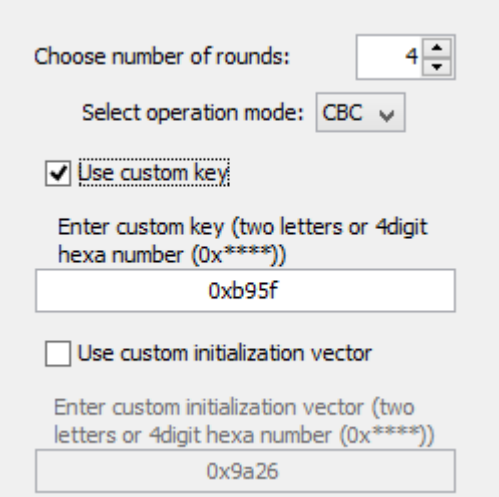
Obrázek 17: Playfair



Obrázek 18: Transpozice, pevná perioda



Obrázek 19: RSA



Obrázek 20: Baby Rijndael

Na této stránce se nachází snímky všech typů SpecialSettingsSubpanelu. Popisek určuje název algoritmu, který tento subpanel využívá.

Třída SpecialSettingsSubpanel poskytuje funkce k získávání dat a k označení nesprávného vstupu. Rozhodnutí o nesprávném vstupu je na mediátoru. Pokud je mediátor se vstupem spokojený, zavolá funkci populateProps, která v závislosti na typu panelu (pro jaký algoritmus je panel vytvořen) zapíše příslušné hodnoty do Properties.

6.7 TextScrollPane

Tato třída reprezentuje textovou oblast, kterou naleznete ve StandardSettingsSubpanelu. Byla vytvořena k tomu, aby umožnila deaktivování a změnu barvy textové oblasti.

K deaktivaci textové oblasti byla přetížena nefunkční metoda setEnabled pomocí následujícího kódu, který byl nabídnut jako obejití této chyby[3].

```
@Override
public void setEnabled (boolean state){
  getHorizontalScrollBar().setEnabled(state);
  getVerticalScrollBar().setEnabled(state);
  getViewport().getView().setEnabled(state);
}
```

S použitím proměnné view – tedy instance textové komponenty (JTextComponent) vložené do scrollovatelného panelu (JScrollPane) – byly implementovány užitečné metody pro nastavení a získání textu a změnu podbarvení.

```
public String getText(){
  return view.getText();
}

public void setText(String text){
  view.setText(text);
}

@Override
public void setBackground(Color bg){
  if(view!=null)
    view.setBackground(bg);
}
```

6.8 SpinnerNumberListModel

SpinnerModel je objekt, který využívá spinner k získání další nebo předchozí hodnoty v posloupnosti hodnot. Spinner rovněž modelu předává uživatelem zadanou hodnotu k dekódování a rozhodnutí, zda náleží mezi povolené hodnoty.

Třída SpinnerNumberListModel je použita tam, kde netvoří hodnoty souvislý sled čísel (tam, kde jsou přípustné hodnoty všechna přirozená čísla v určitém intervalu je použit SpinnerNumberModel). SpinnerNumberListModel se používá například u volby parametrů *p*, *q* a *e* u RSA, jelikož zde se vybírá z posloupnosti prvočísel.

Třída SpinnerNumberListModel dědí z třídy SpinnerListModel. Hlavním důvodem pro její vytvoření bylo možnost vstupu hodnoty z klávesnice. Původní funkce setValue přijímá pouze objekt stejného typu, jakého byl List nebo pole objektů, které byly vloženy do konstruktoru. Protože z klávesnice zadaný objekt je typu String, dojde k vyhození výjimky a hodnota je tedy uznána za neplatnou. Přetížená metoda nejprve získaný řetězec převede

3 http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4286743

na Integer a teprve ten předá metodě předka. Metoda předka si s ním tedy již správně poradí a hodnota (pokud je platná) je uznána.

Pro jednoduchost použití byl rovněž přidán konstruktor, který přijímá pole int místo pole Objektů. Při volání konstruktoru předka je jako argument použit výsledek statické funkce makeIntegerArrayfromInt, která pole int převede na pole objektů typu Integer. Při převádění je využit auto-boxing, tedy automatické převádění int na Integer (toto převádění ovšem samozřejmě nefunguje pro pole, tudíž je nutné převést v cyklu jeden prvek po druhém).

```
public SpinnerNumberListModel (int[] values){
  super(makeIntegerArrayFromInt(values));
}

@Override
public void setValue(Object elt){
  Integer num;
  if(elt instanceof String){
    try{
      num=Integer.parseInt((String) elt);
      super.setValue(num);
    } catch (NumberFormatException e){
      throw new
IllegalArgumentException(elt.toString());
     }
  }
  else{
    super.setValue(elt);
  }
}

public static Integer[] makeIntegerArrayFromInt(
                                        int[] intArray){
  Integer[] val=new Integer[intArray.length];
  for(int i=0; i<val.length;i++)
    val[i]=intArray[i];
  return val;
}
```

6.9 AlgPropsMediator

AlgPropsMediator je rozhraní, které definuje tyto dvě funkce:

```
public javax.swing.JPanel getSettingsPanel();
public boolean propertiesEditFinished();
```

První z nich je použita k vytvoření JPanelu s nastavením algoritmu. Druhá je zavolána když uživatel stiskne tlačítko OK v AlgorithmPropertiesDialogu. Funkce zpravidla otestuje správnost dat a pokud jsou správná, načte je do instance třídy Properties a tu předá šifrovacímu algoritmu a vrátí true, v opačném případě vrací false.

Mediator, prostředník, funguje jako takový předěl mezi algoritmy a GUI. Získá stávající instanci třídy Properties z algoritmu a zobrazí parametry algoritmů v panelu a po vyplnění uživatelských dat je načte a uloží do nové instance třídy Properties, kterou algoritmu předá.

6.10 MediatorManager

Třída, která algoritmu přidělí mediátor. Obsahuje jedinou funkci getAlgPropsMediator.

```
public static AlgPropsMediator
getAlgPropsMediator(EncodeAlgorithm ea){
  if (ea instanceof Slozena)
    return new CompoundMediator();

  switch(ea.getName()){ //fall-through
    case SL_TRANS:  case DV_SL_TRANS:  case LOWERCASE:
      return new OTOnlyMediator(ea);
    case SUBS_HES:  case VIGENER:  case BEAUFORT:
    case VIGENER_OT:  case VIGENER_ST:
    case SL_TRANS_HES:
      return new StandardMediator(ea);
//standardni, protoze ho vyuziva nejvice algoritmu,
    case SIMPLE_SHIFT:
      return new SimpleShiftMediator(ea);
    case AFFINE:
      return new AffineMediator(ea);
    case PLAYFAIR:
      return new PlayfairMediator(ea);
    case TRANS_FIXED:
      return new TransFixedMediator(ea);
    case RSA:
      return new RSAMediator(ea);
    case RIJNDAEL:
      return new RijndaelMediator(ea);
    default:
      return new EmptyMediator();
  }
}
```

Jak zdrojový kód MediatorManageru napovídá, bylo vytvořeno celkem deset mediátorů. Ty si nyní stručně představíme. Jejich zdrojový kód naleznete samozřejmě na přiloženém CD.

6.11 Mediátory

OTOnlyMediator

GetSettingsPanel vytvoří Panel s jedním StandardSettingsSubpanelem pro otevřený text.

StandardMediator

Funkce getSettingsPanel vytvoří panel o oběma variantami StandardSettingsSubpanelu.

SimpleShiftMediator

Mediátor určený pro algoritmus jednoduchý posun. Výstup funkce getSettingsPanel je tvořen standardním panelem pro otevřený text a speciálním panelem pro tento algoritmus.

AffineMediator

Používán výhradně afinní šifrou. Panel s nastavením tvoří standardní subpanel pro otevřený text a speciální subpanel pro afinní šifru. SpinnerNumberListModel ve speciálním subpanelu využívá funkci getAA(), kterou poskytuje třída AfinniSifra, která vrátí pole možných hodnot parametru *a* afinní šifry.

PlayfairMediator

Tento mediátor je používán pouze u šifry playfair. Funkce getSettingsPanel vytvoří panel se třemi subpanely. Dvojicí standardních subpanelů s otevřeným textem a heslem a speciálním subpanelem pro šifru playfair, kde je možné si určit typ substituce (pro získání možností substituce pro tvorbu comboboxu je v konstruktoru SpecialSettingsSubpanelu zavolána funkce getSubsTypes() ze třídy Playfair).

TransFixedMediator

Mediátor používaný transpozicí s pevnou periodou. Funkce getSettingsSubpanel vytvoří panel se standardním subpanelem pro otevřený text, druhý subpanel je speciální určený pro transpozici s pevnou periodou.

Ve funkci propertiesEditFinished probíhá testování spinnerů standardního subpanelu. Dále pokud uživatel zaškrtne, že si chce zvolit vlastní pořadí sloupců, mediátor načte řetězec a ověří jeho správnost tím, že použije funkci isValid(String str), kterou poskytuje třída TranspositionFixedPeriod.

RSAMediator

Tento mediátor používá algoritmus RSA. Panel s nastavením obsahuje standardní subpanel pro otevřený text a speciální subpanel šifry RSA.

Ve funkci propertiesEditFinished probíhá testování parametrů šifry, zda odpovídají požadavkům, tedy parametry p a q musí být rozdílná prvočísla, jejich součin musí být větší než 100 a největší společný dělitel součinu $(p-1)(q-1)$ s exponentem musí být roven jedné. Pokud jsou všechny požadavky splněny, jsou data načtena do Properties (funkcí populateProps()) a předána algoritmu RSA (setProperties()).

RijndaelMediator

Mediátor používaný šifrou BabyRijndael. V panelu se nachází speciální subpanel pro šifru Rijndael a standardní subpanel pro otevřený text.

Ve funkci propertiesEditFinished probíhá ověřování textu v závislosti na jeho typu. Jelikož šifra Baby Rijndael přijímá jak standardní text, tak hexadecimální řetězce, je nejprve rozhodováno, zda obsahují předponu 0x nebo ne. V případě, že ano, jsou ověřeny funkcí isValidHexaString(), kterou poskytuje třída BabyRijndael. Zároveň musí platit, že klíč i inicializační vektor mají správnou délku – tedy šest znaků (0x a čtyři hexadecimální číslice) respektive dva znaky.

CompoundMediator

Panel s nastavením určený pro složené algoritmy obsahuje pouze nápis, že se jedná o složený algoritmus a tedy se řídí nastavením jednotlivých atomických algoritmů. Funkce propertiesEditFinished je prázdná, jen vrací true.

EmptyMediator

Popisek v panelu s nastavením říká, že algoritmus neposkytuje žádné možnosti nastavení. I zde funkce propertiesEditFinished jen vrací true.

7 Export do pdf

Implementace exportu do pdf je podrobněji popsána v [2], oddíl 4, zde shrnu pouze nejdůležitější informace a změny oproti informacím uvedeným ve zprávě k projektu.

PDF je formát vytvořený společností Adobe k usnadnění tisku a zobrazování dokumentů se zachováním jejich vzhledu napříč různými zařízeními. Formát pdf popisuje norma [22].

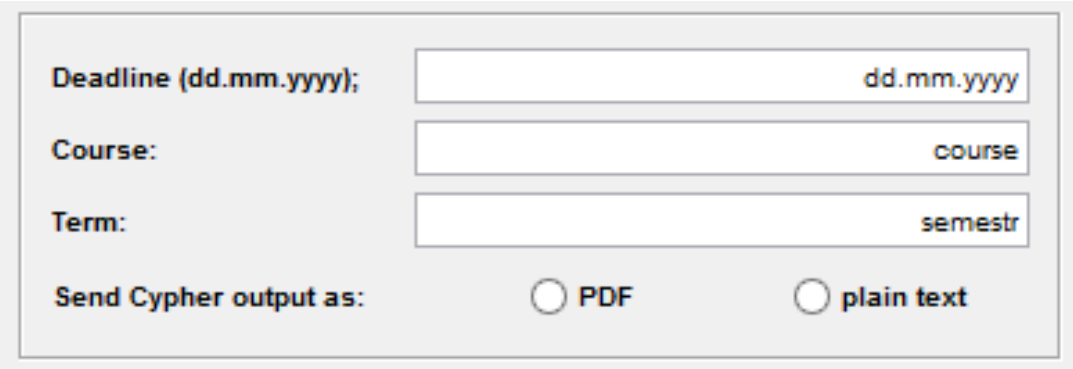
K implementaci exportu do pdf byla použita knihovna Apache PDFBox [23], na jejíž stránkách lze nalézt i užitečnou kuchařku, kterou jsem se při programování částečně inspiroval [24].

Tvorbu pdf zajišťují dvě třídy PDFMaker a DocumentCursor. Třída DocumentCursor slouží k emulaci psaní do „normálního“ dokumentu, protože formát pdf umožňuje řádky textu zapisované v libovolném sledu, klidně i přes sebe. DocumentCursor posouvá pozici zápisu následujícího řádku nebo informuje třídu PDFMaker o dosažení konce stránky aby mohl přidat novou.

PDFMaker vytvoří pdf dokument ve své funkci makePDF, kde získá z TaskDefinition sadu řetězců, které je nutné vložit do dokumentu. V případě, že jsou příliš dlouhé na to, aby se vešly na jeden řádek, (o tom informuje třída DocumentCursor), jsou rozděleny. Pokud v řetězci nalézá mezera, rozdělení proběhne v mezeře.

Během psaní třída PDFMaker odřádkovává kurzor. Když zjistí, že je na konci stránky, založí novou a resetuje kurzor.

Ovládání možnosti výstupu do pdf naleznete v hlavním okně aplikace jako dva přepínače (radiobuttons). Je možné si zvolit výstup aplikace jako „plain text“, tehdy bude aplikace Cypher pracovat jako předtím, nebo pdf, a pak budou odeslaný soubor i kontrolní soubor pro učitele uloženy jako pdf.



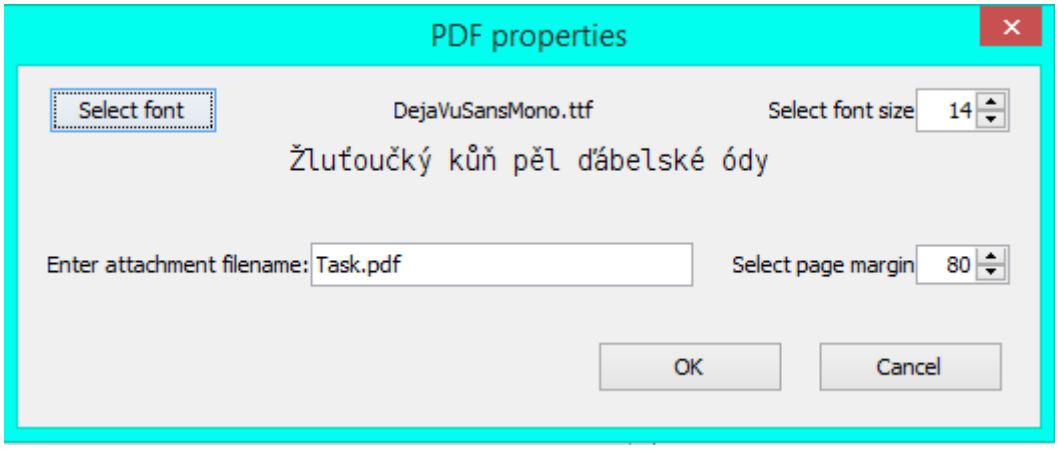
Obrázek 21: Panel s nastavením výstupu aplikace (převzato z [2])

Odesílání pdf souboru jako přílohy bylo vyřešeno použitím již předpřipravené metody sendMessage ve třídě MailSender. Bohužel ji bylo nutné trochu upravit, protože soubor přicházel poškozený (úpravu naleznete v [2]).

Oproti předchozí verzi popsané v [2] bylo vytvořeno dialogové okno, třída PDFPropertiesDialog (jako na obrázku 22), kde je možné nastavit parametry výstupu do pdf. Jedná se o použitý font, velikost písma, velikost okrajů a název pdf přílohy odesílané studentům. Tyto parametry jsou uložené v nové třídě PDFConfig v balíku conf. V souvislosti s tím byly sem ze třídy CypherProperties přesunuty parametry spravující výstup do pdf (informace ískaná z radiobuttons v GUI a název pdf přílohy). Tudíž znění strany 29 z [2] již není platné. Načítání nyní probíhá z instance třídy PDFConfig.

Třída PDFConfig obsahuje interně instanci třídy Properties a poskytuje řadu metod, kterými je možné jednotlivé položky uložené v Properties získat a nastavit.

Do menu Edit v hlavním okně aplikace byla přidána položka PDF Settings. Pokud na ni uživatel klikne, zobrazí se instance třídy PDFPropertiesDialog. Snímek dialogu vidíte na obrázku 22 na další stránce.



Obrázek 22: PDF Properties Dialog

V levém horním rohu vidíte tlačítko Select font. Po kliknutí se objeví JFileChooser, kde je možné vybrat soubor fontu (je nastavený filtr pro soubory s příponou .otf a .ttf). Název zvoleného souboru se objeví na popisku nahoře uprostřed a překreslí se vzorový text uprostřed panelu. Vpravo nahoře se nachází spinner, kde je možné zvolit velikost fontu. Po změně hodnoty dojde rovněž k překreslení textu. V dolní části dialogového okna pak naleznete textové pole, kam lze zapsat jméno souboru, který bude odesílán studentům jako příloha a pomocí spinneru lze nastavit velikost okrajů stránky dokumentu. Po kliknutí na tlačítko OK dojde k uložení nastavených hodnot do instance třídy PDFConfig.

Zdrojový kód naleznete samozřejmě na přiloženém CD. Vedle toho se na přiloženém CD nachází vzorový výstup programu (zadání pro studenty a soubor pro učitele).

8 Složená šifra

8.1 Úvod

Složené šifry byly vytvořeny dvěma autory bez návaznosti na sebe. V nich se tak objevily dva přístupy. Přístup pana Kouby – zřetězit šifry až ve třídě Model ([2], strana 13), přístupem pana Kohouta bylo vytvořit abstraktní třídu, která algoritmy zřetězí a navenek se chová jako jednoduchá. Já jsem se rozhodl tyto přístupy sjednotit. Líbí se mi přístup pana Kohouta, kdy to, že se jedná o složenou šifru je zapouzdřeno a třída Model s ní zachází jako s jednoduchou. Bohužel kvůli přehlédnutí při programování dojde při výběru některé ze složených šifer pana Kohouta k zamrznutí programu.

```
public int getMinPasswordLength() {
    return Math.max(prvni.getMinPasswordLength(),
                druha.getMinPasswordLength());
}

public int getMaxPasswordLength() {
    return Math.min(prvni.getMinPasswordLength(),
                druha.getMinPasswordLength());
}
```

Metody getMinPasswordLength a getMaxPasswordLength jsou sice naprogramované správně, nicméně nepočítají s tím, že délka hesla u transpozic je nulová (protože žádné heslo nepotřebují, viz tabulka 1 na straně 17). Poté program totiž hledá například heslo o délce větší než pět a zároveň menší než nula, a takové samozřejmě nenajde. Proto je nutné při programování mého sjednocení brát v potaz nulovou délku hesla u jednoho nebo více algoritmů, ze kterých se složená šifra skládá.

Zároveň bylo mým cílem zachovat „look and feel“ původních složených šifer od pana Kouby, jakožto zakladatele této aplikace. Tudíž jsem do třídy Složená, kterou jsem vytvořil, přenesl část kódu pana Kouby z třídy CompoundAlgorithmName.

Metoda getName v abstraktní třídě EncodeAlgorithm původně vrací pole řetězců (názvy jednotlivých elementárních šifer). Prvky tohoto pole se složí ve třídě CompoundAlgorithmName. Díky tomu, že jméno složené šifry bude nyní připravováno ve třídě Složená, odpadá nutnost vracet pole řetězců v rozhraní EncodeAlgorithm. Proto byla návratová hodnota funkce getName z pole řetězců změněna jen na jeden řetězec.

U funkce encode byla naopak návratová hodnota z jednoho řetězce převedena na pole řetězců. Původně totiž byly parametry šifry uvedené na začátku výstupního řetězce, které musely být následně ve třídě TaskDefinition oddělovány (úkolem studenta je parametry šifry zjistit, takže mu je nesmíme poslat), tudíž výpočetně mnohem méně náročné je vracet šifrovaný text a řetězec parametrů odděleně jako dva prvky pole řetězců.

Jelikož je díky nastavení algoritmů (kapitola 6) možné změnit otevřený text (ve funkci encode se nebere zřetel na otevřený text předložený aplikací, ale vezme se text definovaný uživatelem), je nutné vracet i jej, protože jinak by v textu pro učitele byl otevřený text vybraný generátorem otevřeného textu v aplikaci a ne ten od uživatele. Otevřený text tedy zaujal třetí místo výsledného pole řetězců.

Změnou rozhraní EncodeAlgorithm bylo samozřejmě nutné upravit zdrojové kódy všech šifrovacích algoritmů, jelikož toto rozhraní implementují.

Nyní se podívejme na samotný kód abstraktní třídy Slozena.

8.2 Vlastní popis kódu

8.2.1 Třída Slozena

Jednotlivé šifrovací algoritmy ze kterých se skládá složená šifra jsou uložené jako prvky pole s názvem šifry. To zajišťuje možnost zřetězení teoreticky neomezeného počtu šifrovacích algoritmů.

```
EncodeAlgorithm [] sifry;
```

Jelikož třída Slozena implementuje rozhraní EncodeAlgorithm, musí samozřejmě obsahovat funkce getName, encode, getSettingsPanel, propertiesEditFinished getMinTextLength, getMaxTextLength getMinPasswordLength a getMaxPasswordLength.

Funkce getName vrací řetězec s názvem šifry. K vytvoření řetězce používá funkci getCompoundName přesunutou ze třídy CompoundAlgorithmName (jako parametr funkce getCompoundName (v tomto případě null) se vkládají parametry šifer (např. použité heslo nebo rozměry tabulky u transpozic), které jsou samozřejmě známy až během šifrování, tudíž se tato funkce užívá rovněž ve funkci encode (viz dále)).

```
@Override
public String getName(){
  return getCompoundName(null);
}
```

Funkce encode v cyklu prochází jednotlivé elementární šifry a volá jejich funkci encode. Výstupní pole řetězců (o dvou prvcích) rozdělí takto: první řetězec obsahuje šifrovaný text, proto jej předá další šifře v pořadí. Druhý prvek obsahuje název šifry a parametry. Ty ukládá do pomocného pole params, které poté vloží jako vstup do funkce getCompoundName. Ta je spojí dohromady do jednoho řetězce. Výstup této funkce tvoří stejně jako u elementárních šifer dva řetězce. První je zašifrovaný text a druhý je tvořen zmíněným výstupem funkce getCompoundName.

```
@Override
public String [] encode(String plainText,
                        String password){
  String[] tmp=null;
  String[] params=new String[sifry.length];
  for (int k=0; k<sifry.length; k++) {
    tmp=sifry[k].encode(plainText, password);
    params[k]=tmp[1];
    plainText=tmp[0];
  }

  return new String[] {tmp[0],
                       getCompoundName(params)};
}
```

Ve třídě Slozena se samozřejmě musí nacházet funkce getProperties(), která pouze vrací null a dále setProperties a resetProperties, které nedělají nic. Je to proto, že složená šifra se řídí nastavením atomických algoritmů.

```
@Override
public Properties getProperties(){
  return null;
}

@Override
public void setProperties (Properties props){}

@Override
public void resetProperties(){}
```

Třídu Složená dále tvoří už jen čtveřice funkcí getMinTextLength, getMaxTextLength getMinPasswordLength a getMaxPasswordLength. Jelikož se však od sebe téměř neliší, uvedu zde pouze první z nich. Celý zdrojový kód naleznete na přiloženém CD.

```
@Override
public int getMinTextLength() {
  int tmp;
  int min=Integer.MAX_VALUE;
  for (EncodeAlgorithm sifra : sifry) {
    tmp=sifra.getMinTextLength();
    if(tmp<min)
      min=tmp;
  }
  return min;
}
```

Funkce nejprve projde všechny šifrovací algoritmy a hledá maximum (předpokládá, že intervaly možných délek textů jednotlivých algoritmů mají neprázdný průnik). Nalezené maximum poté vrátí. Funkce getMaxTextLength naopak vrací minimum.

Funkce získavající hranice intervalu hesla jsou velmi podobné, jediný rozdíl je v tom, že je nutné vzít v potaz, že délka hesla některých algoritmů je nulová. Proto je do příkazu if vložena další podmínka:

```
if((tmp!=0)&&(tmp<min))
  min=tmp;
```

a jelikož se může stát, že složená šifra se skládá pouze z algoritmů, které nevyžadují heslo (a tedy návratové hodnoty funkcí get???PasswordLength jsou nulové), je nutné zajistit, aby v takovém případě byla nulová návratová hodnota i u složené šifry. To řeší ternární operátor v závěru funkce (pokud všechny algoritmy vyžadují nulovou délku hesla, nedojde ke změně proměnné min, resp. max).

```
return (min==Integer.MAX_VALUE)?0:min;
```

8.2.2 Potomci třídy Slozena

Z třídy Slozena dědí všechny složené šifry. Dědění je jednoduché. Jelikož veškeré potřebné funkce jsou již v abstraktní třídě Slozena napsány, stačí ve dceřinných třídách vedle funkce getInstance pouze napsat konstruktor, který inicializuje pole sifry (viz předchozí oddíl). Jako příklad mohu uvést konstruktor třídy AfinniSifraPlusSloupTrans (jelikož jsou všechny algoritmy singleton, mají private konstruktor a k instanci této třídy se přistupuje pomocí již zmíněné funkce getInstance (viz zdrojový kód rozhraní EncodeAlgorithm)).

```
private AfinniSifraPlusSloupTrans() {
  sifry=new EncodeAlgorithm[2];
  sifry[0]=AfinniSifra.getInstance();
  sifry[1]=SloupcovaTranspozice.getInstance();
}
```

8.2.3 Změny v dalších třídách

Ve třídě Model stačilo z funkce initAlgorithms odstranit volání funkce getCompoundName ze třídy CompoundAlgorithmName, jelikož jméno složeného algoritmu je vráceno již po zavolání funkce getName. Část původního kódu:

```
String[] algNames = alg.getName();
CompoundAlgorithmName compoundAlgName = new
                CompoundAlgorithmName(algNames);
algorithms.put(
     compoundAlgName.getCompoundName(null), alg);
```

byla nahrazena:

```
String algName = alg.getName();
algorithms.put(algName, alg);
```

Vnitřní třída CypherTask ve třídě TaskDefinition byla pozměněna zásadnějším způsobem. Vedle spíše „kosmetických“ změn v názvu a v druhém případě i typu proměnných (coupoundAlgorithm změněn na algorithm a CompoundAlgorithmName compoundAlgName na String algName) a proměně funkce getCompoundAlgName na getAlgName došlo k přepracování inicializační metody init. Původní kód byl díky přesunu zřetězení atomických algoritmů do třídy Složená velmi zjednodušen.

Ve třídě TaskDefinition bylo nahrazeno volání

```
task.getCompoundAlgName().getCompoundName(
                task.getParameterText())
```

voláním

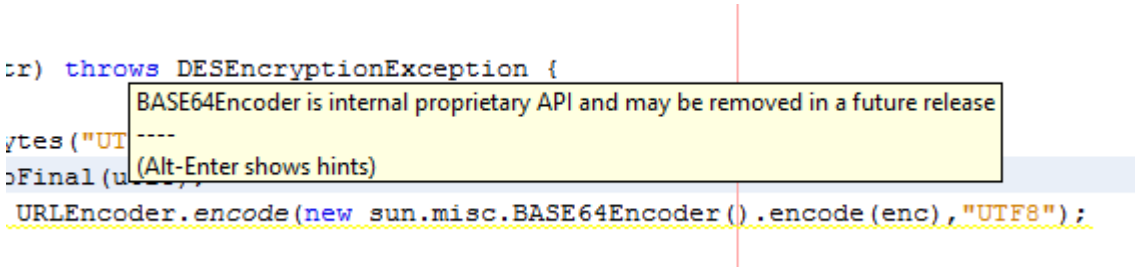
```
task.getParameterText()
```

které vrátí zmíněný druhý prvek pole s názvem šifry a použitými parametry (kupříkladu řetězec „04 - Playfair [key=PASSWORD V->U]“). To bylo možné díky tomu, že řetězec v souboru pro učitele (za textem Algoritmus:) vzniká již v jednotlivých šifrách a je navrácen funkcí encode jako druhý prvek pole řetězců a tedy není nutné jej vytvářet.

9 Další drobnější úpravy

9.1 DESEncryptor

Využívání proprietární API sun.misc.BASE64Encoder a sun.miscBASE64Decoder ve třídě DESEncryptor není doporučované, protože hrozí, že přestane být dostupné v některé z dalších vydání Javy, o čemž informuje IDE na obrázku 23 (převzato z [2])



Obrázek 23: Zobrazení varování v IDE NetBeans(převzato z [2])

Proto bylo jejich volání nahrazeno pomocí třídy Base64 [3].
Původní znění příkazu pro šifrování

```
String encodedString = URLEncoder.encode(
                new sun.misc.BASE64Encoder()
                     .encode(enc),"UTF8");
```

bylo nahrazeno

```
String encodedString=Base64.getEncoder()
                .encodeToString(enc);
```

a příkaz pro dešifrování

```
byte[] dec = new sun.misc.BASE64Decoder()
                .decodeBuffer(URLDecoder
                     .decode(str,"UTF8"));
```

byl nahrazen

```
byte[] dec=Base64.getDecoder()
           .decode(URLDecoder.decode(str,"UTF8"));
```

URLDecoder by zde nemusel být, protože by i tak byl výstup výše uvedené šifrovací funkce správně dekódován, nicméně byl ponechán pro zpětnou kompatibilitu (tedy aby fungovala hesla uložená předchozí verzí DESEncryptoru).

9.2 Switch místo kaskády if

Ve třídě CypherGUIView byla kaskáda příkazů if ve funkci propertyChange nahrazena příkazem switch. Od Javy 1.5 je totiž možné použít switch i na řetězce (podrobnější informace najdete v [2], str. 33).

9.3 Oprava funkce sendToOneHandler

Ve třídě CypherGUIView byl dále také přesunut dotaz zda model obsahuje dalšího adresáta před odesílání e-mailu (volání funkce sendToOne). Původně totiž funkce sendToOne odeslala e-mail předposlednímu adresátovi, posunula iterátor na poslední prvek a poté se sendToOneHandler ptala modelu, zda obsahuje další prvky (hasNextRecipient). Ten odpověděl, že ne, a funkce tedy oznámila, že dorazila na konec seznamu adresátů a přesunula iterátor (rewindRecipients()). Poslední adresát tak nemohl dostat e-mail. Po přesunutí je dotaz na další prvky umístěn před odesíláním e-mailu a posouváním iterátoru, proto e-mail dostane i poslední adresát.

Zde uvádím část kódu funkce. Původní umístění dotazu hasNextRecipient() bylo v příkazu if, tedy až po posunutí iterátoru, které probíhá ve funkci sendToOne().

```
boolean hasNextRecipient=model.hasNextRecipient();
try {
  glass.setCursor(Cursor
          .getPredefinedCursor(Cursor.WAIT_CURSOR));
   glass.setVisible(true);
   Controler.sendToOne();
} catch (CypherException ex) {
Logger.getLogger(CypherGUIView.class.getName())
          .log(Level.SEVERE, null, ex);
} finally {
  glass.setCursor(null);
}
if (!hasNextRecipient) {
  JoptionPane.showMessageDialog(null,
          "This was the last recipient in the
          + "recipient list.");
  model.rewindRecipients();
}
```

10 Závěr

V rámci bakalářské práce jsem se věnoval implementováním nových funkcí a vylepšováním stávajících v aplikaci Cypher. V rámci zadání jsem měl za úkol se s aplikací seznámit, opravit chyby a implementovat několik úkolů dle zadání. Nyní shrnu výsledky jednotlivých úkolů.

10.1 Export do pdf

Splněno s použitím knihovny Apache PDFBox. Zároveň přidána do GUI možnost měnit nastavení exportu do PDF. V budoucnu by bylo vhodné vylepšit export do pdf větší variabilitou formátování (nastavení stylů odstavců, přidání možnosti zarovnání textu do bloku, atd).

10.2 Odesílání e-mailu s pdf přílohou

Splněno s použitím upravené již implementované funkce.

10.3 Implementace asymetrických algoritmů

Implementována byla šifra RSA s náhodně generovanými malými moduly. Prvočísla pro tvorbu modulů jsou tedy vybírána pouze mezi prvními 93 prvočísly (tedy z intervalu <2;499>).

10.4 Implementace moderních symetrických algoritmů

Naprogramována byla šifra Baby Rijndael, což je zmenšená varianta šifry Rijndael, tedy vítězného algoritmu pro standard AES. Rovněž byly implemntovány dva režimy činnosti blokové šifry.

10.5 Parametrizace algoritmů

Bylo implementována možnost nastavit algoritmy v GUI. Je možné volit hodnoty parametrů náhodné (původní implementace) nebo definovat konkrétní hodnoty a tedy předem připravit zadání pro studenty eliminováním veškerých generátorů náhodných čisel.

Vedle toho jsem v aplikaci opravil nalezené chyby. Je možné, že jsem na nějaké zapomněl nebo při implementaci nových funkcí další vytvořil. Pokud byste ve zdrojovém kódu případně při chodu aplikace zjistili nějakou chybu, můžete se ozvat na mou školní e-mailovou adresu: smutnmir@fel.cvut.cz případně pokud nebude funkční můžete použít můj soukromý e-mail miroslav.smutny94@seznam.cz. Na stejné e-mailové adresy se můžete rovněž obrátit v případě nečitelnosti či absence přiloženého CD.

11 Seznam použité literatury

[1]: *Welcome to NetBeans* [online] Oracle corporation and/or its affiliates. 2015 [cit. 2015-10-31]. Dostupné z: https://netbeans.org/index.html

[2]: SMUTNÝ, M. *Vylepšení systému pro generování šifrových textů*. Praha: ČVUT 2016. Zpráva k Individuálnímu projektu. ČVUT v Praze, Fakulta elektrotechnická, Katedra telekomunikační techniky

[3]: *Base 64 (Java Platform SE 8)* [online] Oracle Corporation and/or its affiliates. 1993, 2016. Dostupné z: http://docs.oracle.com/javase/8/docs/api/java/util/Base64.html

[4]: *Overview (Java Platform SE 8)* [online] Oracle corporation and/or its affiliates. 2016 [cit. 2016-02-08]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/overview-summary.html

[5]: *The Java (tm) Tutorials* [online] Oracle and/or its affiliates. 1995, 2015 [cit. 2016-04-23]. Dostupné z: http://docs.oracle.com/javase/tutorial/

[6]: *Properties (Java Platform SE 8)* [online] Oracle and/or its affiliates. 1993, 2015 [cit. 2016-04-23]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html

[7]: *BigInteger (Java Platform SE 8)* [online] Oracle and/or its affiliates. 1993, 2016 [cit. 2016-04-24]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

[8]: *BitSet (Java Platform SE 8)* [online] Oracle corporation and/or its affiliates. 1993, 2016 [cit. 2016-05-02]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html

[9]: *Lesson: Laying Out Components Within a Container (The Java (tm) Tutorials > Creating GUI with JFC/Swing)* [online] Oracle and/or its affiliates. 1995, 2015 [cit. 2016-04-24]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html

[10]: *How to use BorderLayout (The Java (tm) Tutorials > Creating a GUI With JFC/Swing > Laying Out Components Within a Container)* [online] Oracle and/or its affiliates. 1995, 2015 [cit. 2016-05-09]. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

[11]: KLIMA, Richard E. a Neil P. SIGMON. *Cryptology: Classical and Modern with Maplets*. Boca Raton: CRC Press, 2012. ISBN: 978-1-4398-7241-3.

[12]: SMUTNÝ, Miroslav. *Kryptografie*. [zápočtová prezentace z předmětu Prezentační dovednosti]. Praha: ČVUT FEL. 30. 11. 2015.

[13]: MARTIN, Keith M.. *Everyday cryptography*. New York: Oxford University Press Inc., 2012. ISBN: 978-0-19-969559-1.

[14]: POLÍVKA, Jaroslav. *Pod pokličkou šifrovacího algoritmu RC4*. [online]. itnetwork.cz. 2016 [cit. 2016-05-12]. Dostupné z: http://www.itnetwork.cz/algoritmy/ostatni/pod-poklickou-algoritmu-rc4/

[15]: DIFFIE, Whitfield a Martin E. HELLMAN. New Directions in Cryptography In: *IEEE Transactions on Information Theory* [online] IEEE. November 1976, vol. it-22, no. 6, 644-654. ISSN 0018-9448. [cit. 2016-05-13]. Dostupné z doi: 10.1109/TIT.1976.1055638 Dostupné také z: http://www-ee.stanford.edu/~hellman/publications/24.pdf

[16]: VANĚK, Tomáš. *Režimy činnosti blokových šifer v1.1.5*. [prezentace k přednášce z předmětu Informační bezpečnost]. Praha: ČVUT FEL

[17]: MAO, Wenbo. *Modern cryptography: theory and practise*. Upper Saddle River: Pearson Education, 2004. ISBN: 0-13-066943-1.

[18]: BERGMAN, Clifford. *A Description of Baby Rijndael*. [online]. Ames: Iowa State University. February 21, 2005 [cit. 2016-05-14]. Dostupné z: http://orion.math.iastate.edu/cbergman/crypto/homework/babyr/babyr.pdf

[19]: VONDRUŠKA, Pavel. Asymetrická kryptografie - RSA In: *Crypto-World* [online] . 2001, 11/2001. ISSN 1801-2140. [cit. 2016-05-16]. Dostupné z: http://crypto-world.info/casop3/crypto11_01.pdf

[20]: *Sample data for Baby Rijndael* [online] BERGMAN, Clifford. [cit. 2016-05-17]. Dostupné z: http://orion.math.iastate.edu/cbergman/crypto/homework/babyr/sampledata3.html

[21]: *SpinnerModel (Java Platform SE 8)* [online] Oracle and/or its affiliates. 1993, 2016 [cit. 2016-05-20]. Dostupné z: https://docs.oracle.com/javase/8/docs/api/javax/swing/SpinnerModel.html

[22]: ISO 32000-1:2008 Document management - Portable Document Format - Part 1: PDF 1.7. ICS: 35.240.30. First Edition 2008-07-01 International Organization for Standardization. Dostupné také z: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf

[23]: *Apache PDF Box | A Java PDF Library* [online] The Apache Software Foundation. 2009-2015 [cit. 2016-04-20]. Dostupné z: https://pdfbox.apache.org/

[24]: *Apache PDFBox | Cookbook - Document Creation* [online] The Apache Software Foundation. 2009-2015 [cit. 2015-02-10]. Dostupné z: http://pdfbox.apache.org/1.8/cookbook/documentcreation.html