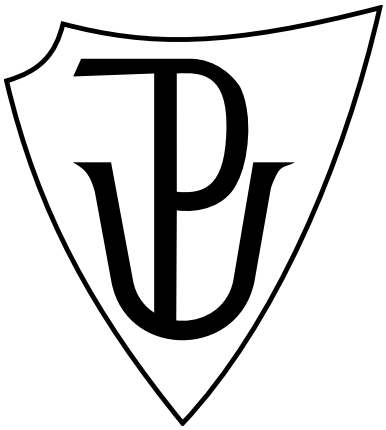
UNIVERZITA PALACKÉHO V OLOMOUCI
PEDAGOGICKÁ FAKULTA
KATEDRA MATEMATIKY

DIPLOMOVÁ PRÁCE

Asymetrická kryptografie



Vedoucí diplomové práce:
doc. RNDr. Tomáš Zdráhal, CSc.
2017

Vypracoval:
Bc. Vojtěch Šoupal
UM-UTIV, 2. ročník

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. RNDr. Tomáše Zdráhala, CSc. a že jsem uvedl veškerou použitou literaturu.

V Olomouci dne

.....

Bc. Vojtěch Šoupal

Poděkování

Mé poděkování patří panu Doc. RNDr. Tomáši Zdráhalovi, CSc. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval.

Bibliografická identifikace

Jméno a příjmení autora	Bc. Vojtěch Šoupal
Název práce	Asymetrická kryptografie
Typ práce	Diplomová
Pracoviště	Katedra matematiky
Vedoucí práce	doc. RNDr. Tomáš Zdráhal, CSc.
Rok obhajoby	2017
Abstrakt	Diplomová práce seznamuje čtenáře s asymetrickým šifrováním RSA. Text je rozdělen do tří kapitol. První kapitola se zaměřuje na matematické pozadí RSA šifrování. Druhá kapitola se věnuje základním znalostem šifrování a detailněji vysvětluje principy RSA. Třetí kapitola patří systému počítačové algebry Maxima a RSA šifrování v něm.
Klíčová slova	Kryptografie, asymetrická kryptografie, RSA šifrování, Maxima
Počet stran	64
Počet příloh	0
Jazyk	český

Bibliographical identification

Autor's first name and surname	Bc. Vojtěch Šoupal
Title	Asymmetric cryptography
Type of thesis	Master
Department	Department of Mathematics
Supervisor	doc. RNDr. Tomáš Zdráhal, CSc.
The year of presentation	2017
Abstract	The diploma thesis focuses on the asymmetric encryption RSA. The theoretical part is divided into three chapters. The first chapter states the mathematical background of RSA encryption. The second chapter describes the basic knowledge of encryption as well as the principles of RSA in detail. The third chapter addresses the system of computer-aided algebra Maxima and the RSA encryption in such system.
Keywords	Cryptography, asymmetric cryptography, RSA encryption, Maxima
Number of pages	64
Number of appendices	0
Language	Czech

Obsah

Seznam obrázků

Použité značení

Symbol	**Význam**
$\mathbb{Z}$	Množina celých čísel
$\mathbb{N}$	Množina přirozených čísel
$[i]_n$	Zbytková třída modulo n obsahující prvek i
$\mathbb{Z}_n$	Množina všech zbytkových tříd modula n
mod	Funkce modulo
$\equiv$	Relace ekvivalence
$\mid$	Relace býti dělitelem
$NSD(a, b)$	Největší společný dělitel čísel a a b
$\varphi(n)$	Eulerova funkce

Úvod

Cílem této práce je čtenáře provést pasážemi modulární aritmetiky, které jsou potřebné pro pochopení matematického pozadí asymetrického šifrování RSA. Tuto teoretickou pasáž textu budeme doplňovat o praktické příklady, které pomohou danou teorii lépe uchytit. V další kapitole se seznámíme se základy kryptografie. Konec této kapitoly budeme věnovat výhradně asymetrickému šifrování RSA. Ukážeme si principy generování klíčů, zašifrování zprávy a samozřejmě dešifrování. Opět budeme části textu prokládat příklady a to jak jednoduchými, tak i příklady odpovídajícími použití v praxi. Čtenář si na těchto příkladech uvědomí složitost výpočtu šifrovacího systému. V poslední kapitole se seznámí se systémem počítačové algebry Maxima. Po získání základních znalostí o programu čtenáře provedeme vytvořením příkazů pro ověření výpočtů šifrování z druhé kapitoly tohoto textu. Na konci této kapitoly vytvoříme nové funkce, které budou sloužit jako demonstrace šifrování v programu Maxima.

1 Matematické pozadí asymetrického šifrování

V problematice šifrování hraje matematika zásadní roli. Proto ještě dříve, než se začneme bavit o samotném asymetrickém šifrování, seznámíme čtenáře s potřebným matematickým aparátem. Očekává se znalost základních algebraických pojmů, které budeme v dalším textu užívat. Čtenář si je může připomenout v druhé kapitole [1]. Naší snahou bude tuto matematickou část textu předložit co nejstručněji. Na druhou stranu se budeme snažit teorii definic a vět doplňovat o názorné příklady a vysvětlení. Ty povedou k lepšímu porozumění kapitoly, která je pro samotné pochopení principu šifrování klíčová. Nejprve si zavedeme kongruenci modulo n a poté množinu zbytkových tříd $\mathbb{Z}_n$.

Definice 1. Kongruence modulo n. Ať $m > 1$ je pevné přirozené číslo. Řekneme, že celá čísla a a b jsou kongruentní modulo n, (značíme $a \equiv b \pmod{n}$), pokud existuje celé číslo k takové, že $a - b = k \cdot n$. Tedy kongruence modulo n je relací ekvivalence, která navíc respektuje operace sčítání a odčítaní, tj. pro všechna celá čísla a, b, c, d platí:

$$a \equiv b(\text{mod } n)) \wedge c \equiv d(\text{mod } n)) \Rightarrow a + c \equiv b + d(\text{mod } n))$$

$$a \equiv b(\text{mod } n)) \wedge c \equiv d(\text{mod } n)) \Rightarrow a \cdot c \equiv b \cdot d(\text{mod } n))$$

Definici můžeme také chápat tak, že čísla a a b jsou kongruentní právě tehdy, když obě čísla dávají stejný zbytek po dělení číslem n. Jako příklad si uvedeme tyto.

Příklad 1. Rovnice kongruence v $\mathbb{Z}_{13}$

$$4 \equiv 17(\text{mod } 13))$$

$$-3 \equiv 10(\text{mod } 13))$$

$$0 \equiv 26(\text{mod } 13))$$

Definice 2. Množina $\mathbb{Z}_n$. Existuje n různých tříd ekvivalence modulo n, jsou to třídy

odpovídající číslům $0, 1, \ldots, n-1$. Množinu všech zbytkových tříd značíme $\mathbb{Z}_n$ a máme

$$\mathbb{Z}_n = \{[0]_n, [1]_n, \ldots, [n-1]_n\}.$$

Příklad 2. Množina $\mathbb{Z}_{13}$.

$$\mathbb{Z}_{13} = \{[0]_{13}, [1]_{13}, [2]_{13}, [3]_{13}, [4]_{13}, \ldots, [11]_{13}, [12]_{13}\}.$$

Z definice je tedy jasné, že množina má tolik prvků, kolik existuje zbytků po dělení číslem n. Jako příklad jsme si uvedli množinu $\mathbb{Z}_{13}$, která má opravdu třináct zbytkových tříd. Nyní připomeneme definice neutrálního a inverzního prvku struktury.

Definice 3. Struktura s neutrálním prvkem. Struktura $(M,*)$ se nazývá struktura s neutrálním prvkem, právě když platí

$$(\exists x \in M)(\forall y \in M)(x * y = y \wedge y * x = y).$$

Každý prvek $x \in M$, pro nějž platí

$$(\forall y \in M)(x * y = y \wedge y * x = y).$$

se nazývá neutrální prvek struktury $(M,*)$. Tento prvek bývá často označován jako e.

Příklad 3. Obsahuje struktura $(\mathbb{Z}_5, +)$ neutrální prvek? Příklad názorně vyřešíme pomocí Cayleho tabulky, která znázorňuje všechny možné kombinace výpočtů prvků množiny s danou operací. Nyní hledáme takový prvek, který po aplikaci dané operace na ostatní prvky jejich hodnotu nezmění. Z tabulky vidíme, že je to prvek $[0]_5$.

$+$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$
$[0]_5$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$
$[1]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$	$[0]_5$
$[2]_5$	$[2]_5$	$[3]_5$	$[4]_5$	$[0]_5$	$[1]_5$
$[3]_5$	$[3]_5$	$[4]_5$	$[0]_5$	$[1]_5$	$[2]_5$
$[4]_5$	$[4]_5$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$

Definice 4. Struktura s inverzním prvkem. Struktura $(M,*)$ se nazývá struktura s

inverzními prvky, právě když má neutrální prvek e a když platí

$$(\forall x \in M)(\exists y \in M)(x * y = e \wedge y * x = e).$$

Je-li $x \in M$, nazývá se každý prvek y pro nějž platí

$$(x * y = e \wedge y * x = e).$$

inverzním prvkem k prvku x. Inverzní prvek k prvku x bývá často označován jako x^{-1}.

Příklad 4. Obsahuje struktura $(\mathbb{Z}_5, +)$ inverzní prvky? Příklad budeme řešit opět pomocí Cayleho tabulky. Z definice víme, že inverzní prvek y k prvku x je takový, pro který platí

$$(x * y = e \wedge y * x = e).$$

V naší tabulce tedy budeme pro každý řádek (prvek množiny) hledat takový sloupec (inverzní prvek), ve kterém se bude nacházet hodnota neutrálního prvku, tedy hodnota $[0]_5$.

$+$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$
$[0]_5$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$
$[1]_5$	$[1]_5$	$[2]_5$	$[3]_5$	$[4]_5$	$[0]_5$
$[2]_5$	$[2]_5$	$[3]_5$	$[4]_5$	$[0]_5$	$[1]_5$
$[3]_5$	$[3]_5$	$[4]_5$	$[0]_5$	$[1]_5$	$[2]_5$
$[4]_5$	$[4]_5$	$[0]_5$	$[1]_5$	$[2]_5$	$[3]_5$

Vypíšeme tedy jednotlivé dvojice prvků:

$$x = [0]_5; x^{-1} = [0]_5$$

$$x = [1]_5; x^{-1} = [4]_5$$

$$x = [2]_5; x^{-1} = [3]_5$$

$$x = [3]_5; x^{-1} = [2]_5$$

$$x = [4]_5; x^{-1} = [1]_5.$$

Dalším potřebným matematickým aparátem je největší společný dělitel a dále prvočísla, bez kterých se v dalším textu neobejdeme. V následujících definicích předpokládáme, že I je oborem integrity a $|$ je relace "dělí".

Definice 5. Společný dělitel. Jsou-li $a_1, a_2, \ldots, a_n$ libovolné prvky z I, nazývá se prvek $u \in I$ jejich společný dělitel, právě když $u|a_1, u|a_2, \ldots, u|a_n$.

Definice 6. Největší společný dělitel. Nechť $a_1, a_2, \ldots, a_n \in I$, Prvek $x \in I$ se nazývá největším společným dělitelem prvků $a_1, a_2, \ldots, a_n$ (značíme $x = NSD(a_1, a_2, \ldots, a_n)$), právě když platí současně

$$x|a_1 \wedge x|a_2 \wedge \ldots \wedge x|a_n \tag{1}$$

$$(\forall x_1 \in I)[(x_1|a_1 \wedge \ldots \wedge x_1|a_n) \Rightarrow x_1|x] \tag{2}$$

tedy x je největším společným dělitelem prvků $a_1, a_2, \ldots, a_n$, právě když je jejich společným dělitelem a když každý společný dělitel x_1 těchto prvků je dělitelem prvku x. Čísla, jejichž největším společným dělitelem je číslo 1 nazýváme čísla nesoudělná.

Příklad 5. Nechť $I = \mathbb{Z}$. Najděte $NSD(8, 20)$. Nejprve najdeme podle definice společné dělitele těchto čísel. Těmito čísly jsou čísla $1, -1, 2, -2, 4, -4$. Všimněme si, že definici vyhovují nejen kladná, ale také záporná čísla. Největšími společným dělitelem pak není pouze číslo 4, ale také číslo -4.

Definice 7. Prvočíslo. Prvočíslem nazýváme takové přirozené číslo $n \in \mathbb{N}$, které je beze zbytku dělitelné právě dvěma různými přirozenými čísly a to jedničkou a samo sebou.

Na základě této definice by mělo být jasné, že číslo 1 není prvočíslem a existuje pouze jedno sudé prvočíslo 2. Ostatní prvočísla musí být zákonitě lichá. Dále lze z definice odvodit, že pro každé prvočíslo $n \in \mathbb{N}$ a libovolné číslo $x \in \mathbb{N}, 0 < x < n$ je $NSD(n, x) = 1$.

Definice 8. Eulerova funkce. Pro přirozené číslo n definujeme přirozené číslo $\varphi(n)$ jako počet čísel nesoudělných s n, větších než 0 a menších než n. Funkci $\varphi : \mathbb{N} \to \mathbb{N}$ nazýváme Eulerova funkce.

Všimněme si, že pokud do Eulerovy funkce dosadíme jakékoli prvočíslo p bude výsledkem vždy číslo $p-1$. Tento fakt založen na definici prvočísel, kde jsme odvodili, že pro každé prvočíslo $n \in \mathbb{N}$ a libovolné číslo $x \in \mathbb{N}, 0 < x < n$ je $NSD(n,x) = 1$. Pro každé prvočíslo p tedy platí:

$$\varphi(p) = p - 1$$

Příklad 6. Vypočítejte Eulerovy funkce pro hodnoty 5, 8, 12.

$$\varphi(5) = 4 \qquad \textit{hodnoty} \quad [1,2,3,4]$$

$$\varphi(8) = 4 \qquad \textit{hodnoty} \quad [1,3,5,7]$$

$$\varphi(12) = 4 \qquad \textit{hodnoty} \quad [1,5,7,11]$$

Věta 1. (Bezoutova rovnost). Pro libovolná celá čísla a_1 a a_2 existuje jejich největší společný dělitel $NSD(a_1, a_2)$, přitom existují celá čísla k_1, k_2 tak, že

$$NSD(a_1, a_2) = k_1 \cdot a_1 + k_2 \cdot a_2. \tag{3}$$

Důkaz: Důkazem je právě rozšířený Euklidův algoritmus, který slouží k nalezení celých čísel k_1, k_2.

Rozšířený Euklidův algoritmus. Tato rozšířená verze Euklidova algoritmu umožňuje nalézt multiplikativní inverzi na tělese $\mathbb{Z}_p$, kde p je prvočíslo, za předpokladu, že $NSD(x,p) = 1$. Není nezbytně nutné, aby se jednalo o těleso, ale v takovém případě nemáme zaručeno, že inverze bude vůbec existovat. Principem algoritmu je rozepisování zbytků tak, aby byly vyjádřeny jako násobky čísel p a x. Pokračujeme tak dlouho, dokud se nedostaneme do situace

$$a \cdot x + b \cdot p = 1.$$

Zde totiž víme, že

$$b \cdot p = 0 \text{ v } \mathbb{Z}_p,$$

tudíž musí platit

$$a \cdot x = 1 \rightarrow a = x^{-1}.$$

Algoritmus si ukážeme na následujícím příkladě.

Příklad 7. Nalezněte multiplikativní inverzi čísla 40 v $\mathbb{Z}_{127}$. Výpočet algoritmu:

$$127 = 3 \cdot 40 + 7 \quad \rightarrow \quad 7 = 127 - 3 \cdot 40 \tag{4}$$

$$40 = 5 \cdot 7 + 5 \rightarrow 5 = 40 - 5 \cdot 7 \rightarrow 5 = 40 - (5 \cdot (127 - 3 \cdot 40)) \rightarrow 5 = 16 \cdot 40 - 5 \cdot 127 \tag{5}$$

$$7 = 1 \cdot 5 + 2 \rightarrow 2 = 7 - (1 \cdot 5) \rightarrow 2 = (127 - 3 \cdot 40) - (1 \cdot (16 \cdot 40 - 5 \cdot 127)) \rightarrow \\ \rightarrow 2 = 6 \cdot 127 - 19 \cdot 40 \tag{6}$$

$$5 = 2 \cdot 2 + 1 \rightarrow 1 = 5 - (2 \cdot 2) \rightarrow 1 = 16 \cdot 40 - 5 \cdot 127 - (2 \cdot (6 \cdot 127 - 19 \cdot 40)) \rightarrow \\ \rightarrow 1 = 54 \cdot 40 - 17 \cdot 127 \tag{7}$$

Popis jednotlivých kroků:

- Nejprve pomocí rovnice (4) rozepíšeme číslo 127 jako násobek čísla 40, označme si ho jako n a určíme zbytek z.
- V dalších rovnicích (5), (6) a (7) vždy rozepisujeme prvek n z minulého kroku zbytkem z z minulého kroku. Tedy v každém opakování tohoto kroku získáváme menší z a pokračujeme do té doby dokud $z \neq 1$.
- V rovnici (7) dostáváme zbytek 1 a tedy jsme u konce našeho algoritmu. Inverzním prvkem je prvek 54.

Dále si připomeneme z hlediska šifrování důležitou větu a to větu Eulerovu.

Věta 2. (Eulerova věta). Je-li $NSD(a,m) = 1$, potom platí

$$a^{\varphi(m)} = 1 \text{ v } \mathbb{Z}_m.$$

Vztah by se dal také napsat jako

$$a^{\varphi(m)} = 1 \bmod m.$$

Důkaz: Protože každé číslo od 0 do $m-1$, které je nesoudělné s m, má inverzi v $\mathbb{Z}_m$, je podle definice Eulerovy funkce v $\mathbb{Z}_m$ přesně $\varphi(m)$ různých invertibilních prvků. Označme je $b_1, b_2, \ldots, b_{\varphi(m)}$. Nejprve ukážeme, že jsou-li i a j dvě různá čísla z množiny $\{0, 1, \ldots, \varphi(m)\}$, potom

$$b_i \cdot a \neq b_j \cdot a \text{ v } \mathbb{Z}_m$$

Kdyby totiž nastala rovnost, znamenalo by to, že v $\mathbb{Z}_m$ platí rovnost $b_i = b_j$, protože předpokládáme, že $NSD(a, m) = 1$. To je spor. Posloupnosti

$$b_1 \cdot a, b_2 \cdot a, \ldots, b_{\varphi(m)} \cdot a \text{ a } b_1, b_2, \ldots, b_{\varphi(m)}$$

musí v $\mathbb{Z}_m$ obsahovat (až na pořadí) stejná čísla. Součiny všech členů obou posloupností v $\mathbb{Z}_m$ jsou si tedy v $\mathbb{Z}_m$ rovny. To znamená, že platí

$$a^{\varphi(m)} \cdot b_1 \cdot \ldots \cdot b_{\varphi(m)} = b_1 \cdot b_{\varphi(m)}$$

Protože součin $b_1 \cdot b_{\varphi(m)}$ má inverzi v $\mathbb{Z}_m$ platí

$$a^{\varphi(m)} = 1 \text{ v } \mathbb{Z}_m$$

a to jsme chtěli ukázat.

Příklad 8. Vypočítejte pomocí Eulerovy věty pro hodnoty $a = 8$, $n = 5$.

$$a = 8,\ n = 5,\ \varphi(5) = 4 \Rightarrow 8^4 = 1 \bmod 5 \Leftrightarrow 4096 = 1 \bmod 5$$

Pokud ve výše zmíněné větě položíme $n = p$, kde p je prvočíslo dostaneme s ohledem na výpočet Eulerovy funkce následující větu.

Věta 3. (Malá Fermatova věta). Ať p je prvočíslo. Potom pro libovolné celé číslo a nesoudělné s p platí

$$a^{p-1} = 1 \text{ v } \mathbb{Z}_p.$$

nebo

$$a^{p-1} = 1 \bmod p.$$

Důkaz: (Malá Fermatova věta) Nejprve ukážeme, že jsou-li i a j dvě různá čísla z množiny $\{0, 1, \ldots, (p-1)\}$, potom

$$i \cdot a \neq j \cdot a \text{ v } \mathbb{Z}_p.$$

Kdyby totiž nastala rovnost, znamenalo by to, že platí $p|a \cdot (i-j)$ a protože předpokládáme, že $NSD(a, p) = 1$, musí platit, že $p|(i-j)$. To je spor. Ukázali jsme tedy, že posloupnosti

$$1 \cdot a, 2 \cdot a, \ldots, (p-1) \cdot a \text{ a } 1, 2, \ldots (p-1)$$

musí v $\mathbb{Z}_p$ obsahovat (až na pořadí) stejná čísla. Součiny všech členů obou posloupností v $\mathbb{Z}_p$ jsou si tedy v $\mathbb{Z}_p$ rovny. To znamená ,že platí

$$a^{p-1} \cdot (p-1)! = (p-1)! \text{ v } \mathbb{Z}_p$$

Protože p je prvočíslo,je $NSD(p, (p-1)! = 1)$ a tudíž

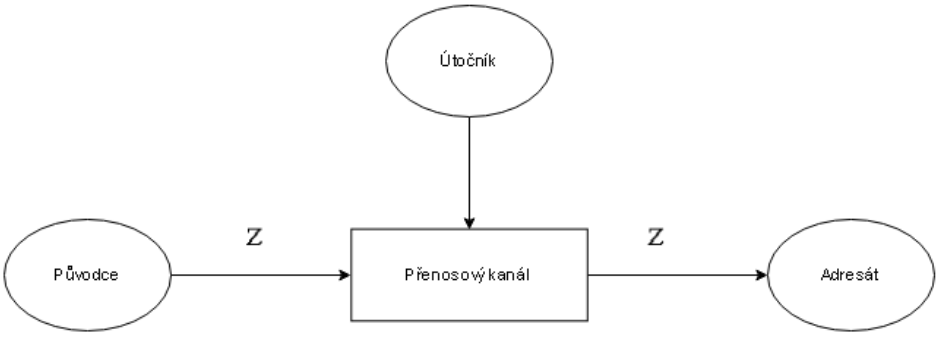
$$a^{p-1} = 1 \text{ v } \mathbb{Z}_p.$$

2 Základy kryptografie

V následujících podkapitolách si osvojíme základní pojmy v oblasti šifrování. Nahlédneme do historie vzniku šifrování a k čemu ho vlastně potřebujeme. Dále se zaměříme na jedno z nejznámějších asymetrických šifrování a to šifrování RSA. Vysvětlíme si princip tohoto šifrování a vyzkoušíme si jeho aplikaci na jednoduchých, ale i složitějších příkladech. Po přečtení této kapitoly bude čtenář schopen vysvětlit, co je to šifrování, popsat jak funguje RSA šifrování a dokáže zašifrovat jednoduché zprávy. Pojďme si tedy osvětlit pojmy, se kterými se můžeme v dané problematice setkat.

2.1 Základní pojmy

Lidé spolu komunikují již od dávných dob. Způsoby komunikace se v průběhu vývoje lidstva měnily a vyvíjely. Postupem času bylo potřeba, aby se potřebná informace dostala z jednoho místa A na místo druhé B, aniž by byla prozrazena či pozměněna. Ono místo A si označíme jako původce zprávy. Místo B označíme jako adresáta zprávy. Zprávou (někdy také kryptogramem) se rozumí posloupnost symbolů, ve které je nějakým způsobem "schována" původní informace. Ta může být ve formě písmen, obrazových bodů, spojitého signálu (hovor) a je tedy přetransformována do podoby zprávy. Transformace máme dvojího typu - kódování a šifrování. Velice často dochází k mylné představě, že se jedná o jeden a ten samý pojem. Ovšem není tomu tak. Kódování chápeme jako takovou transformaci, která nevyužívá žádné utajované informace. Jako příklad si můžeme uvést kódování v ASCII tabulce, která každému znaku přiřazuje číslo. V tomto procesu opravdu nehraje roli žádná utajená konstanta nebo funkce. Naopak při šifrování dochází k transformaci informace na zprávu a samozřejmě také i naopak, vždy s určitou tajnou konstantou nebo funkcí. Tuto tajnou konstantu v tomto procesu, který se také někdy označuje jako šifrovací algoritmus, budeme nazývat klíč. Nyní si můžeme na obrázku č. (1) ukázat schéma přenosu zprávy. Na obrázku se vyskytuje ještě třetí osoba, o které jsme se nezmínili. Jedná se o tzv. útočníka. Útočníkem se rozumí neoprávněná osoba, která se snaží buď odposlouchávat v přenosovém kanálu nebo dokonce obsah zpráv modifikovat. Obě tato chování jsou jak pro původce, tak adresáta nebezpečná a mohou



Obrázek č. 1: Schéma přenosu zpráv

způsobit velké škody. Představte si situaci, kdy by se útočníkovi podařilo odposlechnout Vaši komunikaci při platbě na internetu a byl poté schopen neomezeně disponovat s Vaší kartou. Kdyby dokázal modifikovat zprávy v přenosovém kanálu, mohl by se prakticky vydávat přímo za Vaši osobu a řídit Vaši e-mailovou komunikaci. Pokud je zpráva známá jen původci a adresátovi, budeme takovou zprávu označovat jako důvěrnou. Autentičnost zprávy je pak potvrzení předpokladu, že adresát dostal zprávu přímo od původce a ta nebyla nijak modifikována. Systémy, které zajišťují důvěrnost nebo autentičnost zprávy, se nazývají kryptosystémy. My si je přiblížíme v další kapitole. Problémem bezpečnosti zpráv (důvěrnost a autentičnost) se zabývá kryptografie. Naopak kryptoanalýza se zabývá překonáváním kryptografie. Spolu navzájem tvoří vědu nazývanou kryptologie.

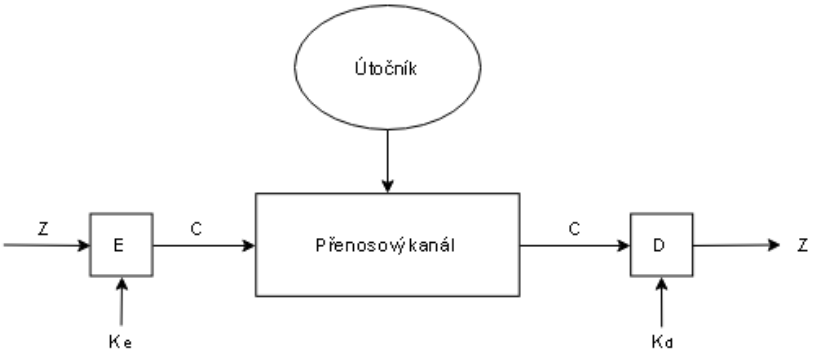
`Poznámka:` V dalším textu budeme zprávou označovat původní informaci a kryptogramem zašifrovanou zprávu.

2.2 Kryptosystémy

Rozlišujeme dva základní druhy kryptosystémů.

- Utajovací - zajišťují důvěrnost zprávy
- Autentizační - zajišťují autentičnost zprávy

Pak také existují tzv. *Hybridní* kryptosystémy, které zajišťují jak důvěrnost, tak autentičnost zprávy. Hybridním kryptosystémům se detailněji v této práci věnovat nebudeme. Vystačíme si s prvními dvěma kryptosystémy.



Obrázek č. 2: Schéma utajovacího kryptosystému

2.2.1 Utajovací kryptosystémy

Úkolem utajovacího kryptosystému je zajistit důvěrnost zprávy. Kryptosystém šifruje zprávu Z pomocí šifrovací funkce E. Tuto šifrovací funkci definujeme jako

$$E(Z, K_e),$$

kde K_e je šifrovacím klíčem. Tato funkce každé zprávě Z přiřadí kryptogram C, tedy platí

$$C = E(Z, K_e)$$

Kryptogram C je samozřejmě u adresáta potřeba dešifrovat pomocí dešifrovací (inverzní) funkce D, kterou definujeme takto

$$D(C, K_d),$$

kde K_d je dešifrovacím klíčem. Funkce každému kryptogramu C přiřadí původní zprávu Z takto

$$Z = D(C, K_d)$$

Výše zmíněnou teorii si můžeme prohlédnout na obrázku č. (2). Je potřeba, aby šifrovací funkce byla konstruována tak, aby se útočníkovi nepodařilo ji prolomit v požadovaném čase. Tímto časem je myšlen čas, kdy by útočník mohl provést škody ať už původci nebo adresátovi. Tento čas se může v různých případech použití lišit. Mohou to být řády dnů nebo také roků. Došli jsme tedy k závěru, že potřebujeme dvě funkce, které jsou závislé na klíčích K_e a K_d. Existuje mezi klíči nějaká závislost? Ano. Pokud totiž dokážeme v

požadovaném čase z K_e odvodit K_d je potřeba, aby oba klíče byly tajné. Pokud by byl K_e veřejný, utajovací kryptosystém by selhal, protože by útočník byl schopen odvodit dešifrovací klíč K_d a tím by dokázal šifrovat zprávy a dešifrovat kryptogramy. V tomto případě je tedy nutnou podmínkou, aby klíče byly utajeny. Z hlediska utajení pak klíče zastávají stejnou pozici a říkáme, že kryptosystém je symetrický. V opačném případě, kdy nelze odvodit K_d (soukromý) z K_e (veřejný), nazýváme kryptosystém asymetrický.

2.2.2 Autentizační kryptosystémy

Již víme, že autentičnost zprávy znamená pravdivost předpokladu adresáta o původci zprávy. Nyní si popíšeme kryptosystémy, které tuto vlastnost zajišťují. Celou situaci si přiblížme na praktickém příkladu autentizace listinných dokumentů. Jak víme, ty se opatřují tzv. pečetí. Pokud se nám dostane do ruky taková listina obsahující pečeť, jsme schopni na základě této pečeti jednoznačně určit původce zprávy (pokud nebyla původci tato pečeť odcizena). Stejně tak fungují i autentizační kryptosystémy. Na straně adresáta se vytvoří již zmíněná pečeť. Tu vytvoříme pomocí tzv. pečetící funkce Q, kterou můžeme definovat jako

$$Q(Z, K_p),$$

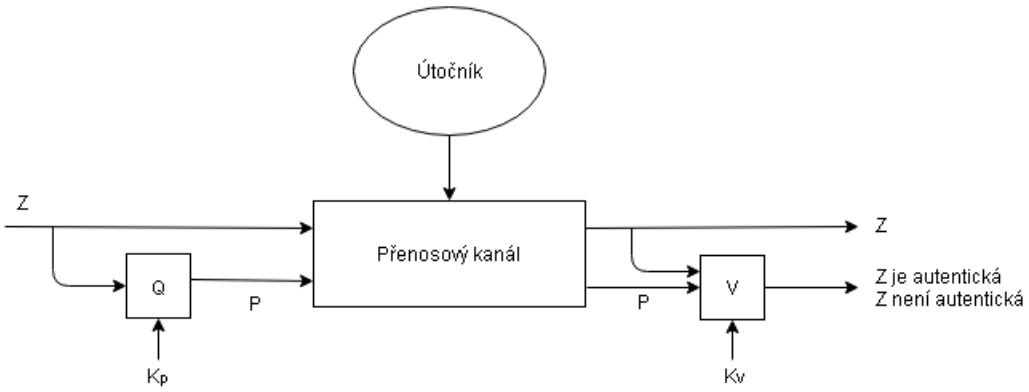
kde K_p je tajným pečetícím klíčem. Tato funkce každé zprávě Z přiřadí pečeť P, tedy platí

$$P = Q(Z, K_p).$$

Zpráva Z se společně s pečetí P pošle přenosovým kanálem adresátovi. Tato dvojice se označuje jako zapečetěná zpráva. Tuto zapečetěnou zprávu potřebuje adresát nejprve ověřit. To provede pomocí ověřovací funkce V, kterou definujeme následovně

$$V(Z, P, K_v) = \begin{cases} \text{Zpráva Z je autentická} \\ \text{Zpráva Z není autentická} \end{cases}$$

kde K_v je ověřovacím klíčem. Funkce nabývá logicky pouze dvou hodnot, které určují, zda zpráva je nebo není autentická. Opět si můžeme celý princip kryptosystému prohlédnout na obrázku č. (3). Tak jako u utajovacích, tak i u autentizačních kryptosystémů



Obrázek č. 3: Schéma autentizačního kryptosystému

se zaměříme na vztah mezi klíči. Pokud by někdo byl schopen z ověřovacího klíče K_v zjistit pečetící klíč K_p, musí být oba klíče tajné. Pokud by tak nebylo, mohl by adresát zjistit pečetící klíč a vytvářet zprávy, které by se tvářily jako vytvořené skutečným majitelem pečetícího klíče.V tomto případě klíče zastávají stejnou pozici z hlediska utajení a budeme proto hovořit o symetrických autentizačních kryptosystémech. Jako příklad si uveďme kryptosystém HMAC. Pokud v požadovaném čase nelze zjistit pečetící klíč, pak ověřovací klíč může být veřejný a budeme tyto autentizační kryptosystémy nazývat jako asymetrické. U těchto kryptosystémů nelze odvodit pečetící klíč a proto je vždy o původci zpráv jasno. Zde si jako příklad uvedeme techniku digitálního podepisování dokumentů.

2.2.3 Bezpečnost kryptosystémů

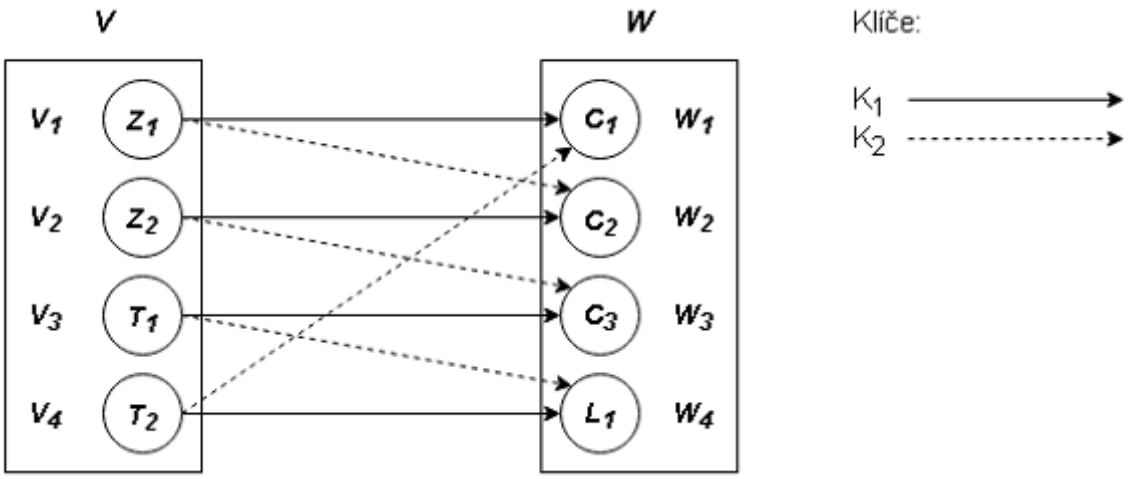
V kapitole 2.2.1 jsme si uvedli šifrovací funkci takto:

$$E(Z, K_e),$$

kde K_e je šifrovacím klíčem. Tato funkce každé zprávě Z přiřadí kryptogram C, tedy platí

$$C = E(Z, K_e)$$

Jedná se tedy o zobrazení z množiny vzorů $\boldsymbol{V}$ (zpráv) do množiny obrazů $\boldsymbol{W}$ (kryptogramů). Předpokládáme, že množina zpráv $\boldsymbol{V}$ může obsahovat smysluplné zprávy Z, ale také posloupnost symbolů s nulovým významem. Tyto zprávy s nulovým významem budeme označovat jako T. Prvky množiny $\boldsymbol{W}$, pro které existuje alespoň jedna smyslu-



Obrázek č. 4: Ukázka šifrovací funkce

plná zpráva Z a šifrovací klíč K_e, nazveme kryptogramem a budeme je značit C. Všechny ostatní prvky množiny $\boldsymbol{W}$ nazveme prázdným obrazem L, jelikož k nim neexistuje vzor Z. V tuto chvíli si můžeme šifrovací funkci zobrazit následovně. Jak se můžeme přesvědčit k obrazu L_1 opravdu existují pouze prázdné vzory T_1 a T_2. Všechny ostatní prvky množiny $\boldsymbol{W}$ mají alespoň jeden vzor Z. Na základě toho, kolik vzorů Z má daný obraz W_i (s použitím všech existujících klíčů), definujeme tzv. valenci Val.

Příklad 9. Vypište valenci všech prvků množiny obrazů $\boldsymbol{W}$ na obrázku č. 4

$$Val(C_1) = 1$$

$$Val(C_2) = 2$$

$$Val(C_3) = 1$$

$$Val(L_1) = 0$$

Kryptogram C_2 má valenci 2, protože mohl vzniknout zašifrováním zprávy Z_1 klíčem K_2 nebo zašifrováním zprávy Z_2 klíčem K_1.

Proč má vůbec tahle valence smysl? Odpověď je snadná. Pokud se útočníkovi podaří v přenosovém kanále odposlechnout kryptogram C_2 a pokusí se dešifrovat všemi možnými klíči (tzv. útok hrubou silou), získá dvě zprávy Z_1 a Z_2. Útočník nemá šanci zjistit, který

tajný klíč se použil při šifrování a nemůže tak s přesností určit původní zprávu. Čím větší je valence u jednotlivých kryptogramů, tím je větší bezpečnost celého systému. Valence může u kryptogramů nabývat hodnot 1 až minimum z dvojice N a K, kde N je počet zpráv (myšleno smysluplných zpráv Z) a K je počet klíčů daného systému. V předchozí větě byla záměrně uvedena velikost valence pouze pro kryptogramy, protože již víme, že valence prázdného obrazu L musí být zákonitě rovna nule. Pokud by se nerovnala nule, pak by se nejednalo o prázdný obraz, nýbrž o kryptogram. Pokud se na valenci podíváme z hlediska celého kryptosystému, budeme minimální valenci všech kryptogramů označovat jako M.

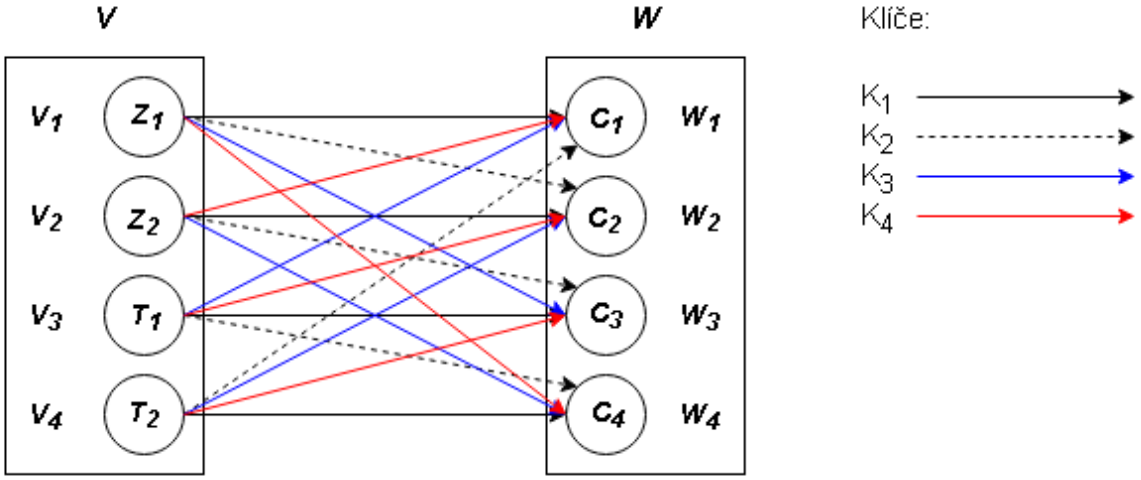
Příklad 10. Určete minimální valenci M na obrázku č. 4

$$\text{Min}\{Val(C_1), Val(C_2), Val(C_3)\} = \text{Min}\{1, 2, 1\} = 1$$

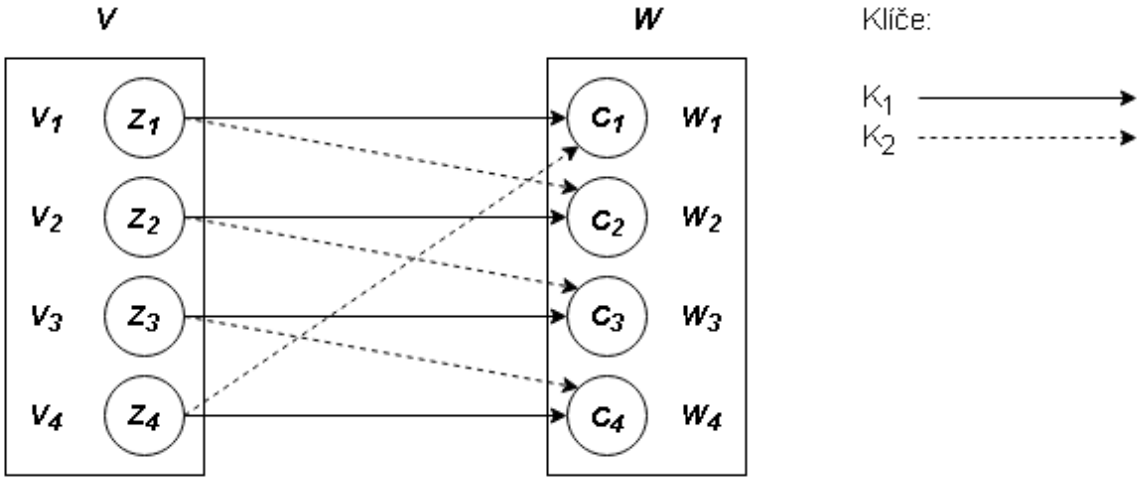
Důvěrnost kryptosystémů se hodnotí právě podle minimální valence. Toto kritérium se může jevit jako nespravedlivé, protože právě jeden slabší kryptogram může určit stupeň bezpečnosti celého kryptosystému. Je to dáno tím, že u tohoto kryptogramu má útočník největší šanci zjistit původní zprávu. Kryptosystémy z pohledu důvěrnosti dělíme následovně:

- maximální důvěrnost $M = N$ (viz. obrázek č. 5)
- redukovaná důvěrnost $1 < M < N$ (viz. obrázek č. 6)
- minimální důvěrnost $M = 1$ (viz. obrázek č. 7)

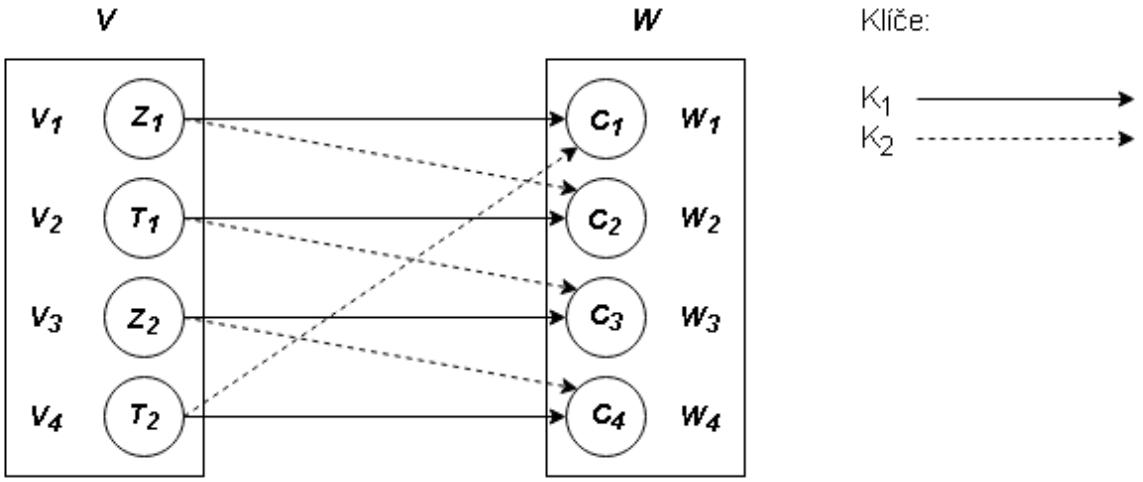
V praxi se nejvíce setkáváme s nejslabším typem co se důvěrnosti týče. Je to dáno tím, že počet všech možných klíčů, které by musel útočník vyzkoušet, je tak velký, že za dobu rezistence programu není schopen původní zprávu získat. Ostatní dva typy jsou náročné na správu klíčů a proto se v praxi prakticky nepoužívají. Pojďme se podívat na bezpečnost ještě jedním pohledem. Na kryptosystémy se můžeme dívat jako na neprolomitelné a prolomitelné. Za neprolomitelné pokládáme ty kryptosystémy, které útočník neprolomí



Obrázek č. 5: Kryptosystém s maximální důvěrností zpráv



Obrázek č. 6: Kryptosystém s redukovanou důvěrností zpráv



Obrázek č. 7: Kryptosystém s minimální důvěrností zpráv

ani s neomezenou výpočetní kapacitou. Sem bychom mohli zařadit kryptosystémy s maximální nebo redukovanou důvěrností. Prolomitelné systémy dále rozdělujeme na teoreticky a prakticky prolomitelné. Zde nám spadají systémy s minimální důvěrností. Teoreticky prolomitelné jsou pouze ty, pro které útočník potřebuje takovou výpočetní techniku, která je mimo reálné možnosti útočníka. Pokud útočník je schopen mít takovou výpočetní techniku, aby kryptosystém prolomil, jedná se o prakticky prolomitelný kryptosystém.

2.3 Historie kryptografie

První zmínky o šifrování na hliněné destičky jsou staré více jak 3500 let a pochází z Mezopotámie. O dalších 1000 let později se začaly objevovat různé substituční šifry. Jako příklad si můžeme uvést substituční šifru "atbaš". Ta prvnímu písmenu hebrejské abecedy přiřadila poslední písmeno, druhému předposlední a tak dále. V latince si můžeme tuto šifru znázornit následující převodní tabulkou.

a	b	c	...	x	y	z
Z	X	Y	...	C	B	A

Římané například používali Ceasarovu šifru, která každé písmenko zprávy zamění za jiné písmeno posunuté v abecedě o pevně daný počet míst. Ve středověku se kryptografie vyvíjela zejména v Evropě, a to metodou pokus-omyl. Největšího rozmachu kryptografie dosáhla teprve nedávno. V roce 1974 byl publikován první autentizační symetrický kryptosystém. V roce 1978 započala éra asymetrických kryptosystémů. V daném roce byl zveřejněn první asymetrický utajovací kryptosystém RSA a princip digitálního podpisu. RSA šifrování se budeme věnovat v další kapitole.

2.4 Asymetrické šifrování RSA

V minule kapitole jsme zmínili, že asymetrická šifra RSA byla zveřejněna v roce 1978 a odstartovala novou éru v oblasti kryptografie. Jistě se ptáte, jak tento název zkratky RSA vznikl. Určitě vás napadlo, že by poslední písmenko "A" mohlo mít spojitost s tím, že šifrovací systém je asymetrický. Není tomu tak. Název vznikl jako zkratka tří matematiků, kteří tento systém navrhli. Byli to pánové Rivest, Shamir, Adleman. Na základě

této šifry je založeno mnoho dnešních šifrovacích systémů a dá se říci, že je RSA takovým standardem v oblasti šifrování. Každý účastník, který využívá tuhle šifru, disponuje veřejným a soukromým klíčem. Víme, že je nemožné z veřejného klíče zjistit klíč soukromý. Mezi dvojicí klíčů existuje další spojitost. Všechny zprávy, které zašifrujeme veřejným klíčem, lze dešifrovat pouze příslušným soukromým klíčem a naopak. Jak zjistíme v další podkapitole, bezpečnost RSA je založena na volbě prvočísel p a q, sloužících pro vytvoření klíčů. Je totiž relativně jednoduché najít součin těchto prvočísel. Ovšem zjistit ze součinu samotná prvočísla p a q (faktorizace), je při dostatečně velkých p a q a v rozumném čase prakticky nemožné. Momentálně totiž není znám žádný algoritmus, který by tento problém efektivně řešil. V praxi se dnes používají takové klíče, které vytváří onen součin o velikosti 309 cifer a více. Pokud se tedy nenajde nový algoritmus pro řešení faktorizace, bude s rostoucím výkonem počítačů stačit zvyšovat velikost prvočísel.

2.4.1 Generování klíčů

Klíče generuje záměrně přímo adresát. Je tím totiž zajištěno, že soukromý klíč zná jen on. Postup si můžeme popsat následujícím algoritmem.

1. Adresát náhodně zvolí dvě velká prvočísla p a q.
2. Jejich vynásobením získá modulus $n = p \cdot q$.
3. Pomocí Eulerovy funkce vypočítá $\varphi(n) = (p-1) \cdot (q-1)$.
   (pokud vám výpočet tohoto bodu není jasný, vraťte se k definici Eulerovy funkce).
4. Dále zvolí veřejný exponent e pro který musí platit $1 < e < \varphi(n)$ a současně $D(e, \varphi(n)) = 1$.
5. pomocí Euklidova algoritmu vypočítá soukromý exponent d, pro který musí platit $d \cdot e \bmod \varphi(n) = 1$.
6. Dvojici $VK = (e, n)$ nazveme veřejným klíčem. Druhou dvojici $SK = (d, n)$ nazveme soukromým klíčem adresáta. Adresát zveřejní pouze veřejný klíč. Ostatní proměnné, které použil pro jejich vytvoření, nechá v tajnosti, mohly by totiž vést k odhalení soukromého klíče.

Uveďme si názornou ukázku tvorby klíčů. Pro jednoduchost použijeme malá prvočísla.

Příklad 11. Vytvořte dvojici klíčů pro prvočísla 7 a 11. Budeme postupovat přesně podle výše zmíněného algoritmu.

1. ze zadání máme určená prvočísla $p = 7$ a $q = 11$.

2. získáme modulus $n = p \cdot q = 7 \cdot 11 = 77$

3. $\varphi(n) = (p-1) \cdot (q-1) = 6 \cdot 10 = 60$

4. zvolíme exponent veřejného klíče $e = 13$

5. vypočítáme pomocí Euklidova algoritmu soukromý exponent d.

$$60 = 4 \cdot 13 + 8 \rightarrow 8 = 60 + (-4 \cdot 13)$$

$$13 = 1 \cdot 8 + 5 \rightarrow 5 = 13 - (1 \cdot 8) \rightarrow 5 = (-1 \cdot 60) + (5 \cdot 13)$$

$$8 = 1 \cdot 5 + 3 \rightarrow 3 = 8 - (1 \cdot 5) \rightarrow 3 = (2 \cdot 60) + (-9 \cdot 13)$$

$$5 = 1 \cdot 3 + 2 \rightarrow 2 = 5 - (1 \cdot 3) \rightarrow 2 = (-3 \cdot 60) + (14 \cdot 13)$$

$$3 = 1 \cdot 2 + 1 \rightarrow 1 = 3 - (1 \cdot 2) \rightarrow 1 = (5 \cdot 60) + (-23 \cdot 13)$$

Jelikož hledáme inverzní prvek v $\mathbb{Z}_{60}$, bude levá strana součtu poslední rovnice vždy rovna nule. Tedy dostaneme v $\mathbb{Z}_{60}$ následující rovnici

$$1 = (-23) \cdot 13 \bmod 60.$$

A je tedy jasné, že $d = (-23 + 60) = 37$ je hledaným inverzním prvkem.

6. Našli jsme tedy dvojici klíčů

$$VK = (e, n) = (13, 77)$$

$$SK = (d, n) = (37, 77).$$

Abychom si udělali obrázek, jak generování klíčů probíhá v praxi, uvedeme si další příklad.

Příklad 12. Vytvořte dvojici klíčů, aby vyhovovala použití v praxi.

1. určíme dostatečně velká prvočísla p a q.

$$p = 774196690813211106932703436330736974742561435635584587189767$$
$$467538305380320622220857229741217686043056139217455800374092$$
$$598119526553100754871637971794904570391695941600884305716749$$
$$604988340858129204579164537470194616440313953079206249473499$$
$$510535300861464863071981555907634664293926737095254285109732$$
$$72600609091$$

$$q = 497600993746759829337668454735094736764707883422813387791917$$
$$924959003937512095393006283634430113137460865380058626649130$$
$$748136562206438424438441319057545656720753583911355371087959$$
$$916381554744526108743097428672313605025423083821990536755928$$
$$252407886139918985672771168817937493408077283357953943012616$$
$$29479871213$$

`Poznámka:` O tom, že čísla p a q jsou opravdu prvočísla, se přesvědčíme v dalších kapitolách za pomoci počítačového systému algebry MAXIMA.

2. získáme modulus $n = p \cdot q$.

$$n = 385241042704106813033803611494574237566808028572472702330760$$
$$774666396956069708914696330602379701047498681649897202019075$$
$$142662676309567026083149293994946318504106686229401066560000$$
$$610499565901134547123603733487203792862131418132266914115633$$
$$220791373209618353693611496989279248927564728930035716007965$$
$$276616217877526761239735799704609354149415456460199004230605$$
$$372048308930699453855461933896706080551695246535052076065056$$
$$556428581929262811461139850935454439831634061527422264203572$$

842630681738309436650229370826260808941810905161847254031535
697308689074573146696686698077411442955392604271268821186535
294328030318803699738 3

3. $\varphi(n) = 385241042704106813033803611494574237566808028572472702330 7$
607746663969560697089146963306023797010474986816498972020190
751426626763095670260831492939949463185041066862294010665600
006104995659011345471236037334872037928621314181322669141156
332207913732096183536936114969892792489275647289300357160079
652766162178648087843941360903419056352387571393451263110400
213922984920767744807622836067205674454179387685432469060190
814121583496958002505735444591423536469231095592562977689484
504458626346430957376942028176930381892803858229471898836625
237294463947969437148266728595899639157081368826942487809820
032120468068285956517080

4. zvolíme exponent veřejného klíče $e = 65537$

5. hledáme exponent soukromého klíče d, aby platilo $d \cdot e = 1 \bmod \varphi(n)$.

$d = 333765451649284905382598670379509669485074993703480477262318$
946328913730650442836511553040314927834306958574258253057709
181079188257888150830846955353968780454753462076466615183435
840275956358490922466365869469207186149988890574028646161491
283954623660560143471981640890661393626609782392349786455471
999119106006741882873781943171237651538163638433499732072759
123162616233871782621762717084911112056251633091010563861077
598644293997704170584034276669710851462443517014397780198865
012375915819078932302010100990443694208467080076009606700918
524771751782511621341509147008837010145536295518248406002077
3361615223725172208673

6. Našli jsme tedy dvojici klíčů

$$VK = (e, n)$$

$$SK = (d, n).$$

2.4.2 Vyhodnocení RSA funkce

Teď, když víme, jak si účastníci šifrování vygenerují klíče, pojďme si načrtnout, jak šifrování funguje v praxi. Pro šifrování a dešifrování si zavedeme následující funkci:

$F(Z, K_e) = Z^e \bmod n$, kde Z je zpráva a $K_e = (e, n)$ je šifrovací klíč.

Zda půjde o šifrování nebo dešifrování bude rozhodovat v daném použití zpráva Z a použití klíče. Mějme tedy dvě osoby, které si pracovně označíme jako Petru a Martina. Každý z nich má svůj vlastní veřejný a soukromý klíč. Mohou nastat tyto situace:

1. Petra chce poslat soukromou zprávu Z Martinovi. Petra zašifruje zprávu pomocí Martinova veřejného klíče $VK_M = (e_M, n_M)$. Nyní je Martin jediný, kdo si může zprávu dešifrovat, protože jako jediný disponuje příslušným soukromým klíčem $SK_M = (d_M, n_M)$. Situaci můžeme matematizovat následovně:

   Petra zašifruje zprávu pomocí VK_M : $C_1 = F(Z, VK_M) = Z^{e_M} \bmod n_M$

   Martin zprávu dešifruje pomocí SK_M : $Z = F(C_1, SK_M) = C_1^{d_M} \bmod n_M$

2. Martin chce pomocí zprávy vydat nějaké veřejné prohlášení. Danou zprávu tedy zašifruje svým soukromým klíčem $SK_M = (d_M, n_M)$. Zpráva půjde dešifrovat pouze Martinovým veřejným klíčem $VK_M = (e_M, n_M)$ (tím bude ověřeno, že zprávu vydal opravdu Martin). Tento klíč je pro všechny volně dostupný a tak si zprávu bude moci přečíst kdokoli. Tento princip je použit v tzv. digitálním podpise. Zapsáno

matematicky:

Martin zašifruje zprávu pomocí SK_M : $C_1 = F(Z, SK_M) = Z^{d_M} \bmod n_M$

kdokoliv zprávu dešifruje pomocí VK_M : $Z = F(C_1, VK_M) = C_1^{e_M} \bmod n_M$

3. V první uvedené možnosti posílala Petra soukromou zprávu Martinovi. Martin si zprávu dokázal dešifrovat, ale nedokázal ověřit původce zprávy. Pokud tedy chce Petra poslat Martinovi soukromou zprávu a zaručit Martinovi, že je zpráva od ní, je potřeba zkombinovat dva předchozí příklady. Petra tedy zašifruje zprávu pomocí svého soukromého klíče $SK_P = (d_P, n_P)$, a poté ji zašifruje Martinovým veřejným klíčem $VK_M = (e_M, n_M)$. Teď bude Martin jediný schopný zprávu dešifrovat svým soukromým klíčem $SK_M = (d_M, n_M)$. A poté pomocí veřejného klíče Petry $VK_P = (e_P, n_P)$ ověří, že je zpráva od Petry. Matematicky:

Petra zašifruje zprávu klíčem SK_P: $C_1 = F(Z, SK_P) = Z^{d_P} \bmod n_P$

poté ji zašifruje klíčem VK_M: $C_2 = F(C_1, VK_M) = C_1^{e_M} \bmod n_M$

Martin dešifruje klíčem SK_M: $C_3 = F(C_2, SK_M) = C_2^{d_M} \bmod n_M$

nyní platí, že $C_1 = C_3$ a konečně

Martin dešifruje klíčem VK_P: $Z = F(C_3, VK_P) = C_3^{e_P} \bmod n_P$

Poslední uvedený scénář si vyzkoušíme na následujícím příkladě.

Příklad 13. Petra posílá zprávu Martinovi a chce zaručit Martinovi, že je zpráva opravdu od ní. V rámci jednoduchosti budeme za zprávu Z považovat číslo 2 a budeme používat následující klíče:

- Petra

$$VK_P = (7, 55)$$

$$SK_P = (23, 55)$$

- Martin

$$VK_M = (13, 77)$$

$$SK_M = (37, 77)$$

dále budeme postupovat tak, jak jsme si to uvedli výše:

Petra zašifruje zprávu klíčem SK_P: $C_1 = F(2, SK_P) = 2^{23} \bmod 55 = 8$,

poté ji zašifruje klíčem VK_M: $C_2 = F(8, VK_M) = 8^{13} \bmod 77 = 50$

Martin dešifruje klíčem SK_M: $C_3 = F(50, SK_M) = 50^{37} \bmod 77 = 8$

Martin dešifruje klíčem VK_P: $Z = F(8, VK_P) = 8^7 \bmod 55 = 2$

Martin tedy dostal zprávu, která je bez pochyby od Petry.

Opět čtenáře neochudíme o příklad z praxe. Ukážeme si tedy scénář číslo 2, kdy Martin chce vydat veřejné prohlášení.

Příklad 14. Martin chce pomocí zprávy vydat nějaké veřejné prohlášení. V rámci jednoduchosti budeme za zprávu Z považovat číslo 2 a budeme používat klíče, které jsme vytvořili v příkladě 12.

- Martin

$$VK_M = (e_M, n)$$

$$SK_M = (d_M, n)$$

kde

$$n = 385241042704106813033803611494574237566808028572472702330760$$
$$774666396956069708914696330602379701047498681649897202019075$$
$$142662676309567026083149293994946318504106686229401066560000$$
$$610499565901134547123603733487203792862131418132266914115633$$

220791373209618353693611496989279248927564728930035716007965
276616217877526761239735799704609354149415456460199004230605
372048308930699453855461933896706080551695246535052076065056
556428581929262811461139850935454439831634061527422264203572
842630681738309436650229370826260808941810905161847254031535
697308689074573146696686698077411442955392604271268821186535
2943280303188036997383

$e_M = 65537$

$d_M = 333765451649284905382598670379509669485074993703480477262318$
946328913730650442836511553040314927834306958574258253057709
181079188257888150830846955353968780454753462076466615183435
840275956358490922466365869469207186149988890574028646161491
283954623660560143471981640890661393626609782392349786455471
999119106006741882873781943171237651538163638433499732072759
123162616233871782621762717084911112056251633091010563861077
598644293997704170584034276669710851462443517014397780198865
012375915819078932302010100990443694208467080076009606700918
524771751782511621341509147008837010145536295518248406002077
3361615223725172208673

Budeme postupovat podle následujících kroků:

Martin zašifruje zprávu klíčem $SK_M : C_1 = F(2, SK_M) = 2^{d_M} \bmod n$

$C_1 = 289931361338498190652587876769864048024569835142088345605802$
966677504169405742454599771267406191817133616221563738904118
119124293969999846263651753991690616243038607506037936923609
975856712174892809102261906812464804275845990101680491181537
706190141619222080641656713462361455770426782661564350498968

613751912025887275552685363648168683799108499834467406290879

467606663279745553279494463085453157953527174107097674801224

716414265218594893856898885878512868059311392959475507841678

554977966285233317339650162049716445767975797963579941398627

195553271426010800732725020603038494988682622692947845819461

7529377415967667870926

kdokoliv zprávu dešifruje klíčem $VK_M : Z = F(C_1, VK_M) = C_1^{65537} \bmod n = 2$

Dostali jsme tedy opět původní zprávu Z.

`Poznámka:` V dalších kapitolách si správnost řešení ověříme s využitím programu MAXIMA.

http://mayor.fri.uniza.sk/krypto/09/rsa.pdf

2.4.3 Důkaz, že RSA funguje

Na začátku této kapitoly jsme uvedli následující tvrzení. *Všechny zprávy, které zašifrujeme veřejným klíčem, lze dešifrovat pouze příslušným soukromým klíčem a naopak.* Nyní si toto tvrzení napíšeme jako větu a následně si ji dokážeme.

Věta 4. Nechť $K_e = (e, n)$, $K_d = (d, n)$ jsou šifrovací klíče. Dále pro zprávu Z označíme $E(Z, K_e) = Z^e \bmod n$ jako šifrovací funkci a $D(Z, K_d) = Z^d \bmod n$ jako dešifrovací funkci. Pak platí, že zpráva Z, kterou zašifrujeme veřejným klíčem, lze dešifrovat pouze příslušným soukromým klíčem a naopak. Matematicky lze tuto větu vyjádřit takto:

$$D(E(Z, K_e), K_d) = Z = E(D(Z, K_d), K_e)$$

Nyní slibovaný důkaz věty.

Důkaz: Zpráva Z může být volena v tomto rozmezí $0 \leq Z < n$. Budeme se tedy zabývat následujícími třemi případy.

- $Z = 0$, pak

$$E(0, K_e) = 0^e \bmod n = 0$$
$$D(0, K_d) = 0^d \bmod n = 0$$

a věta je pro tento případ dokázána.

- $NSD(n, Z) = 1$, potom víme

$$e \cdot d = 1 \bmod \varphi(n) = 1 + k \cdot \varphi(n), k \in \mathbb{Z}$$

$$Z^{\varphi(n)} = 1 \bmod n$$

a tedy s využitím Eulerovy věty platí

$$D(E(Z, K_e), K_d) = D(Z^e \bmod n, K_d) = (Z^e)^d \bmod n = Z^{e \cdot d} \bmod n =$$

$$= Z^{1+k \cdot \varphi(n)} \bmod n = Z \cdot \left(Z^{\varphi(n)}\right)^k \bmod n = Z \bmod n$$

- $NSD(n, Z) \neq 1$. Potom buď p dělí Z nebo q dělí Z. Nemohou ovšem nastat oba případy, jelikož $Z \in (0, n)$. Bez újmy na všeobecnosti předpokládejme, že $Z = l \cdot p^s$, kde $s \leq 1$ a $NSD(l, n) = 1$, (s, $l \in \mathbb{N}$). Pak platí

$$D(E(Z, K_e), K_d) = Z^{e \cdot d} \bmod n = (l \cdot p^s)^{1+k \cdot \varphi(n)} \bmod n = l \cdot (p^{1+k \cdot \varphi(n)})^s \bmod n \tag{8}$$

Z malé Fermatovy věty víme, že pro prvočíslo q platí

$$p^{q-1} = 1 \bmod q.$$

Odtud

$$p^{(q-1)\cdot(p-1)} = 1 \pmod{q}$$

$$p^{k\cdot\varphi(n)} = 1 + a \cdot q, \text{ pro } a \geq 1$$

$$p^{k\cdot\varphi(n)+1} = p + a \cdot p \cdot q = p + a \cdot n$$

$$p^{k\cdot\varphi(n)+1} = 1 \pmod{n}.$$

Po dosazení do rovnice (3) dostaneme

$$D(d, E(e, Z)) = l \cdot p^s \bmod n = Z.$$

3 CAS Maxima

Jako první nás určitě napadne otázka: *"Co znamená zkratka CAS v názvu kapitoly?"*. CAS neboli Computer algebra system (systém počítačové algebry) je označením pro skupinu programů zabývající se manipulací symbolických nebo numerických výrazů. Takovýchto programů existuje celá řada. Některé programy jsou komerční (Maple, Matlab), některé naopak volně šiřitelné (Maxima). Teď trošku do historie. Jedním z prvních takových programů byl program Macsyma (MAC's SYmbolic MAnipulator). Ten byl vyvíjen od roku 1968 do roku 1982 v rámci projektu MAC (Mathematics and Computation) v MIT (Massachusetts Institute of Technology). Ve stejném roce se začal o jednu z verzí starat profesor William F. Shelter z Univerzity Texas. Tato verze dostala název Maxima. Profesor Shelter dostal v roce 1998 svolení uveřejnit zdrojové kódy této verze pod licencí GNU GPL. Tato licence spočívá v tom, že všechna odvozená díla musí být vydána pod stejnou licencí. Zdrojové kódy Maximy jsou veřejně dostupné tzv. "open source". Pokud je někdo využije, musí také zveřejnit zdrojové kódy svého programu. Právě z tohoto důvodu jsme si vybrali pro tuto práci program Maxima, je totiž volně dostupný a může ho využívat kdokoliv. Není potřeba investovat nemalé peníze za používání programu, což je zejména ve školství problém. Profesor se o vývoj programu staral až do své smrti v roce 2001. Po jeho smrti vývoj programu stále pokračuje a stará se o něj nezávislá skupina vývojářů a uživatelů. Ti vydávají nové verze programu jak pro Windows tak pro Linux, MacOS a Android. Týdně si v průměru Maximu stáhne necelých 6000 uživatelů. Je tedy jasné, že Maxima je opravdu hojně využívána. V dalším textu se zaměříme na instalaci ve všech možných operačních systémech a provedeme čtenáře prvním spuštěním programu.

3.1 Instalace a prohlídka programu

Program Maxima je dostupný na těchto operačních systémech.

- Windows
- Linux
- MacOS

- Android

Zdrojový kód a instalační soubory pro Windows, Linux a MacOS najdeme na webové stránce `https://sourceforge.net/projects/maxima/files/`. Zde si vybereme požadovanou verzi a stáhneme ji. V případě Androidu stačí aplikaci najít v Google Play pod názvem *"Maxima on Android"*.

3.1.1 Instalace programu - Windows

Na výše zmíněném odkazu stáhneme poslední verzi pro operační systém Windows. Aktuálně se jedná o verzi 5.39 ze dne 12. 12. 2016. Po stažení a spuštění staženého souboru se nám zobrazí průvodce instalací. Po odsouhlasení licenčních podmínek zvolíme název záložky v nabídce start a spustíme samotnou instalaci. Za zmínku stojí fakt, že se nás instalace nezeptá, kam chceme program nainstalovat. Automaticky jej nainstaluje do složky *C:\maxima-5.39.0*. Stisknutím tlačítka *"Dokončit"* ukončíme instalaci.

`Poznámka:` V dalších kapitolách se budeme věnovat programu Maxima nainstalovaném na operačním systému Windows.

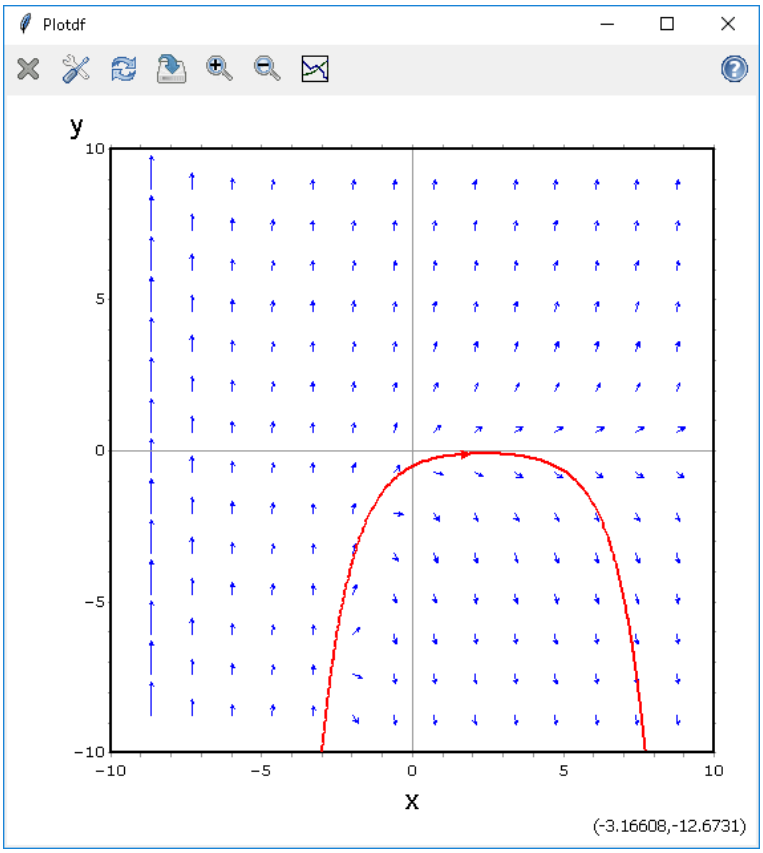
3.1.2 Instalace programu - MacOS

Stáhneme tedy poslední verzi pro MacOS. V textovém souboru "*How to install*" se dočteme pokyny instalace, které nebudou pro uživatelé zvyklé na tento operační systém problémem.

3.1.3 Instalace programu - Linux

Názvy souborů pro Linux jsou tvořeny dvěma částmi oddělenými pomlčkou. První je název a druhá je verze souboru. Pro minimální instalaci je potřeba nainstalovat .rpm soubory s názvem *"maxima-"* a *"maxima-exec-clisp-"*, které jsou následovány stejnou verzí. Jelikož jsou tyto dva soubory na sobě závislé, je potřeba je nainstalovat jedním společným příkazem:

$$rpm - ivhmaxima - x.y.z - n.i386.rpmmaxima - exec - clisp - x.y.z - n.i386.rpm$$



Obrázek č. 8: Příklad použití příkazu plotdf

nebo pro 64-bit rpm soubory

$$rpm - ivhmaxima - x.y.z - n.x86_64.rpmmaxima - exec - clisp - x.y.z - n.x86_64.rpm$$

kde *"x.y.z-n"* v příkazech je číslo verze. Nepovinné je nainstalovat balíček s názvem *"maxima-xmaxima-"*, který s sebou přináší grafické uživatelské rozhraní, které je pro některé příkazy Maximy vyžadováno (například příkaz plotdf, který zobrazuje směr pole na osách x a y viz obrázek č. 8)

3.1.4 Instalace programu - Android

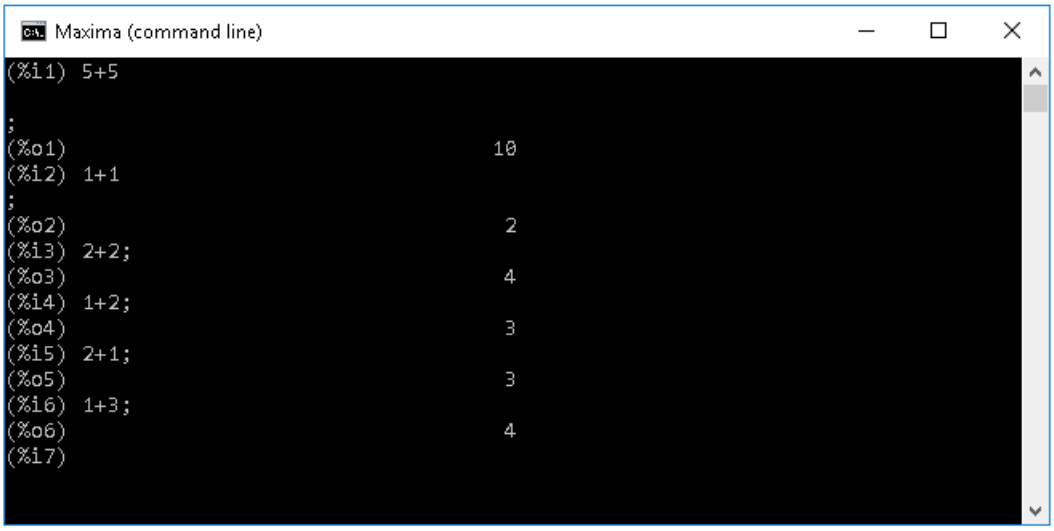
Pro instalaci na zařízení běžící pod operačním systémem Android je potřeba najít aplikaci s názvem *"Maxima on Android"* na Google Play a nainstalovat. Instalace vyžaduje Android 2.2 a vyšší, kterou disponuje naprostá většina chytrých telefonů a tabletů. Tuto aplikaci již nainstalovalo více než 50000 uživatelů. Aktuální verzí aplikace je verze 2.9, která vyšla 18. února 2017.

3.1.5 Průvodce prvním spuštěním

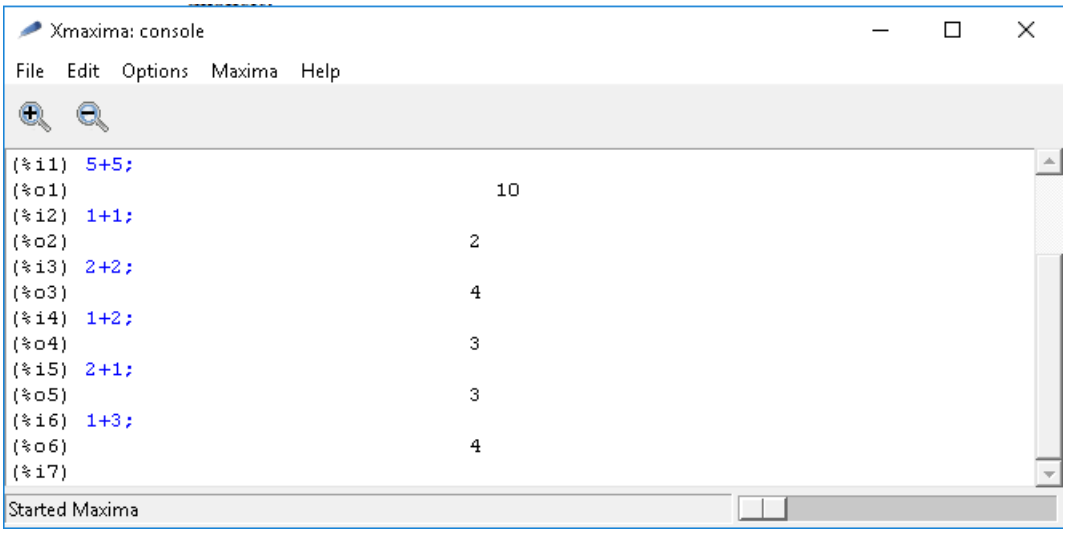
Program Maxima lze spustit dohromady ve třech různých variantách, které se liší různou mírou podpory grafického uživatelského rozhraní (zkráceně GUI z anglického Graphical User Interface). Máme k dispozici tyto režimy spuštění:

- Maxima (command line)
- XMaxima (simple GUI)
- wxMaxima (GUI for Maxima)

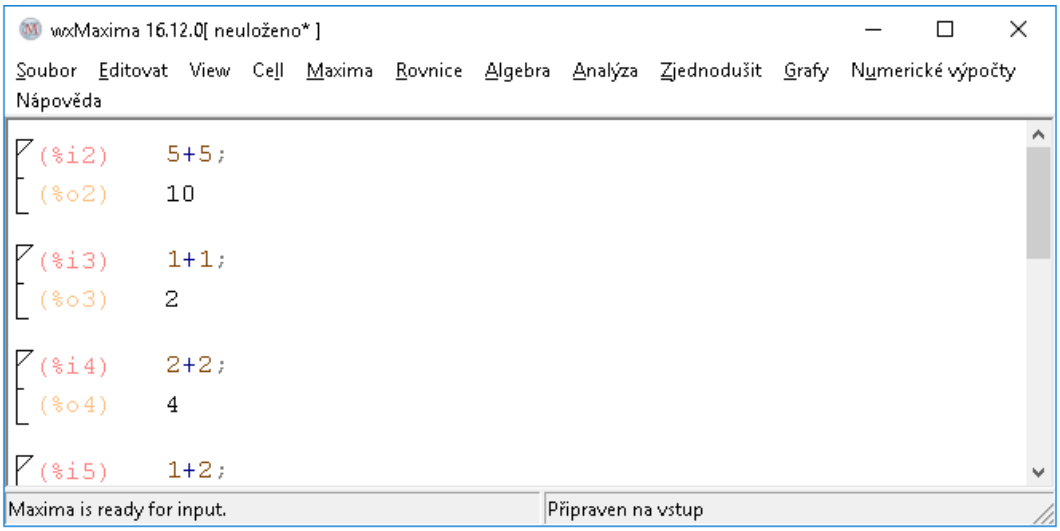
Nulovou podporu GUI představuje spuštění Maximy v příkazovém řádku. Jedná se stále o plnohodnotný nástroj, který je však do jisté míry omezen (například o funkce, které jako výstup vykreslují grafy). Další variantou je XMaxima, která je grafickou nadstavbou příkazové řádky, která navíc umožňuje vykreslování grafů. Poslední verzí je wxMaxima, která je vylepšena bohatým menu, díky kterému uživatel nemusí znát některé příkazy zpaměti. Jednoduše je pustí kliknutím na danou možnost v menu. To je zvláště pro začínající uživatele velmi přínosné. Zkušenější uživatelé ovšem častěji využívají přímo psaní příkazů do příkazové řádky. My se v dalším textu budeme věnovat pro uživatele nejpříznivějšímu prostředí wxMaxima.



Obrázek č. 9: Maxima (command line)



Obrázek č. 10: XMaxima (simple GUI)



Obrázek č. 11: wxMaxima (GUI for Maxima)

Příkazový řádek

Příkazový řádek vyplňuje skoro většinu prostoru pracovní plochy programu. Je to prostředník, díky kterému nám Maxima poskytuje obrovské možnosti početních operací, funkcí, výpočtů. Abychom mohli tyto výpočty realizovat, musíme se seznámit s pravidly psaní příkazů. Jednotlivé výrazy, které chceme vyhodnotit, musí být odděleny středníkem. Samotné vyhodnocení program provede na základě stisknutí kláves *Shift* + *Enter*. Při psaní výrazů není nutné psát každý výraz jako samostatný příkaz. Můžeme je oddělit středníkem a vyhodnotit zároveň. To si nyní ukážeme.

```
(%i1)  1;
```

(%o1) 1

```
(%i2)  1+2;
```

(%o2) 3

A nyní oba výrazy vyhodnotíme v jednom příkazu.

```
(%i4)  1;1+2;
```

(%o3) 1

(%o4) 3

Další možností je vytvořit víceřádkový příkaz pomocí klávesy *Enter*.

```
(%i6)  1;
       1+2;
```

(%o5) 1

(%o6) 3

Důležité je oddělit jednotlivé příkazy středníkem. Uživatel si může chování kláves pro odřádkování a vyhodnocování výrazů vyměnit. Tuto možnost lze nastavit v menu *Editovat* → *Nastavení* → zaškrtnutí možnosti *Enter vyhodnocuje výraz*. Můžeme si všimnout, že jednotlivé příkazy a výsledky jsou v programu označovány. Všechny vstupy (z anglického *input*) jsou označeny jako (%iX), kde X je číslo vstupu. A všechny výstupy (z anglického *output*) jsou označeny jako (%iY), kde Y je číslu výstupu. Výstupy jednotlivých vstupů

můžeme použít při psaní dalších příkazů. Chceme-li z nějakého důvodu potlačit zobrazení výsledku vyhodnocení, vložíme na konec příkazu místo středníku $. Pro zobrazení výsledku posledního vyhodnocení stačí použít znak %.

```
(%i7)  (%o1) + (%o2);
```

(%o7) 4

```
(%i8)  (%o1) + (%o2)$
```

```
(%i9)  %;
```

(%o9) 4

Poslední možností jak ovlivnit průběh vyhodnocení výrazu je použití apostrofu. Apostrof zajistí, že se daný objekt umístěný bezprostředně za ním nevyhodnotí a pouze se do výsledku přepíše nevyhodnocen.

```
(%i10) '(%o1) + (%o2);
```

(%o7) $\%o1 + 3$ Příkazový řádek disponuje našeptávačem funkcí. Tento našeptávač spustíme kombinací kláves $CTRL + K$.

Uložení, načtení, export a tisk souboru

V programu fungují klasické klávesové zkratky.

- $CTRL + N$ založení nového souboru
- $CTRL + S$ pro uložení souboru
- $CTRL + O$ pro otevření souboru
- $CTRL + P$ tisk souboru

Všechny tyto možnosti můžeme nalézt v záložce menu *"Soubor"*. Také tam najdeme možnost exportu. Program dokáže exportovat do souboru HTML, pdfLaTeX a do dávkového souboru ve formátu *.mac. K čemu jsou jednotlivé možnosti dobré? HTML se bude hodit, pokud budeme chtít výpočty prezentovat na webu. Možnost pdfLaTeX nám vytvoří soubor v jazyku TEX, který byl použit i při psaní této práce. Dávkový soubor nám uloží

příkazy daného souboru. Po načtení tohoto dávkového souboru se všechny příkazy načtou a vyhodnotí. Můžeme si tedy vytvořit dávkový soubor obsahující vlastní funkce a ty pak načíst do dalších souborů. Možnost načtení dávkového souboru je opět ve výše zmíněné záložce menu.

Nápověda

Zejména začínající uživatele programu bude zajímat nápověda programu. Nápovědu můžeme vyvolat pomocí záložky *"Nápověda"* v menu. Program obsahuje dvě nápovědy. První je nápověda programu wxMaxima (možnost *"wxMaxima Help"* v dané záložce), obsahující nápovědu k uživatelskému rozhraní, seznam nových funkcí v této verzi. Nás však bude zajímat druhá nápověda (*"Maxima help"*). Tato nápověda obsahuje rozsáhlou dokumentaci programu a jeho funkcí. Bohužel ani tato verze neobsahuje českou verzi nápovědy, což je trošku škoda. Jsme tedy opět závislí na úrovni naší angličtiny. Věřím však, že tato překážka uživatele neodradí.

Ukončení programu

Pro ukončení programu stačí zmáčknout kombinaci kláves $CTRL + Q$. Tuto možnost také najdeme v záložce *"Soubor"* menu. V případě neuložené práce se program před ukončením zeptá, zda chcete svou práci uložit.

3.2 Základní uživatelská výbava pro konstrukci RSA

Předmětem této práce není čtenáře naučit všechny funkce programu. Naším cílem je naučit čtenáře vše potřebné, aby mohl v Maximě šifrovat pomocí RSA. Proto se v dalším textu omezíme pouze na to, co autor pokládá za nejnutnější pro samotné šifrování. Bude se jednat o standardní operace, přiřazování hodnot a funkcí, možnosti zobrazení celých čísel a výčet jednotlivých funkcí, které se v průběhu šifrování mohou hodit.

3.2.1 Standardní operace

Klasické binární operace *plus*, *mínus*, *krát*, *děleno* zastupují znaky +, -, *, /. N-tou mocninu čísla x můžeme zapsat pomocí stříšky nebo **. Odmocninu zapisujeme formou mocniny. v případě druhé odmocniny můžeme využít funkce *sqrt()*, které předáme jako argument číslo, které chceme odmocnit. Tak jako na papíře, i zde je potřeba dodržovat priority jednotlivých operací a závorek. Při chvilce nepozornosti totiž může dojít k úplně jinému výsledku, než jsme čekali.

```
(%i11) 2**2+1;
```

(%o11) 5

```
(%i12) 2**(2+1);
```

(%o12) 8

3.2.2 Přiřazení hodnot výrazů a funkcí

Přiřazení hodnot a výrazů provádíme pomocí dvojtečky. Výraz může obsahovat další proměnné. Vše si ukážeme na následujícím příkladu.

```
(%i13) a: 5;
       b: a+2;
```

(a) 5

(b) 7

Pro vytváření funkcí používáme ":=". vytvořme tedy funkci $g(x, y)$.

```
(%i14) g(x,y) := x + y;
```

(%o14) $g(x, y) := x + y$

Nyní zavoláme vytvořenou funkci s argumenty a a b, které jsme vytvořili v předešlé ukázce.

```
(%i15) g(a,b);
```

(%o15) 12

3.2.3 Zobrazení maximálního počtu cifer

Program Maxima umožňuje v základním nastavení celé zobrazení maximálně 100 ciferných čísel. Pro větší čísla používá zkrácený zápis. Tento zápis může vypadat takto.

```
(%i16) 12345678901234567890123456789012345678901234567890112345678901234
       567890123456789012345678901234567890123456789012345678901;
```

(%o16) 123456789012345678901234567890[42*digits*]23456789012345678901234567890 1

Tuto maximální hodnotu zobrazených cifer můžeme změnit. Možnost najdeme přes záložku *"Editovat"* → *"Nastavení"* → *"Maximum zobrazených cifer"*. Zvýšíme hodnotu na číslo 150. A zkusíme opět vyhodnotit číslo z příkladu (%i16).

```
(%i17) (%i16);
```

(%o17) 123456789012345678901234567890123456789012345678901123456789012345678
901234567890123456789012345678901

Pro další výpočty v tomto textu zůstaneme s ohledem na úsporu místa na původní maximálně hodnotě 100 cifer.

3.2.4 Funkce

V této podkapitole si postupně představíme všechny funkce, které by se čtenáři při šifrování mohly hodit. Pokud to půjde, budeme funkce demonstrovat na příkladech z kapitoly č. 1.

Modulo

Funkce ***mod(x,n)*** vypočítává modulo n z čísla x.

```
(%i18) mod(12,5);
```

(%o18) 2

Největší společný dělitel

Pro výpočet největšího společného dělitele čísel a a b využijeme funkci ***gcd(a,b)***.

```
(%i19) gcd(8,20);
```

(%o19) 4 Všimněme si, že funkce vrací pouze kladná čísla. Nevrací nám tedy hodnotu (-4), která je podle definice také největším společným dělitelem.

Prvočísla

Z pohledu prvočísel nás budou zajímat následující funkce:

- primep(n) - vrací hodnotu *true* v případě, že n je prvočíslo. Jinak *false*.
- next_prime(n) - vrací nejmenší prvočíslo větší než n.

```
(%i22) primep(4);
       primep(5);
       next_prime(5);
```

(%o20) $false$

(%o21) $true$

(%o22) 7

V ukázce jsme ověřili, že číslo 4 není na rozdíl od čísla 5 prvočíslem. Dále jsme našli číslo 7 jakožto nejmenší prvočíslo větší než číslo 5.

Eulerova funkce

Výpočet hodnoty Eulerovy funkce provedeme pomocí funkce ***totient(n)***.

```
(%i25) totient(5);
       totient(8);
       totient(12);
```

(%o23) 4

(%o24) 4

(%o25) 4

Rozšířený Euklidův algoritmus

Využijeme funkci ***inv_mod(x,p)***, kde x je číslo ke kterému hledáme inverzi a p je modulo, ve kterém inverzi počítáme. Složitý výpočet z příkladu č. 7 nyní vypočítáme jednořádkovým příkazem.

```
(%i26) inv_mod(40,127);
```

(%o26) 54

Umocnění v modulu

Pro potřeby šifrování budeme často umocňovat pomocí obrovských klíčů v určitém modulu. V programu Maxima budeme využívat funkci ***power_mod(a,n,m)***, která vypočítá rovnici:

$$x = a^n \bmod m$$

```
(%i27) power_mod(8,4,5);
```

(%o27) 1

3.3 Ukázka RSA šifrování v programu Maxima

V kapitole č. 2.4 jsme si ukázali generování klíčů a samotné šifrování RSA. Nyní, když máme potřebné znalosti programu Maxima, ověříme si, že jsme v příkladech č. 12 a č. 14 počítali správně.

Příklad 15. Ověřte pomocí programu Maxima vytvoření klíčů z příkladu č. 12.

Nejprve založíme proměnné p a q.

```
(%i29) p:774196690813211106932703436330736974742561435635584587189767467538305380320622220857229741217686043056139217455800374092598119526553100754871637971794904570391695941600884305716749604988340858129204579164537470194616440313953079206249473499510535300861464863071981555907634664293926737095254285109732726006090 91;
       q:497600993746759829337668454735094736764707883422813387791917924959003937512095393006283634430113137460865380058626649130748136562206438424438441319057545656720753583911355371087959916381554744526108743097428672313605025423083821990536755928252407886139918985672771168817937493408077283357953943012616294798712 13;
```

(p) 774196690813211106932703436330[251*digits*]673709525428510973272600609091

(q) 497600993746759829337668454735[251*digits*]728335795394301261629479871213

Dále ověříme, zda zvolená čísla p a q jsou prvočísla.

```
(%i31) primep(p);
       primep(q);
```

(%o30) *true*

(%o31) *true*

Získáme modulus $n = p \cdot q$.

```
(%i32) n:p*q;
```

(n) 385241042704106813033803611494[562*digits*]211865352943280303188036997383

Vypočítáme hodnotu Eulerovy funkce $\varphi(n) = (p-1) * (q-1)$, kterou si v programu uložíme do proměnné r.

```
(%i33) r: (p-1)*(q-1);
```

(r) 385241042704106813033803611494[562*digits*]809820032120468068285956517080

Zvolíme veřejný exponent $e = 65537$.

```
(%i34) e: 65537;
```

(e) 65537

Hledáme exponent soukromého klíče d, aby platilo $d \cdot e = 1 \bmod \varphi(n)$.

```
(%i35) d: inv_mod(e,r);
```

(d) 333765451649284905382598670379[562*digits*]060020773361615223725172208673

Tímto jsme našli dvojici klíčů

$$VK = (e, n)$$

$$SK = (d, n).$$

, kde

```
(%i38) e;
       d;
       n;
```

(%o36) 65537

(%o37) 333765451649284905382598670379[562*digits*]060020773361615223725172208673

(%o38) 385241042704106813033803611494[562*digits*]211865352943280303188036997383

Příklad 16. Ověřte správnost vyhodnocení RSA funkce z příkladu č. 14.

Zadání: *Martin chce pomocí zprávy vydat nějaké veřejné prohlášení. V rámci jednoduchosti budeme za zprávu Z považovat číslo 2 a budeme používat klíče, které jsme vytvořili v příkladě č. 15.* Nejprve založíme zprávu $Z = 2$.

```
(%i39) Z: 2;
```

(Z) 2

Martin zašifruje zprávu klíčem $SK_M : C_1 = F(2, SK_M) = 2^{d_M} \bmod n$

Vypočítáme tedy proměnnou $C1$ pomocí funkce ***power_mod***

```
(%i40) C1: power_mod(Z,d,n);
```

(C1) $289931361338498190652587876769[562 digits]458194617529377415967667870926$

A vrhneme se na dešifrování.

kdokoliv zprávu dešifruje pomocí $VK_M : Z = F(C_1, VK_M) = C_1^{e_M} \bmod n$

```
(%i41) power_mod(C1,e,n);
```

(C1) 2

Opět jsme tedy získali původní zprávu Z. Ověřili jsme tedy správnost výpočtu.

3.4 Komplexní šifrování v programu Maxima

V této kapitole se pokusíme vytvořit komplexní knihovnu pro šifrování. Tato knihovna bude obsahovat funkce pro generování náhodných klíčů, šifrování a dešifrování. Při tvorbě funkce pro generování náhodných klíčů jsme čerpali z [4]. Při vytváření knihovny, ale budeme vytvářet i nové vlastní funkce. Vše se budeme snažit popisovat tak, aby čtenář na každém řádku věděl, co daná funkce provádí. Čtenáře tedy nebudeme učit programovat, ale nabídneme mu detailní popis těchto funkcí a samozřejmě možnost je využít. Pro správné fungování následujících funkcí je potřeba načíst knihovnu *"distrib"*, která nám umožňuje používat funkce pro získání náhodného čísla z daného intervalu. Knihovnu načteme následujícím příkazem **Load()**.

```
(%i42) Load(distrib);
```

(%o42) $C:\maxima-5.39.0\share\maxima\5.39.0\_2\_g5a49f11\_dirty\share\distrib\distrib.mac$

3.4.1 Funkce pro generování náhodných klíčů

Jako první tedy vytvoříme funkci pro generování náhodných klíčů, kterou nazveme ***generateRandomKey***. Při generování klíčů je z hlediska bezpečnosti zásadní jejich velikost. Abychom mohli tuto velikost dynamicky ovlivňovat, bude naše funkce mít jeden vstupní parametr. Díky tomuto parametru (označme si ho jako l) budeme náhodně hledat prvočísla v okolí čísel intervalu $(2^l, 2^{l+1})$. Těm čtenářům, kteří s programováním nemají žádnou zkušenost, se může kód zdát nepřehledný a složitý. I proto si kód popíšeme řádek po řádku, abychom věděli co funkce přesně v jednotlivých krocích dělá. Pro pozdější komfort při popisu začneme se seznámením dosud neznámých funkcí, které se v kódu vyskytnou:

Datové typy

Tak jako všechny ostatní programovací jazyky, Maxima disponuje určitými datovými typy. Pro potřeby některých funkcí je potřeba předávat proměnné různého datového typu.

Některá funkce očekává řetězec znaků, jiná například číslo. A právě funkce ***random()*** očekává číslo, které je navíc datového typu **float**. Nebudeme tuto problematiku dále rozebírat a spokojíme se s myšlenkou, že některé funkce potřebují bezpodmínečně na vstupu tento datový typ. Proto dané parametry funkcí budeme tzv. ***"přetypovávat"*** funkcí ***float(x)***, kde x je číslo, které chceme převést na float. V příkladu si ukážeme různé způsoby interpretace čísla $\frac{1}{2}$.

```
(%i45) rationalize(1/2);
       bfloat(1/2);
       float(1/2);
```

(%o43) $\frac{1}{2}$

(%o44) $5.0b-1$

(%o45) 0.5

Funkce ***rationalize()*** nám vrací hodnotu ve tvaru zlomku. Funkce ***bfloat()*** vrací hodnotu ve tvaru xby, který bychom mohli matematicky přepsat jako $x \cdot 10^y$. V našem případě tedy na $5 \cdot 10^{-1} = \frac{1}{2}$. Funkce ***float()*** vrací hodnotu ve tvaru desetinného čísla.

Náhodné přirozené číslo

Pro získání náhodného přirozeného čísla z intervalu $\langle 0, x)$, použijeme funkci ***random(x)***, kde x je krajní bod intervalu.

```
(%i46) random(4);
```

(%o46) 3

Náhodná hodnota z intervalu

Pro získání náhodného čísla z intervalu použijeme funkci, která je součástí knihovny "distrib". Funkce se nazývá ***random_continuous_uniform(a,b)***, kde a a b jsou krajní body intervalu (datového typu float) a musí platit podmínka $a < b$

```
(%i47) random_continuous_uniform(4,8);
```

(%o47) 6.772068727764502

Berme na vědomí, že získáváme **náhodnou** hodnotu z intervalu, tudíž při každém vyhodnocení příkazu bude výsledek s největší pravděpodobností odlišný.

Dolní celá část

Funkce ***floor(x)*** vrací hodnotu dolní celé části čísla x.

```
(%i49) random_continuous_uniform(4,8);
       floor(%)
```

(%o48) 5.853820661579186

(%o49) 5

Nyní již známe vše potřebné a můžeme si náš kód prohlédnout a popsat.

```
1  generateRandomKey(l):=
2  block([p,q,n,r,e,d,K],
3      p:next_prime(floor(float(random_continuous_uniform(2**l
           ,2**(l+1))))),
4      q:next_prime(floor(float(random_continuous_uniform(2**l
           ,2**(l+1))))),
5      n:p*q,
6      r:(p-1)*(q-1),
7      e:next_prime(floor(float(random(floor(r/2))))),
8      d:inv_mod(e,r),
9      K:[e,n,d],
10     K
11 );
```

Zdrojový kód č. 1: Generování náhodných klíčů RSA

1. Pojmenování funkce a určení počtu parametrů pro následné volání funkce. V našem případě se bude funkce volat s jedním číselným parametrem l, což si ukážeme také v dalším textu.

2. Založení "výpočetního" bloku. Tento blok si představme jako prostor, ve kterém si vytvoříme potřebné proměnné a provedeme dané výpočty. Založili jsme tedy pro-

měnné p, q, n, r, e, d, K.

3. Nyní budeme volit prvočíslo p. Daný řádek budeme vysvětlovat od konce. Nejdříve pomocí funkce ***random_continuous_uniform(2**l,2**(l+1))*** vytvoříme náhodné číslo z intervalu $(2^l, 2^{l+1})$. Následně převedeme na datový typ float a získáme dolní celou část pomocí ***floor()***. Nakonec použijeme funkci ***next_prime()***, které nám najde nejmenší prvočíslo větší než číslo předané jako parametr.

4. Stejně jako v minulém kroku získáme prvočíslo q.

5. Spočítáme modulus $n = p * q$.

6. Spočítáme hodnotu Eulerovy funkce a uložíme do r.

7. Určíme veřejný klíč e. Nejdříve získáme náhodné přirozené číslo v intervalu $\langle 0, r/2)$ a poté najdeme nejbližší prvočíslo pomocí ***next_prime()***.

8. Pomocí funkce ***inv_mod(e,n)***, najdeme inverzní prvek k e v modulu n.

9. Vytvoříme proměnnou K, do které si uložíme hodnoty e, n, d.

10. Vrátíme K jako výsledek vyhodnocení funkce

11. Ukončení bloku.

Nejprve vyhodnotíme definici funkce a poté ji můžeme začít volat.

```
(%i50) generateRandomKey(l):=
       block([p,q,n,r,e,d,K],
       p:next_prime(floor(float(random_continuous_uniform(2^l,2^(l+1))))),
       q:next_prime(floor(float(random_continuous_uniform(2^l,2^(l+1))))),
       n:p*q,
       r:(p-1)*(q-1),
       e:next_prime(floor(float(random(floor(r/2))))),
       d:inv_mod(e,r),
       K:[e,n,d],
       K
       );
```

(%o50) $generateRandomKey(l) := block([p, q, n, r, e, d, K], p : next\_prime(floor(float($
$random\_continuous\_uniform(2^l, 2^(l+1))))), q : next\_prime(floor(float($
$random\_continuous\_uniform(2^l, 2^(l+1))))), n : p * q, r : (p-1) * (q-1),$
$e : next\_prime(floor(float(random(floor(r/2))))), d : inv\_mod(e, r), K : [e, n, d]$
$, K)$

```
(%i51) key: generateRandomKey(3);
```

(%o51) $[53, 169, 125]$

Z hlediska zvolení kompromisu mezi bezpečností a rychlosti výpočtu je v [4] doporučeno funkci volat s parametrem $l = 125$. Pak mohou klíče vypadat následovně:

```
(%i52) key: generateRandomKey(125);
```

(%o52) $[213722[64digits]512781, 446782[64digits]967777, 125243[64digits]392161]$

Jednotlivé části našeho vytvořeného klíče si uložíme do samostatných proměnných.

```
(%i55) e: key[1];
       n: key[2];
       d: key[3];
```

(e) $213722[64digits]512781$

(n) $446782[64digits]967777$

(d) $125243[64digits]392161$

3.4.2 Funkce pro šifrování

Prozatím jsme se setkali se šifrováním čísel. My ovšem chceme šifrovat textové zprávy. Je tedy potřeba text interpretovat jako seznam čísel, které poté budeme šifrovat. Z těchto zašifrovaných čísel stvoříme dlouhý textový řetězec a ten zakódujeme v kódování **base64**.

Převod textu na seznam čísel a zpět

Pro potřeby převodu textu na čísla využijeme funkci ***string_to_octets(s)***, která jednotlivé znaky řetězce s, převede na čísla. Převod může vypadat takto:

```
(%i56) string_to_octets("test");
```

(%o56) $[116, 101, 115, 116]$

Pro opačný převod na text využijeme funkci ***octets_to_string(o)***, která seznam čísel o převede zpátky na textový řetězec.

```
(%i57) octets_to_string(%o56);
```

(%o57) *test*

Délka seznamu

Délku seznamu zjistíme pomocí funkce ***length(seznam)***.

```
(%i58) length(%o56);
```

(%o58) 4

Spojení řetězce

Pro spojení libovolného počtu řetězců použijeme funkci ***sconcat(s1,s2,...,sn)***

```
(%i59) sconcat("t","e","s","t");
```

(%o59) *test*

Kódování a dekódování base64

Funkce ***base64*** slouží pro zakódování řetězce, seznamu, čísla. Naopak funkce ***base64_decode*** opět dekóduje. Uveďme si názorný příklad.

```
(%i60) base64("test");
```

(%o60) $dGVzdA==$

```
(%i61) base64_decode(%);
```

(%o61) *test*

Můžeme tedy přistoupit ke zdrojovému kódu pro šifrování textu. Funkci nazveme ***encrypt(text,e,n)***, která bude na vstupu brát text, který budeme šifrovat, exponent klíče e a modulus n. Funkce vypadá následovně.

```
1  encrypt(text,e,n) :=
2  block([list,i],
3      list:string_to_octets(text),
4      returnvalue: "",
5      for i:1 thru length(list) step 1 do
6      block([value],
7          value:power_mod(list[i],e,n),
8          returnvalue: sconcat(returnvalue,O,value)
9      ),
10     return(base64(returnvalue))
11 );
```

Zdrojový kód č. 2: RSA šifrovací funkce

1. Pojmenování funkce a určení počtu parametrů pro následné volání funkce.
2. Založení výpočetního bloku s proměnnými *list* a *i*. Proměnnou *list* využijeme pro uložení převodu textu na čísla. Proměnnou *i* využijeme pro potřeby cyklu.
3. Uložíme do listu převod textu na čísla.
4. Nastavíme návratový řetězec *returnvalue* na prázdný řetězec.
5. Nastavíme cyklus for. Budeme provádět příkazy na řádcích 6, 7, 8 tolikrát, kolik prvků máme v proměnné *list*. Kolikrát jsme již tuto část kódu prošli, si bude pamatovat proměnná *i*.
6. Založení výpočetního bloku pro cyklus for s proměnnou *value*. Do této proměnné budeme ukládat šifry jednotlivých prvků proměnné *list*.
7. Do proměnné *value* zašifrujeme hodnotu *i*-tého prvku v proměnné *list*.
8. Do řetězcové proměnné *returnvalue* vždy připojíme oddělovač "O" a připojíme právě vypočítanou šifru.
9. Ukončení bloku pro cyklus for

10. Řetězec *returnvalue* zakódujeme pomocí funkce base64 a vrátíme jako výsledek funkce.

11. Ukončení bloku.

Před prvním zavoláním funkce musíme nejprve definici funkce vyhodnotit. Poté zašifrujeme náš první text za pomocí již vytvořeného klíče z ukázky (%i55).

```
(%i62) encrypt(text,e,n) :=
       block([octetslist, i],
           octetslist:string_to_octets(text),
           returnvalue: "",
           for i:1 thru length(octetslist) step 1 do
           block([value],
               value:power_mod(octetslist[i],e,n),
               returnvalue: sconcat(returnvalue,0,value)
           ),
           return(base64(returnvalue))
       );
```

(%o62) $encrypt(text, e, n) := block([octetslist, i],\ octetslist : string\_to\_octets(text),$ $returnvalue :, for\ i\ thru\ length(octetslist)\ do\ block([value], value : power\_mod$ $(octetslist[i], e, n), returnvalue : sconcat(returnvalue, O, value)), return(base64$ $(returnvalue)))$

```
(%i63) encrypt("test",e,n);
```

(%o63) $TzE3MzM2Mjk5NTY4NDE3NzUxODUyNzA3MDA5Nzg4ODUyNjc5N$ $TAyNDA0OTYyODc3NzM4MTM2MjgzNjcyOTM4NDU3MTA2Mzc5N$ $jgzMzhPMjMzNDg5MjY1NjM2Nzc4MjU2MzYwNDk1NzI5Mzk5OD$ $EyNjEyODg4NjMwNjc4NDM4NTc2NzQ5NTUxMjE2NTUyMTI2MT$ $k3MDk5ODY0MU8yNTA3MjU0MDYwMTUxNTcwNTU5OTc4MDk2N$ $TE2MDkzMDQwMTI4MjMzMjkyODEzMzc0MTM2MTQzNzc1ODk2$ $MTYzMjMxODE4NDg5NTE4TzE3MzM2Mjk5NTY4NDE3NzUxOD$ $UyNzA3MDA5Nzg4ODUyNjc5NTAyNDA0OTYyODc3NzM4MTM2Mj$ $gzNjcyOTM4NDU3MTA2Mzc5NjgzMzg =$

3.4.3 Funkce pro dešifrování

Aby byla naše množina funkcí kompletní, zbývá poslední funkce pro dešifrování textu. Tato funkce bude podle očekávání vykonávat v opačném pořadí inverzní operace, které jsme prováděli ve funkci pro zašifrování. Nejprve tedy dekódujeme text pomocí funkce ***base64_decode(text)***. Poté ho rozdělíme pomocí oddělovače "O". Tyto jednotlivé prvky dešifrujeme pomocí zadaného klíče. A nakonec tato čísla převedeme na původní text pomocí funkce ***octets_to_string(list)***. Ještě než přistoupíme ke zdrojovému kódu, seznámíme se s poslední dvojicí funkcí.

Rozdělení textu pomocí oddělovače

Program Maxima disponuje funkcí ***split(text, sep)***, která rozdrobí text pomocí oddělovače předaném v parametru *sep* a vytvoří seznam prvků.

```
(%i64) split("t*e*s*t","*");
```

(%o64) $[t, e, s, t]$

Vyhodnocení řetězce

V některých situacích je potřeba vyhodnotit řetězec. Budeme-li mít v řetězci uložené číslo, nebudeme s ním moci provádět početní operace, dokud tento řetězec nevyhodnotíme pomocí funkce eval_string(text).

```
(%i65) eval_string("1") + 1;
```

(%o65) 2

Vrhněme se tedy na funkci ***decrypt(crypt, d, n)***, která bude dešifrovat text uložený v proměnné *crypt* za pomoci exponentu d a modulusu n.

```
1  decrypt(crypt,d, n) :=
2  block([list , i],
3      list: split (base64_decode(crypt), "O"),
4      for i:1 thru length(list) step 1 do
5      block([value],
```

```
6           value : power_mod(eval_string(list[i]),d,n),
7           list[i]: value
8       ),
9       return(octets_to_string(list))
10  );
```

Zdrojový kód č. 3: RSA dešifrovací funkce

1. Pojmenování funkce a určení počtu parametrů pro následné volání funkce.
2. Založení výpočetního bloku s proměnnými *list* a *i*. Proměnnou *list* využijeme pro uložení převodu zašifrovaného textu na čísla. Proměnnou *i* využijeme pro potřeby cyklu.
3. Dekódujeme proměnnou *crypt*. Dekódovaný text rozdělíme pomocí oddělovače a uložíme do proměnné *list*.
4. Nastavíme cyklus for. Budeme provádět příkazy na řádcích 5, 6, 7 tolikrát, kolik prvků máme v proměnné *list*. Kolikrát jsme již tuto část kódu prošli, si bude pamatovat proměnná *i*.
5. Založení výpočetního bloku pro cyklus for s proměnnou *value*. Do této proměnné budeme dešifrovat jednotlivých prvky proměnné *list*.
6. *I*-tý prvek proměnné *list* přepíšeme dešifrovanou hodnotou uloženou v proměnné *value*.
7. Ukončení bloku pro cyklus for
8. Dešifrovaná čísla poté opět převedeme na text pomocí funkce octets_to_string() a vrátíme jako výsledek funkce.
9. Ukončení bloku.

Vyzkoušejme tedy dešifrovat text.

```
(%i66) decrypt(crypt,d, n) :=
       block([list, i],
           list: split (base64_decode(crypt), "O"),
           for i:1 thru length(list) step 1 do
           block([value],
               value : power_mod(eval_string(list[i]),d,n),
               list[i]: value
           ),
           return(octets_to_string(list))
       );
```

(%o66) $decrypt(crypt, d, n) := block([list, i], list : split(base64\_decode(crypt), "O"),$
$for\ i\ thru\ length(list)\ doblock([value], value : power\_mod(eval\_string(list[i])),$
$d, n), list[i] : value), return(octets\_to\_string(list)))$

```
(%i67) decrypt(%o63,d, n);
```

(%o67) *test*

Opět jsme tedy dostali původní zprávu. Tímto je naše malá šifrovací knihovna RSA kompletní.

4 Závěr

Práce čtenáře v první kapitole seznamuje s matematickým pozadím asymetrického šifrování RSA. Po zvládnutí této kapitoly čtenář získá v další kapitole znalost základních pojmů šifrování. Osvojí si principy asymetrického šifrování RSA a své nabyté znalosti utvrdí na přiložených příkladech. Řešením těchto příkladů si uvědomuje složitost výpočtů a uvítá pomoc systému počítačové algebry Maxima. V poslední kapitole se po seznámení s programem Maxima pustí do ověření příkladů z druhé kapitoly, a poté dokonce vytvoří vlastní funkce pro potřeby šifrování. To jen potvrzuje čtenářovu nabytou znalost dané problematiky. To bylo hlavním cílem této práce. Dalším cílem bylo práci napsat tak, aby byla zajímavá i pro ty, kteří o šifrování čtou vůbec poprvé. To jsem se snažil podpořit častými obrázky, příklady a ukázkami příkazů z programu. Tento text by mohl sloužit jako podpůrný text pro studium asymetrického šifrování.

Literatura

[1] BLAŽEK, Jaroslav a a spol. *Algebra a teoretická aritmetika.* Státní pedagogické nakladatelství. Praha, 1985.

[2] DEMLOVÁ, Marie. *Diskrétní matematika a logika: Přednáška 11 – 12/12/2005.* [online] Praha : [s.n.], 2005. Dostupné z `http://math.feld.cvut.cz/demlova/teaching/dml/pred11.pdf`.

[3] DROBULIAK, Matůš. *RSA šifra.* [online], 2015. Dostupné z `http://www.karlin.mff.cuni.cz/~tuma/Aplikace15/Prace15/Drobuliak_RSA.pdf`.

[4] MONTERDE J. a VALLEJO J., *Implementing the RSA Cryptosystem with Maxima CAS*,The Electronic Journal of Mathematics and Technology, Vol. 6, No. 1, PP. 34-53, 2012.

[5] PŘIKRYL, Jan. *Matematické algoritmy (11MAG): Přednáška 3 - prezentace.* [online] Praha : [s.n.], 2013. Dostupné z `https://zolotarev.fd.cvut.cz/static/mag/mag-2014-03-slides.pdf`. http://mayor.fri.uniza.sk/krypto/09/rsa.pdf

[6] STANĚK, Martin. *Kryptografia a bezpečnosť: Text k přednášce.* Dostupné z `http://mayor.fri.uniza.sk/krypto/09/rsa.pdf`.

[7] ŠÍR, Zbyněk. *Teorie čísel a úvod do šifrování.* Dostupné z `http://www.talnet.cz/documents/18/54ffba6e-4e85-4475-b14d-4a63c19b4e3c`.

[8] VELEBIL, Jiří. *Diskrétní matematika: Text k přednášce.* [online] Praha : [s.n.], 2007. 193 s. Dostupné z `ftp://math.feld.cvut.cz/pub/velebil/y01dma/dma-notes.pdf`.

[9] *Algoritmy, Euklidův algoritmus.* [online]. Dostupné z `http://www.algoritmy.net/article/44/Eukliduv-algoritmus`.