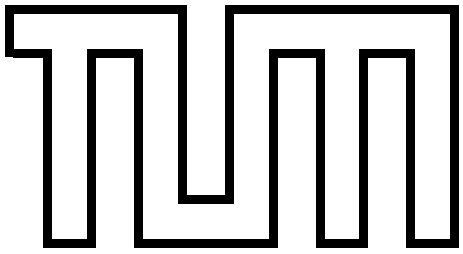

Technische Universität München

Fakultät für Informatik

Forschungs- und Lehreinheit Informatik VII

Lattice-based Cryptography
Kryptographie in Gittern

Masterseminar

Sebastian Rettenberger

Betreuer: Dr. Michael Luttenberger

Abgabetermin: 23. Juni 2011

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Asymmetrische Kryptographie ist heute ein wichtiger Bestandteil vieler Computersysteme und wird u.a. zu Verschlüsselung, in digitalen Signaturen oder zur Identifikation eingesetzt. In asymmetrischen Kryptosystemen spielen zahlentheoretische Probleme eine wichtige Rolle. So basiert RSA z.B. auf dem schweren Problem große Integer zu faktorisieren. Dabei ist dieses Problem eigentlich nur worst-case schwer. Das heißt, sehr viele Zahlen können trivial faktorisiert werden, wie z.B. alle geraden Zahlen.

In der Kryptographie braucht man aber Probleme, bei denen möglichst alle Instanzen schwer sind. Beim RSA wird deshalb das Produkt aus 2 $\frac{n}{2}$-Bit Primzahlen verwendet. Für solche Produkte vermutet man, dass sie average-case schwer sind. Ist n groß genug, ist es nach gegenwärtigem Kenntnisstand also nicht möglich, aus dem Produkt selbst, die zwei Primzahlen zu berechnen.

Auch wenn Quantencomputer noch von keiner praktischen Bedeutung sind, so wurde von Shor [Sho95] gezeigt, dass bei entsprechendem Stand der Technik mit ihrer Hilfe die relevanten zahlentheoretischen Probleme (Ganzzahlfaktorisierung und diskreter Logarithmus) effizient gelöst, und damit Verfahren wie RSA oder ElGamal gebrochen werden können.

Viele Probleme in Gittern sind vermutlich average-case schwer[1] und bist jetzt gibt es noch keinen Quantenalgorithmus, der die Probleme schneller als auf herkömmlichen Computern lösen könnte.

1.2 Gitter

Anschaulich sind Gitter eine Menge von Punkten in einem n-dimensionalen Raum mit einer periodischen Struktur.

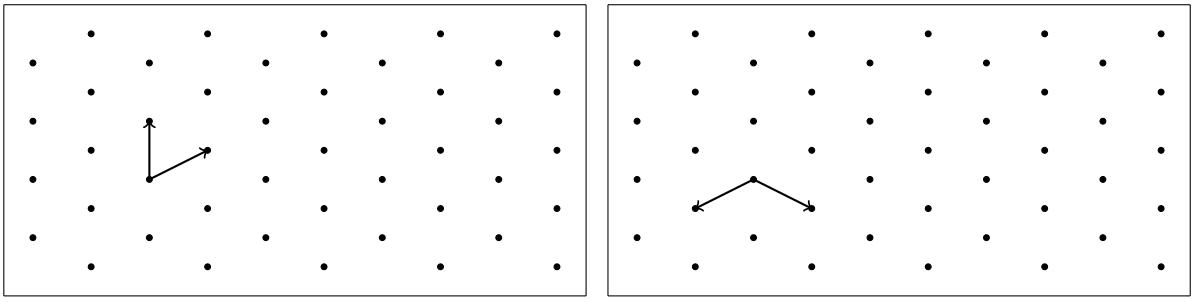

Abbildung 1: 2-dimensionales Gitter mit 2 möglichen Basen

[1] Für manche wurde dies sogar nachgewiesen.

Jedes Gitter $\mathcal{L}$ wird durch eine Basis aus n linear unabhängigen Vektoren $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$ beschrieben:

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}$$

bzw. in Matrixschreibweise:

$$\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\}$$

Für die Kryptographie sind vor allem **q-adische Gitter** für ein $q \in \mathbb{Z}$ interessant. Für q-adische Gitter gilt:

$$\mathbf{x} \in \mathcal{L} \iff \mathbf{x} \mod q \in \mathcal{L}$$

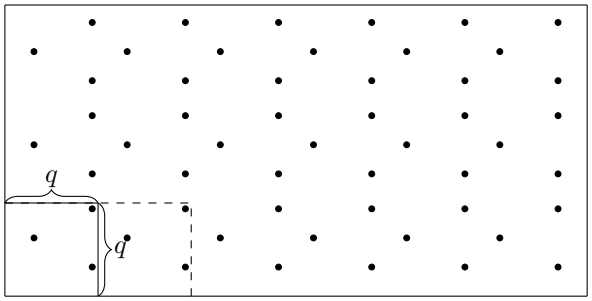

Abbildung 2: 2-dimensionales q-adisches Gitter

Setzt man $\det(\mathcal{L}(\mathbf{B})) = |\det(\mathbf{B})|$, so ist jedes Gitter $\mathcal{L}$ q-adisch, falls q ein Vielfaches von $\det(\mathcal{L})$ ist. In der Kryptographie interessieren jedoch hauptsächliche q-adische Gitter mit $q < \det(\mathcal{L})$. [MR08]

Insbesondere sind folgende Gitter interessant:

$$\begin{aligned} \Lambda_q(\mathbf{A}) &= \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y} = \mathbf{A}^T\mathbf{s} \mod q, \mathbf{s} \in \mathbb{Z}^n\} \\ \Lambda_q^{\perp}(\mathbf{A}) &= \{\mathbf{y} \in \mathbb{Z}^m : \mathbf{Ay} = \mathbf{0} \mod q\} \end{aligned}$$

mit $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $q, m, n \in \mathbb{Z}^+$ und $n \leq m$.

Dabei wird die Dimension der Gitter um $m-n$ Dimensionen erweitert. Alle Gitterprobleme in $\Lambda_q(\mathbf{A})$ und $\Lambda_q^{\perp}(\mathbf{A})$ lassen sich jedoch auf Probleme in $\mathcal{L}(\mathbf{B})$ für ein $\mathbf{B} \in \mathbb{R}^{n \times n}$ reduzieren. Das heißt insbesondere, dass die zusätzlichen $m - n$ Dimensionen keinen Einfluss auf die Schwierigkeit der Probleme haben. [MR08]

1.3 Gitterprobleme

Auf Gittern lassen sich verschiedene schwere Probleme definieren. Die wichtigsten drei sind:

Shortest Vector Problem (SVP) Gegeben ein Gitter mit Basis $\mathbf{B}$, finde den kürzesten Vektor in $\mathcal{L}(\mathbf{B})$ ungleich 0.

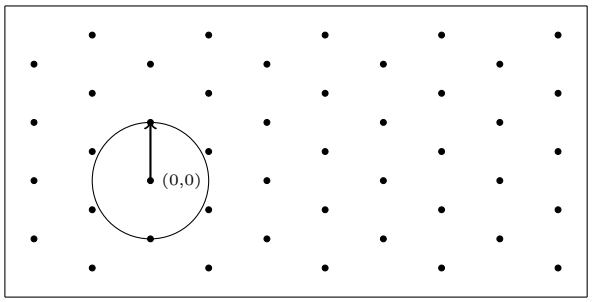

Abbildung 3: SVP mit einer möglichen Lösung

Closest Vector Problem (CVP) Gegeben ein Gitter mit Basis $\mathbf{B}$ und einem beliebigen Zielvektor $\mathbf{t}$, finde $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ mit der kürzesten Distanz zu $\mathbf{t}$.

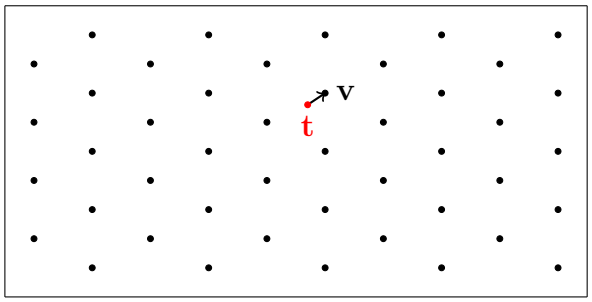

Abbildung 4: Ein Closest Vector Problem

Shortest Independent Vectors Problem (SIVP) Gegeben ein Gitter mit Basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, finde n linear unabhängige Vektoren $\mathbf{s}_i$, so dass $\max_i \|\mathbf{s}_i\|$ minimal ist.

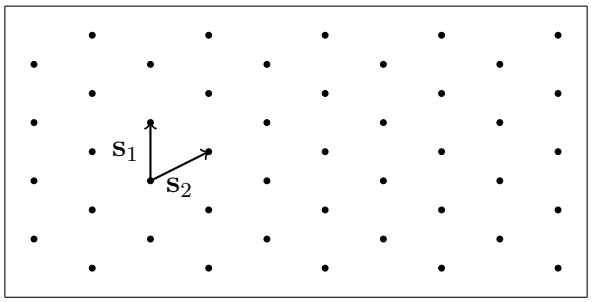

Abbildung 5: SIVP mit einer möglichen Lösung

Oftmals interessiert nicht einmal die exakte Lösung für ein Problem, sondern nur eine Nährung mit dem Faktor $\gamma(n) > 1$. Angenommen, die optimale Lösung für das SVP eines Gitters mit Dimension n ist $\mathbf{x}^*$, dann langt es also eine Lösung $\mathbf{x}$ mit $\|\mathbf{x}\| \leq \gamma(n)\|\mathbf{x}^*\|$ zu finden.

Für $\gamma(n)$ (vgl. [MR08])

- $\geq \sqrt{n/\log n}$ ist das SVP_γ vermutlich[2] nicht NP-schwer
- $\leq n^{O(1/\log\log n)}$ ist SVP_γ nachweislich NP-schwer

Von dem CVP_γ weiß man, dass es mindestens so schwer wie das SVP_γ ist, da sich mit Hilfe eines Algorithmus für das CVP_γ ein Algorithmus für das SVP_γ konstruieren lässt, der höchstens polynomiell längere Laufzeit hat. [GMSS99]

Es existieren verschiedene Ansätze um Gitterprobleme zu lösen. Einer davon ist der LLL Algorithmus[3]. Zu einer gegebenen Basis $\mathbf{B}$ konstruiert der Algorithmus eine Basis $\mathbf{B}'$ mit möglichst orthogonalen Vektoren. Der Algorithmus besitzt zwar eine polynomielle Laufzeit, findet aber nur eine Näherung mit $\gamma(n) = 2^{O(n)}$ für das SVP. Andere Algorithmen, wie z.B. die kombinatorische Methode, finden Lösungen mit $\gamma(n) = \text{poly}(n)$, haben jedoch Laufzeiten in NP. [MR08]

Trotzdem spielt die kombinatorische Methode eine wichtige Rolle in der Kryptoanalyse. Da sie oftmals bessere Ergebnisse als LLL erzielt, wird sie meist zum Festlegen der Parameter verwendet, um eine gewisse Sicherheit zu gewährleisten.

Zusammengefasst lässt sich also folgende Vermutung aufstellen:
Vermutung 1. *Es gibt keinen Algorithmus mit polynomieller Laufzeit, der Gitterprobleme auf einen polynomiellen Faktor annähert. [MR08]*

Es gilt sogar noch mehr. Bis jetzt gibt es noch nicht einmal einen Quantenalgorithmus, der die Gitterprobleme in polynomieller Zeit lösen kann.
Vermutung 2. *Es gibt keinen Quantenalgorithmus, der Gitterprobleme in Polynomialzeit auf polynomielle Faktoren annähert. [MR08]*

2 Hashfunktionen

Hashfunktionen spielen in der Informatik eine wichtige Rolle. Die kryptographischen Hashfunktionen sind sogenannte Einwegfunktionen. Das heißt, gegeben ein $x \in D$ ist es leicht ein $h(x) \in R$ zu berechnen. Umgekehrt ist es aber schwer, aus einem $y \in R$ ein $h^{-1}(x) \in D$ zu berechnen. Wie bei allen Hashfunktionen gilt $|R| \ll |D|$.

Bei kryptographischen Hashfunktionen unterscheidet man zwei wesentliche Sicherheitsklassifizierungen (vgl. [Buc08]):

Schwache Kollisionsresistenz Für einen gegebenen Wert x ist es praktisch unmöglich einen Wert $x' \neq x$ mit $h(x) = h(x')$ zu finden.

[2]Solange die Zeithierarchie gilt.
[3]benannt nach Lenstra, Lenstra und Lovástra

Starke Kollisionsresistenz Es ist praktisch unmöglich 2 Werte x, x' mit $x \neq x'$ und $h(x) = h(x')$ zu finden.

Hashfunktionen, die die starke Kollisionsresistenz erfüllen, werden auch als kollisionsresistente Funktionen bezeichnet.

Ein Vorschlag für eine kollisionsresistente schlüsselabhängige Hashfunktion, deren Sicherheit auf Gitterproblemen beruht, kommt von Ajtai:

Algorithmus 1 Ajtai's Konstruktion

Parameter: $n, m, q, d \in \mathbb{N}$
Mögliche Werte: $d = 2, q = n^2$
Schlüssel: $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ zufällig
Hashfunktion: $f_{\mathbf{A}} : \mathbb{Z}_d^m \to \mathbb{Z}_q^n$
mit $f_{\mathbf{A}}(\mathbf{y}) = \mathbf{A}\mathbf{y} \mod q$

Um eine sinnvolle Hashfunktion zu erhalten, muss $|\mathbb{Z}_d^m| > |\mathbb{Z}_q^n|$ gelten. Eine Bedingung für die Parameter ist also $m \log d > n \log q$, bzw. $m > n \log q / \log d$.

Behauptung 1. *$f_{\mathbf{A}}$ ist stark kollisionsresistent.*

Beweis. Angenommen, man findet eine Kollision $\mathbf{y} \neq \mathbf{y}'$ mit $f_{\mathbf{A}}(\mathbf{y}) = f_{\mathbf{A}}(\mathbf{y}')$, dann ist

- $\mathbf{0} = f_{\mathbf{A}}(\mathbf{y}) - f_{\mathbf{A}}(\mathbf{y}') = \mathbf{A}\mathbf{y} - \mathbf{A}\mathbf{y}' = \mathbf{A}(\mathbf{y} - \mathbf{y}')$
  $\Rightarrow \mathbf{y} - \mathbf{y}' \in \Lambda_q^{\perp}(\mathbf{A})$
- $\|\mathbf{y} - \mathbf{y}'\| = \sqrt{\sum_i (y_i - y_i')^2} \leq \sqrt{\sum_i d^2} = d\sqrt{m}$

$\Rightarrow \mathbf{y} - \mathbf{y}'$ ist eine Näherung für das SVP
Für $d = 2$, $q = n^2$ und $m \approx 2n \log q / \log d$:

$$\|\mathbf{y} - \mathbf{y}'\| < \text{poly}(n)$$

□

Ajtai's Konstruktion ist sehr einfach und kommt lediglich mit Addition und Multiplikation aus. Wählt man q als Zweier-Potenz, kann man sogar auf Multiplikationen verzichten. Allerdings braucht man für 100 Bits Sicherheit einen Schlüssel mit ∼500.000 Bits und ∼50.000 arithmetische Operationen, um $\mathbf{f}_A$ zu berechnen, was für eine einfache Hashfunktion deutlich zu groß ist. [MR08]

3 Public-Key Verschlüsselung

Public-Key Verfahren verwenden im Gegensatz zu Private-Key Verfahren zwei verschiedene Schlüssel zum ver- und entschlüsseln. Dabei wird der öffentliche Schlüssel zum Ver-

schlüsseln verwendet und der private Schlüssel zum Entschlüsseln. Wichtig ist dabei, dass der private Schlüssel aus dem öffentlichen nicht einfach berechnet werden kann.

Möchte A eine verschlüsselte Nachricht von B empfangen, kann A einfach seinen öffentlichen Schlüssel veröffentlichen und B kann dann damit die Nachricht verschlüsseln. Der Vorteil gegenüber Private-Key Verfahren ist, dass A und B kein Geheimnis (z.B. einen privaten Schlüssel) austauschen müssen, um sich gegenseitig verschlüsselte Nachrichten zu schicken. [Buc08]

Ein Public-Key Verfahren, dessen Sicherheit auf der Schwierigkeit von Gittern beruht, ist ein LWE (Learning with errors) basierendes Verschlüsselungsverfahren:

Algorithmus 2 LWE basierendes Verschlüsselungsverfahren

Parameter: $n, m, l, t, r, q \in \mathbb{N}$ und $\alpha \in \mathbb{R}^+ \setminus \{0\}$
Schlüsselerzeugung: $\mathbf{S} \in \mathbb{Z}_q^{n \times l}$, $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ zufällig uniform, $\mathbf{E} \in \mathbb{Z}_q^{m \times l}$ zufällig mit Verteilung Ψ_α
Privater Schlüssel: $\mathbf{S} \in \mathbb{Z}_q^{n \times l}$
Öffentlicher Schlüssel: $(\mathbf{A}, \mathbf{P} = \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times l}$
Verschlüsselung: Klartext $\mathbf{v} \in \mathbb{Z}_t^l$, $\mathbf{a} \in \{-r, -r+1, \ldots, r\}^m$ zufällig
Geheimtext $(\mathbf{u} = \mathbf{A}^T\mathbf{a}, \mathbf{c} = \mathbf{P}^T\mathbf{a} + f(\mathbf{v})) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^l$
Entschlüsselung: Geheimtext $(\mathbf{u}, \mathbf{c})$
Originaltext $f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{u})$

Die Verteilung Ψ_α ähnelt dabei der Normalverteilung mit Erwartungswert 0 und Standardabweichung $\frac{\alpha q}{\sqrt{2\pi}}$. Jedoch wird beim Erzeugen der Zufallszahlen das Ergebnis zusätzlich auf die nächste ganze Zahl gerundet und modulo q genommen. [MR08]

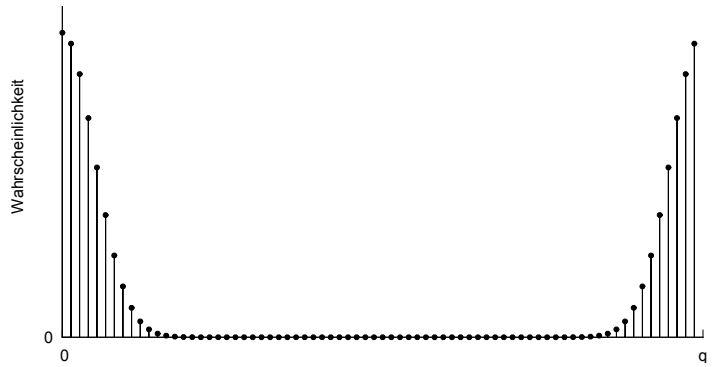

Abbildung 6: Ein Beispiel für Ψ_α

Die Funktion f bildet Vektoren von $\mathbb{Z}_t^l$ auf $\mathbb{Z}_q^l$ ab, in dem jeder Eintrag des Vektors mit q/t multipliziert und anschließend auf die nächste ganze Zahl gerundet wird. f^{-1} ist die entsprechende „Umkehrfunktion“ dazu, die jeden Wert mit t/q multipliziert und danach rundet. [MR08] Da jedes Mal auf ganze Zahlen gerundet wird, folgt daraus nicht unbedingt $f^{-1} \circ f(\mathbf{x}) = \mathbf{x}$ bzw. $f \circ f^{-1}(\mathbf{x}) = \mathbf{x}$.

Entschlüsselt man nun einen Geheimtext $(\mathbf{u}, \mathbf{c})$, so erhält man nicht zwangsweise den Originaltext $\mathbf{v}$:

$$\begin{aligned} f^{-1}(\mathbf{c} - \mathbf{S}^T\mathbf{u}) &= f^{-1}(\mathbf{P}^T\mathbf{a} + f(\mathbf{v}) - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ &= f^{-1}((\mathbf{AS} + \mathbf{E})^T\mathbf{a} + f(\mathbf{v}) - \mathbf{S}^T\mathbf{A}^T\mathbf{a}) \\ &= f^{-1}(\mathbf{E}^T\mathbf{a} + f(\mathbf{v})) \end{aligned}$$

Für $t \ll q$ gilt zwar, $\mathbf{v} = f^{-1}(f(\mathbf{v}))$ mit sehr großer Wahrscheinlichkeit, jedoch kann der Term $\mathbf{E}^T\mathbf{a}$ zu Fehlern führen. Die Wahrscheinlichkeit für einen Fehler lässt sich zwar durch eine geschickte Wahl der Parameter klein halten, ist jedoch nicht ganz auszuschließen.

Die Sicherheit des Verschlüssungsverfahrens garantiert der folgende Satz:
Satz 1. *Angenommen, man könnte das LWE Problem mit Parametern n, m, q, Ψ_α mit $\alpha q > \sqrt{n}, q \leq \text{poly}(n)$ Primzahl und $m \leq \text{poly}(n)$ in Polynomialzeit lösen, dann gibt es einen Quantenalgorithmus, der alle $SIVP_{\Theta(n/\alpha)}$ in Polynomialzeit löst.*

Satz 1 macht bereits erste Einschränkungen an die Parameter. Möchte man die Fehlerwahrscheinlichkeit, die Schlüsselgröße und den Vergrößerungsfaktor der verschlüsselten Nachricht möglichst klein halten, so bekommt man weitere sinnvolle Bedingungen für die Parameter. Tabelle 1 zeigt einige mögliche Parameterbelegungen.

n, l	136	166	192	214	233	233
m	2008	1319	1500	1333	1042	4536
q	2003	4093	8191	16381	32749	32749
r	1	4	5	12	59	1
t	2	2	4	4	2	40
α	0,0065	0,0024	0,0009959	0,00045	0,000217	0,000217
Öffentlicher Schlüssel (in Bits)	6×10^6	$5,25 \times 10^6$	$7,5 \times 10^6$	8×10^6	$7,3 \times 10^6$	$31,7 \times 10^6$
Vergrößerungsfaktor	21,9	24	13	14	30	5,6
Fehlerwahrscheinlichkeit	0,9%	0,56%	1%	0,8%	0,9%	0,9%
Gittergröße beim Angriff	322	372	417	457	493	493

Tabelle 1: Mögliche Parameter für das LWE-Verfahren [MR08]

Sehr positiv am LWE basierenden Verfahren ist, dass die verschlüsselte Nachricht nur um einen konstanten Faktor größer als der Originaltext ist. Auch mit $\Theta(n)$ Operationen pro verschlüsseltem Bit ist der Algorithmus nicht schlecht. Da jedoch der Schlüssel aus einer Matrix besteht, ist man hier in der Größenordnung $\Theta(n)$. Auch die Wahrscheinlichkeit für einen Fehler fällt negativ auf. Dieser lässt sich jedoch mit fehlerkorrigierenden Codes noch weiter senken.

Außerdem ist das Verfahren zwar sicher gegen Chosen-Plaintext-Attacken (CPA)[4], kann aber mit einer Chosen-Ciphertext-Attacke (CCA) gebrochen werden. [MR08]

[4]Dieser Angriff funktioniert bei Public-Key Verfahren immer. [Buc08]

4 Ausblick

Auch wenn Kryptographie auf Gittern noch in den Kinderschuhen steckt, sind bereits deutliche Fortschritte gemacht worden. Wählt man in Ajtai's Konstruktion für $\mathbf{A}$ eine spezielle Struktur, so lässt sich zum einen die Schlüsselgröße deutlich verkleinern und zum anderen die Berechnung von $f_{\mathbf{A}}$ beschleunigen. Der SWIFFTX, eine Hashfunktion, die diese Verbesserungen gegenüber Ajtai's Konstruktion beinhaltet, wurde sogar für SHA-3 vorgeschlagen [NIS]. Auch wenn SWIFFTX angelehnt wurde, ist er bereits konkurrenzfähig zu anderen kryptographischen Hash-Algorithmen.

Eine ähnliche Verbesserung der Schlüsselgröße lässt sich auch für das LWE basierende Verfahren erhoffen. Genauso wünschenswert wäre auch eine klassische Reduktion auf Gitterprobleme. Dies würde die Sicherheit des Algorithmus unabhängig von Vermutung 2 machen. Ein Verbesserungsvorschlag kommt von Peikert und Waters. Durch die Einführung von *lossy trapdoor functions* konnte ein CCA-sicheres Public-Key Verfahren entwickelt werden.

Literatur

[Buc08] J. Buchmann. *Einführung in die Kryptographie*. Springer, 2008.

[GMSS99] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999.

[MR08] D. Micciancio and O. Regev. Lattice-based cryptography. *Post-quantum Cryprography. Springer*, 2008.

[NIS] Cryptographic hash algorithm competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/` [Letzer Zugriff: 22.06.2011].

[Sho95] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *Arxiv preprint quant-ph/9508027*, 1995.