Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Bc. Jana Divišová

Kryptografie založená na mřížkách

Katedra algebry

Vedoucí diplomové práce:
RNDr. David Stanovský, Ph.D.

Studijní program:
Matematika
Matematické metody informační bezpečnosti

2010

Na tomto místě bych velmi ráda poděkovala svému vedoucímu diplomové práce RNDr. Davidovi Stanovskému, Ph.D. za pomoc a cenné rady, kterými mi pomohl. Dále bych ráda poděkovala Ivanu Štubňovi za podporu a za podnětné připomínky k implementaci kryptosystémů.

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 14. dubna 2010 Jana Divišová

Obsah

Kapitola 1. Úvod 5

Kapitola 2. Mřížky 6
2.1. Problémy spojené s mřížkami 7
2.2. Kryptografie ve spojitosti s mřížkami 11

Kapitola 3. Kryptosystémy založené na mřížkách 14
3.1. Ajtai–Dwork kryptosystém 14
3.2. Goldreich–Goldwasser–Halevi kryptosystém 17
3.3. NTRU kryptosystém 20

Kapitola 4. Mřížky v kryptoanalýze 27
4.1. Knapsack kryptosystém 27
4.2. Hidden number problem 31

Kapitola 5. RSA vs. NTRU 33
5.1. Implementace 34
5.2. Výsledky testů 40

Kapitola 6. Závěr 46

Literatura 47

Příloha A. Tabulky 48

Příloha B. Grafy 54

Název práce: Kryptografie založená na mřížích
Autor: Jana Divišová
Katedra: Katedra algebry
Vedoucí bakalářské práce: RNDr. David Stanovský, Ph.D.
e-mail vedoucího: stanovsk@karlin.mff.cuni.cz

Abstrakt: V předložené práci se věnujeme různým pohledům na využití mřížek v kryptografii. Poté, co popíšeme mřížky obecně a problémy s nimi spojené, se věnujeme kryptosystémům založených na mřížích. Popisujeme jejich matematické pozadí i formulaci algoritmů na šifrování a dešifrování. V další části popisujeme využití mřížek v kryptoanalýze. Jedná se především o útoky na knapsack systém a řešení hidden number problému. Významnou součástí práce je také srovnání dvou kryptosystémů RSA a NTRU pro srovnatelnou úroveň bezpečnosti a to z hlediska rychlosti šifrování, dešifrování a generování klíčů.

Klíčová slova: mřížky, kryptografie s veřejným klíčem, kryptoanalýza, NTRU, RSA

Title: Lattice based cryptography
Author: Jana Divišová
Department: The Department of Algebra
Supervisor: RNDr. David Stanovský, Ph.D.
Supervisor's e-mail address: stanovsk@karlin.mff.cuni.cz

Abstract: The aim of this work is several faces of lattices in cryptography. After the section in which we describe lattices in general and lattice problems, we turn to the lattice based cryptosystems. We describe their mathematical background and also formulations of encryption and decryption algorithms. In the next part we describe the usage of lattice in cryptanalysis. It is mainly attacks against knapsack system a solving hidden number problem. The significant part of this work is to compare two cryptosytems RSA a NTRU for the similar level of security. We compare the speed of encryption, decryption and key generation.

Keywords: lattices, public key cryptosystems, cryptanalysis, NTRU, RSA

KAPITOLA 1

Úvod

Potřeba kryptografie s veřejným klíčem a elektronického podpisu v současné době velmi narůstá. Hlavně z důvodu, že čím dál více lidí používá počítače pro předávání důvěrných dokumentů, nakupování a k přístupu k citlivým údajům. Informace předávané při těchto činnostech je potřeba chránit. Ve skutečnosti, některé z těchto problémů ani nelze vyřešit bez použití bezpečné kryptografie s veřejným klíčem.

Historicky byly mřížky zkoumány již od 18. století matematiky jako Lagrange, Gauss nebo později Minkowski. V nedávné době se mřížky dostali do povědomí v informatice. Jsou používány k řešení různých problémů a své uplatnění naleznou především v kryptoanalýze, ale existují i kryptografické schémata založená na mřížkách.

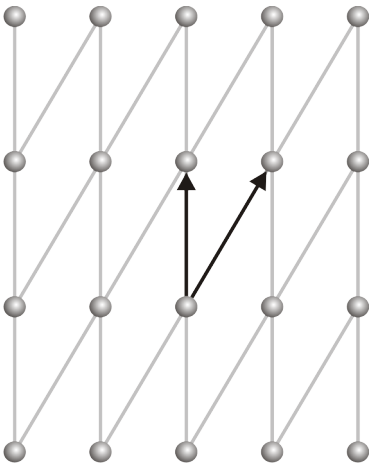
V této práci si nejdříve v Kapitole 2 popíšeme mřížky a problémy s nimi spojené. Především si ale v Kapitole 3 představíme několik kryptosystémů založených na mřížkách. Pro Kapitolu 3 byly zejména využity zdroje [**5**], [**6**] a [**9**]. Nejvýznamnější kryptosystém, který využívá problémů mřížek, je NTRU. Jeho popis vychází z [**7**] a [**8**]. Dále si v Kapitole 4 ukážeme metody, jak využít mřížky v kryptoanalýze. Kapitola 4 čerpá především ze [**11**] a [**12**].

Poslední Kapitola 5 této práce je i srovnání NTRU a RSA z hlediska rychlosti. Budeme testovat rychlost šifrování, dešifrování i generování klíčů. RSA je pravděpodobně nejpoužívanější kryptosystém pro podepisování a šifrování asymetrickou kryptografií. Na druhou stranu, NTRU je nový a v poslední době velmi diskutovaný šifrovací systém. Populární je zejména pro svou bezpečnost, protože je založen na problému, který je těžký v nejhorším případě, na rozdíl od RSA, kde musíme spoléhat na správnou volbu parametrů. Zároveň s rostoucí výpočetní silou faktorizační algoritmy, díky kterým lze RSA prolomit a které se dříve zdály nepoužitelné, dnes mohou běžet na domácích počítačích. Prozatímní řešení je zvyšovat délku klíče, ale to nemůže do budoucna stačit, protože již dnes existuje kvantový algoritmus, který dokáže faktorizovat velká čísla v polynomiálním čase. Z tohoto pohledu je změna nevyhnutelná. Otázkou ale zůstává, který kryptosystém dokáže RSA nahradit. NTRU je jednou z možností.

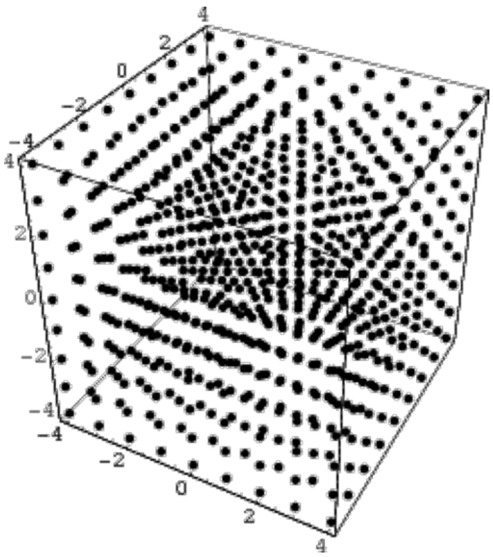
KAPITOLA 2

Mřížky

Mřížka je volně řečeno množina bodů v n–rozměrném prostoru s periodickou strukturou. Formální definice je uvedena níže.



Obrázek 1. Příklad mřížky ve dvojrozměrném prostoru



Obrázek 2. Příklad mřížky ve trojrozměrném prostoru

DEFINICE 2.1. Nechť n je přirozené číslo a $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$ lineárně nezávislé vektory nad $\mathbb{R}$. Pak množinu

$$L = \sum_{i=1}^{n} \mathbb{Z}\mathbf{b}_i = \left\{ \sum_{i=1}^{n} a_i \mathbf{b}_i \,;\ a_1, \ldots, a_n \in \mathbb{Z} \right\}$$

nazýváme *mřížka* nad vektory $\mathbf{b}_1, \ldots, \mathbf{b}_n$.

DEFINICE 2.2. Řekneme, že $\mathbf{b}_1, \ldots, \mathbf{b}_n$ tvoří *bázi mřížky* L, pokud L je mřížka nad těmito vektory. Libovolná podmnožina $M \subseteq \mathbb{R}^n$ je mřížka, pokud existují vektory $\mathbf{b}_1, \ldots, \mathbf{b}_n$ takové, že M je mřížka právě nad $\mathbf{b}_1, \ldots, \mathbf{b}_n$. Číslo n se nazývá *hodnost mřížky*.

Jak je vidět z Definice 2.2, každá mřížka nemá pouze jednu bázi, ale má nekonečně mnoho bází. Žádná báze není ničím privilegovaná před jinou.

2.1. Problémy spojené s mřížkami

V této části si nejdříve popíšeme dva základní problémy týkající se mřížek a následně si ukážeme závislost jejich složitostí.

2.1.1. *SVP*

Hlavní problém týkající se mřížek je hledání nejkratšího vektoru báze (shortest vector problem – *SVP*). Při řešení tohoto problému máme zadánu mřížku L s nějakou libovolnou bází a naším cílem je najít nejkratší nenulový vektor, který patří do mřížky.

DEFINICE 2.3. Označením $SVP(L)$ rozumíme problém hledání nenulového vektoru $\mathbf{u} \in L$, pro který platí

$$\|\mathbf{u}\| \leq \|\mathbf{v}\|,$$

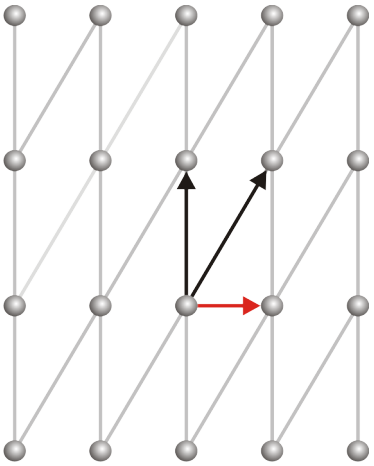
kde $\mathbf{v}$ je libovolný vektor mřížky.

SVP nemusíme řešit takto přesně, ale někdy nám stačí aproximační varianta. Při aproximaci tedy nehledáme jeden konkrétní nejkratší vektor, ale hledáme vektor, který se neliší od tohoto nejkratšího o násobek větší než je námi stanovená konstanta.

DEFINICE 2.4. Označením $SVP_\gamma(L)$ rozumíme problém hledání nenulového vektoru $\mathbf{u} \in L$, pro který platí

$$\|\mathbf{u}\| \leq \gamma \cdot \|\mathbf{v}\|,$$

kde $\mathbf{v}$ je libovolný vektor mřížky a γ je aproximační konstanta.



Obrázek 3. Příklad *SVP*

Složitost tohoto problému plyne jak z faktu, že jedna mřížka má mnoho různých bází, tak z předpokladu, že typicky dostaneme mřížku zadanou pomocí báze, která obsahuje velmi dlouhé vektory, mnohem delší než je nejkratší nenulový vektor. Algoritmy, které řeší SVP_γ, fungují v exponenciálním čase pro γ polynomiální. A naopak algoritmy, které jsou schopny řešit SVP_γ v polynomiálním čase, dávají γ exponenciální.

Existuje známý polynomiální algoritmus označovaný zkratkou LLL (Lenstra–Lenstra–Lovász, prezentovaný v [**10**]), který řeší SVP_γ s aproximační konstantou $\gamma = 2^{\mathcal{O}(n)}$, kde n je hodnost mřížky. Přesto, že se to zdá jako velmi špatný výsledek, je tento algoritmus velmi užitečný a používá se v problematice faktorizace polynomů nad racionálními koeficienty a v mnohých aplikacích v kryptoanalýze. Později Schnorr představil vylepšení, které snížilo aproximační konstantu γ na $2^{\mathcal{O}(n(\log\log n)^2/\log n)}$.

Pokud vezmeme v úvahu výše uvedené výsledky, mohli bychom předpokládat, že SVP_γ je NP–těžké. Nicméně zatím nejlepší známý výsledek je, že to platí pouze pro konstantu $\gamma < 2^{(\log n)^{\frac{1}{2}-\varepsilon}}$. Navíc obecně SVP_γ není považováno za NP–těžké pro $\gamma > \sqrt{n/\log n}$.

Z praktického hlediska je tedy problém říci, jak veliké mřížky by se měli používat, tj. jaké n volit, aby byl problém spočítat SVP_γ s dnešní výpočetní silou. Podle [**3**] platí, že pokud se n zvolí řádově 10^2, je SVP_γ extrémně složité.

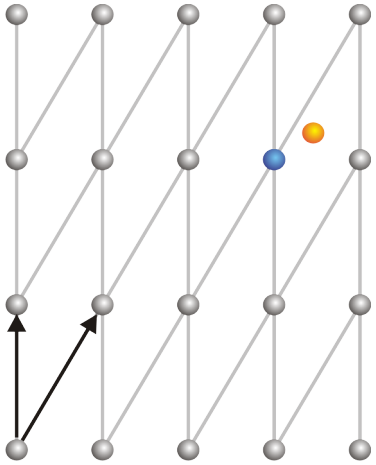
2.1.2. *CVP*

Druhý a obecně využívanější problém týkající se mřížek je hledání nejbližšího bodu mřížky (closest vector problem – *CVP*). Pokud máme zadán terč (terčový vektor, terčový bod) $\mathbf{w}$ v prostoru $\mathbb{R}^n$ a mřížku L ve stejném prostoru, pak máme najít bod mřížky, který je nejblíž k zadanému terči.

Definice 2.5. *CVP*$(L, \mathbf{w})$ je problém hledání vektoru $\mathbf{u} \in L$, pro který platí

$$\|\mathbf{u} - \mathbf{w}\| \leq \|\mathbf{v} - \mathbf{w}\|,$$

kde $\mathbf{v}$ je libovolný vektor mřížky.



Obrázek 4. Příklad *CVP*

Pokud budeme uvažovat aproximační variantu, pak hledáme bod mřížky, který není více vzdálený, než je vzdálenost nejbližšího bodu vynásobená aproximační konstantou.

Definice 2.6. $CVP_\gamma(L, \mathbf{w})$ je problém hledání vektoru $\mathbf{u} \in L$, pro který platí

$$\|\mathbf{u} - \mathbf{w}\| \leq \gamma \cdot \|\mathbf{v} - \mathbf{w}\|,$$

kde $\mathbf{v}$ je libovolný vektor mřížky a γ je aproximační konstanta.

2.1.3. *SVP* není složitější než *CVP*

Obecně se předpokládá, že *SVP* není těžší než *CVP*. Z empirického hlediska je tento předpoklad oprávněný, především z toho důvodu, že dokázat NP–těžkost problému SVP_γ trvalo o mnoho déle než pro problém CVP_γ. Navíc, aproximace problému CVP_γ v n–rozměrné mřížce s faktorem $\gamma = 2^{\log^{0,999} n}$ je NP–těžká, ale stejný výsledek pro SVP_γ dostáváme pouze pro $\gamma = 2^{(\log n)^{\frac{1}{2}-\varepsilon}}$. V této části práce ukážeme, že pokud máme orákulum, ze kterého získáme jako výstup aproximaci CVP_γ, pak dokážeme v polynomiálním čase najít aproximaci SVP_γ.

Mezi *SVP* a *CVP* jsou dva základní rozdíly. Za prvé *SVP* hledá bod mřížky, který je nejblíže nulovému bodu, na druhé straně *CVP* hledá bod mřížky, který je nejblíže libovolnému zadanému bodu. Za druhé *SVP* nedovoluje nulový vektor jako výsledek, ale *CVP* může dát jako přípustný výsledek námi zadaný terčový vektor (s podmínkou, že patří do mřížky). To znamená, že tyto dva problémy nejsou jednoduše propojené. Triviální redukce *SVP* na *CVP* tedy nebude fungovat, protože *CVP* orákulum by mohlo vracet vždy terčový vektor a to by byla nula pro *SVP*. Tuto možnost potřebujeme vyloučit.

Budeme postupovat jako v [**4**]. První krok, který tedy uděláme, je, že nebudeme hledat v okolí nuly, ale v okolí jiného bodu $\mathbf{w} \in L$. Abychom se vyhnuli tomu, že nám orákulum vrátí právě tento bod, tak se zeptáme orákula na mřížku $L' \subsetneq L$, která neobsahuje vektor $\mathbf{w}$.

Nyní ukážeme, jak převést SVP_γ na řešení n případů CVP_γ. Nejdříve popíšeme algoritmus redukce, poté dokážeme několik pomocných tvrzení a na závěr kapitoly formulujeme hlavní větu o redukci SVP_γ na CVP_γ.

Teď ve stručnosti popíšeme redukci, která se dá vyjádřit Algoritmem 2.7. Nechť je dána báze $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ mřížky $L(B) = \{\sum_{i=1}^n c_i \mathbf{b}_i : c_1, \ldots, c_n \in \mathbb{Z}\}$. Zkonstruujeme postupně n instancí CVP_γ tak, že do j–té instance CVP_γ vstupuje báze $B^{(j)} = [\mathbf{b}_1, \ldots, \mathbf{b}_{j-1}, 2\mathbf{b}_j, \mathbf{b}_{j+1}, \ldots, \mathbf{b}_n]$ a terčový vektor $\mathbf{b}_j$. Vidíme tedy, že $\mathbf{b}_j \notin L(B^{(j)})$ a $L(B^{(j)}) \subsetneq L(B)$. Orákulum nám pak vrátí nejmenší rozdíl ze všech volání CVP_γ. Tj. pokud označíme $\mathbf{u}_j$ odpověď orákula $CVP_\gamma(L(B^{(j)}), \mathbf{b}_j)$, pak vracíme nejkratší vektor z množiny $\mathbf{u}_1 - \mathbf{b}_1, \ldots, \mathbf{u}_n - \mathbf{b}_n$.

Algoritmus 2.7 (Redukce).

`Vstup`: Báze B, kde $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$
`Výstup`: $SVP_\gamma(L(B))$

`j.` Pro $j = 1$ až n zavolej orákulum na vstupu $(B^{(j)}, b_j)$,
tj. $\mathbf{u}_j = CVP_\gamma(L(B^{(j)}), \mathbf{b}_j)$

`n+1.` vrať vektor $\mathbf{v}$, nejkratší vektor z množiny $\mathbf{u}_1 - \mathbf{b}_1, \ldots, \mathbf{u}_n - \mathbf{b}_n$

Správnost Algoritmu 2.7 plyne z následujících tří tvrzení, které dávají do souvislosti výsledky problému SVP_γ a výsledky CVP_γ instancí.

Lemma 2.8. *Nechť je $\boldsymbol{v} = \sum_{i=1}^n c_i \boldsymbol{b}_i$ nejkratší vektor mřížky L. Pak existuje i takové, že c_i je liché.*

Důkaz. Pokud by všechna c_i byla sudá, pak by byl vektor $\frac{1}{2} \cdot \mathbf{v} = \sum_{i=1}^n \frac{c_i}{2} \mathbf{b}_i$ kratší než vektor $\mathbf{v}$ v mřížce L. □

Tvrzení 2.9. *Nechť je $\boldsymbol{v} = \sum_{i=1}^n c_i \boldsymbol{b}_i$ vektor v mřížce $L(B)$ a c_j nechť je liché. Pak $\boldsymbol{u}_j = \frac{c_j+1}{2}(2\boldsymbol{b}_j) + \sum_{i \neq j} c_i \boldsymbol{b}_i$ je vektor mřížky $L(B^{(j)})$ a vzdálenost $\boldsymbol{u}_j$ a $\boldsymbol{b}_j$ je délka vektoru $\boldsymbol{v}$.*

Důkaz. Nejdříve ukážeme, že vektor $\mathbf{u}_j$ patří do mřížky $L(B^{(j)})$. Vzhledem k jeho rozepsání na součet vektorů báze stačí, že $\frac{c_j+1}{2}$ je celé číslo, a to je, protože c_j je liché.

Dále potřebujeme ukázat, že vzdálenost $\mathbf{u}_j$ a $\mathbf{b}_j$ je délka vektoru $\mathbf{v}$

$$\mathbf{u}_j - \mathbf{b}_j = \frac{c_j + 1}{2}(2\mathbf{b}_j) + \sum_{i \neq j} c_i \mathbf{b}_i - \mathbf{b}_j = c_j \mathbf{b}_j + \sum_{i \neq j} c_i \mathbf{b}_i = \mathbf{v} \ .$$

□

TVRZENÍ 2.10. *Nechť máme* $c'_j = \frac{c_j+1}{2}$, c_j *liché, a* $\boldsymbol{u}_j = c'_j(2\boldsymbol{b}_j) + \sum_{i\neq j} c_i\boldsymbol{b}_i$ *je vektor v mřížce* $L(B^{(j)})$. *Pak platí, že vektor* $\boldsymbol{v} = (2c'_j - 1)\boldsymbol{b}_j + \sum_{i\neq j} c_i\boldsymbol{b}_i$ *je nenulový vektor v mřížce* $L(B)$ *a délka* $\boldsymbol{v}$ *je vzdálenost* $\boldsymbol{u}_j$ *od terčového vektoru* $\boldsymbol{b}_j$.

DŮKAZ. Fakt, že $\mathbf{v}$ je nenulový vektor, plyne z toho, že $2c'_j - 1 = c_j$ a to je liché z předpokladů.

Nyní ukážeme, že $\mathbf{v}$ je vzdálenost $\mathbf{u}_j$ od terče $\mathbf{b}_j$:

$$\mathbf{v} = (2c'_j - 1)\mathbf{b}_j + \sum_{i\neq j} c_i\mathbf{b}_i = c'_j(2\mathbf{b}_j) + \sum_{i\neq j} c_i\mathbf{b}_i - \mathbf{b}_j = \mathbf{u}_j - \mathbf{b}_j \ .$$

□

Nyní můžeme vyřknout Větu 2.11, která nám propojí Lemma 2.8, Tvrzení 2.9 a Tvrzení 2.10.

VĚTA 2.11. *Pro každou funkci* $f : \mathbb{N} \to \{r \in \mathbb{R} : r \geq 1\}$, *problém* SVP_f *je redukovatelný v polynomiálním čase na* CVP_f.

DŮKAZ. Definujme instance problému $CVP_f(L(B^{(j)}), \mathbf{b}_j)$ pro $j = 1, \ldots, n$, stejně jako v Algoritmu 2.7. Chceme dokázat, že nejkratší vektor mřížky $L(B)$ se vyskytne alespoň pro jedno $j = 1, \ldots, n$ jako odpověď orákula na dotaz $CVP_f(L(B^{(j)}), \mathbf{b}_j)$.

Nechť je vektor $\mathbf{v} = \sum_{i=1}^{n} c_i\mathbf{b}_i$ nejkratší vektor v mřížce $L(B)$. Tedy víme, že $\|\mathbf{v}\| \leq \gamma \cdot \|\mathbf{w}\|$ pro libovolný vektor $\mathbf{w} \in L(B)$. Z Lemmatu 2.8 vyplývá, že existuje j, že koeficient c_j je lichý. Použijeme Tvrzení 2.9 a definujeme vektor $\mathbf{u}_j = \frac{c_j+1}{2}(2\mathbf{b}_j) + \sum_{i\neq j} c_i\mathbf{b}_i$. Platí tedy, že $\mathbf{u}_j$ náleží do $L(B^{(j)}) \subset L(B)$ a splňuje $\|\mathbf{u}_j - \mathbf{b}_j\| = \|\mathbf{v}\| \leq \gamma \cdot \|\mathbf{w}\|$. Vektor $\mathbf{u}_j$ je tedy řešení j–té instance CVP_γ.

□

2.2. Kryptografie ve spojitosti s mřížkami

V této části popíšeme vztahy mřížek a některých kryptografických pojmů.

2.2.1. Jednosměrná funkce s padacími dvířky

Jednosměrná funkce s padacími dvířky je funkce, kterou je snadné spočítat a všeobecně se věří, že je jí těžké invertovat bez informace navíc (padací dvířka).

Základní myšlenka, o kterou se opírá celá konstrukce pomocí mřížek je, že pokud máme k dispozici jakoukoliv bázi mřížky, je jednoduché vygenerovat bod, který leží blízko bodu mřížky. Například tak, že vezmeme bod mřížky a přičteme k němu malý chybový vektor. Ale zároveň je složité k takovému bodu najít původní bod mřížky, pokud máme k dispozici libovolnou bázi – *CVP*. To znamená, že přičtení chybového vektoru je dobrý kandidát na jednosměrnou funkci s padacími dvířky.

Abychom si ukázali, jak funguje mechanismus padacích dvířek, tedy, co vlastně v našem případě jsou padací dvířka, musíme využít fakt, že různé báze stejné mřížky dávají různou možnost nalezení nejbližšího bodu mřížky k libovolnému bodu v $\mathbb{R}^n$. Proto by naše padací dvířka měla být taková báze, která nabízí velmi dobrou aproximaci nejbližšího bodu mřížky. Budeme tedy používat dvě báze – jednu pro výpočet funkce, a druhou pro její invertování.

2.2.2. Šifrovací schéma

Nyní, když máme jednosměrnou funkci, zbývá nám vložit zprávu do argumentu funkce a tím zprávu zašifrovat. Základní metoda je použít těžký bit jednosměrné funkce. Takto potom můžeme vkládat zprávu bit po bitu. Toto schéma je velmi bezpečné, ale nepoužitelné z důvodu obrovského nárůstu velikosti zprávy. Při použití mřížek můžeme ale využít jinou možnost, která by mohla být efektivnější. Můžeme vybrat z veřejné báze vektory a udělat z nich celočíselnou lineární kombinaci "specifikovanou nějakým způsobem" bity zprávy a k tomuto bodu přidat malý náhodný chybový vektor.

Dešifrování potom probíhá tak, že najdeme nejbližší bod mřížky (pomocí soukromé báze). Soukromou bázi jsme volili tak, aby dešifrování (tj. nalezení nejbližšího bodu) bylo správně s vysokou pravděpodobností.

2.2.3. Podpisové schéma

Pomocí jednosměrné funkce je možnost sestavit podpisové schéma. V případě mřížek se na zprávu budeme dívat jako na n - rozměrný vektor, pak podpis takovéto zprávy je jednoduše nejbližší bod mřížky. Soukromá báze byla zvolena tak, abychom mohli takovýto bod spočítat. Pokud chceme podpis ověřit, musíme zjistit, zda se jedná skutečně o bod mřížky a zda je blízko zprávy.

Je nutné říci, že zprávy, které leží blízko sebe, mají při tomto použití stejný podpis. Z toho důvodu je doporučeno nejdříve zprávy hashovat a pak až podepisovat. Tím se taky ubráníme tomu, že pokud by někdo našel dvě blízké zprávy s různým podpisem, pak mi mohl dopočítat velmi malou bázi mřížky, což by útočníkovi umožnilo počítat blízké body v mřížce.

2.2.4. Složitost

Všechna dnes běžně používaná kryptografická schémata jsou založena na složitosti v průměrném případě. Například při použití jakékoliv schématu založeném problému na faktorizace velkých čísel musíme vycházet z předpokladu, že počáteční parametry budou vhodně zvoleny. Ale co přesně znamená vhodně zvoleny? Je zřejmé, že bychom se měli vyhnout číslům s malými prvočíselnými děliteli. Existuje pak ještě mnoho dalších předpokladů, které musí naše parametry splnit, aby bylo schéma bezpečné. Ale co když existují omezení na parametry, o kterých zatím nevíme?

Z tohoto pohledu je vhodné použít schéma, které není bezpečné jen v průměrném případě, ale které je bezpečné i v nejhorším případě. A právě schéma založené na mřížkách tyto vlastnosti splňuje. Důkazem se nebudeme v této práci blíže zabývat, podrobnosti lze nalézt v [**1**].

2.2.5. Hashovací funkce

Podobně jako jsme výše definovali šifrovací schéma, lze zavést i rodinu hashovacích funkcí – za hash budeme považovat nejbližší bod mřížky. První konstrukce byla předvedena v [**2**], kde bylo ukázáno, že pokud bychom dokázali invertovat funkci z této rodiny, pak bychom byli schopni vyřešit jakýkoliv příklad aproximace SVP s aproximační konstantou $\gamma = n^c$.

KAPITOLA 3

Kryptosystémy založené na mřížkách

V této kapitole popíšeme několik algoritmů, které využívají problémy spojené s mřížkami. V první části popíšeme Ajtai–Dwork kryptosystém, poté ukážeme jednosměrnou funkci s padacími dvířky a rozebereme ji i z algoritmického hlediska v GGH kryptosystému. Již jsme nastínili v závěru předchozí kapitoly, jak by se dala takováto funkce využít v šifrovacím i podepisovacím schématu. V této kapitole si ale ukážeme konkrétní algoritmy, které tuto jednosměrnou funkci využívají. Materiál k oběma kryptosystémům byl čerpán z [**5**] a [**6**]. Nakonec si představíme nejvýznamnější algoritmus založený na mřížkách NTRU.

3.1. Ajtai–Dwork kryptosystém

První kryptosystém, který si představíme, je i historicky nejstarší. Jeho autorem je dvojice Ajtai a Dwork. Původní šifrovací algoritmus ale může způsobovat chyby při dešifrování, proto si ukážeme ještě vylepšení, díky kterému k těmto chybám již nedochází. Tento kryptosystém je přelomový v tom, že využívá složitost v nejhorším případě, ale ukázalo se, že jakákoliv reálně použitelná implementace vede k výraznému snížení bezpečnosti.

3.1.1. Teoretické zázemí algoritmu

K popisu algoritmu zavedeme následující značení.

Definice 3.1. Nechť n je přirozené číslo, $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$. Pro lineárně nezávislé vektory $\mathbf{b}_1, \ldots, \mathbf{b}_n$ definujeme *rovnoběžník nad vektory* $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ jako množinu

$$P(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{b}_i ; \alpha_i \in [0,1) \right\} .$$

Šířka rovnoběžníku $P(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ je minimální vzdálenost $\mathbf{b}_i$ od podprostoru generovaného vektory $\mathbf{b}_j$, $j \neq i$, přes všechna i.

Pokud máme zadaný rovnoběžník $P(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ a vektor $\mathbf{v}$, potom můžeme *redukovat vektor* $\boldsymbol{v}$ *modulo* P. Vektor $\mathbf{v}' \in P$ a platí $\mathbf{v}' = \mathbf{v} + \sum_{i=1}^{n} a_i \mathbf{b}_i$, kde a_i jsou celá čísla. Píšeme $\mathbf{v}' = \mathbf{v} \bmod P$.

3.1.2. Popis algoritmu

Algoritmus popíšeme ve třech částech. Nejdříve vygenerujeme klíče a poté ukážeme, jak probíhá šifrování a nakonec dešifrování.

3.1.2.1. Zvolení parametrů a klíčů

Číslo n je bezpečnostní parametr. Čím vyšší je n, tím bezpečnější schéma je. Nechť $m = n^3$ a $\rho_n = 2^{n \log n}$. Označíme B_n jako n–rozměrnou krychli o délce hrany ρ_n

$$B_n = \{\mathbf{w} \in \mathbb{R}^n; 0 \leq w_i \leq \rho_n, i = 1, \ldots, n\} \ ,$$

kde w_i jsou souřadnice vektoru $\mathbf{w}$. Dále pak označíme S_n n–rozměrnou kouli o poloměru n^{-8}

$$S_n = \{\mathbf{w} \in \mathbb{R}^n; \|\mathbf{w}\| \leq n^{-8}\} \ .$$

Jako soukromý klíč zvolíme náhodně vektor $\mathbf{u}$ v n–rozměrné jednotkové kouli.

Poté musíme zvolit veřejný klíč $\mathbf{v}_1, \ldots, \mathbf{v}_m$ a $\mathbf{w}_1, \ldots, \mathbf{w}_n$. Vektory $\mathbf{v}_1, \ldots, \mathbf{v}_m$ budeme volit náhodně následujícím postupem

(1) Zvolíme náhodně vektory $\mathbf{b}_1, \ldots, \mathbf{b}_m$ tak, aby byl jejich skalární součin se soukromým klíčem $\mathbf{u}$ celočíselný, tedy $\mathbf{b}_j \in \{\mathbf{w} \in B_n; \langle \mathbf{w}, \mathbf{u} \rangle \in \mathbb{Z}\}$, pro $j = 1, \ldots, m$.
(2) Pro $i = 1, \ldots, n$ zvolíme náhodně vektory $\mathbf{w}_1, \ldots, \mathbf{w}_n$ z koule S_n.
(3) Vrátíme $\mathbf{v}_j = \mathbf{b}_j + \sum_{i=1}^{n} \mathbf{w}_i$, pro $j = 1, \ldots, m$.

3.1.2.2. Šifrování

V Ajtai–Dwork kryptosystému probíhá šifrování bit po bitu. Tedy pro řetězec $s = c_1 c_2 \ldots c_l$ bude každý bit c_i šifrovaný zvlášť.

Pro zašifrování "0" zvolíme náhodně $a_1, \ldots, a_m \in \{0, 1\}$ a redukujeme vektor $\sum_{i=1}^{m} a_j \mathbf{v}_j$ modulo rovnoběžník $P(\mathbf{w}_1, \ldots, \mathbf{w}_n)$. Zašifrování "0" je tedy vektor

$$\mathbf{x} = \Big(\sum_{j=1}^{m} a_j \mathbf{v}_j \Big) \bmod P(\mathbf{w}_1, \ldots, \mathbf{w}_n) \ .$$

Pro zašifrování "1" zvolíme náhodně vektor $\mathbf{x} \in P(\mathbf{w}_1, \ldots, \mathbf{w}_n)$. Zašifrování "1" je pak právě tento vektor.

3.1.2.3. Dešifrování

Pokud máme šifrovaný text $\mathbf{x}$ a soukromý klíč $\mathbf{u}$, tak spočítáme $\gamma = \langle \mathbf{x}, \mathbf{u} \rangle$. Dešifrujeme "0", pokud je číslo γ vzdálené od celého čísla méně jak $\frac{1}{n}$. V opačném případě dešifrujeme "1".

3.1.2.4. Chyby při dešifrování

Je jednoduché vidět, že pokud $\mathbf{x}$ je zašifrovaná "1", je desetinná část skalárního součinu $\langle \mathbf{x}, \mathbf{u} \rangle$ téměř rovnoměrně rozdělená v intervalu $[0, 1)$. Na druhou stranu, pokud $\mathbf{x}$ je zašifrovaná "0", je desetinná část $\langle \mathbf{x}, \mathbf{u} \rangle$ vždy menší než $\frac{1}{n}$. Tedy šifrovaná nula bude vždy i dešifrovaná jako nula, ale šifrovaná jednička bude s pravděpodobností $\frac{2}{n}$ dešifrovaná jako nula.

3.1.3. Konstrukce bez dešifrovacích chyb

Trojice matematiků Goldreich, Goldwasser a Halevi modifikovali v [**9**] Ajtai-Dwork kryptosystém tak, aby nedocházelo k výše popsaným chybám při dešifrování. Schéma je velmi podobné. Šifrování "1" probíhá stejným způsobem, a šifrování "0" se jen o málo liší. Při dešifrování tedy využíváme stejné vlastnosti jako v předchozím – pokud je $\langle \mathbf{x}, \mathbf{u} \rangle$ blízko celému číslu, dešifrujeme "0".

3.1.3.1. Zvolení parametrů a klíčů

Čísla n, m, ρ_n a kouli S_n zvolíme stejně jako v Ajtai–Dwork kryptosystému.

Stejný zůstane i soukromý klíč, tedy zvolíme náhodně vektor $\mathbf{u}$ v n–rozměrné kouli S_n.

Poté musíme zvolit vektory $\mathbf{v}_1, \ldots, \mathbf{v}_m$ a $\mathbf{w}_1, \ldots, \mathbf{w}_n$. Tyto vektory zvolíme také stejně jako v předchozím schématu. Navíc ale vybereme náhodně index j_1 mezi všemi indexy j, pro které platí $\langle \mathbf{b}_j, \mathbf{u} \rangle \in 2\mathbb{Z} + 1$, kde $\mathbf{b}_j$ je vektor, který byl vybrán pro generování $\mathbf{v}_j$ (z rovnosti $\mathbf{v}_j = \mathbf{b}_j + \sum_{i=1}^{n} \mathbf{w}_i$). Platí tedy, že $\langle \mathbf{b}_j, \mathbf{u} \rangle$ je liché celé číslo. Veřejný klíč tedy sestává z vektorů $\mathbf{v}_1, \ldots, \mathbf{v}_m$ a $\mathbf{w}_1, \ldots, \mathbf{w}_n$ a indexu j_1.

3.1.3.2. Šifrování

Pro zašifrování "0" postupujeme přesně stejně jako v původním schématu. Zvolíme náhodně $a_1, \ldots, a_n \in \{0, 1\}$ a redukujeme vektor $\sum_{j=1}^{m} a_j \mathbf{v}_j$ modulo rovnoběžník $P(\mathbf{w}_1, \ldots, \mathbf{w}_n)$. Zašifrování "0" je tedy vektor

$$\mathbf{x} = \Big(\sum_{j=1}^{m} a_j \mathbf{v}_j \Big) \bmod P(\mathbf{w}_1, \ldots, \mathbf{w}_n) \ .$$

Rozdíl nastává ve chvíli, kdy chceme zašifrovat "1". Pro zašifrování "1" zvolíme náhodně čísla $a_1, \ldots, a_m \in \{0, 1\}$. Redukujeme vektor $\frac{1}{2}\mathbf{v}_{j_1} + \sum_{j=1}^{m} a_j \mathbf{v}_j$ modulo rovnoběžník P. Tedy platí, že zašifrování "1" je vektor

$$\mathbf{x} = \left(\frac{1}{2}\mathbf{v}_{j_1} + \sum_{j=1}^{m} a_j \mathbf{v}_j\right) \bmod P(\mathbf{w}_1, \ldots, \mathbf{w}_n) \ .$$

3.1.3.3. Dešifrování

Pokud máme šifrovaný text $\mathbf{x}$ a soukromý klíč $\mathbf{u}$, tak spočítáme $\gamma = \langle \mathbf{x}, \mathbf{u} \rangle$. Dešifrujeme "0", pokud je číslo γ vzdálené od celého čísla méně jak $\frac{1}{4}$. V opačném případě dešifrujeme "1".

Tvrzení 3.2 (Error–free decryption). *Pro každý bit $\sigma \in \{0, 1\}$, každou volbu soukromé i veřejné báze a každou volbu koeficientů a_j platí, že šifrovaný text $\boldsymbol{x}$ splňuje $\langle \boldsymbol{x}, \boldsymbol{u} \rangle \in \mathbb{Z} + \frac{\sigma}{2} \pm \frac{1}{n}$.*

3.2. Goldreich–Goldwasser–Halevi kryptosystém

Goldreich–Goldwasser–Halevi kryptosystém využívá jednosměrnou funkci, jejíž složitost plyne z implementace problému *CVP*. Tedy máme problém najít nejbližší bod mřížky $L(B)$ k nějakému zadanému terčovému bodu $\mathbf{w}$. Jako soukromý i veřejný klíč budeme používat dvě různé báze stejné mřížky.

3.2.1. Popis algoritmu

Pro konstrukci kryptosystému postavíme čtyři subalgortimy - Generate, Sample, Evaluate a Invert. Tento algoritmus je podrobněji popsán v [**5**], kde lze nalézt i některé experimentální výsledky.

3.2.1.1. Generate

V subalgoritmu Generate najdeme konstanty pro popis jednosměrné funkce a přidáváme k ní informaci o padacích dvířkách. Potřebujeme vygenerovat dvě báze B a R stejné hodnosti a kladné reálné číslo σ. První, co musíme určit, je hodnost obou matic, tedy hodnota n. Čím větší n, tím bezpečnější schéma se dá očekávat. Na druhou stranu ale tím také roste potřebný prostor a čas na výpočet. Podle [**5**] se dá předpokládat, že dimenze 250 - 300 je dostatečně veliká.

Nejprve vygenerujeme "soukromou bázi" R jako náhodnou matici. Tj. vybereme matici z uniformního rozdělení na $\{-l, \ldots, +l\}^{n \times n}$, kde l je nějaká celočíselná hranice. Podle [**5**] není l pro bezpečnost schématu nijak zásadně důležité, mnohem důležitější je zvolit dostatečně velké n.

Pak vytvoříme "veřejnou bázi". Pro vytvoření veřejné báze jsou možné dva přístupy. První možnost je brát z báze vektor po vektoru a přičíst k němu lineární kombinaci zbylých vektorů. Koeficienty lineární kombinace lze volit náhodně z $\{-1, 0, 1\}$ s vyšší pravděpodobností zvolení nuly. Druhá možnost je modifikovat všechny vektory současně pomocí násobení báze několika náhodnými maticemi nad $\{-1, 0, 1\}$. Bohužel v tomto případě narostou koeficienty veřejné báze velmi rychle a proto se tato metoda nepoužívá.

Na dvojici (B, σ) se díváme jako na popis jednosměrné funkce $f_{B,\sigma}$ a R je informace o padacích dvířkách.

3.2.1.2. Sample

Subalgoritmus SAMPLE je pouze pomocný algoritmus pro ukázku fungování celého kryptosystému. Vracíme dva vektory $\mathbf{v}$ a $\mathbf{e}$ z $\mathbb{R}^n$. Vektor $\mathbf{v}$ volíme náhodně v $\mathbb{Z}^n$. Například ho můžeme vytvořit tak, že každou jeho souřadnici postupně uniformně náhodně zvolíme z rozsahu $\{-n, \ldots, +n\}$. Tento vektor bude simulovat zprávu, kterou chceme zašifrovat. Vektor $\mathbf{e}$ je volen tak, že každou souřadnici volíme $+\sigma$ nebo $-\sigma$ s pravděpodobností $\frac{1}{2}$.

3.2.1.3. Evaluate

Subalgoritmus EVALUATE pouze určí hodnotu na základě daných vstupních parametrů B, σ, $\mathbf{v}$ a $\mathbf{e}$. Spočítá $\mathbf{w} = f_{B,\sigma}(\mathbf{v}, \mathbf{e}) = B\mathbf{v} + \mathbf{e}$.

3.2.1.4. Invert

Poslední subalgoritmus INVERT slouží, jak název napovídá, k invertování funkce za použití informace o padacích dvířkách. To znamená, že jako vstup dostaneme $\mathbf{w}$ a máme soukromou bázi R. Invertování bude probíhat tak, že si vektor $\mathbf{w}$ zapíšeme jako lineární kombinaci sloupců z R, pak zaokrouhlíme koeficienty na celá čísla a tím dostaneme vektor mřížky. Tento bod mřížky vyjádříme jako lineární kombinaci vektorů z veřejné báze B a tato kombinace je pak právě vektor $\mathbf{v}$. Nyní, když máme $\mathbf{v}$, můžeme dopočítat $\mathbf{e}$.

Formálně řečeno spočítáme $T = B^{-1}R$, pak můžeme dopočítat $\mathbf{v} = T\lceil R^{-1}\mathbf{w}\rfloor$, kde symbolem $\lceil \mathbf{a} \rfloor$ myslíme zaokrouhlení všech koeficientů $\mathbf{a}$ na nejbližší celá čísla. Nakonec pak máme $\mathbf{e} = \mathbf{w} - B\mathbf{v}$.

Aby byl subalgoritmus INVERT úspěšný s vysokou pravděpodobností, je potřeba vhodně zvolit σ. Pro určení hranice na σ využijeme Lemma 3.3.

LEMMA 3.3. *Nechť je R soukromá báze použitá na počítání inverze funkce $f_{B,\sigma}(\boldsymbol{v}, \boldsymbol{e})$. Chyba při invertování nastane, právě tehdy když $\lceil R^{-1}\boldsymbol{e}\rfloor \neq \vec{\boldsymbol{0}}$.*

Důkaz. Nechť je T matice přechodu, tedy $T = B^{-1}R$. Pak můžeme dopočítat $\mathbf{v} = T\lceil R^{-1}\mathbf{w}\rfloor$ a nakonec mám $\mathbf{e} = \mathbf{w} - B\mathbf{v}$. Je zřejmé, že pokud je $\mathbf{v}$ spočítáno správně, pak i $\mathbf{e}$ je spočítáno správně. Proto se budeme v důkazu věnovat pouze výpočtu vektoru $\mathbf{v}$. Připomeňme, že $\mathbf{w} = B\mathbf{v} + \mathbf{e}$, proto platí

$$\begin{aligned} T\lceil R^{-1}\mathbf{w}\rfloor &= T\lceil R^{-1}(B\mathbf{v}+\mathbf{e})\rfloor \\ &= T\lceil R^{-1}B\mathbf{v} + R^{-1}\mathbf{e}\rfloor \\ &= T\lceil (BT)^{-1}B\mathbf{v} + R^{-1}\mathbf{e}\rfloor \\ &= T\lceil T^{-1}\mathbf{v} + R^{-1}\mathbf{e}\rfloor\ . \end{aligned}$$

Ale protože T a $\mathbf{v}$ jsou celočíselné, tak i $T^{-1}\mathbf{v}$ je celočíselné a proto

$$\lceil T^{-1}\mathbf{v} + R^{-1}\mathbf{e}\rfloor = T^{-1}\mathbf{v} + \lceil R^{-1}\mathbf{e}\rfloor\ .$$

Pokud dosadíme tuto rovnost, dostáváme, že

$$T\lceil R^{-1}\mathbf{w}\rfloor = T(T^{-1}\mathbf{v} + \lceil R^{-1}\mathbf{e}\rfloor) = \mathbf{v} + T\lceil R^{-1}\mathbf{w}\rfloor\ .$$

Proto algoritmus Invert uspěje pouze v případě, že $\lceil R^{-1}\mathbf{w}\rfloor = \vec{\mathbf{0}}$.

□

Tvrzení 3.4. *Nechť je R soukromá báze použitá na počítání inverze funkce $f_{B,\sigma}(\boldsymbol{v}, \boldsymbol{e})$. Označme ρ jako maximální L_1 normu z řádků R^{-1}. Pak pokud platí $\sigma < \frac{1}{2\rho}$, nenastanou žádné chyby při invertování.*

Důkaz. Označme si řádky matice R^{-1} jako $\mathbf{b}_1, \ldots, \mathbf{b}_n$. Podle Lemmatu 3.3 chyba nenastane právě tehdy, když je $\lceil R^{-1}\mathbf{w}\rfloor = \vec{\mathbf{0}}$. To znamená, že aby algoritmus dal správný výsledek, musí pro každý řádek $\mathbf{b}_1, \ldots, \mathbf{b}_n$ platit, že $|\mathbf{b}_i \cdot \mathbf{e}| \doteq 0$. Souřadnice i–tého vektoru si označíme jako $\mathbf{b}_i(j)$ a víme, že souřadnice vektoru $\mathbf{e}(j)$ jsou v absolutní hodnotě rovny σ. Pak výše uvedený skalární součin můžeme zapsat jako

$$|\mathbf{b}_i \cdot \mathbf{e}| = |\sum_{j=1}^{n} \mathbf{b}_i(j) \cdot \mathbf{e}(j)| \leq \sum_{j=1}^{n} |\mathbf{b}_i(j) \cdot \mathbf{e}(j)| = \sum_{j=1}^{n} |\mathbf{b}_i(j)| \cdot |\mathbf{e}(j)| =$$

$$= \sum_{j=1}^{n} |\mathbf{b}_i(j)| \cdot \sigma = \sigma \sum_{j=1}^{n} |\mathbf{b}_i(j)| \leq \sigma \cdot \rho < \frac{1}{2}\ .$$

□

3.3. NTRU kryptosystém

Další kryptosystém, kterým se budeme zabývat je NTRU. Tento algoritmus byl představen v roce 1998 trojicí Jeffrey Hoffstein, Jill Pipher a Joseph Silverman v [**7**]. Bezpečnost tohoto kryptosystému závisí na kombinaci nezávislé redukce modulo dvě celá čísla a počítání v okruzích polynomů. Šifrování a dešifrování s NTRU probíhá velmi rychle a generování klíčů je rychlé a jednoduché. Konkrétně šifrování a dešifrování bloku o délce N proběhne v čase $\mathcal{O}(N^2)$, což je o řád rychlejší než RSA, které počítá v čase $\mathcal{O}(N^3)$. Na první pohled by se mohlo zdát, že měříme v různých rozměrech, protože N sice znamená v obou kryptosystémech délku bloku, ale ta je jiná RSA a jiná NTRU. Tento rozdíl ale můžeme zanedbat, protože se jedná o asymptotickou složitost a rozdíl je pouze o násobek konstantou.

Podrobnosti neuvedené v této práci včetně možností zrychlení nebo komentáře k útokům lze nalézt v [**22**].

3.3.1. Teoretické zázemí algoritmu

NTRU kryptosystém závisí na trojici celočíselných parametrů (N, p, q). Čísla p a q nemusí být prvočísla, ale budeme předpokládat, že $\mathrm{NSD}(p, q) = 1$ a $q \gg p$. Všechny základní operace budou probíhat v okruhu

$$R = \mathbb{Z}[X]/(X^N - 1) \ .$$

Element $a(X) \in R$ můžeme chápat jak jako polynom, tak jako vektor,

$$a(X) = \sum_{i=0}^{N-1} a_i X^i = [a_1, \ldots, a_{N-1}] \ .$$

Nyní nadefinujeme operaci dvou polynomů značenou jako '$*$'.

Definice 3.5. Nechť $a, b \in R$, pak $a * b = c$ s koeficienty

$$c_k = \sum_{i=0}^{N-1} a_i b_{k-i} + \sum_{i=k+1}^{N-1} a_i b_{N+k-i} = \sum_{i+j \equiv k \bmod N} a_i b_j \ .$$

Pokud provádíme operaci $*$ *modulo* q, myslíme tím, že všechny koeficienty násobku redukujeme modulo q.

Dále budeme potřebovat měřit velikost polynomu. Mohli bychom použít Euklidovskou normu $\|a\|^2 = \sum_{i=0}^{N-1} a_i$, ale pro naše potřeby bude vhodnější nadefinovat vlastní normu, která bude centrovaná.

DEFINICE 3.6. Nechť $a \in R$, pak *centrovaná norma vektoru* a je

$$\|a\|^2 = \sum_{i=0}^{N-1}(a_i - \mu_a)^2 \qquad \text{, kde } \mu_a = \frac{1}{N}\sum_{i=0}^{N-1} a_i \ .$$

Tato centrovaná norma je Euklidovská norma po provedení projekce na prostor kolmý k vektoru $(1, 1, \ldots, 1)$. Důvod, proč používáme právě takovouto normu, je ten, že útočník může vždy pracovat s centrovanými vektory, takže práce s necentrovanými normami nepřinese žádnou výhodu. V matematickém vyjádření vypadá redukce jako izomorfismus

$$\begin{aligned}
&\mathbb{Z}[X]/(X^N - 1) \sim \\
&\mathbb{Z}[X]/\{(X-1)(X^{N-1} + X^{N-2} + \ldots + X + 1)\} \sim \\
&\mathbb{Z}[X]/(X-1) \times \mathbb{Z}[X]/(X^{N-1} + X^{N-2} + \ldots + X + 1) \sim \\
&\qquad \sim \mathbb{Z} \ \times \ \mathbb{Z}[X]/(X^{N-1} + X^{N-2} + \ldots + X + 1) \ .
\end{aligned}$$

Tento rozklad také ukazuje, že je vhodné volit za N prvočíslo, nebo aspoň číslo, které nemá malé prvočíselné faktory, protože pak se vystavujeme riziku, že útočník dále rozloží okruh R.

Abychom mohli popsat propojení problémů založených na mřížkách a NTRU algoritmu, musíme definovat matici, která nám bude charakterizovat polynom (nebo vektor – podle toho, jak chápeme prvky okruhu R).

DEFINICE 3.7. *Konvoluční modulární mřížka* L_h asociovaná polynomu

$$h(X) = h_0 + h_1 X + h_2 X^2 + \ldots + h_{N-1} X^{N-1} \in R$$

je množina vektorů $(u, v) \in R \times R$ taková, že platí

$$v(X) \equiv h(X) * u(X) \bmod q \ .$$

Je zřejmé, že L_h je generovaná řádky následující matice

$$\left(\begin{array}{cccc|cccc}
1 & 0 & \ldots & 0 & h_0 & h_1 & \ldots & h_{N-1} \\
0 & 1 & \ldots & 0 & h_{N-1} & h_0 & \ldots & h_{N-2} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1 & h_1 & h_2 & \ldots & h_0 \\
\hline
0 & 0 & \ldots & 0 & q & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & 0 & q & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & 0 & 0 & \ldots & q
\end{array}\right)$$

LEMMA 3.8. *Konvoluční modulární mřížka je invariantní vůči rotaci. Tj. pokud* $(u, v) \in L_h$*, pak*

$$(X^i * u, X^i * v) \in L_h \text{ pro všechna } 0 \le i < N \ .$$

Navíc všechny tyto rotace mají stejnou centrovanou normu.

Důkaz. Dle definice platí, že $(u, v) \in L_h$ pokud $v(X) \equiv h(X) * u(X) \bmod q$. Pokud vynásobíme obě strany rovnice X^i, dostáváme

$$\begin{aligned} X^i * v(X) &\equiv X^i * h(X) * u(X) \bmod q \\ (X^i * v(X)) &\equiv h(X) * (X^i * u(X)) \bmod q \end{aligned}$$

A toho tedy plyne, že $(X^i * u, X^i * v) \in L_h$.

Nyní ukážeme, že norma se při rotaci nemění. Platí

$$\|u\|^2 = \sum_{j=0}^{N-1} (u_j - \mu_u)^2 \qquad \text{, kde } \mu_u = \frac{1}{N} \sum_{i=0}^{N-1} u_i \ .$$

Zvolme libovolný index i, kde $0 \leq i < N$. Označme si $w = X^i * u$. Pak platí, že $w_j \equiv u_{(j+i \bmod N)}$. Je tedy vidět, že $\mu_w = \mu_u = \frac{1}{N} \sum_{i=0}^{N-1} u_i$ nezávisle na volbě i. Norma pak je

$$\|w\|^2 = \sum_{j=0}^{N-1} (w_j - \mu_u)^2 = \sum_{j=0}^{N-1} (u_{(j+i \bmod N)} - \mu_u)^2 = \|u\|^2 \ .$$

□

Definice 3.9. Podmřížku generovanou dvojicí vektorů (u, v) a všemi rotacemi $(X^i * u, X^i * v)$ označíme jako $R * (u, v)$.

Definice 3.10. Polynom g je *malý polynom*, nebo alternativně má *malé koeficienty*, pokud jsou všechny jeho koeficienty $\mathcal{O}(1)$.

Definice 3.11. Pokud se polynom h dá rozložit tak, že

$$h \equiv f^{-1} * g \bmod q \ ,$$

kde polynomy f a g mají malé koeficienty. Pak říkáme, že L_h je *NTRU mřížka*. Označíme ji L_h^{NT}.

Poznámka 3.12. Pomocí Gaussovy heuristiky můžeme odhadnout některé vlastnosti náhodné mřížky.

- Můžeme odhadnout velikost nejkratšího vektoru mřížky L velké dimenze.
$$\lambda_{Gauss}(L) = \sqrt{dim(L)/2\pi\varepsilon} \cdot Det(L)^{1/dim(L)} \ .$$
- Obecná konvoluční modulární mřížka L_h má dimenzi $2N$ a determinant q^N, proto bude pravděpodobně nejkratší vektor velikosti
$$\lambda_{Gauss}(L_h) = \sqrt{Nq/\pi\varepsilon} = \mathcal{O}(N) \ .$$
- Mějme NTRU mřížku L_h^{NT}. Polynom h má rozklad $h = f^{-1} * g \bmod q$, kde polynomy f a g mají malé koeficienty. Pak tedy L_h^{NT} obsahuje krátké vektory (f, g). Obecněji tedy platí, že všechny rotace $(X^i * f, X^i * g)$ jsou krátké vektory v L_h^{NT} mající délku $\mathcal{O}(\sqrt{N})$.
- Na základě předchozích bodů lze předpokládat, že tyto vektory budou pravděpodobně o $\mathcal{O}(\sqrt{N})$ kratší než všechny ostatní vektory v podmřížce $R * (f, g)$.

Definice 3.13. Nechť L_h^{NT} je NTRU mřížka. Potom *NTRU problém založený na mřížkách* je najít vektor délky $\mathcal{O}(\sqrt{N})$ v mřížce L_h^{NT}.

Polynom h budeme považovat za veřejný klíč a dvojici (f, g) za soukromý klíč. Polynom h nám tedy určuje NTRU mřížku L_h^{NT} dimenze $2N$. Tato mřížka obsahuje soukromý klíč (f, g). Dle Poznámky 3.12 víme, že tyto vektory jsou malé. Pokud je f tvaru $1 + pF_p$, kde F_p je inverz k f modulo p, pak útok pomocí mřížek na soukromý klíč obsahuje vyřešení *CVP*. Pokud f není tohoto tvaru, pak bychom museli vyřešit instanci SVP_γ.

3.3.2. Popis algoritmu – NTRU šifrování

Konkrétní popis algoritmu rozdělíme na tři části – generování klíčů, šifrování a dešifrování.

3.3.2.1. Generování klíčů

Pro použití NTRU kryptosystému potřebujeme vygenerovat dva náhodné polynomy s malými koeficienty f a g. Polynom f musí splňovat podmínku, že k němu existuje inverzní polynom modulo q i modulo p. Dále platí $\|f\| = \|g\| = \mathcal{O}(\sqrt{N})$. Pro vhodné parametry bude polynom f invertibilní ve většině případů, a výpočet inverzních polynomů je pouze použití Euklidova algoritmu. Označme si příslušné inverzy F_q a F_p. Tj. platí

$$F_q * f \equiv 1 \bmod q \quad \text{a} \quad F_p * f \equiv 1 \bmod p.$$

Když máme tyto polynomy, pak můžeme spočítat veřejný klíč polynom h.

$$h \equiv f^{-1} * g \bmod q = F_q * g \bmod q$$

Soukromý klíč je polynom f, ale vyplatí se uchovávat i polynom F_q.

3.3.2.2. Šifrování

Šifrování je v kryptosystému NTRU velmi jednoduchý proces. Pokud máme zprávu m a chceme ji zašifrovat, potřebujeme pouze veřejný klíč h, číslo p a náhodně zvolený polynom s. Šifrovanou zprávu e pak spočítáme

$$e \equiv p \cdot s * h + m \bmod q.$$

3.3.2.3. Dešifrování

Dešifrování je na první pohled výpočetně složitější, protože probíhá dvakrát násobení polynomů, ale při podrobnějším zkoumání vidíme, že druhé násobení je pouze modulo p a víme, že $q \gg p$. Koeficienty při druhém násobení budou tedy malé a proto bude násobení rychlé. Pro dešifrování zprávy e potřebujeme pouze

soukromý klíč f a pro zvýšení efektivity polynom F_q, který bychom jinak museli znovu počítat při každém dešifrování.

Nejprve spočítáme polynom a následujícím způsobem

$$a \equiv f * e \bmod q.$$

Koeficienty polynomu a pak budeme brát z intervalu od $-q/2$ do $q/2$. Původní zprávu dostaneme vynásobením

$$F_q * a \bmod p.$$

Tvrzení 3.14. *Tento systém dešifrování dá původní zprávu m.*

Důkaz. Polynom a, který spočítáme splňuje následující ekvivalence

$$\begin{aligned} a &\equiv f * e \bmod q \\ &\equiv f * (p \cdot s * h + m \bmod q) \bmod q \\ &\equiv f * p \cdot s * h + f * m \bmod q \\ &\equiv f * p \cdot s * F_q * g + f * m \bmod q \\ &\equiv (f * F_q) * p \cdot s * g + f * m \bmod q \\ &\equiv p \cdot s * g + f * m \bmod q \end{aligned}$$

Uvažujme nyní polynom $p \cdot s * g + f * m \bmod q$. Pokud tento polynom redukujeme modulo p, dostáváme, $f * e \bmod p$ a když tento polynom vynásobíme polynomem F_p, pak dostaneme původní zprávu m. □

3.3.3. Popis algoritmu – NTRU-Podpis

Pro některé kryptosystémy s veřejným klíčem je velmi jednoduché udělat ze šifrovacího schématu podepisovací a naopak. Využije se vlastnosti takovýchto kryptosystémů, že člověk, který chce podepisovat nebo dešifrovat, použije klíče, který zná jen on sám. A na druhou stranu, pokud chce kdokoliv ověřit podpis, stačí mu k tomu veřejný klíč, tedy stejný postup, jako kdyby odesílal šifrovanou zprávu. NTRU nepatří mezi takovéto systému. NTRU-Podpis je samostatný algoritmus publikovaný v roce 2003 v [**8**], který ani nevyužívá stejné parametry jako původní NTRU kryptosystém. NTRU-Podpis potřebuje pouze dva parametry N a q.

3.3.3.1. Generování klíčů

Stejně jako v NTRU kryptosystému potřebujeme pro podepisování a ověřování vygenerovat dva náhodné polynomy s malými koeficienty f a g. Protože ale v algoritmu NTRU-Podpis používáme jen jedno prvočíslo, polynom f musí splňovat pouze jednu podmínku, že existuje inverzní f^{-1} modulo q. Veřejný klíč je stejný jako v původním NTRU kryptosystému.

$$h \equiv f^{-1} * g \bmod q$$

K privátnímu klíči potřebujeme ještě dopočítat polynomy F a G, které splňují

$$f * G - g * F = q \quad \text{a zároveň} \quad \|F\| = \|G\| = \mathcal{O}(N).$$

Platí tedy $F^{-1} * G \equiv h \bmod q$. Rotací (f, g) a (F, G) získáme bázi mřížky L_h^{NT}.

3.3.3.2. Podepisování

Dokument D, který chceme podepisovat, je vhodné nejdříve hashovat. Pokud máme hash $m = (m_1, m_2)$ složený ze dvou náhodných polynomů modulo q, můžeme zprávu podepsat. Podpisem dokumentu D bude vektor (s, t), který je blízko zprávě m. Tento vektor najdeme tak, že $m = (m_1, m_2)$ vyjádříme jako $\mathbb{Q}$–lineární kombinaci krátkých vektorů z báze a pak zaokrouhlíme koeficienty na nejbližší celá čísla. Právě v tomto kroku využijeme polynomy F a G. Použijeme rovnost

$$(m_1, m_2) = \varphi * (f, g) + \psi * (F, G),$$

kde, $\varphi, \psi \in \mathbb{Q}[x]$. Pokud bude platit $\varphi = A + B$ a $\psi = a + b$ pro B, b polynomy s celými koeficienty a A, a polynomy s koeficienty mezi $-q/2$ a $q/2$, podpis potom spočítáme

$$\begin{aligned} s &\equiv f * B + F * b \bmod q \\ t &\equiv g * B + G * b \bmod q. \end{aligned}$$

Lze také psát zkráceně, protože $(s, t) \in L_h^{NT}$

$$(s, t) = B * (f, g) + b * (F, G).$$

Jak ale spočítáme pomocné vektory $a, b, A, B \in \mathbb{Z}[X]/(X^N - 1)$? Pokud tuto rovnost rozepíšeme po složkách, dostáváme

$$\begin{aligned} m_1 &= \varphi * f + \psi * F \\ m_2 &= \varphi * g + \psi * G \end{aligned}$$

Dále budeme upravovat první rovnost, nejdříve ji celou vynásobíme G a pak využijeme rovnosti $f * G - g * F = q$ až nakonec dostaneme vyjádření pomocí m_2

$$\begin{aligned} m_1 &= \varphi * f + \psi * F && / * G \\ m_1 * G &= \varphi * f * G + \psi * F * G \\ &= \varphi * (q + g * F) + \psi * F * G \\ &= \varphi * q + \varphi * g * F + \psi * F * G \\ &= \varphi * q + F * (\varphi * g + \psi * G) \\ &= \varphi * q + F * m_2 \end{aligned}$$

Protože $\varphi \in \mathbb{Q}[x]$, potřebujeme mu zaokrouhlit koeficienty na nejbližší celá čísla. To se nejlépe vyjádří tak, že $\varphi * q$ rozepíšeme na součet $A + q * B$, kde A bude mít koeficienty mezi $-q/2$ a $q/2$. Tedy máme

$$m_1 * G = A + q * B + F * m_2 \ .$$

Podobnými úpravami rovnosti $m_2 = \varphi * g + \psi * G$ dostáváme

$$\begin{aligned} G * m_1 - F * m_2 &= A + q * B, \\ -g * m_1 + f * m_2 &= a + q * b, \end{aligned}$$

kde mají polynomy A, a koeficienty mezi $-q/2$ a $q/2$ a B, b jsou polynomy s celými koeficienty.

3.3.3.3. Ověřování

Nechť s je údajný podpis z algoritmu NTRU-Podpis pro zprávu $m = (m_1, m_2)$ a nechť h je veřejný klíč. Podpis bude uznán ověřeným pouze tehdy, pokud ten, kdo podepisuje a jehož podpis se má ověřit, prokáže znalost bodu mřížky L_h^{NT}, který je blízko zpráv m. Verifikaci tedy provedeme ve dvou krocích

(1) Spočítáme polynom $t \equiv h * s \bmod q$. (Uvědomme si, že (s, t) je bod mřížky L_h^{NT})
(2) Dále pak musíme spočítat vzdálenost (s, t) od (m_1, m_2) a ověřit, zda je menší než předem zadaná hodnota.

KAPITOLA 4

Mřížky v kryptoanalýze

Kromě toho, že existují kryptosystémy, které jsou založeny na mřížkách, lze mřížky použít i v opačném smyslu k prolamování kryptosystémů. Díky algoritmu LLL (Lenstra-Lenstra-Lovász), který byl představen v [**10**], bylo první využití mřížek nikoliv, že by vznikaly nové kryptosystémy, ale spíše takové, že mnoho systémů bylo prolomeno. Jedno z nejpřirozenějších využití mřížek, konkrétně jejich redukce, tedy algoritmu LLL, je hledání malých kořenů lineárních rovnic o více neznámých. Mezi nejslavnější aplikace LLL patří vyřešení HNP (hidden number problem), my se ale budeme věnovat nejdříve knapsack kryptosystému.

4.1. Knapsack kryptosystém

Jeden z prvních útoků, při kterém bylo použito redukce mřížek, je útok na knapsack kryptosystém. V této části si ukážeme jak útok na původní Merkle-Hellmanovo schéma, tak metodu, jak vyřešit téměř každý knapsack problém s tzv. malou hustotou.

Nejdříve definujeme knapsack problém. Definice 4.1 a 4.2 jsou ekvivalentní, pouze v druhé je pro přehlednost použito značení pomocí vektorů.

Definice 4.1. Mějme posloupnost přirozených čísel $a_1, \ldots, a_n$ a přirozené číslo s, pak *knapsack problém* je nalézt čísla $x_1, \ldots, x_n \in \{0, 1\}$ takové, že splňují $s = \sum_{i=1}^{n} x_i a_i$. Čísla $a_1, \ldots, a_n$ označíme jako *posloupnost knapsack vah* a číslo s jako *cíl*.

Definice 4.2. Mějme vektor $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ a $s \in \mathbb{N}$, pak *knapsack problém* je nalézt vektor $\mathbf{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ takové, že $s = \mathbf{x} \cdot \mathbf{a}$. Vektor $\mathbf{a}$ pak označujeme jako *váhový vektor* a s jako *cíl*.

Knapsack problém je NP-úplný, přesto ale existují případy, které jsou řešitelné v polynomiálním čase. Jeden z velmi známých příkladů jednoduchého knapsack problému je použití superrostoucí posloupnosti vah. Superrostoucí posloupností rozumíme takovou posloupnost, že každý následující člen posloupnosti je ostře větší než součet předchozích členů. Tj. $a_j > \sum_{i=1}^{j-1} a_j$ pro každé $j = 1, \ldots, n$. V takovém případě mohou být všechna x_i postupně dopočítána od x_n až po x_1

pomocí podmínky

$$x_i = 1 \quad \Leftrightarrow \quad s - \sum_{j=i+1}^{n} x_j a_j \geq a_i \ .$$

4.1.1. Merkle–Hellman kryptosystém

Merkle–Hellmanův kryptosystém byl prezentován jako jeden z prvních kryptosystémů s veřejným klíčem již roku 1978. Fakticky to byla jediná alternativa k RSA až do roku 1982, kdy byl předveden útok na jednoduché implementace Merkle–Hellmanova kryptosystému. Stejný rok pak byl prezentován i útok, který obsahoval využití LLL algoritmu. Kryptoanalýza Merkle–Hellmanova kryptosystému bylo vůbec prvním použitím mřížek v kryptografii.

4.1.1.1. Popis kryptosystému

Merkle–Hellmanův kryptosystém je založen na převedení jednoduchého knapsack problému se superrostoucí posloupností vah na systém, kde by nemělo jít invertovat transformaci bez znalosti soukromého klíče.

Nechť je $\mathbf{a} = (a_1, \ldots, a_n)$ je váhový vektor, kde $a_1, \ldots, a_n$ je superrostoucí posloupnost vah a nechť M a W jsou čísla splňující

$$M > \sum_{i=1}^{n} a_i \quad \text{a zároveň} \quad \mathrm{NSD}(M, W) = 1 \ .$$

Merkle a Hellman vytvořili novou posloupnost vah $\mathbf{b}$.

$$b_i = W \cdot a_i \bmod M \quad 1 \leq i \leq n \ .$$

Veřejný klíč jsou právě tyto nové váhy $\mathbf{b}$. Soukromý klíč sestává z čísel M, W a z původních vah $\mathbf{a}$.

Šifrování zprávy m probíhá pouhým vynásobením. Zprávu m můžeme považovat za vektor $\mathbf{x}$ z Definice 4.2. Tedy šifrovanou zprávu c dostaneme $c = m \cdot \mathbf{b}$. K dešifrování pak potřebujeme vyřešit knapsack problém s vahami $\mathbf{b}$ a cílem c. Díky znalosti soukromého klíče můžeme tento problém převést na původní knapsack váhovým vektorem $\mathbf{a}$ a cílem $c' = c \cdot W^{-1} \bmod M$, který lze jednoduše vyřešit.

$$c' = c \cdot W^{-1} \equiv (m \cdot \mathbf{b})W^{-1} \equiv m(\mathbf{b} \cdot W^{-1}) \equiv m \cdot \mathbf{a} \bmod M$$

4.1.1.2. Kryptoanalýza

Kryptoanalýzu Merkle–Hellmanova kryptosystému popíšeme jen v základních bodech. Podrobnosti lze najít v [**11**] nebo v [**12**]. Útok na toto schéma je založen na redukci mřížek, konkrétně využití LLL, a diofantických aproximacích.

Nejdříve z ekvivalence $b_i = W \cdot a_i \bmod M$ definujeme celá čísla k_i tak, že $b_i W^{-1} - k_i M = a_i$ pro $i = 1, \ldots, n$. Z této rovnice nám pak vychází

$$\frac{W^{-1}}{M} - \frac{k_i}{b_i} = \frac{a_i}{b_i M} \ .$$

Vidíme tedy, že každé $\frac{k_i}{b_i}$ je dobrá aproximace $\frac{W^{-1}}{M}$.

Další krok je uvědomit si, že každý knapsack problém lze převést na jednoduchý knapsack se superrostoucí posloupností vah. Podrobnosti jsou uvedeny v Tvrzení 4.3, které je shrnutím výsledků z [**13**] a [**14**].

Tvrzení 4.3. *Nechť jsou M, $\boldsymbol{b}$, k_i definovány jako výše a $U = W^{-1}$. Potom existuje $\epsilon > 0$ takové, že pokud $\frac{U'}{M'}$ je zlomek splňující $|\frac{U'}{M'} - \frac{U}{M}| \leq \epsilon$, pak posloupnost vah $\boldsymbol{a'} = (a'_1, \ldots, a'_n)$, kde $a'_i = b_i U' - k_i M'$ pro $i = 1, \ldots, n$, je superrostoucí.*

Pokud tedy nalezneme k_i definované výše, pak $\frac{k_i}{b_i}$ může být použito pro splnění podmínek Tvrzení 4.3, tedy nalezení vhodného $\frac{U'}{M'}$. Z Tvrzení 4.3 pak plyne, že dokážeme najít superrostoucí posloupnost knapsack vah a tak prolomit celý šifrovací systém.

Nyní tedy potřebujeme najít k_i. Podrobnosti jsou uvedeny v [**12**]. Prvním krokem je uvědomit si, že každé $\frac{k_i}{k_1}$ je dobrá aproximace $\frac{b_i}{b_1}$. Tato aproximace je tzv. UGSDA (unusually good simultaneous Diophantine approximation). Úkolem tedy je pro nějaké $t \leq n$ najít UGSDA posloupnosti $(\frac{b_2}{b_1}, \ldots, \frac{b_t}{b_1})$. Z té jsme pak schopni dopočítat k_1.

Pro nalezení UGSDA definujme mřížku L generovanou řádky matice B.

$$B = \begin{pmatrix} b_2 & b_3 & \ldots & b_t & \lfloor b_1^{1/t} \rfloor \\ -b_1 & & & & \\ & -b_1 & & & \\ & & \ddots & & \\ & & & -b_1 & \end{pmatrix}$$

Použitím LLL algoritmu pro dostatečně velké t nalezneme malý vektor $\mathbf{w}$ tvaru $(v_1 b_2 - v_2 b_1, \ldots, v_1 b_t - v_t b_1, v_1 \lfloor b_1^{1/t} \rfloor)$. Protože se jedná o nejmenší vektor nalezený LLL algoritmem, pak jeho prvních $t-1$ souřadnic splňuje

$$|v_1 b_i - v_i b_1| \leq 2^{(t-1)/4} a^{-(t+1)/t^2} \leq a^{-1/(t-1)} \ .$$

Proto vektor určený souřadnicemi $(\frac{v_2}{v_1}, \ldots, \frac{v_t}{v_1})$ je blízko $(\frac{b_2}{b_1}, \ldots, \frac{b_t}{b_1})$. A protože jsou UGSDA vzácné, tak se dá předpokládat, že $v_i = k_i$ pro $i = 1, \ldots, n$.

I přes úspěšnou kryptoanalýzu Merkle–Hellmanova kryptosystému výzkum v oblasti knapsack kryptosystému neustával. Pokračoval zejména z toho důvodu, že implementace takových systémů je jednoduchá a rychlá. Ale všechny vzniklé systémy byly prolomeny často i s využitím mřížek.

4.1.2. Knapsack systémy s malou hustotou

Další druh knapsack problémů, který byl prolomen na základě mřížek je tzv. knapsack problém s malou hustotou.

DEFINICE 4.4. Nechť je $\mathbf{a} = (a_1, \ldots, a_n)$ váhový vektor knapsack problému a nechť A je maximum těchto vah. Pak *hustota* d vah je definována jako $\frac{n}{\log_2 A}$.

Budeme uvažovat knapsack problém s váhovým vektorem $\mathbf{a} = (a_1, \ldots, a_n)$, cílem s a řešením $\mathbf{x} = (x_1, \ldots, x_n)$. Podívejme se nyní na mřížku L_0 generovanou řádky matice M_0, kde $N > \sqrt{n}$ nějaké celé číslo.

$$M_0 = \begin{pmatrix} 1 & & & & Na_1 \\ & 1 & & & Na_2 \\ & & \ddots & & \vdots \\ & & & 1 & Na_n \\ & & & & Ns \end{pmatrix}$$

Pokud zadefinujeme vektor $\mathbf{c_0}$ jako $(x_1, \ldots, x_n, -1) \in \mathbb{Z}^{n+1}$, pak $\mathbf{x_0} = \mathbf{c_0} M_0$ je vektor mřížky L_0. Protože s je cíl a tedy $s = \sum_{i=1}^{n} x_i a_i$, pak platí

$$\mathbf{x_0} = \mathbf{c_0} M_0 = (x_1, \ldots, x_n, -1) M_0 = \Big(x_1, \ldots, x_n, N(x_1 a_1 + \ldots + x_n a_n - s)\Big) = (x, 0).$$

Vidíme tedy, že nalezením $\mathbf{x_0}$ v mřížce L_0 dokážeme spočítat i $\mathbf{x}$. Pokud předpokládáme, že maximálně polovina všech x_i je nenulových, pak je tento vektor malý tj. $\|\mathbf{x_0}\|^2 \leq \frac{1}{2} n$. Tímto způsobem lze podle [**15**] vyřešit téměř všechny knapsack problémy s velkým n a hustotou $d < 0,6463...$.

Tato hranice na hustotu byla později v [**16**] zvýšena na $d < 0,9408...$ použitím mřížky L_1 generovanou řádky matice M_1.

$$M_1 = \begin{pmatrix} 1 & & & & Na_1 \\ & 1 & & & Na_2 \\ & & \ddots & & \vdots \\ & & & 1 & Na_n \\ \frac{1}{2} & \frac{1}{2} & \ldots & \frac{1}{2} & Ns \end{pmatrix}$$

Budeme používat stejný vektor jako v předchozím případě, který označíme $\mathbf{c_1}$ (tj. $\mathbf{c_1} = (x_1, \ldots, x_n, -1) \in \mathbb{Z}^{n+1}$). Pak $\mathbf{x_1} = \mathbf{c_1} M_1$ je vektor mřížky L_1.

$$\mathbf{x_1} = \Big(x_1 - \frac{1}{2}, \ldots, x_n - \frac{1}{2}, N(x_1 a_1 + \ldots + x_n a_n - s)\Big) = (x_1 - \frac{1}{2}, \ldots, x_n - \frac{1}{2}, 0).$$

Protože $x_i \in \{0, 1\}$, pak všechny souřadnice vektoru $\mathbf{x_1}$ jsou z $\{-\frac{1}{2}, 0, \frac{1}{2}\}$ pro všechna $1 \leq i \leq n$. V tomto případě, nezávisle na původních hodnotách x_i, vektor $\mathbf{x_1}$ splňuje $\|\mathbf{x_1}\|^2 \leq \frac{1}{4} n$. Vektor $\mathbf{x_1}$ je tedy malým vektorem v mřížce L_1.

Samozřejmě, že v současné době neexistuje žádné orákulum *SVP*, které by našlo krátké vektory $\mathbf{x_0}$ nebo $\mathbf{x_1}$, ale ve skutečnosti stačí pouze LLL algoritmus.

4.2. Hidden number problem

Hidden number problem (tj. problém skrytého čísla, ale český překlad se nepoužívá) je první využití mřížek v "pozitivním" smyslu (důkaz, že nejvýznamnější bity v soukromém klíči Diffie-Hellmanova schématu jsou těžké bity). Nás ale bude zajímat především aplikace v kryptoanalýze. Nejčastější použití se týká případů, kdy máme o tajných údajích nějaké částečné informace, např. z postranních kanálů. Popis kryptografických aplikací byl publikován v roce 2002 v [**17**].

Nyní zadefinujeme pojem nejvýznamnějších bitů (most significant bits – *MSB*) a instanci hidden number problému – *HNP*.

Definice 4.5. Nechť je $x \in \mathbb{Z}$, $p \in \mathbb{N}$. Označme zbytek x po dělení číslem p jako $\lfloor x \rfloor_p$ (tj. $\lfloor x \rfloor_p = x \bmod p$). Nechť $\ell > 0$, potom $MSB_{\ell,p}(x)$ je celé číslo u, které splňuje

$$\big|\lfloor x \rfloor_p - u\big| \leq \frac{p}{2^{\ell+1}} \ .$$

Pokud je ℓ celé číslo, pak Definice 4.5 určuje ℓ nejvýznamnějších bitů čísla x ze $\mathbb{Z}_p$. Ve skutečnosti je ale definice flexibilnější, protože ℓ nemusí být celé číslo.

Definice 4.6. Označením *HNP* rozumíme problém nalezení čísla $\alpha \in \mathbb{Z}_p$ takového, že pro k čísel $t_1, \ldots, t_k \in \mathbb{Z}_p^*$ zvolených nezávisle a náhodně, a pro nějaké $\ell > 0$, máme dány dvojice

$$\big(t_i, MSB_{\ell,p}(\alpha t_i)\big) \ , \quad i = 1, \ldots, k \ .$$

Při řešení *HNP* budeme uvažovat vektor $\mathbf{w} = (\lfloor \alpha t_1 \rfloor_p, \ldots, \lfloor \alpha t_k \rfloor_p, \alpha/2^{\ell+1})$, který náleží do $(k+1)$-dimenzionální mřížky L. Mřížka L je generována řádky následující matice M.

$$M = \begin{pmatrix} p & & & & \\ & p & & & \\ & & \ddots & & \\ & & & p & \\ t_1 & t_2 & \ldots & t_k & \frac{1}{2^{\ell+1}} \end{pmatrix}$$

Pokud nyní označíme $a_i = MSB_{\ell,p}(\alpha t_i)$ pro všechna $i = 1, \ldots, k$, pak vektor $\mathbf{a} = (a_1, \ldots, a_k, 0)$ je velmi blízko mřížce L, protože je velmi blízko vektoru $\mathbf{w}$. Ve skutečnosti je tato vzdálenost řádově $p/2^\ell$.

Tvrzení 4.7. *Existuje polynomiální algoritmus, který, když dostane na vstupu r-dimenzionální mřížku L a vektor $\boldsymbol{v} \in \mathbb{R}^r$, pak najde vektor $\boldsymbol{u} \in L$, který splňuje nerovnost*

$$\|\boldsymbol{u} - \boldsymbol{v}\| \leq 2^{\mathcal{O}(r \log^2 \log r / \log r)} \min\{\|\boldsymbol{x} - \boldsymbol{v}\|, \ \boldsymbol{x} \in L\} \ .$$

Důkaz. Jedná se pouze o kombinaci dosažených výsledků v redukci báze mřížek. □

Použitím Tvrzení 4.7 pro $r = k+1$ a $\mathbf{v} = \mathbf{a}$ nalezneme vektor $\mathbf{u}$. Doufáme, že takto jsme našli $\mathbf{u} = \mathbf{w}$, ze kterého můžeme dál najít α. Proto, abychom mohli tvrdit, že jsme tímto postupem našli $\mathbf{w}$, musíme ukázat, že algoritmus může najít pouze pro zanedbatelné množství k-tic $(t_1, \ldots, t_k)$ vektor $\mathbf{u} \neq \mathbf{w}$, který splňuje

$$\|\mathbf{u} - \mathbf{a}\| \leq 2^{\mathcal{O}((k+1)\log^2\log(k+1)/\log(k+1))} \min\{\|\mathbf{x} - \mathbf{a}\|,\ \mathbf{x} \in L\} \ .$$

Pro zjednodušení zápisu označme $2^{\mathcal{O}((k+1)\log^2\log(k+1)/\log(k+1))} p/2^\ell = K$. Předpokládejme, že $\min\{\|\mathbf{x} - \mathbf{a}\|,\ \mathbf{x} \in L\} \leq p/2^\ell$. Pak tedy platí

$$\|\mathbf{u} - \mathbf{a}\| \leq K \ .$$

Z poslední nerovnosti plyne, že hledáme vektory $\mathbf{u} \neq \mathbf{w}$, pro které platí

$$\|\mathbf{u} - \mathbf{w}\| \leq K \ .$$

Z tvaru matice M vyplývá, že každý vektor mřížky L, tedy i vektor $\mathbf{u}$, je tvaru

$$\mathbf{u} = \big(\beta t_1 - \gamma_1 p, \ldots, \beta t_k - \gamma_k p, \beta/2^{\ell+1}\big),$$

pro nějaká čísla $\gamma_1, \ldots, \gamma_k$ a β. Aby mohl vektor $\mathbf{u}$ tohoto tvaru splňovat výše uvedenou nerovnost, musí být každá souřadnic vektoru $\mathbf{u} - \mathbf{w}$ menší než K. Konkrétně tedy musí platit

$$(\alpha - \beta) t_i \equiv y_i \bmod p$$

pro nějaké $y_i \in [-K, K]$. Zbývá nám ještě určit pravděpodobnost výskytu takovýchto $y \in \mathbb{Z}_p$ pro $\lambda \neq 0$,

$$\Pr_{y \in Z_p}(\lambda t \equiv y \bmod p \quad | \ y \in [-K, K]) \leq \frac{2K+1}{p} \ .$$

To znamená, že pravděpodobnost P, že prvních k souřadnic vektoru $\mathbf{u} - \mathbf{w}$ splňuje danou ekvivalenci alespoň pro jedno $\beta \neq \alpha$ je zhora ohraničena

$$P \leq (p-1)\left(\frac{2K+1}{p}\right)^k \leq p\left(\frac{3K}{p}\right)^k = \frac{p}{2^{\ell k}}\ 2^{\mathcal{O}(k(k+1)\log^2\log(k+1)/\log(k+1))} \ .$$

Vhodným zvolením parametrů ℓ a k tedy docílíme, že pravděpodobnost, že $\mathbf{w}$ je jediný vektor blízko $\mathbf{a}$ se exponenciálně blíží k jedné. Proto aproximační algoritmus *CVP* vrátí skoro vždy vektor $\mathbf{w}$ a pokud známe $\mathbf{w}$, není problém z poslední souřadnice $\alpha/2^{\ell+1}$ dopočítat α.

Vhodně zvolené parametry jsou podle [**17**] například

$$\ell = \left\lceil C\ \frac{(\log p \log\log\log p)^1/2}{\log\log p} \right\rceil \quad \text{pro libovolné } C > 0 \quad \text{a} \quad k = \left\lceil \frac{3\log p}{\ell} \right\rceil \ .$$

KAPITOLA 5

RSA vs. NTRU

Tato kapitola obsahuje srovnání rychlosti dvou kryptosystémů RSA a NTRU. Vzhledem k důležitosti a rozšířenosti asymetrické kryptografie je v dnešní době nutné, aby kryptosystémy byly nejen bezpečné, ale také dostatečně rychlé, protože velikost šifrovaných zpráv významně roste.

Právě testování rychlosti šifrování a dešifrování obou zmiňovaných kryptosystémů je cílem této kapitoly. Při teoretickém srovnání je NTRU o řád rychlejší. Konkrétně šifrování a dešifrování bloku o délce N proběhne v čase $\mathcal{O}(N^2)$ narozdíl od RSA, které počítá v čase $\mathcal{O}(N^3)$. Oba kryptosystémy jsou implementovány podle jejich matematických základů pouze s využitím knihovny NTL. Kvůli objektivnímu srovnání samotných kryptosystémů nejsou použita žádná jiná vylepšení, ačkoliv jsou u obou algoritmů možná, jak z hlediska implementace, tak z hlediska hardwarové akcelerace. RSA bylo programováno naprosto přímočaře, pro NTRU jsme využili částečně strukturu prezentovanou v [**18**].

Budeme postupně testovat rychlost generování klíčů pro různé úrovně bezpečnosti. Dále budeme testovat rychlost šifrování a dešifrování a to nejen pro různou úroveň bezpečnosti, ale i pro různě veliké zprávy. Zprávy pro šifrování budou náhodné, pouze se zadanou velikostí, stejné pro oba kryptosystémy.

Testovány budou tři úrovně bezpečnosti. Délky klíčů jsou uvedeny v Tabulce 1 včetně ohodnocení bezpečnosti pomocí MIPS roků (MIPS = milion instructions per second, MIPS rok = počet kroků provedených za jeden rok, při vykonání milionu instrukcí za vteřinu).

Kryptosystém	Úroveň bezpečnosti	Předpokládaný čas prolomení
RSA	512 bitů	10^5 MIPS roků
NTRU	N = 167	10^6 MIPS roků
RSA	1024 bitů	10^{12} MIPS roků
NTRU	N = 263	10^{14} MIPS roků
RSA	4096 bitů	10^{33} MIPS roků
NTRU	N = 503	10^{35} MIPS roků

Tabulka 1. Srovnání bezpečnosti RSA a NTRU

5.1. Implementace

Pro implementaci byly použity následující moduly knihovny NTL. Dokumentaci ke knihovně lze najít v [**19**].

- `ZZ` velká čísla
  - obsahuje aritmetiku velkých čísel
  - využito především v RSA pro uložení parametrů, ale i v NTRU, protože některé další datové struktury potřebují vstup právě v tomto formátu
- `ZZ_p` čísla modulo p
  - obsahuje aritmetiku v $\mathbb{Z}_p$
  - nejdříve je potřeba zadefinovat modul p příkazem `ZZ_p::init(p)`, a poté probíhají veškeré operace s proměnnými typu `ZZ_p` pouze modulo p
  - využito pouze v RSA, modul nastaven na $\varphi(N)$ při generování klíče a na N při šifrování a dešifrování
- `ZZX` polynomy nad `ZZ`
  - obsahuje aritmetiku polynomů nad celými čísly, včetně modulární aritmetiky pro počítání modulo f
  - využito pouze v NTRU při dešifrování, kdy je potřeba počítat se zápornými koeficienty
- `ZZ_pX` polynomy nad `ZZ_p`
  - obsahuje veškerou aritmetiku polynomů nad $\mathbb{Z}_p$, včetně modulární aritmetiky pro počítání modulo f, tato modulární aritmetika může být urychlena pomocí předpočítání informací o f
  - nejdříve je potřeba zadefinovat modul p příkazem `ZZ_p::init(p)`, a poté probíhají veškeré operace s proměnnými typu `ZZ_pX` s koeficienty modulo p
  - využito pouze v NTRU
- `tools`
  - další přídavné funkce, využito pouze funkce na měření času v obou algoritmech

5.1.1. Generování zpráv

Program `gener_file.exe` generuje určený počet zpráv K o zadané velikosti v bitech. Zpráva je vytvořena tak, že se náhodně generují bity "0" nebo "1" a zapisují se do souboru s názvem `Messagek.txt`, kde `k` $= 0, 1, \ldots K-1$ je index generované zprávy. Z tohoto souboru jsou pak bity načítány do obou naprogramovaných kryptosystémů.

5.1.2. Generování klíče

Standardně je potřeba generovat pouze jeden klíč, který se uloží do souboru. Pro účely testování rychlosti je ale možné generovat více klíčů najednou, které se do souboru neukládají. Program se na počet klíčů ptá hned na začátku. Jako další je potřeba zadat parametry pro určitou úroveň bezpečnosti.

5.1.2.1. Klíč RSA

Klíč RSA je generován programem `RSA_gener_key.exe`. Při generování klíče je nutné zadat jeho velikost a veřejný exponent. Pro testování budeme zadávat v současné době velmi často používaný exponent $e = 65\,637\ (= 2^{16} + 1)$. Velikost klíče je zadávána podle požadované velikost N v bitech.

Samotný soukromý klíč je generovaný Algoritmem 5.1. Pro účely testování nebudeme ukládat veřejný a soukromý klíč odděleně. Klíč veřejný i soukromý budeme mít tedy uložený v souboru `RSA_key.txt`.

Algoritmus 5.1. Generování klíče RSA

`Vstup`: velikost klíče $KeyLength$, veřejný exponent e
`Výstup`: soukromý klíč d

1. dokud je $\mathrm{NSD}(\varphi(N), e) \neq 1$, opakuj
   a) generuj prvočíslo p délky $KeyLength/2$
   b) generuj prvočíslo q délky $KeyLength/2 + 1$
   c) spočítej $\varphi(N)$ jako $(p-1) \cdot (q-1)$
2. $N = p \cdot q$
3. spočítej d tak, že $e \cdot d \equiv 1 \bmod \varphi(N)$
4. ulož do souboru N, e, d

V tomto algoritmu můžeme plně vycházet z možností NTL knihovny. Modul `ZZ`, kromě toho, že obsahuje všechny potřebné funkce pro počítání s velkými čísly, jako je sčítání, odečítání, násobení nebo umocňování, obsahuje také funkci pro generování prvočísel. Tato funkce potřebuje na vstupu pouze délku hledaného prvočísla. Lze ještě zadat další nepovinný parametr a to pravděpodobnost, že dané číslo je prvočíslo. Tato hodnota určuje počet průchodů Rabin–Millerovým testem. Další netriviální funkce, kterou budeme z NTL využívat je počítání inverzu modulo $\varphi(N)$ ve třetím kroku algoritmu.

5.1.2.2. Klíč NTRU

Klíč NTRU vygenerujeme programem `NTRU_gener_key.exe`. Pro generování klíče NTRU je nutné zadat celkem tři parametry. První je dimenze mřížky N, která nám zaručuje bezpečnost, a pak dva parametry p a q, pro které platí, že $\mathrm{NSD}(p, q) = 1$. Číslo p budeme volit vždy 3 a číslo q jako 2^i. Tím budeme mít zajištěnou nesoudělnost. Konkrétní parametry budeme volit podle Tabulky 2.

N	q	p
167	128	3
263	128	3
503	256	3

Tabulka 2. Parametry NTRU

Všechny operace ve všech částech NTRU, ať už se jedná o generování klíče, šifrování nebo dešifrování, budou probíhat vždy v $R = \mathbb{Z}[X]/(X^N - 1)$, tedy modulo polynom $x^N - 1$, proto musíme používat příslušné modulární funkce.

Při generování klíče NTRU budeme potřebovat náhodný polynom f, který bude invertibilní modulo p i q. Aby byl polynom s velkou pravděpodobností invertibilní, budeme chtít, aby $f(1) = 1 \bmod p$ a $f(1) = 1 \bmod q$. Podrobnější vysvětlení těchto podmínek je v [**20**].

Narozdíl od RSA musíme pro NTRU generovat jak soukromý, tak veřejný klíč. Jak vidíme v Algoritmu 5.2, soukromý klíč je pouze vhodně vygenerovaný náhodný polynom, veřejný klíč je nutné dopočítat. Stejně jako v případě RSA, ani v NTRU nebudeme pro účely testování ukládat veřejný a soukromý klíč odděleně. Veřejný i soukromý klíč uložíme do souboru `NTRU_key.txt`. Tento soubor bude kromě klíčů obsahovat i parametry šifrování N, p, q a polynom F_p, který se využívá při dešifrování, abychom ho nemuseli počítat při každém dešifrování.

Algoritmus 5.2. Generování klíče NTRU

`Vstup`: dimenze mřížky N, parametry p a q
`Výstup`: veřejný klíč h, soukromý klíč f

1. zvol náhodný polynom f, tak že $f(1) = 1 \bmod p$ a $f(1) = 1 \bmod q$
2. zvol náhodný polynom g
3. spočítej $F_q = f^{-1} \bmod q$ a $F_p = f^{-1} \bmod p$
4. spočítej veřejný klíč $h = F_q \cdot g \bmod q$
5. ulož do souboru N, p, q, h, f, F_p

Po náhodných polynomech generovaných v krocích 1 a 2 chceme, aby měly malé koeficienty. Podmínku na hodnotu v kroku 1 můžeme jednoduše splnit tak, že polynom bude mít určitý počet koeficientů roven 1, a o jeden koeficient méně bude rovno -1. Ostatní koeficienty budou nulové. Které koeficienty budou rovny 1 a které -1 bude zvoleno náhodně.

Výpočet inverzního polynomu v kroku 3 modulo q je založen podle [**21**] na metodě Newtonových iterací. To znamená, že pokud máme spočítaný inverz modulo p, pak velice jednoduše dopočítáme inverz modulo p^r Algoritmem 5.3. Z knihovny NTL využijeme funkci pro hledání inverzu modulo 2. Nezapomeňme, že všechny výpočty probíhají modulo polynom $x^N - 1$, proto musíme používat příslušné funkce pro modulární výpočty.

ALGORITMUS 5.3. Výpočet inverzu modulo p^r

`Vstup`: polynom $f(x)$, parametry p a r, $g(x) = f(x)^{-1} \bmod p$
`Výstup`: $g(x) = f(x)^{-1} \bmod p^r$

`1.` $p = q$
`2.` dokud $q < p^r$
  `a)` $q = q^2$
  `b)` $g(x) = g(x) \cdot (2 - f(x) \cdot g(x)) \bmod q$

5.1.3. Šifrování

Do obou algoritmů šifrování vstupují shodně soubor s klíčem a zprávy, které byly vygenerovány dříve. Žádné jiné vstupy nejsou předpokládány. Pokud program klíč nenajde, vyhlásí chybu. Pokud nenajde zprávu, kterou má šifrovat, tak proběhne, ale nevznikne žádný šifrový text. V případě, že je ve složce, kde pouštíme šifrování více zpráv, pak algoritmus zašifruje všechny zprávy klíčem ze souboru. Algoritmus skončí, pokud již nenajde žádné zprávy.

Pro jednoduchost implementace ani jeden z šifrovacích algoritmů nepřevádí šifrový text na bity, ale nechává ho v původním formátu, tedy číslo zapsané v desítkové soustavě v případě RSA a koeficienty polynomu zapsané také v desítkové soustavě v případě NTRU. Tím usnadníme i budoucí načítání šifrového textu.

Oba šifrovací algoritmy fungují po blocích, to znamená, že algoritmus vždy načte potřebný počet bitů v délce jednoho bloku a s tím pracuje. Délky bloků jsou pro oba algoritmy různé a jsou patrné z názvů šifer.

Další zjednodušení se týká bloků, které končí jednou nebo více nulami. Vzhledem k tomu, že oba algoritmy načítají posloupnost bitů tak, že poslední bit je nejvýznamnější, poslední nula nebo více nul se neprojeví při zpracovávání. Konkrétně to znamená, že zprávu "1100" budou oba algoritmy chápat jako "11" a tak ji i zašifrují. Tato vlastnost by se dala ošetřit například doplněním posledního bloku nulami na stejný počet bitů, jako všechny mají ostatní bloky. Pak bychom při dešifrování pouze doplnili zprávu o potřebný počet nul. Ale vzhledem k tomu, že takovéto doplňování nemá vliv na rychlost algoritmů a oba algoritmy se chovají stejně (pouze s tím rozdílem, že v RSA je potřeba doplnit větší počet nul, protože pracuje s delšími bloky), necháváme tuto vlastnost neošetřenou.

5.1.3.1. RSA šifrování

Zprávu zašifrujeme programem `RSA_encrypt.exe`. Ve složce, kde spouštíme program musí být soubor s klíčem a pak zpráva (případně zprávy), kterou chceme zašifrovat.

Blok v RSA může být vždy nejvýše tak veliký, že číslo, kterým je reprezentovaný (provádíme převod do desítkové soustavy), je ostře menší než N. Aby byl

blok vždy stejně velký a určitě menší jak N, budeme v první fázi načítat ze zprávy pokaždé stejný počet bitů, a to o jeden bit méně, než je počet bitů N.

Blok, který jsme načetli máme nyní k dispozici jako číslo, se kterým můžeme dále počítat, tedy ho šifrovat. Po provedení šifrování výsledné číslo zapíšeme do souboru. Nebudeme ho nijak převádět, protože tím pak urychlíme a zjednodušíme načítání šifrového textu při dešifrování.

Algoritmus 5.4. RSA šifrování

`Vstup`: `RSA_key.txt` a zprávy `Messagek.txt`, kde `k` je index zprávy
`Výstup`: soubory `RSACipherTextk.txt`, kde `k` je index zprávy

1. načti klíč ze souboru
2. dokud není konec zprávy
   a) načti maximálně $(\log_2 N - 1)$ bitů ze zprávy a ulož je jako číslo do m
   b) spočítej $c = m^e \bmod N$
   c) vypiš c do souboru

Šifrování je s pomocí NTL opět velmi přímočaré. Nastavíme modul pomocí `ZZ_p::init(N)` a knihovna obsahuje všechny potřebné funkce včetně optimalizace umocňování v kroku `2b`.

5.1.3.2. NTRU šifrování

Na šifrování kryptosystémem NTRU použijeme program `NTRU_encrypt.exe`. Stejně tak, jako v případě RSA, je nutné, aby složka, kde je program, obsahovala jak soubor s klíčem, tak zprávy, které chceme zašifrovat.

Další podobnost najdeme v tom, že NTRU pracuje také po blocích. Tyto bloky jsou ale výrazně menší než v případě RSA. Bity zprávy načítáme jako koeficienty polynomu, proto načteme vždy maximálně N bitů.

Algoritmus 5.5. NTRU šifrování

`Vstup`: `NTRU_key.txt` a zprávy `Messagek.txt`, kde `k` je index zprávy
`Výstup`: soubory `NTRUCipherTextk.txt`, kde `k` je index zprávy

1. načti klíč ze souboru
2. zvol náhodný polynom $s \in \mathbb{Z}_3[x]$
3. spočítej e jako $e = s * p \cdot h \bmod q$
4. dokud není konec zprávy
   a) načti maximálně N a ulož je jako koeficienty polynomu m
   b) spočítej $c = m + e \bmod q$
   c) vypiš c do souboru

Náhodný polynom v kroku `1` volíme stejným způsobem jako polynomy v Algoritmu 5.2 při generování klíče. Násobení v kroku `2` je v NTL také implementováno jako modulární násobení a stejně tak sčítání modulo q v kroku `2b`.

5.1.4. Dešifrování

Při dešifrování budeme potřebovat soukromý klíč, tedy soubor s klíčem a šifrové texty. Program testuje přítomnost potřebných souborů a stejně jako při šifrování vyhlásí chybu, pokud nenalezne žádný klíč. Programy nekladou důraz na tvary vypisování dešifrované zprávy, to znamená že fakticky se otevřený a dešifrovaný text liší (například mezerou). Ale číselně si obě zprávy odpovídají. Algoritmy tedy fungují správně, pro testování rychlosti samotných algoritmů není úprava výstupu nutná.

5.1.4.1. RSA dešifrování

Při dešifrování zpráv, které byly zašifrovány kryptosystémem RSA, použijeme program `RSA_decrypt.exe`.

Algoritmus 5.6. RSA dešifrování

`Vstup`: `RSA_key.txt` a zprávy `RSACipherTextk.txt`, kde `k` je index zprávy
`Výstup`: soubory `RSADecipherTextk.txt`, kde `k` je index zprávy

1. načti klíč ze souboru
1. dokud není konec šifrového textu
   a) načti řádek šifrového textu a ulož ho do c
   b) spočítej $m = c^d \bmod N$
   c) vypiš m do souboru

Načítání šifrového textu je jednoduché, protože máme šifrový text rozdělený po blocích do řádků souboru `RSACipherTextk.txt`. Dešifrovací algoritmus využívá pouze umocňování modulo N. Stačí tedy nastavit modul na N příkazem `ZZ_p::init(N)` a další je již implementováno v NTL.

5.1.4.2. NTRU dešifrování

Pro NTRU dešifrování slouží program `NTRU_decrypt.exe`.

Algoritmus 5.7. NTRU šifrování

`Vstup`: `NTRU_key.txt` a zprávy `NTRUCipherTextk.txt`, kde `k` je index zprávy
`Výstup`: soubory `NTRUDecipherTextk.txt`, kde `k` je index zprávy

1. načti klíč ze souboru
2. dokud není konec šifrového textu
   a) načti řádek šifrového textu a ulož ho do polynomu c jako koeficienty
   b) spočítej a jako $a = f * c \bmod q$
   c) posuň všechny koeficienty polynomu a do intervalu $(-q/2, q/2]$
   d) spočítej $m = a * Fp \bmod p$
   e) vypiš m do souboru

Načítání šifrového textu je opět jednoduché, protože máme šifrový text rozdělený po blocích do řádků souboru `NTRUCipherTextk.txt`. Dešifrování NTRU probíhá do kroku `2b` pouze pomocí funkcí implementovaných v NTL. V kroku `2c` ale nastává problém, protože pokud počítáme s koeficienty modulo q, pak v případě, že bychom se dostali do záporných hodnot, NTL je za nás automaticky přepočítá do kladných. Proto musíme použít konverzi a převést polynom a na polynom nad celými čísly, provést výpočet v kroku `2d` a poté konvertovat koeficienty polynomu zpět do $\mathbb{Z}_p$.

5.2. Výsledky testů

V této části si popíšeme výsledky testování naprogramovaných algoritmů RSA a NTRU. Asymptotická výpočetní složitost RSA je $\mathcal{O}(N^3)$ a NTRU $\mathcal{O}(N^2)$. Podle teoretických výsledků a také podle dosud prezentovaných výsledků především v [**22**] se dá předpokládat, že NTRU bude pracovat výrazně rychleji, než RSA. Vzhledem k tomu, že jsme zvolili pro RSA vždy stejný veřejný exponent 65 537, dá se odhad složitosti na šifrování snížit na $\mathcal{O}(N^2)$. Dešifrování ale stále zůstává na původních $\mathcal{O}(N^3)$. Z tohoto bychom tedy mohli odvozovat, že šifrování bude trvat stejně dlouho při použití RSA i NTRU, ale v dešifrování by mohl být rozdíl.

Kromě teoretického pohledu je zajímavý i praktický pohled. Obě šifry pracují po blocích, ale velikosti bloků jsou rozdílné. RSA pracuje až na několikanásobně větších blocích než NTRU. Máme tedy důvod se domnívat, že čas potřebný k výpočtu roste přibližně lineárně s délkou zprávy, ale to bude pravděpodobně platit až u větších zpráv. U menších zpráv by se mohlo projevit, že např. RSA 4096 pracuje s blokem 4096 bitů, ale zpráva je ve skutečnosti výrazně kratší, proto by rychlost výpočtu nemusela s velikostí zprávy narůstat až do zprávy velikosti jednoho bloku. Abychom dokázali odpovědět na všechny tyto otázky, testovali jsme tři úrovně bezpečnosti pro zprávy různých délek.

Níže uvedené délky zpráv jsme volili ze dvou důvodů. Nejprve u kratších zpráv nás zajímalo, jestli se nějak projeví, že zpráva je optimalizovaná vzhledem k délce bloku jedné ze šifer. Na druhou stranu, testy s velkými zprávami by nám pak měly nezávisle porovnat obě šifry bez dalších vlivů. Zprávy jsme tedy volili s velikostmi:

- 150 bitů – tato délka by měla být výhodná pro NTRU 167
- 500 bitů – tato délka by měla být výhodná pro NTRU 503 a RSA 512
- 4 000 bitů – tato délka by měla být výhodná pro RSA 4096
- 30 000 bitů – při větších zprávách by neměla mít vliv velikost bloků
- 400 000 bitů – porovnání šifer na velkých zprávách

Dalším faktorem je doba generování klíče. Toto hledisko může být pro nás jako jednotlivce méně zajímavé, protože klíč vygenerujeme pouze jednou a pak s jeho pomocí šifrujeme mnohonásobně víckrát. Ale například pro certifikační autority

je doba generování klíče kritická. Proto se v této práci vracíme i k testování doby generování klíče pro všechny tři úrovně bezpečnosti.

Testy byly prováděny na počítači s procesorem AMD Turion 1.8 GHz. Čas byl měřen pomocí vnitřní funkce NTL `GetTime`. Aby mohly být časy při testování vůbec měřitelné, byly testy prováděny vždy na takovém množství zpráv (resp. generováno tolik klíčů), aby se naměřené hodnoty pohybovaly nejméně v jednotkách sekund. Počet zpráv při šifrování i dešifrování byl určený rychlostí šifrování. Počty zpráv (resp. klíčů) v testu jsou uvedeny v přílohách společně se všemi měřeními. Porovnávat budeme průměrnou dobu běhu algoritmu. Z důvodu možné nepřesnosti měření budeme uvažovat pouze rozdíly, které přesáhnou 5% doby běhu rychlejšího algoritmu.

5.2.1. Generování klíčů

Pro všechny tři úrovně bezpečnosti je generování klíčů pro NTRU rychlejší, než pro RSA. Při nejnižší úrovni se rozdíl může zdát nepatrný, řádově 10 ms, ale vzhledem k celkové době generování je RSA o více jak 1/3 pomalejší. Jak potom dále narůstá bezpečnost šifry, výrazně roste i doba generování klíče RSA. Při nejvyšší bezpečnosti je doba generování klíče RSA téměř 27x delší než pro NTRU. Dá se předpokládat, že pokud bychom pro RSA 4096 zvolili větší veřejný exponent, doba generování klíče by se mohla zkrátit, protože odhadujeme, že nejdelší čas trvá výpočet soukromého klíče. Zároveň by se tím ale snížila rychlost šifrování, proto jsme zůstali u standardního veřejného exponentu.

Zajímavé je, že časy v testech se příliš neliší, přestože při generování klíčů obou algoritmů spoléháme na generování náhodných prvků, které splňují určité parametry (pro RSA hledáme prvočísla a pro NTRU chceme invertibilní polynom). To by mohlo ukazovat na to, že podmínky, které jsme stanovili, jsou splnitelné s vysokou pravděpodobností v obou kryptosystémech.

Průměrné časy jsou zapsány v Tabulce 3.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na klíč (ms)	26,9154	59,2498	92,746	36,7262	271,9743	26 927,78

Tabulka 3. Rychlost generování klíčů

5.2.2. Šifrování

Při testech šifrování musíme přihlížet k velikosti zprávy. U nejmenší zprávy 150 bitů je jasně rychlejší RSA a to ve všech třech úrovních bezpečnosti. Pro NTRU 167 se sice velikost zprávy blíží k optimální velikosti bloku, ale rychlosti

srovnatelně bezpečného RSA 512 zdaleka nedosahuje. Nejmenší rozdíl můžeme pozorovat až u nejvyšší úrovně bezpečnosti, kde se pohybuje okolo 0,6 ms. Vzhledem k celkové délce šifrování je to necelých 10% doby šifrování RSA. Průměrné výsledky měření pro zprávy o velikosti 150 bitů jsou v Tabulce 4.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	3,30545	5,49372	6,01200	1,27091	1,36871	5,3562

Tabulka 4. Rychlost šifrování zprávy o velikosti 150 bitů

Velikost zprávy 500 bitů je vhodná pro NTRU 503 a pro RSA 512. V prvních dvou úrovních bezpečnosti je opět výrazně rychlejší RSA. Doba šifrování NTRU 167 je téměř trojnásobná oproti RSA 512 a v případě NTRU 263 a RSA 1024 se rozdíl blíží čtyřnásobku. K vyrovnání dochází až při nejvyšší bezpečnosti. RSA 4096 je stále rychlejší než NTRU 503, ale rozdíl je pod 5%. Můžeme si všimnout, že šifrování pomocí NTRU 503 bylo v tomto jediném testu rychlejší o 0,23 ms než NTRU 263. Rozdíl je ale menší než 5% doby běhu, proto jej nepovažujeme za významný. Průměrné výsledky testů šifrování zprávy o velikosti 500 bitů jsou v Tabulce 5.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	4,20653	6,63075	6,4058	1,44235	1,74507	6,1607

Tabulka 5. Rychlost šifrování zprávy o velikosti 500 bitů

Zprávy o velikosti 4 000 bitů je již nutné šifrovat po blocích téměř ve všech případech. Při velikostech bloků 167 a 263 bitů již nemá smysl uvažovat, jak moc je tato délka zprávy blízko násobkům bloků. Na druhou stranu pro všechny další šifry je velikost zprávy 4 000 bitů vhodná. RSA 512 a NTRU 503 zpracují zprávu téměř přesně v osmi blocích, RSA 1024 pak ve čtyřech blocích. Jediná šifra, která dokáže zpracovat celou zprávu naráz je RSA 4096. RSA je výrazně rychlejší pro první dvě úrovně bezpečnosti. Dalo by se říci až o polovinu. Ale při nejvyšší bezpečnosti jsou obě šifry vyrovnané. Průměrné doby šifrování zpráv o velikosti 4 000 bitů nalezneme v Tabulce 6.

Výsledky testování na zprávách o velikosti 30 000 bitů a 400 000 bitů jsou velmi podobné. Při nejnižší bezpečnosti je RSA rychlejší než NTRU přibližně o 20%. Střední úroveň bezpečnosti se NTRU a RSA liší jen velmi málo. A při nejvyšší úrovni bezpečnosti je NTRU o 2/3 rychlejší než RSA. Průměrné výsledky měření pro zprávy o velikosti 30 000 bitů jsou v Tabulce 7 a pro zprávy o velikosti 400 000 bitů v Tabulce 8.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	10,51540	12,6777	13,6597	7,4121	8,6732	14,1766

Tabulka 6. Rychlost šifrování zprávy o velikosti 4 000 bitů

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	62,54567	63,5145	67,657	51,681	59,4895	105,136

Tabulka 7. Rychlost šifrování zprávy o velikosti 30 000 bitů

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	784,74	782,555	829,96	673,18	776,78	1 346,36

Tabulka 8. Rychlost šifrování zprávy o velikosti 400 000 bitů

Výsledky všech měření a zakreslení v grafech lze nalézt v přílohách. Celkově lze tedy říci, že NTRU je sice považováno za rychlé, ale pro nižší úrovně bezpečnosti je stále vhodnější RSA. O používání NTRU lze uvažovat až v případě, že předpokládáme, že budeme potřebovat střední nebo vysokou bezpečnost. Pokud tedy víme, že budeme chtít střední nebo vysokou bezpečnost, pak je ještě důležitá velikost zpráv, které budeme šifrovat. Při střední bezpečnosti se NTRU z hlediska rychlosti vyrovná RSA až pro zprávy velké desítky tisíc bitů a větší. Nejvyšší bezpečnost je srovnatelně rychlá pro oba algoritmy již od malých zprávy ale pouze do velikosti zpráv jednotky tisíc bitů. U větších zpráv se pak z hlediska rychlosti vyplatí použít NTRU.

	1. úroveň	2. úroveň	3. úroveň
150 bitů	RSA 512	RSA 1024	RSA 4096
500 bitů	RSA 512	RSA 1024	NTRU 503 / RSA 4096
4 000 bitů	RSA 512	RSA 1024	NTRU 503 / RSA 4096
30 000 bitů	RSA 512	NTRU 263 / RSA 1024	NTRU 503
400 000 bitů	RSA 512	NTRU 263 / RSA 1024	NTRU 503

Tabulka 9. Vhodné kryptosystémy podle rychlosti šifrování

5.2.3. Dešifrování

I při testech dešifrování hraje roli velikost zprávy, ale ne tak výraznou jako při testech šifrování. Téměř výlučně při všech testech se při nejnižší úrovni bezpečnosti vyplatí používat RSA a u střední a vysoké úrovně bezpečnosti NTRU. Výjimku tvoří pouze velikost zprávy 150 bitů. Zde se projevila vhodnost délky zprávy pro NTRU 167 a tedy i při nejnižší úrovni je RSA o více jak 20% pomalejší než NTRU. I při střední úrovni dává NTRU lepší výsledky než při ostatních testech, pracuje téměř čtyřikrát rychleji než RSA. Nejvyšší úroveň bezpečnosti vychází ve všech testech jasně pro NTRU. U zprávy o velikosti 150 bitů pracuje téměř stokrát rychleji. Průměrné výsledky měření pro zprávy o velikosti 150 bitů jsou v Tabulce 10.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	5,17484	9,43154	18,1641	6,35683	35,93467	1 716,46

Tabulka 10. Rychlost dešifrování zprávy o velikosti 150 bitů

U 500 bitové zprávy již zaznamenáváme standardní průběh testu popsaný výše. Nejnižší úroveň bezpečnosti je výrazně rychlejší v případě RSA, NTRU pracuje dvojnásobně dlouhou dobu. Při střední úrovni se role obou šifer vyměňují a NTRU je v tomto případě dvakrát rychlejší než RSA. Nejvyšší úroveň bezpečnosti dává téměř stejné výsledky jako u zprávy o velikosti 150 bitů. Průměrné výsledky testů dešifrování zprávy o velikosti 500 bitů jsou v Tabulce 11.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	13,79137	18,24436	20,8078	6,05804	36,08184	1 700,431

Tabulka 11. Rychlost dešifrování zprávy o velikosti 500 bitů

Další test pro zprávu o velikosti 4 000 bitů se od předchozího liší jedině pro střední úroveň bezpečnosti. Zatímco v předchozích dvou testech bylo jasně rychlejší NTRU, zde není rozdíl mezi oběma šiframi tak velký, je přibližně 12%. Ani rozdíl při nejvyšší úrovni bezpečnosti se nevyrovná předchozím testů, ale přesto NTRU pracuje desetkrát rychleji než RSA. Narozdíl od šifrování se při testech dešifrování ukázalo, že má smysl uvažovat, jak dlouhé zprávy vstupují do algoritmu vzhledem k délce bloku. RSA 4096 totiž dešifruje zprávy o velikosti 150 bitů, 500 bitů i 4 000 bitů stejně rychle narozdíl od NTRU, kde doba dešifrování narůstá, i když ne tak výrazně, aby dosáhla času RSA. Průměrné doby dešifrování zpráv o velikosti 4 000 bitů nalezneme v Tabulce 6.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	100,6935	128,1715	161,4593	46,081	144,8525	1 707,625

Tabulka 12. Rychlost dešifrování zprávy o velikosti 4 000 bitů

Testy na dvojici největších zpráv jenom potvrzují, že při nízké úrovni bezpečnosti se z časového hlediska vyplatí používat RSA. Při střední úrovni bezpečnosti vychází lépe NTRU o více než 20%. A při nejvyšší úrovni bezpečnosti NTRU pracuje více než desetkrát rychleji. Průměrné výsledky měření pro zprávy o velikosti 30 000 bitů jsou v Tabulce 13 a pro zprávy o velikosti 400 000 bitů v Tabulce 14.

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	747,384	918,998	1 134,261	350,967	1 111,193	13 716,02

Tabulka 13. Rychlost dešifrování zprávy o velikosti 30 000 bitů

	NTRU			RSA		
	167	263	503	512	1024	4096
Průměr na zprávu (ms)	8 066,15	11 578,43	14 994,84	4 535,9	14 530,4	168 130

Tabulka 14. Rychlost dešifrování zprávy o velikosti 400 000 bitů

Výsledky všech měření a zakreslení v grafech lze nalézt v přílohách. Kvůli vysokým dobám dešifrování RSA 4096 při nejvyšší úrovni bezpečnosti jsou první dvě úrovně znázorněny pro všechny testy ještě v detailu pod grafem, aby byly patrné rozdíly v rychlosti i v těchto úrovních.

	1. úroveň	2. úroveň	3. úroveň
150 bitů	NTRU 167	NTRU 263	NTRU 503
500 bitů	RSA 512	NTRU 263	NTRU 503
4 000 bitů	RSA 512	NTRU 263	NTRU 503
30 000 bitů	RSA 512	NTRU 263	NTRU 503
400 000 bitů	RSA 512	NTRU 263	NTRU 503

Tabulka 15. Vhodné kryptosystémy podle rychlosti dešifrování

KAPITOLA 6

Závěr

V práci jsme věnovali různému využití mřížek v kryptografii. Z uvedených aplikací je vidět, že mřížky mají v kryptografii významnou roli. Výsledky jejich prvotního využití v kryptoanalýze (zde se jedná především o redukci báze a LLL algoritmus) byly také použity v oblasti dokazování bezpečnosti a stejně tak pro tvorbu kryptosystémů založených na problémech spojených s mřížkami (*SVP* a *CVP*). Některé kryptosystémy byly poměrně brzy prolomeny (např. Ajtai-Dwork kryptosystém), ale jiné (např. NTRU) stále útokům odolávají. Z tohoto pohledu by se dalo očekávat, že alternativa v oblasti kryptografie s veřejným klíčem a do budoucna i možná náhrada RSA bude kryptosystém, který je nějakým způsobem spjatý s mřížkami.

Ukázali jsme, že z hlediska rychlosti by NTRU bylo vhodným nástupcem především pro vyšší úrovně bezpečnosti a delší zprávy, které chceme šifrovat. Zůstává ale otázka bezpečnosti. Jedna strana je složitost problému, na kterém je kryptosystém založen, druhá potom vlastní schéma a využití problému a třetí strana je vlastní implementace. Některé kryptosystémy byly v minulosti prolomeny právě kvůli nevhodnému schématu a mnoho i kvůli špatné implementaci. Ale ani z hlediska složitosti problémů spojených s mřížkami není stále vše vyřešené. Není například jasné, jestli SVP_γ je nebo není snadné řešit pro polynomiální γ. A také musíme uvážit, že problémy spojené s mřížkami nebyly v minulosti tolik zkoumány jako například faktorizace. Z tohoto důvodu stále existuje pouze velmi málo algoritmů na redukci báze mřížky.

Lze tedy říci, že v současné době a podle současných výsledků je NTRU zajímavá alternativa k RSA. Otázkou ale zůstává, jestli v době, kdy budeme nuceni od RSA ustoupit, bude NTRU stále tak bezpečné a zároveň dostatečně rychlé.

Literatura

[1] Miklós Ajtai: *Generating Hard Instances of Lattice Problems*, 1996.

[2] Oded Goldreich, Shafi Goldwasser, Shai Halevi: *Collision–Free Hashing from Lattice Problem*, 1997.

[3] Oded Regev: *Lattice–based Cryptography*, 2006.

[4] Oded Goldreich, Daniele Micciancio, Shmuel Safra, Jean-Pierre Seifert: *Approximating shortest lattice vectors is not harder than approximating closest lattice vectors*, 1999.

[5] Oded Goldreich, Shafi Goldwasser, Shai Halevi: *Public–Key Cryptosystems from Lattice Reduction Problems*, 1997.

[6] Phong Q. Nguyen, Jacques Stern : *Lattice Reduction in Cryptology: An Update*, 2002.

[7] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman: *NTRU: A Ring–Based Public Key Cryptosystem*, 1998.

[8] Jeffrey Hoffstein, Nick Howgrave–Graham, Jill Pipher, Joseph H. Silverman, William Whyte: *NTRUSign: Digital Signatures Using the NTRU Lattice*, 2003.

[9] Oded Goldreich, Shafi Goldwasser, Shai Halevi: *Eliminating Decryption Errors in the Ajtai-Dwork Cryptosystem*, 1997.

[10] Arjen K. Lenstra, Hendrik W. Lenstra jr., László Lovász : *Factoring Polynomials with Rational Coefficients*, 1982.

[11] Phong Q. Nguyen, Jacques Stern : *The Two Faces of Lattices in Cryptology*, 2001.

[12] Jason Hinek: *Lattice Attacks in Cryptography: A Partial Overview*, 2004.

[13] Richard Eier, Helmut Lagger: *Trapdoors in knapsack cryptosystems*, 1983.

[14] Yvo G. Desmedt, Joos P. Vandewalle, René J. M. Govaerts: *A critical analysis of the security of knapsack public–key algorithms*, 1985.

[15] Jeffrey C. Lagarias, Andrew M. Odlyzko: *Solving low–density subset sum problems*, 1985.

[16] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr a Jacques Stern: *Improved Low–Density Subset Sum Algorithms*, 2008.

[17] Igor E. Shparlinski : *Playing "Hide-and-Seek" in Finite Fields: The Hidden Number Problem and Its Applications*, 2002.

[18] Colleen Marie O'Rourke: *Efficient NTRU Implementations*, 2002.

[19] `http://www.shoup.net/ntl`.

[20] Joseph H. Silverman: *Invertibility in Truncated Polynomial Rings*, 1998.

[21] Joseph H. Silverman: *Almost Inverses and Fast NTRU Key Creation*, 1999.

[22] `http://www.ntru.com/cryptolab`.

KAPITOLA A

Tabulky

Šifra	Počet zpráv
NTRU 167, RSA 512	1 000
NTRU 263, RSA 1024	1 000
NTRU 503, RSA 4096	100

Tabulka A–1. Počet klíčů generovaných v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	26,891	58,598	9,187	36,671	271,890	2 716,61
2	27,140	58,593	9,312	36,594	271,093	2 698,94
3	26,937	59,845	9,250	36,500	271,859	2 696,56
4	26,875	57,189	9,312	36,687	271,921	2 696,66
5	26,859	59,841	9,375	36,593	272,437	2 606,30
6	26,906	60,620	9,250	36,765	272,046	2 694,38
7	26,969	59,219	9,296	37,047	271,921	2 695,75
8	26,906	59,845	9,296	36,828	272,296	2 731,80
9	26,828	59,218	9,281	36,859	271,718	2 696,20
10	26,843	59,530	9,187	36,718	272,562	2 694,58
Průměr na test (s)	26,9154	59,2498	9,2746	36,7262	271,9743	2 692,778
Průměr na klíč (ms)	**26,9154**	**59,2498**	**92,746**	**36,7262**	**271,9743**	**26 927,78**

Tabulka A–2. Rychlost generování klíčů

Šifra	Počet zpráv
NTRU 167, RSA 512	10 000
NTRU 263, RSA 1024	10 000
NTRU 503, RSA 4096	1000

Tabulka A–3. Počet zpráv o velikosti 150 bitů v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	34,734	54,578	6,141	15,593	13,250	5,360
2	34,203	60,328	6,171	12,172	13,125	5,375
3	32,593	54,406	5,921	12,328	14,031	5,328
4	32,562	54,031	5,937	12,390	12,390	5,468
5	32,406	52,125	5,921	13,671	13,328	5,328
6	34,063	54,015	5,984	11,563	13,468	5,328
7	31,000	55,562	6,078	12,312	13,078	5,437
8	32,750	55,250	6,046	12,125	13,046	5,360
9	33,375	54,671	5,937	12,546	17,640	5,328
10	32,859	54,406	5,984	12,391	13,515	5,250
Průměr na test (s)	33,0545	54,9372	6,0120	12,7091	13,6871	5,3562
Průměr na zprávu (ms)	**3,30545**	**5,49372**	**6,01200**	**1,27091**	**1,36871**	**5,3562**

Tabulka A–4. Rychlost šifrování zprávy o velikosti 150 bitů

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	55,203	93,359	18,171	67,859	357,515	1 713,27
2	53,859	98,563	18,266	65,609	369,938	1 706,53
3	50,563	92,750	18,094	62,781	359,859	1 707,98
4	50,609	92,328	18,094	63,531	359,062	1 720,09
5	50,953	92,312	18,110	62,046	355,062	1 718,53
6	53,328	92,344	18,141	63,968	358,734	1 718,70
7	50,891	96,031	18,062	60,968	358,031	1 715,92
8	51,094	98,500	18,250	62,687	356,781	1 724,80
9	50,781	92,296	18,203	64,281	363,875	1 717,09
10	50,203	94,671	18,250	61,953	354,610	1 721,69
Průměr na test (s)	51,7484	94,3154	18,1641	63,5683	359,3467	1 716,46
Průměr na zprávu (ms)	**5,17484**	**9,43154**	**18,1641**	**6,35683**	**35,93467**	**1 716,46**

Tabulka A–5. Rychlost dešifrování zprávy o velikosti 150 bitů

Šifra	Počet zpráv
NTRU 167, RSA 512	10 000
NTRU 263, RSA 1024	10 000
NTRU 503, RSA 4096	1000

Tabulka A–6. Počet zpráv o velikosti 500 bitů v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	41,953	67,281	6,390	17,203	19,359	6,000
2	43,718	64,312	6,375	13,109	16,328	6,296
3	45,593	65,281	6,359	14,500	17,375	6,125
4	39,859	72,609	6,437	13,968	17,843	6,110
5	42,234	64,328	6,484	13,688	17,000	6,062
6	42,734	66,078	6,328	13,390	17,000	6,125
7	40,843	72,437	6,468	14,487	18,765	6,171
8	42,000	62,375	6,421	15,609	17,328	6,250
9	39,906	64,015	6,437	14,031	16,681	6,359
10	41,813	64,359	6,359	14,250	16,828	6,109
Průměr na test (s)	42,0653	66,3075	6,4058	14,4235	17,4507	6,1607
Průměr na zprávu (ms)	**4,20653**	**6,63075**	**6,4058**	**1,44235**	**1,74507**	**6,1607**

Tabulka A–7. Rychlost šifrování zprávy o velikosti 500 bitů

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	134,187	182,016	20,906	60,212	360,437	1 703,53
2	136,156	180,703	20,828	61,046	357,468	1 706,67
3	136,031	178,312	20,781	59,625	357,875	1 700,00
4	137,843	191,000	20,906	61,328	362,578	1 699,23
5	138,797	182,610	20,750	59,531	364,906	1 699,05
6	139,484	192,921	20,860	62,688	360,593	1 704,34
7	134,984	177,468	20,765	61,860	359,140	1 694,52
8	145,859	179,250	20,735	58,046	359,640	1 697,73
9	138,109	179,578	20,781	59,343	361,563	1 699,43
10	137,687	180,578	20,766	62,125	363,984	1 699,81
Průměr na test (s)	137,9137	182,4436	20,8078	60,5804	360,8184	1 700,431
Průměr na zprávu (ms)	**13,79137**	**18,24436**	**20,8078**	**6,05804**	**36,08184**	**1 700,431**

Tabulka A–8. Rychlost dešifrování zprávy o velikosti 500 bitů

Šifra	Počet zpráv
NTRU 167, RSA 512	1000
NTRU 263, RSA 1024	1000
NTRU 503, RSA 4096	1000

Tabulka A–9. Počet zpráv o velikosti 4 000 bitů v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	10,390	12,843	13,625	7,328	8,500	14,578
2	10,328	12,671	13,640	7,437	8,609	13,906
3	10,328	12,687	13,687	7,281	8,438	14,328
4	10,703	12,625	13,671	7,250	9,031	14,172
5	10,593	12,687	13,593	7,531	8,703	14,063
6	10,421	12,640	13,765	7,218	8,406	14,109
7	10,578	12,656	13,578	7,578	9,218	14,189
8	10,563	12,671	13,640	7,593	8,500	14,187
9	10,828	12,703	13,609	7,609	8,406	13,875
10	10,422	12,594	13,789	7,296	8,921	14,359
Průměr na test (s)	10,5154	12,6777	13,6597	7,4121	8,6732	14,1766
Průměr na zprávu (ms)	**10,51540**	**12,6777**	**13,6597**	**7,4121**	**8,6732**	**14,1766**

Tabulka A–10. Rychlost šifrování zprávy o velikosti 4 000 bitů

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	100,765	127,203	161,281	45,953	145,687	1 705,88
2	100,375	127,781	161,297	46,546	145,046	1 708,61
3	100,140	128,156	160,968	46,046	145,562	1 705,05
4	100,594	128,312	162,031	46,187	145,843	1 713,03
5	101,578	128,359	161,375	46,234	144,296	1 703,92
6	100,859	128,031	161,469	46,016	143,890	1 712,28
7	100,343	128,593	162,125	46,094	144,296	1 708,53
8	101,609	128,250	160,844	46,031	145,031	1 705,42
9	100,266	127,968	161,953	45,953	145,015	1 704,67
10	100,406	129,062	161,250	45,750	143,859	1 708,86
Průměr na test (s)	100,6935	128,1715	161,4593	46,081	144,8525	1 707,625
Průměr na zprávu (ms)	**100,6935**	**128,1715**	**161,4593**	**46,081**	**144,8525**	**1 707,625**

Tabulka A–11. Rychlost dešifrování zprávy o velikosti 4 000 bitů

Šifra	Počet zpráv
NTRU 167, RSA 512	300
NTRU 263, RSA 1024	200
NTRU 503, RSA 4096	100

Tabulka A–12. Počet zpráv o velikosti 30 000 bitů v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	18,453	12,625	6,812	15,687	11,937	10,468
2	19,687	12,750	6,765	15,359	11,843	10,468
3	18,609	12,625	6,718	15,531	11,968	10,515
4	18,594	12,687	6,718	15,515	11,843	10,437
5	18,593	12,859	6,796	15,593	11,937	10,468
6	18,609	12,703	6,828	15,375	11,937	10,578
7	18,906	12,703	6,719	15,391	11,891	10,531
8	18,984	12,656	6,719	15,593	11,890	10,468
9	18,515	12,734	6,875	15,578	11,937	10,453
10	18,687	12,687	6,707	15,421	11,796	10,750
Průměr na test (s)	18,7637	12,7029	6,7657	15,5043	11,8979	10,5136
Průměr na zprávu (ms)	**62,54567**	**63,5145**	**67,657**	**51,681**	**59,4895**	**105,136**

Tabulka A–13. Rychlost šifrování zprávy o velikosti 30 000 bitů

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	223,421	182,906	115,328	104,890	222,593	1 375,20
2	223,187	183,921	113,953	103,312	222,093	1 372,55
3	223,859	184,250	111,734	104,937	221,875	1 368,39
4	224,531	183,515	114,062	105,234	222,796	1 371,41
5	223,719	185,875	114,546	105,328	223,421	1 365,48
6	225,328	182,953	113,531	105,937	222,328	1 369,31
7	225,328	183,922	112,140	105,625	221,859	1 368,94
8	224,531	183,625	110,890	106,203	221,265	1 374,03
9	223,813	182,890	114,765	105,843	222,031	1 373,88
10	224,437	184,140	113,312	105,593	222,125	1 376,83
Průměr na test (s)	224,2154	183,7997	113,4261	105,2902	222,2386	1 371,602
Průměr na zprávu (ms)	**747,384**	**918,998**	**1 134,261**	**350,967**	**1 111,193**	**13 716,02**

Tabulka A–14. Rychlost dešifrování zprávy o velikosti 30 000 bitů

Šifra	Počet zpráv
NTRU 167, RSA 512	20
NTRU 263, RSA 1024	20
NTRU 503, RSA 4096	10

Tabulka A–15. Počet zpráv o velikosti 400 000 bitů v jednom testu

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	15,703	15,687	8,359	13,453	15,531	13,437
2	15,609	15,546	8,281	13,437	15,531	13,421
3	15,718	15,671	8,250	13,468	15,625	13,390
4	15,687	15,671	8,265	13,468	15,531	13,453
5	15,718	15,640	8,281	13,421	15,546	13,453
6	15,671	15,703	8,390	13,453	15,656	13,421
7	15,734	15,734	8,343	13,468	15,468	13,421
8	15,687	15,656	8,250	13,453	15,453	13,468
9	15,718	15,625	8,281	13,484	15,531	13,406
10	15,703	15,578	8,296	13,531	15,484	13,766
Průměr na test (s)	15,6948	15,6511	8,2996	13,4636	15,5356	13,4636
Průměr na zprávu (ms)	**784,74**	**782,555**	**829,96**	**673,18**	**776,78**	**1 346,36**

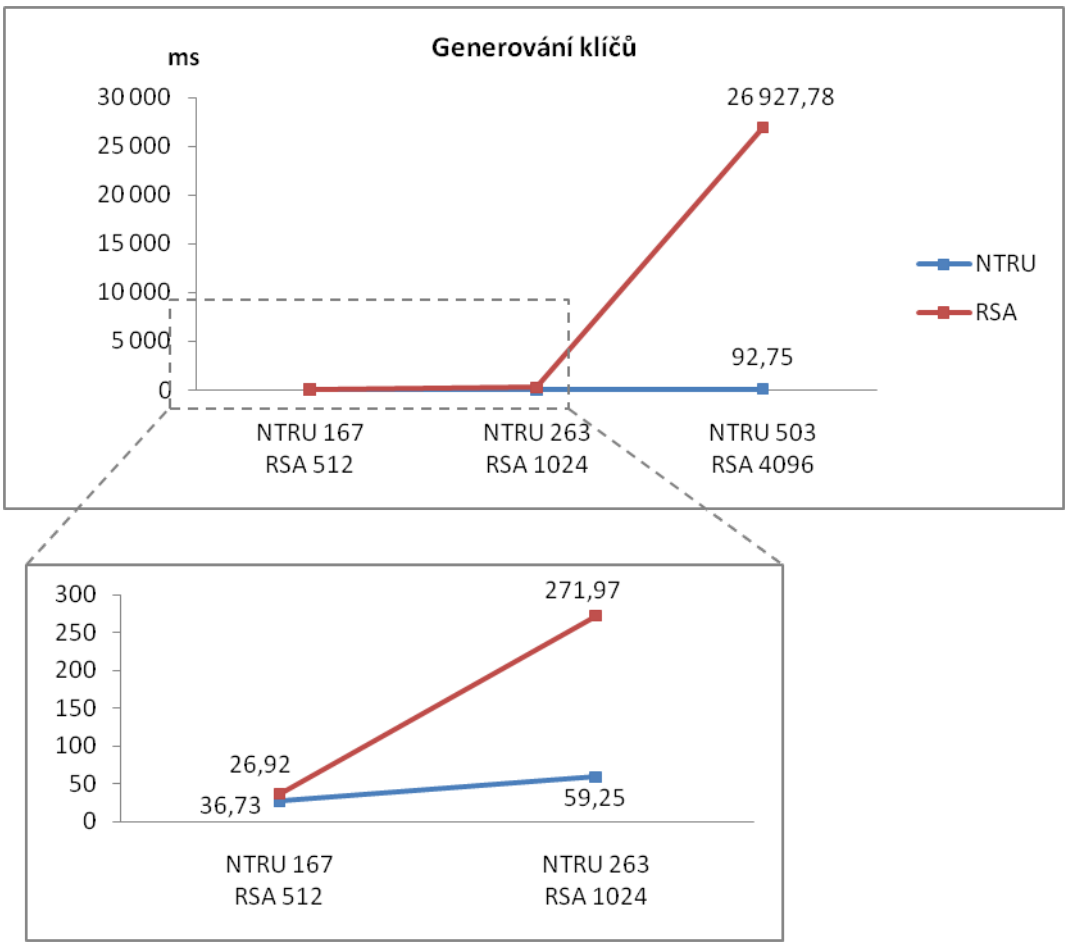
Tabulka A–16. Rychlost šifrování zprávy o velikosti 400 000 bitů

	NTRU			RSA		
č.	**167**	**263**	**503**	**512**	**1024**	**4096**
1	161,750	233,546	148,000	91,078	295,265	1 681,38
2	161,359	229,641	148,843	92,171	293,171	1 680,36
3	161,281	234,360	150,485	90,859	293,906	1 675,78
4	161,203	229,906	150,781	90,343	292,906	1 676,06
5	161,156	233,875	149,250	90,422	291,687	1 678,53
6	161,562	232,109	150,250	90,218	288,313	1 679,17
7	161,468	232,234	151,610	90,656	289,765	1 682,89
8	160,984	229,235	150,688	90,703	287,344	1 683,98
9	161,359	230,641	147,781	90,671	287,000	1 681,92
10	161,109	230,140	151,796	90,078	286,719	1 692,94
Průměr na test (s)	161,3231	231,5687	149,9484	90,7199	290,6076	1 681,301
Průměr na zprávu (ms)	**8 066,15**	**11 578,43**	**14 994,84**	**4 535,9**	**14 530,4**	**168 130**

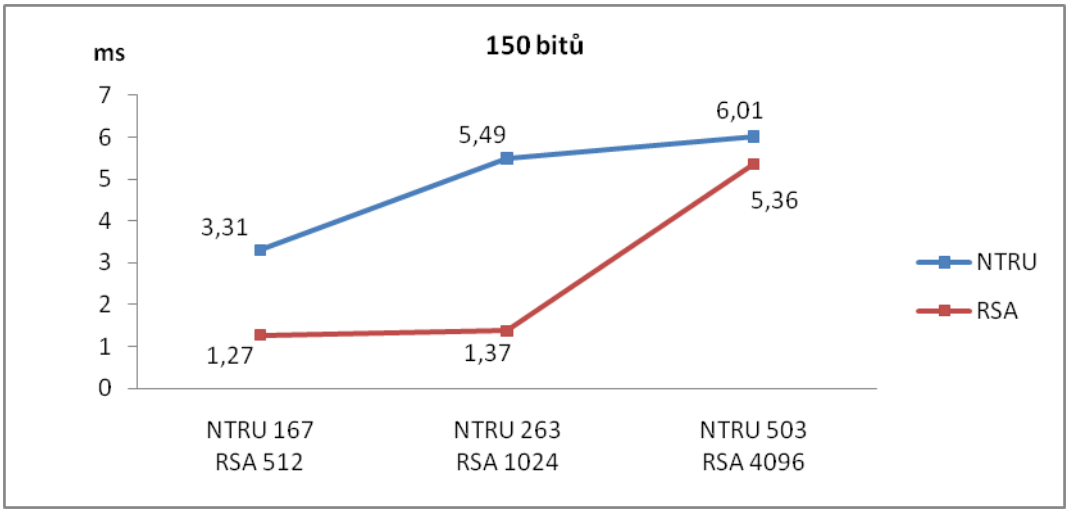
Tabulka A–17. Rychlost dešifrování zprávy o velikosti 400 000 bitů

KAPITOLA B

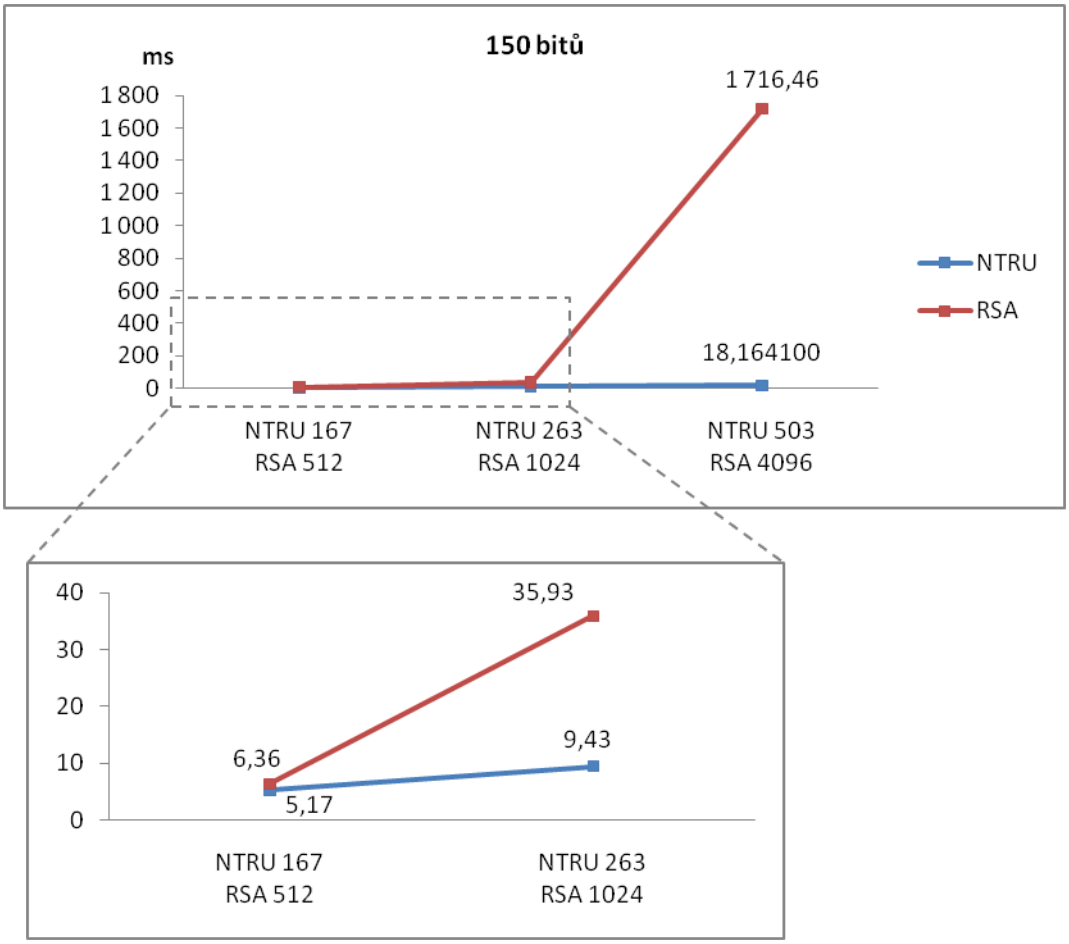
Grafy



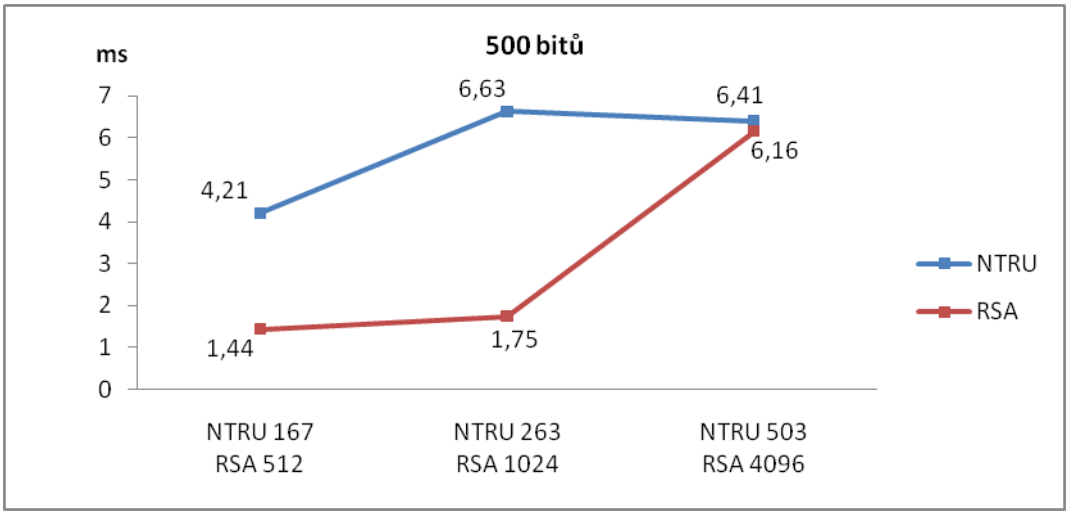
Graf B–1. Rychlost generování klíčů



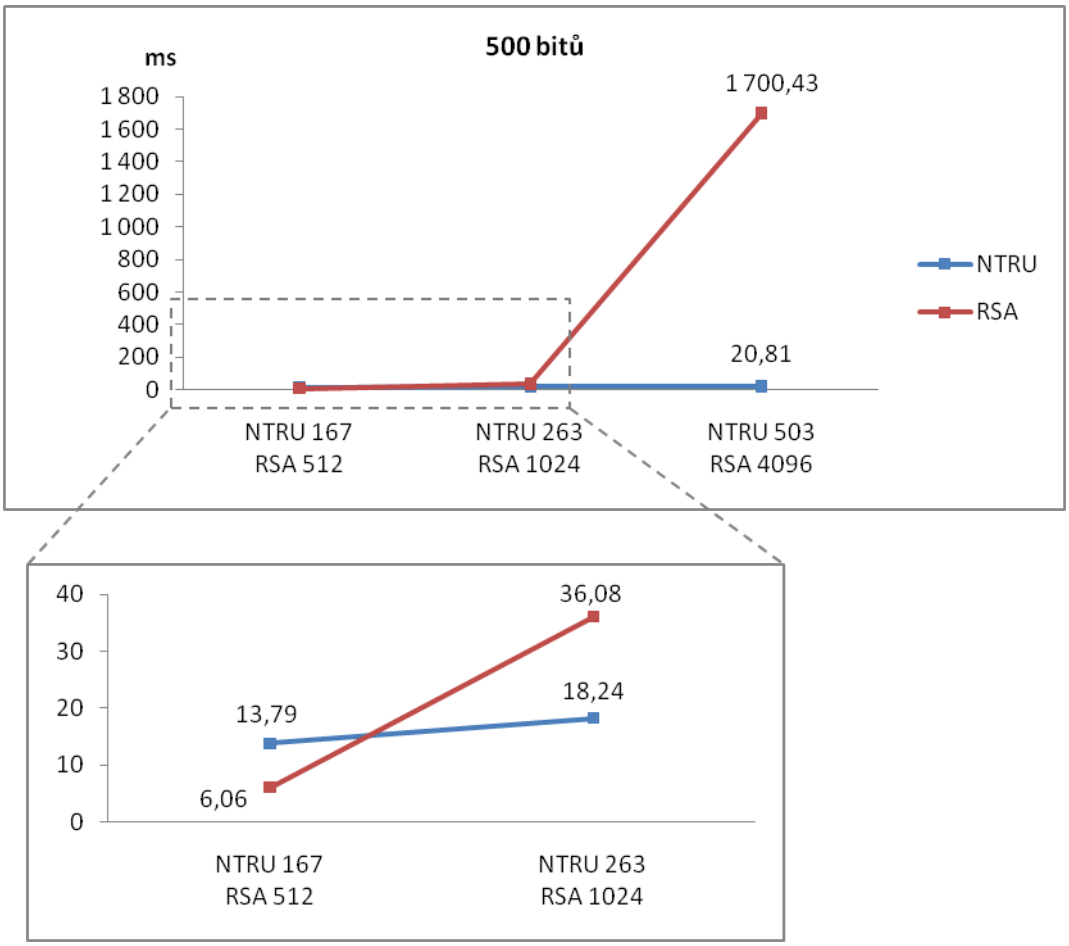
Graf B–2. Rychlost šifrování zprávy o velikosti 150 bitů



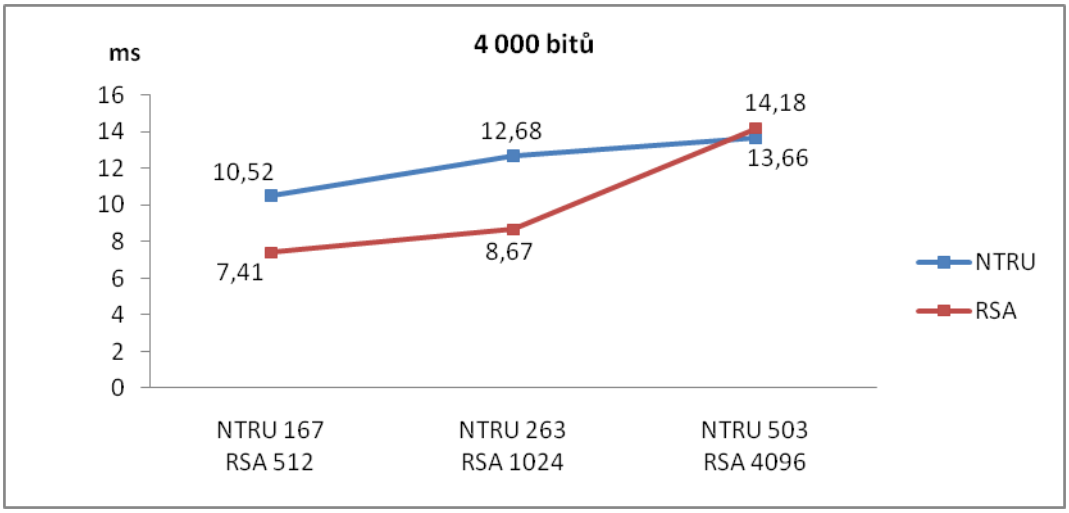
Graf B–3. Rychlost dešifrování zprávy o velikosti 150 bitů



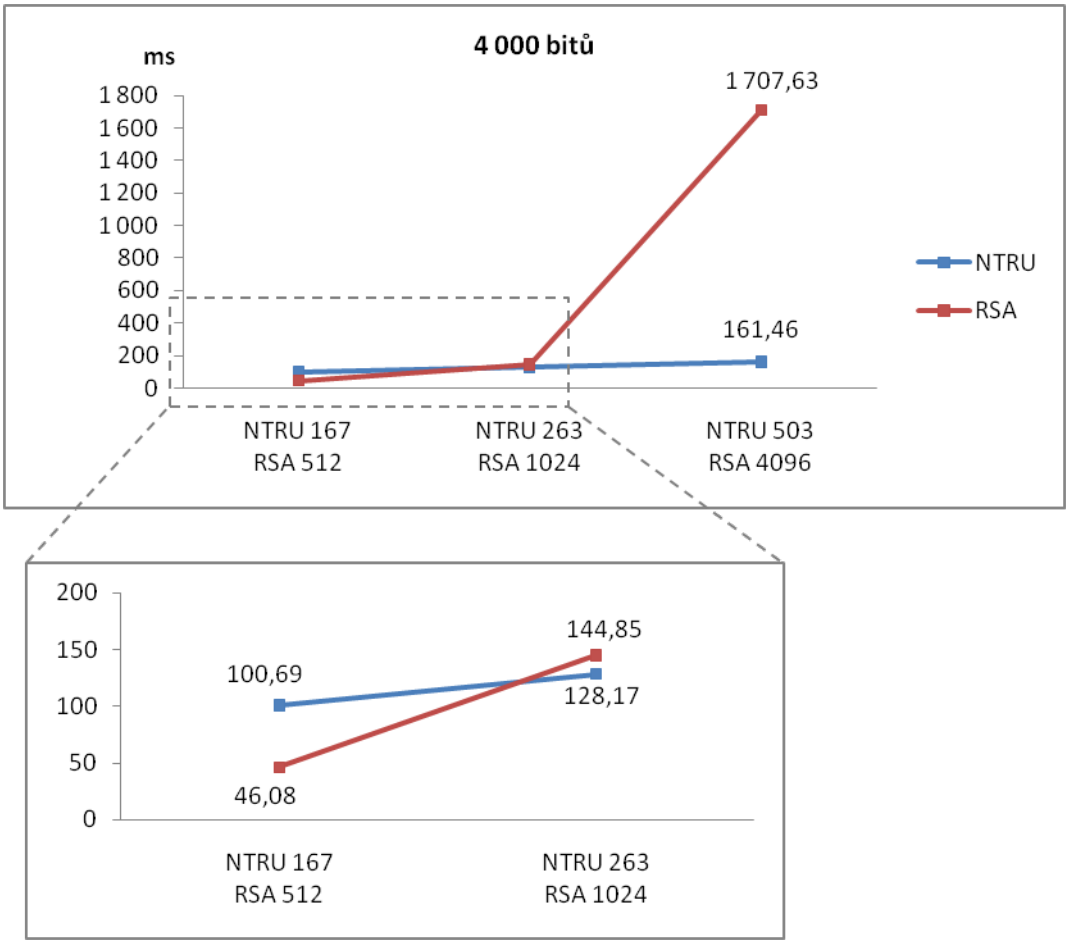
Graf B–4. Rychlost šifrování zprávy o velikosti 500 bitů



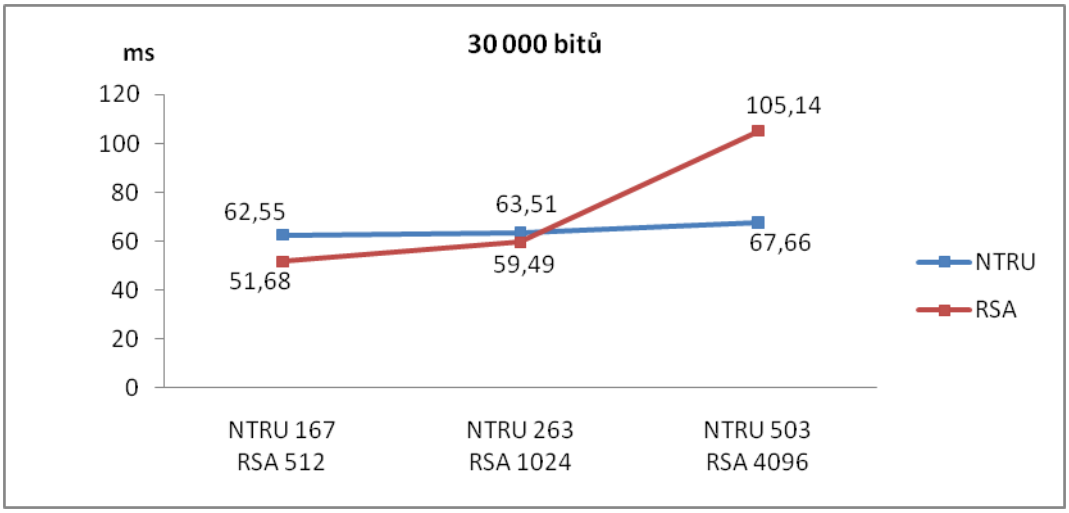
Graf B–5. Rychlost dešifrování zprávy o velikosti 500 bitů



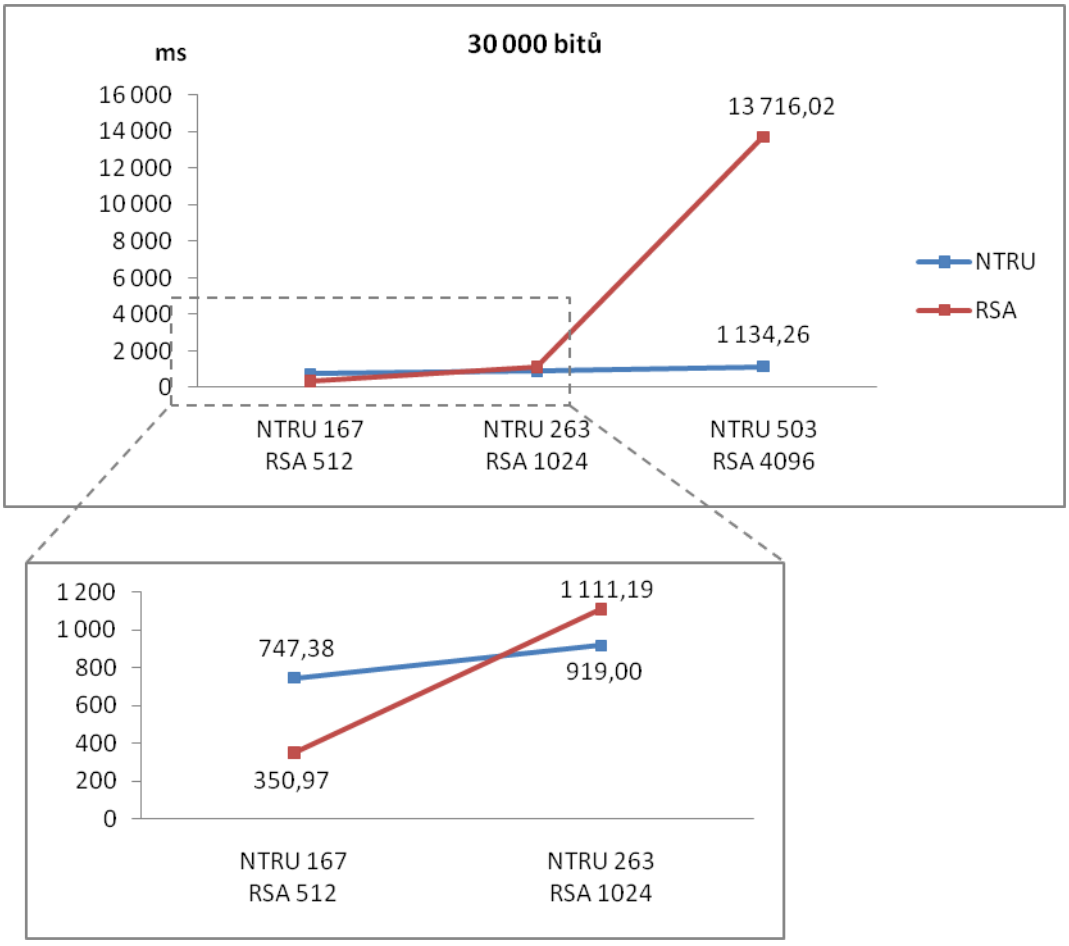
Graf B–6. Rychlost šifrování zprávy o velikosti 4 000 bitů



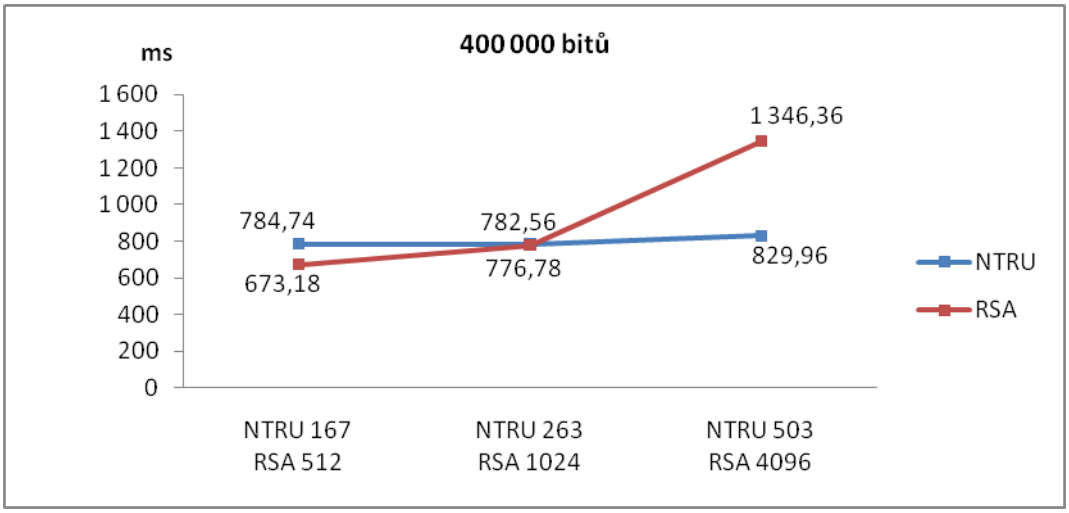
Graf B–7. Rychlost dešifrování zprávy o velikosti 4 000 bitů



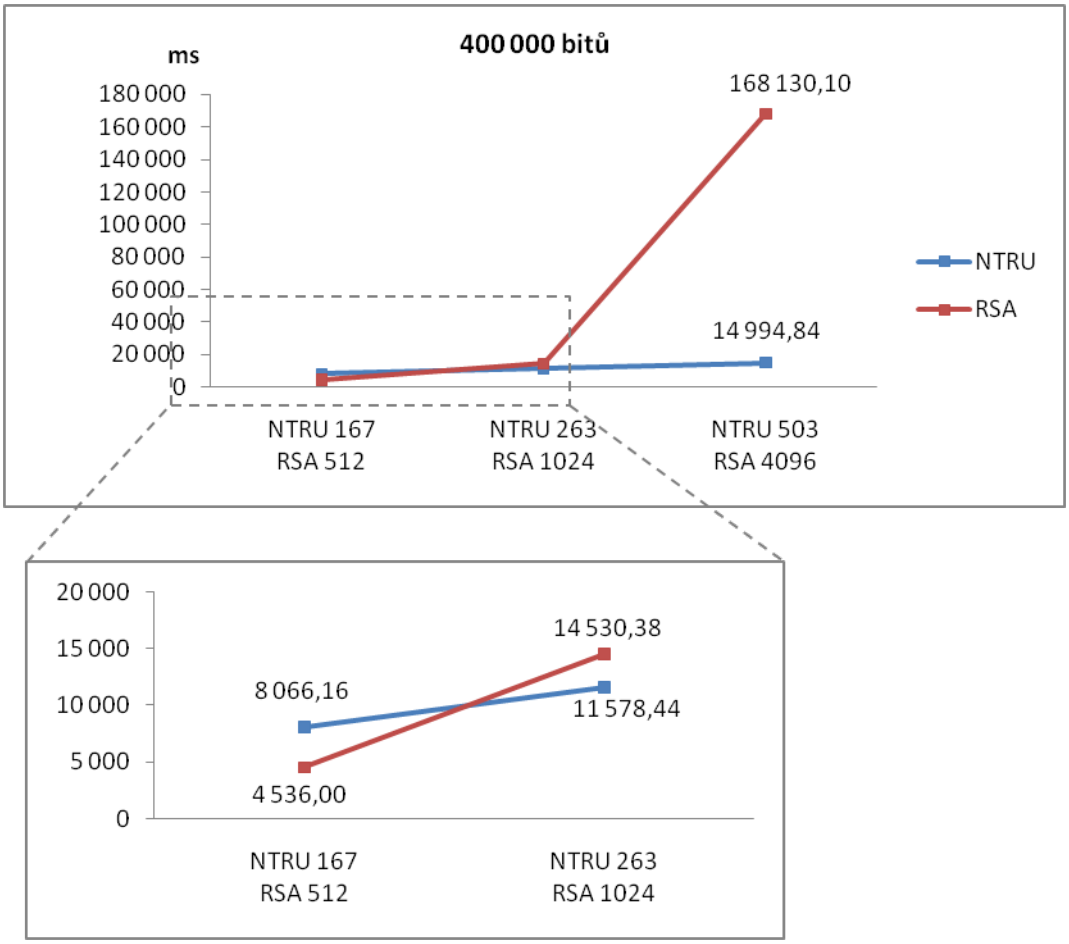
Graf B–8. Rychlost šifrování zprávy o velikosti 30 000 bitů



Graf B–9. Rychlost dešifrování zprávy o velikosti 30 000 bitů



Graf B–10. Rychlost šifrování zprávy o velikosti 400 000 bitů



Graf B–11. Rychlost dešifrování zprávy o velikosti 400 000 bitů