Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Diplomarbeit Nr. 3520

Der Angriff auf Merkle-Hellman Kryptosystem

Sheng Gao

Studiengang:	Informatik
Prüfer:	Prof. Dr. Volker Diekert
Betreuer:	Armin Weiß
begonnen am:	17.05.2013
beendet am:	09.01.2014
CR-Klassifikation:	E.3, E.4, F.2.2

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 09. Jan. 2014 _____

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Jahre 1976 haben Diffie and Hellman das asymmetrischen Kryptosystem erfunden. 1978 entwickelten Merkle und Hellman mit dem Merkle-Hellman Kryptosystem eines der ersten asymmetrischen Kryptoverfahren. Sie haben ein asymmetrisches Kryptosystem erfunden, das auf dem NP-vollständigen Rucksackproblem basiert. Das Rucksackproblem ist eines der berühmtesten NP-vollständigen Probleme. Im Merkle-Hellman Kryptosystem wird eine stark wachsende Folge benutzt, die eine wichtige Information bringt. Adi Shamir hat im Jahre 1982 die erste Angriffsmethode auf Merkle-Hellman Kryptosystem erfasst, und später hat man durch die diophantische Approximation, den Density-Attack usw. das Merkle-Hellman Kryptosystem sowie seine Nachfolgen gebrochen. Außerdem hat man bewiesen, dass solches Kryptosystem mit einer Dichte(Density) kleiner als 0.9408... durch LLL-Algorithmus unsicher ist. Jedoch ist das Kryptosystem mit dem Rucksackproblem sowohl beim Verschlüsseln als auch beim Entschlüsseln sehr effizient. Viele neuen Varianten vom Merkle-Hellman Kryptosystem mit dem modifizierten Rucksackproblem sind noch mit hoher Sicherheit. Deshalb ist das Merkle-Hellman bzw. Rucksackproblem Kryptosystem heutzutage noch sinnvoll.

Dafür habe ich zuerst das Rucksackproblem und das Merkle-Hellman Kryptosystem erklärt. Außerdem habe ich die Einführung der neuen Entwicklung der Analyse von dem Merkle-Hellman Kryptosystem, seinen neuen Varianten und ggf. den Angriffsmethoden darauf. Danach habe ich über den Shamir's Angriff ausführlich beschrieben. Insbesondere habe ich über die Approximation von „Trapdoor Paar" analysiert. Zudem habe ich den Shamir's Angriff implementiert und getestet. Man hat dann die Performance vom Shamir's Angriff in Praxis gesehen. In der praktischen Implementierung sowie Testen habe ich einen Sonderfall bemerkt, dass mehrere „Trapdoor Paare" vorkommen können.

Darüber hinaus finde ich in der Analysis vom Shamir's Angriff eine Eigenschaft der stark wachsenden Folge, die eine Möglichkeit zur Kollision der öffentlichen Schlüssel bringen

kann. Am Ende habe ich eine Zusammenfassung vom Merkle-Hellman Kryptosystem mit dem Shamir's Angriff gegeben, und gleichzeitig ein offenes Thema für die Kollision der öffentlichen Schlüssel gestellt.

Kapitel 2

Merkle-Hellman Kryptosystem

Die Grundidee vom Merkle-Hellman Kryptosystem ist, dass eine relativ einfache Instanz des Rucksackproblems zuerst erzeugt wird, die als privater Schlüssel zum Entschlüsseln zu benutzen ist. Dann wird aus dieser einfachen Instanz eine möglichst schwere Instanz berechnet, die als öffentlicher Schlüssel benutzt wird[1]. Wenn der Absender eine Nachricht an den Empfänger sendet, kann der Absender zuerst mit dem Rucksackproblem verschlüsseln. Weil dieses Problem NP-vollständig ist, kann der Angreifer die Verschlüsselung nicht brechen. Dies war der Gedanke von Merkle und Hellman. Leider ist der Gedanke nicht ganz korrekt. Es liegt daran, dass das Rucksackproblem von Merkle-Hellman eigentlich eine wichtige Lücke hat und nicht wirklich ausreichend sicher ist. Nun folgt eine Einführung zum Rucksackproblem, asymmetrischem Kryptosystem und dem Merkle-Hellman Verfahren:

2.1 Rucksackproblem

Das Rucksackproblem heißst auf Englisch *Knappsack* oder *Subset Problem*. Gegeben sind n Objekte und einen Rucksack mit der Kapazität (Gewichtsgrenze) C.

Eingabe:

$P = (p_1, ..., p_n)$	:	Menge der Rucksäcke
$W = (w_1, ..., w_n)$	:	Gewichtsmenge jedes Rucksacks
C	:	(Gesamte)Kapazität

Gesucht:

Maximiere $\sum_{i=1}^{n} x_i p_i$ unter $\sum_{i=1}^{n} x_i w_i = C$, $x \in \{0, 1\}$

Wir definieren noch die Nutzendichte (vom Objekt i): $\frac{p_i}{w_i}$, wobei alle $\frac{p_i}{w_i}$ sortiert sind. Es soll gelten, dass $\sum_{i=1}^{n} w_i > C$, $w_i < C$, für alle i. Das Ziel ist eine Auswahl einer Teilmenge von Objekten mit maximalem Nutzen unter Beachtung der Kapazität.

2.2 Asymmetrisches Kryptosystem

Für das asymmetrische Kryptosystem kann man in mathematischer Formulierung so definieren[2]:

Pr	Menge der privaten Schlüssel
Pb	Menge der öffentlichen Schlüssel
U	Menge der Nutzer (User), Teilnehmer
	Jeder Teilnehmer $A \in U$ hat einen privaten Schlüssel $Pr(A)$
	sowie einen öffentlichen Schlüssel $Pb(A)$
M	Menge von Klartexten
C	Menge der chiffrierten Nachrichten
$f : Pb \times M \to C$	Chiffrierfunktion
$g : Pr \times C \to M$	Dechiffrierfunktion

Bei einem Asymmetrischen Verschlüsselungsverfahren (auch als *Public-Key-Verfahren* genannt), gibt es im Gegensatz zu einem Symmetrischen, nicht nur einen Schlüssel, sondern gleichzeitig zwei. Dieses sogenannte Schlüsselpaar setzt sich aus einem privaten Schlüssel (*engl.:* private key) und einem öffentlichen Schlüssel (nicht geheim, *engl.:* public key) zusammen. Für die Anwendung muss man einen Algorithmus finden, dass das Entschlüsseln für den Besitzer mit dem privaten Schlüssel relativ leicht ist, während der Angriff möglichst schwer sein kann, obwohl der öffentliche Schlüssel schon bekannt ist.

2.3 Merkle-Hellman Verfahren

Beim Rucksackproblem hat man eine Folge positiver reeller Zahlen, im grundlegenden Merkle-Hellman Verfahren[1] ist diese Folge natürliche Zahlen. Und diese Folge $(a_1, \ldots, a_n)$ ist stark wachsend (*engl.:* super increasing) und zwar $a_i > \sum_{j=1}^{i-1} a_j$ für alle $1 \leq i \leq n$. Die Idee von Merkle-Hellman ist diese „stark wachsende“ Eigenschaft zu nutzen und erzeugt eine neue Folge durch eine Modulo-Rechnung, um das Kryptosystem zu bilden. Diese Folge sowie deren Eigenschaft sind die Kernidee, sowohl von Merkle-Hellman zu ver- und

entschlüsseln, aus auch von Shamir anzugreifen[3, 4]. Neben der stark wachsenden Folge wird ein Modulus M mit $M > \sum_{i=1}^{n} a_i$ gewählt, um eine (quasi-) zufällige Folge zu erzeugen und dann zu verschlüsseln. Der richtige Empfänger braucht für Entschlüsseln noch einen Multiplikator U sowie dessen Inverse W mit $UW = 1 \bmod M$ und $U > 1$, $W > 1$. Um U sowie W schnell zu finden, soll M idealerweise prim sein.

Privater Schlüssel:

Wie alle asymmetrischen Kryptosysteme hat Merkle-Hellman Verfahren einen Geheimteil und zwar $(a_1, \ldots, a_n)$, M, U und W bilden den privaten Schlüssel für Alice.

Öffentlicher Schlüssel:

Alice veröffentlicht eine Folge $(b, \ldots, b_n)$ mit

$$0 \leq b_i < M$$

$$b_i = a_i U \bmod M$$

was sehr „zufällig“ aussieht, und veröffentlicht das als öffentlicher Schlüssel $(b_1, \ldots, b_n)$, den alle kennen können. Natürlich kann Alice noch eine Permutation π einsetzen, dann sieht die Folge $(b_1, \ldots, b_n)$ noch „zufälliger“ aus.

Seien alle Codierung binär, vorgeschlagen sind $n = 100$ und die erste Zahl der Folge, nämlich a_1, 100 Bits. Jede weitere i-te Zahl ist (maximal) $n+i-1$ Bits, damit $(a_1, \ldots, a_n)$ eine stark wachsende Folge wird. Daher sollte a_n 199 Bits lang und der Modulus M 200 Bits lang sein. Nach einer Modulo-Rechnung ist jedes b_i auch maximal $2n$ Bits lang.

Zu verschlüsseln:

Der Absender Bob nimmt einen Klartext durch Binärcodierung, nämlich eine n-Bits-lange Folge $x = (x_1, \ldots, x_n) \in B^n$, und verschlüsselt mit dem öffentlichen Schlüssel $(b, \ldots, b_n)$ duch die Berechnung der Summe $y = \sum_{i=1}^{n} x_i b_i$. Schließlich sendet er den Ciphertext y über den öffentlichen Kanal an den Empfänger Alice. Die Bitlänge von y ist maximal $2n + log(n)$, weil $y < n2^{2n}$ ist.

Zu Entschlüsseln:

Alice empfängt y und berechnet $c = yU \bmod M$, das ist nichts anders als

$$c = \sum_{i=1}^{n} x_i b_i W \bmod M = \sum_{i=1}^{n} x_i a_i \bmod M$$

Wegen $M > \sum_{i=1}^{n} a_i$ folgt $c = \sum_{i=1}^{n} x_i a_i$ und außerdem ist $(a_1, \ldots, a_n)$ stark wachsend, dann ist die Rekonstruktion von der Folge $x = (x_1, \ldots, x_n) \in B^n$, $B = \{0, 1\}$ leicht

zu haben. Falls Alice am Anfang noch eine Permutation π nimmt, die sie kennt, ist der Wiederherstellung der Reihenfolge beim Entschlüsseln für sie nicht schwer.

Das Kryptosystem ist für den Empfänger, der den privaten Schlüssel besitzt, polynomial zu n berechenbar, und auf der anderen Seite ist für Angreifer mit der Hoffnung möglichst nicht polynomial zu lösen, da das Kryptosystem auf dem Rucksackproblem mit der Komplexität von NP-schwer basieren sollte. Ein interessantes Bild, um das Mechanismus von Merkle-Hellman Verfahren darzustellen, und auch die Prinzip des asymmetrischen Kryptosystems, sieht man in Abbildung 2.1[5].

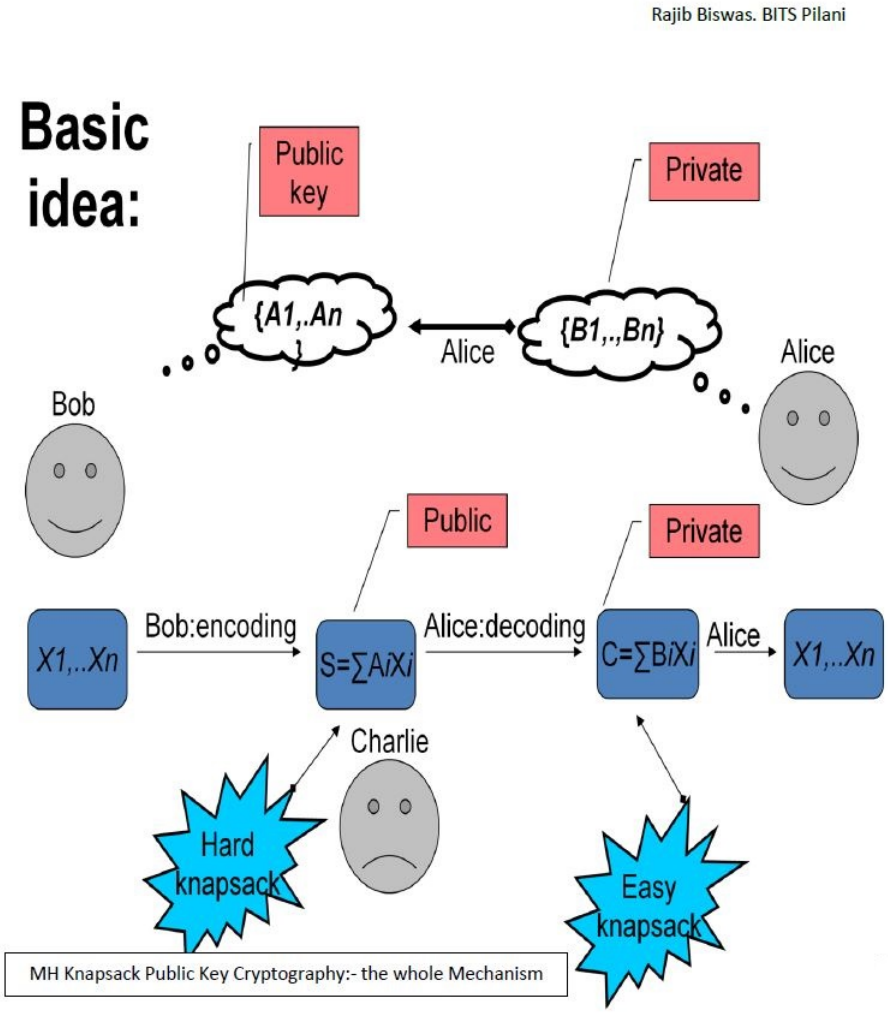

Abbildung 2.1: M-H Kryptosystem

2.4 Lücke von Merkle-Hellman Verfahren

Dass das Rucksackproblem NP-vollständig ist, soll aber etwas irreführend sein, dass das zugehörige Rucksackproblem im Merkle-Hellman Verfahren eigentlich nicht auch NP-vollständig ist. Insbesondere ist die Folge $(a_1, \ldots, a_n)$ im Vergleich zum normalen Rucksackproblem nicht nur „streng monoton steigend“, sondern auch „stark wachsend“. Vielleicht hatten Merkle-Hellman den Gedanke, dass das einfach beim Entschlüsseln und durch Modulo-Rechnung usw. noch schwer genug für Angreifen ist. Aber diese Eigenschaft der Folge bringt viele Informationen und man kann diese Lücke mit verschiedenen Methoden angreifen, damit das Merkle-Hellman Verfahren nicht mehr sicher ist. Folgende zwei Kapiteln werden erklärt, wie man das das Merkle-Hellman Verfahren bricht. Und für die erste Angriffsmethode von Shamir wird ausführlich beschrieben.

Kapitel 3

Neue Entwicklung der Angriffsmethoden

3.1 Überblick

Die erste Angriffsmethode gegen Merkle-Hellman Kryptosystem wurde von Adi Shamir aus Israel im Jahr 1982 erfunden. Sein Algorithmus ist gegen der grundlegenden Version und zwar das zugehörige Rucksackproblem von Merkle-Hellman ist mit einer stark wachsenden Folge für die Gewichtsmenge und daher kein normales Rucksackproblem. Die Bitlänge vom Modulus M usw. für einen n-Bits-langen Klartext sind dn, $1 \leq d \leq 2$, und das Element in der stark wachsenden Folge ist mit einer Bitlänge von $(d-1)n+i-1$. Diese originale Version von Merkle-Hellman, die auch eine Permutationen erlaubt, und sogar mit multi-iterierten Rucksäcken sein kann, werden später von Lagarias durch die diophantische Approximation gebrochen[6]. Außerdem werden andere Versionen, insbesondere „Niedrige Dichte Angriff“ erfunden, somit die Varianten mit niedrigem Verhältnis von Anzahl der *Elemente / Bitlänge* des öffentlichen Schlüssels auch unsicher sind. Dazu werden vielfältige Verfahren entwickelt und in diesem Kapitel wird eine Übersicht dafür erstellt.

Definition: *Informationsrate*, *Dichte(Density)*

Ein grundlegender aber wichtiger Parameter zur Analysis fürs Merkle-Hellman Kryptosystem ist der Begriff *Informationsrate*, und definiert durch

$$R = \frac{\#KlartextBits}{\#CipherertextBits} = \frac{n}{log_2 \sum_{i=1}^{n} a_i}$$

wobei a_i die Gewichte des Rucksacks, nämlich der öffentliche Schlüssel ist.

In manchen Papers wird auch „*Dichte*“, auf Englisch heißt „*Density*“, benutzt und ähnlich wie oben definiert:

$$d = \frac{n}{log_2 \, max \, a_i}$$

Offensichtlich gilt, dass $0 < R \leq 1$, $0 < d$ ist.

Wir können fürs Merkle-Hellman Kryptosystem seine *Dichte* sehen, also $d = 0,5$, wenn n sehr groß ist.

Lagarias und Odlyzko haben gezeigt, dass für $d < 0,6463$ „fast alle“ Merkle-Hellman Verfahren gelöst werden können[7].

Was zu beachten ist, dass hier das Rucksackproblem nicht genau gleich wie Merkle-Hellman Verfahren sein kann, d.h., eine stark wachsende Folge der Gewichte muss nicht unbedingt gewählt wird.

Seit einigen Jahren gibt es viele neuen Varianten von Merkle-Hellman Verfahren, deren Dichte wesentlich größer als Merkle-Hellman Kryptosystem ist. In folgenden Abschnitten werden wir sehen.

3.2 Dichte-Angriff

3.2.1 Grundlagen

Theorem *(Lagarias-Odlyzko)*

Sei A eine positive natürliche Zahle und sei $a_1, \ldots, a_n$ zufällige positive ganze Zahlen mit $0 < a_i < A$, $1 \leq i \leq n$. Und $e = (e_1, \ldots, e_n) \in \{0,1\}^n$ willkürlich. Ferner sei $S = \sum_{i=1}^{n} e_i a_i$, falls die Dichte $d < 0,6463\ldots$. Dann ist das Rucksack-Problem durch Lattice in Polynomialzeit „fast immer“ berechenbar.

Mit diesem Theorem kann man sehen, dass Merkle-Hellman Verfahren auch unsicher ist, da die Dichte $0,5$ zu klein ist. Die stark wachsende Folge führt zu einer verbreiterten Bitlänge des öffentlichen Schlüssels.

Hier ist eine Einführung in den Angriff mit Lattice:

Definition *Lattice*

Sei $\mathbb{R}^m$ ist ein m-dimensionaler Euklidischer Raum. Eine Lattice(man kann das auch auf Deutsch Gitter übersetzen.) ist die Klasse

$$L(b_1, \ldots, b_n) = \{\sum_{i=1}^{n} x_i b_i, x_i \in \mathbb{R}\}$$

für alle ganzzahligen Kombinationen von n linearen unabhängigen Vektoren $b_1, \ldots, b_n$ im Raum $\mathbb{R}^m$ mit $m > n$. Die natürlichen Zahlen n und m heißen der Rang und die Dimension der Lattice. $B = \{b_1, \ldots, b_n\} \in \mathbb{R}^{m \times n}$ heißt ein *Lattice Basis*, mit $L = L(B)$. [8]

3.2.2 Niedrige-Dichte-Angriff

Der Algorithmus von Lenstra–Lenstra–Lovász (LLL) Reduction ist ein wichtiges Verfahren für den Niedrige-Dichte-Angriff.

Lenstra–Lenstra–Lovász (LLL) Reduction[9]

Zuerst definieren wir $S_n(R)$ die Anzahl der Integer-Lösungen für die Ungleichung

$$\sum_{i=1}^{n} x_i^2 \leq R$$

Ferner sei ein Basis $B = \{b_1, \ldots, b_n\} \in \mathbb{R}^n$ und sein zugehöriges *Gram-Schmidt orthogonal Basis* $B^* = \{b_1, \ldots, b_n\} \in \mathbb{R}^n$ sowie die *Gram-Schmidt-Koeffiziente*

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$$

für alle $1 \leq j < i \leq n$.

Dann heißt ein Basis B reduziert, wenn

1. (Größe-Reduktion) $u_{i,j} \leq \frac{1}{2}$, für alle $1 \leq j < i \leq n$ ist.

2. (Lovász Kondition) es existiert ein Parameter $\delta \in (0,25,\ 1]$, sodass für alle $2 \leq k \leq n$:

$$\delta \left\| b_{k-1}^* \right\|^2 \leq \left\| b_k^* \right\|^2 + \mu_{i,j}^* \left\| b_{k-1}^* \right\|^2$$

Lenstra, Lenstra und Lovász haben einen wichtigen Algorithmus erfunden, der in vielen theoretischen ganzzahligen Problemen angewandt wird. Dieser *LLL*-Algorithmus kann wie folgende Schritte erfasst werden[10]:

EINGABE:
Ein Lattice-Basis $B = \{b_1, \ldots, b_n\} \in \mathbb{Z}^n$
Ein Paramepter $\delta \in (0,25,\ 1]$

Prozedur:

Gram-Schmidt Phase

$b_1^* := b_1$, $B_1 := \langle b_1^*,\, b_1^* \rangle$

for alle i von 2 bis n, do

$b_i^* := b_i$

for alle j von 2 bis $i-1$, do

$\mu_{i,\,j} := \frac{\langle b_i^*,\, b_j^* \rangle}{B_j}$

$b_i^* := b_i^* - \mu_{i,\,j} b_j^*$

end for

$B_i := \langle b_i^*,\, b_j^* \rangle$

end for

$k := 2$ ($b_1, \ldots, b_{k-1}$ *sind reduziert nach Größe-Reduktion*)

if $|\mu_{i,\,j}| > \frac{1}{2}$ then (*tut die Reduktion von* $RED(k,\, k-1)$)

for l von $k-1$ bis 1 do

$r := \lfloor 0,5 + \mu_{k,\,l} \rfloor$

$b_k := b_k - r b_l$

for j von 1 bis $l-1$ do

$\mu_{k,\,j} := \mu_{k,\,j} - r \mu_{l,\,j}$

end for

$\mu_{k,\,l} := \mu_{k,\,l} - r$

end for

end if

Berechne $\mu_{i,\,j}$ für $1 \leq j < i \leq n$ und B_i für i von 1 bis n

while $k \leq n$ do

Größe-Reduktion b_k und richtiges $\mu_{k,\,j}$ nach $RED(k,\, k-1)$, für j von 1 bis $k-1$

if $B_k < \left(\frac{3}{4} - \mu_{k,\,k-1}^2\right) B_{k-1}$ then

Tausche b_k und b_{k-1}

$k := max(2,\, k-1)$

else

$k := k + 1$

end if

end while

AUSGABE: LLL reduziertes Basis $b_1, \ldots, b_n$

Mit diesem LLL-Algorithmus haben Lagarias und Odlyzko ihr Theorem für „Niedrige-Dichte-Angriff“ den Grenzwert $0,6463\ldots$ gefunden und zwar wie folgt [11] [12]:

Lagarias-Odlyzko Algorithmus

1. Eingabe: $a_1, \ldots, a_n, s$, Ausgabe: $e_1, \ldots, e_n$.
2. Nehme $N > \sqrt{n}$.
3. Bilde eine Lattice mit den folgenden Vektoren:

$$b_1 = (1, 0, \ldots, 0, -Na_1)$$

$$b_2 = (0, 1, \ldots, 0, -Na_2)$$

$$\ldots$$

$$b_n = (0, 0, \ldots, 1, -Na_n)$$

$$b_{n+1} = (0, 0, \ldots, 0, -Ns)$$

Finde einen kürzesten nicht-negativen Vektor $v = (v_1, \ldots, v_{n+1})$ durch LLL-Algorithmus.

Falls $s = \sum_{i=1}^{n} a_i v_i$, dann $v = e$ und der Verktor v ist gefunden.

Coster, Joux, LaMacchia, Odlyzko, Schnorr und Stern haben den Lagarias-Odlyzko Algorithmus noch ein bisschen modifiziert und damit wird gezeigt, dass fast alle Rucksackprobleme mit einer Dichte von $d < 0,9408\ldots$ auch gelöst werden können.[13]

Die Verbesserung wird durch Einsetzen des Vektors $b_{n+1} = (\frac{1}{2}, \ldots, \frac{1}{2}, Ns)$ statt

$b_{n+1} = (0, 0, \ldots, 0, -Ns)$.

Man kann für die Verbesserung noch die Vektoren $B \in \mathbb{R}^{n+2}$ so setzen:

$$b_1 = (n + 1, -1, -1, \ldots, 0, Na_1)$$

$$b_2 = (-1,\ n+1,\ -1,\ \ldots,\ -1,\ Na_2)$$

$$\ldots$$

$$b_n = (-1,\ -1,\ \ldots,\ n+1,\ -1,\ Na_n)$$

$$b_{n+1} = (-1,\ -1,\ \ldots,\ -1,\ ,n+1,\ -Ns)$$

damit die Grenze $0,9408\ldots$ auch gezeigt wird. [13]

3.2.3 Diskussion über den Grenzwert der Dichte

Lemma

Jede Kugel mit dem Radius von $\sqrt{\alpha n}$ *in* $\mathbb{R}^n$ *mit* $\alpha < \frac{1}{4}$ *hat maximal* $(2-\gamma)$ *Punkte von* $\{0,\ 1\}^n$, *für* $\gamma = \gamma(\alpha) > 0$.

Mit diesem Lemma kann man deshalb die Grenze $0,9408\ldots$ durch eine Reduktion der Vektoren von polynomialer Anzahl von Basis mit verschiedenen b_{n+1} nicht mehr verbessern, wenn n sehr groß ist.

D.h., $0,9408\ldots$ ist die obere Grenze für einen *Niedrige-Dichte-Angriff* [13]. Jedoch kann man für ein bestimmtes kleines n den Grenzwert möglicherweise noch verbessern.

Japanische Experten haben 2003 gezeigt[12], dass für alle Dichte $d < d_a$ ein Kryptosystem mit dem Rucksackproblem mittels $a_1, \ldots, a_n$ und s *„fast immer"* in Polynomialzeit berechenbar ist, wenn man einen Parameter $h \leq \frac{n}{4}$ nimmt, wobei Radius der Kugel $\sqrt{\beta n}$ ist und

$$d_a = max \frac{1}{(log_2 e)\eta(\beta,\ u)}$$

$$\eta(\beta,\ u) = u\beta + ln\,\theta(e^{-u})$$

$$\theta(z) = 1 + 2\sum_{k=1}^{\infty} z^{k^2}$$

$$e = (e_{1,}\ \ldots,\ e_n) \text{ und } u \in \mathbb{R}$$

Ein Beispiel dafür ist $n = 64$, $\beta = 6$, und $d_a > 0,9408\ldots$, also ist eine Dichte über dem Grenzwert zu sehen.

Somit werden manche Kryptosysteme mit dem Rucksackproblem und einer Dichte gegen 1 auch unsicher.

3.3 Neue Verfahren des Rucksack-Typ-Kryptosystems

Nach dem Merkle-Hellman Kryptosystem hat man viele Techniken beim Schlüssel-Erzeugen eingesetzt, somit sehen wir heutzutage viele neuen Verfahren.

3.3.1 Diffusion und Konfusion

Im symmetrischen Kryptosystem benutzt man die Prinzipien der Diffusion und Konfusion nach Shannon, um den Ciphertext noch schwerer für Angreifer zu erkennen.

Diffusion: Jeder Bitblock von Klartext oder Schlüsseln soll viele Bitblöcke von Ciphertext beeinflussen.

Hier sind lineare Funktionen dazu relevant.

Konfusion: Jeder Bitblock von Ciphertext soll hoch nicht-linear von einigen Bitblöcken von Klartext und Schlüsseln abhängen.

Dafür sind nicht-lineare Funktionen zuständig.

Nach Shannon sollen die Funktionen von Ver- und Entschlüsseln sowohl gute Diffusion als auch Konfusion für die Bitblöcke behalten.[14]

Dieses Prinzip kann man aber auch im asymmetrischen Kryptosystem, insbesondere in Rucksack-Problem-Kryptosystemen weiter nutzen und zwar sind die Verfahren von Chor Rivest und Wang Bao-Cang zwei der berühmtesten Varianten.

3.3.2 Verfahren mit hoher Dichte

Chor-Rivest Kryptosystem [15, 16]

Sei $q = p^h$ eine Primzahlpotenz. Wir nehmen den endlichen Körper $GF(q)$ und veröffentlichen den. (Z.B.: ein h-Grad Polynom $P(x)$ ist nicht reduzierbar auf $GF(p)$ und Elemente von $GF(q)$ sind Polynome modulo $P(x)$.) Gleichzeitig nehmen wir einen Ring α für den Unterkörper $GF(p) \subseteq GF(q))$, z.B. $\{\alpha_{0,} \ldots, \alpha_{p-1}\} = GF(p)$.

Der private Schlüssel besteht aus

Ein Element $t \in GF(q)$ mit algebraischem Grad h
Ein Generator g von $GF(q)^*$
Eine natürliche Zahl $d \in \mathbb{Z}_{q-1}$
Eine Permutation π von $\{0, \ldots, p-1\}$

Der öffentliche Teil besteht aus allen

$$c_i = d + log_g(t + \alpha_{\pi(i)}) \bmod \mathrm{q} - 1$$

mit $i = 0, \ldots, p-1$.

Die öffentlichen Parameter müssen so ausgewählt werden, dass der diskrete Logarithmus einfach in $GF(q)$ berechenbar ist. Vorgeschlagen werden ein relativ kleiner p und ein *weicher* h genommen und beispielsweise sind solche Körper von $GF(197^{24})$, $GF(211^{24})$, $GF(243^{24})$ sowie $GF(256^{25})$.

Die Nachrichten in Chor-Rivest Kryptosystem sind alle p-Bits-Strings $m = [m_0 \ldots m_{p-1}]$ mit Hamming-Abstand h.

Fürs Erzeugen vom öffentlichen Schlüssel berechnen wir zuerst

$$p(t) = g^{E(m)-hd}$$

wobei $E(m) = m_0 c_0 + \ldots + m_{p-1} c_{p-1} \bmod q - 1$ und $h = m_0 + \ldots m_{p-1}$.

Für m brauchen wir noch eine Faktorisierung von

$$\Pi_{m_i=1}(t + \alpha_{\pi(i)})$$

Das Entschlüsseln ist aber wie traditionelles Merkle-Hellman Kryptosystem mit Rucksackproblem.

Der Angriff von Odlyzko ist durch eine Analysis von endlichem Körper und zwar wenn der Parameter h einen kleinen Faktor r hat, mit $r > \frac{1}{2} + \sqrt{h + \frac{1}{4}}$, ist der Angriff mit Zeitkomplexität von $\mathcal{O}(h^3 p^r / r^2)$[16]. Darüber hinaus wird gezeigt, dass die Veröffentlichung aller Koeffizienten von c_i durch die *Shortcut Methode* auch eine Lücke ist.[17]*[16]*

Wang-Kryptosystem:[18]

Wang Bao-Cang hat bei Schlüssel-Erzeugen den Chinesischen Restsatz eingesetzt und die Dichte kann gegen 1 sein. Wir nehmen hier sein Verfahren als ein Beispiel für solche neuen Entwicklungen nach Merkle-Hellman Kryptosystem seit einigen Jahren.

Schlüssel-Erzeugen:

Wähle zuerst einen n-dimensionalen zufälligen Vektor $U = (u_1, \ldots, u_n)$, $u_i > 0$.

Berechne den Vektor $V = (v_1, \ldots, v_n)$ mit

$$v_i = u_i – 2^{n-i},\ 1 \leq i \leq n.$$

Wähle noch zwei zufällige Primzahlen p und q, sodass

$$p > \sum_{i=1}^{n} u_i \ , \ q > 2\, max\left\{\sum_{v_i>0} v_i,\ -\sum_{v_i<0} v_i\right\}$$

Mit dem chinesischen Restsatz berechnen wir noch den Vektor $A = (a_1, \ldots, a_n)$, $0 \leq a_i \leq pq - 1$ und zwar

$$a_i \equiv u_i \bmod p,\ a_i \equiv u_i \bmod q$$

Privater Schlüssel: U, V, p und q

Öffentlicher Schlüssel: A

Zu verschlüsseln:

Sei den Klartext m von n-Bits, nämlich $m := (m)_2 = m_1 \cdots m_n$, $m_i = 0$ oder 1.

Der Ciphertext ist $c = a_1 m_1 + \ldots + a_n m_n$

Zu entschlüsseln:

Mit den Nachrichten c und dem privaten Schlüsseln p und q berechnen wir

$$c_p = c \bmod \mathrm{p},\ 0 \leq c_p < p$$

$$c_q = c \bmod \mathrm{q},\ -\tfrac{q}{2} < c_q \leq p$$

Beachte den Wertbereich von c_q, der auch negativ sein kann.

Dann ist der binäre Klartext

$$(m)_2 = m_1 \cdots m_n = (c_q - c_q)_2$$

Wir können noch sehen, dass bei Verschlüsseln und Entschlüsseln die Vektoren U und V nicht direkt benutzt werden.

Wang hat noch eine Verbesserung beim Schlüssel-Erzeugen gegeben und zwar wird neben dem chinesischen Restsatz noch eine Modulo-Mutiplikation eingesetzt, damit die Information von $v_i = u_i – 2^{n-i}$, $1 \leq i \leq n$ im öffentlichen Schlüssel noch tiefer versteckt.

Analysis von Wang-Kryptosystem:

Beim Schlüssel-Erzeugen benötigt es $\mathcal{O}(n)$ und bei Entschlüsseln zwei Modulo-Rechnung, also ist das Verfahren sehr effizient.

Nach der Definition ist die Dichte von Wang-Kryptosystem $d = \frac{n}{log_2 \, max \, a_i}$.

Da $0 \leq a_i \leq pq - 1$, gilt $log_2 \, max \, a_i \leq log_2 \, pq$.

Nach Abschnitt 3.2.3 soll die Länge von pq nicht größer als $\frac{n}{0{,}9408}$ sein, sonst ist dieses Verfahren unsicher.

Dafür nehemen wir r die größte mögliche Länge von q. Es muss gelten

$$0,9408 < \frac{n}{n-2+r}$$

Wegen $v_i = u_i – 2^{n-i}$ kann man am Ende sehen, dass

$$u_i + (2^r - 1) > u_{i+1} + u_{i+2} + \ldots + u_n$$

wobei alle v_i eigentlich negative ganze Zahlen sind, mit $|v_i| < 2^r - 1$.

Diese Ungleichungen von u_i und r kann man noch mit der Eigenschaft der stark wachsenden Folge in $v_i = u_i – 2^{n-i}$, $1 \leq i \leq n$ vergleichen, und dann kann mit 2^{n-i}, $1 \leq i \leq n$ dieses Verfahren auch angreifen[19].

Noch andere Varianten

Neben Wang Bao-Cang haben Min-Shiang Hwang und noch andere Experten verschiedene auf dem Rucksackproblem basierte Kryptosysteme mit hoher Dichte erfunden. Manche haben sogar eine Dichte über $0.9408\ldots$.

Min-Shiang Hwang wählt zuerst eine stark wachsende Folge und einen Modulus M usw. wie Merkle-Hellman, dann setzt noch sehr komplizierte Permutaionen-Kombination-Algorithmen ein. Am Ende hat man eine Dichte $d > 0.9408\ldots$[20]. Aber ein Angriff auf dem Hwang-Kryptosystem wurde 2013, also drei Jahre später, von Rastaghi mit Nutzen der stark wachsenden Folge detailliert beschrieben[21].

Ähnlich wie Wang-Kryptosystem haben noch einige Experten mit anderen Techniken der Diffusion und Konfusion erfunden und behaupteten, dass ihre Verfahren „unter Umständen“ sicher seien. Eine genaue Prüfung dafür bleibt aber noch offen.

Kapitel 4

Der Angriff vom Shamir-Algorithmus

Nun wird gezeigt, wie genau das Merkle-Hellman Kryptosystem unsicher ist und wie man das nach dem Shamir-Algorithmus brechen kann. Hier sind viele Inhalte aus dem Paper von Shamir[3], einem Skript von Lagarias[22] und dem Buch von Diekert[23].

4.1 Verlassen der Abhängigkeit von M

Wie Kapital 2 ist die stark wachsende Folge der „Schlüssel“ für Angreifer. Im Merkle-Hellman Verfahren kann zwar die Reihenfolge durch Permutationen noch verstecken, aber das bringt nicht $n!$ mehr Aufwand, sondern nur weniger als n^4 mehr Aufwand zu raten und man weiß dann, dass eine stark wachsende Folge mit Sicherheit existiert[23]. Den Grund dafür sehen wir später in der Analysis.

Der erste Gedanke ist, wie man die Folge im öffentlichen Schlüssel, umgekehrt zur ursprunglichen stark wachsende Folge des privaten Schlüssels bringt? Oder kann man eine äquivalent stark wachsende Folge bekommen?

Die Umwandlung für Alice ist einfach, sie weiß M und W, damit durch $a_i = b_i W \bmod M$ die originale stark wachsende Folge wieder erstellt wird, was die M und W (oder in äquivalenter weise U, da $W = U^{-1} \bmod M$) benötigt. Wir wissen M und W nicht, können aber die direkte Abhängigkeit von M und W verlassen und stattdessen ein „*Trapdoor Paar*“ V mit $0 < V < 1$ finden und dann eine stark wachsende Folge auch bekommen.

Wir defineiren für rationale Zahlen r und s mit $r = s \bmod 1$, wobei $r \equiv s \bmod 1$ und $0 \leq r < 1$ ist. Das bedeutet, dass r gleich der Wert des Bruchteils von s ist. Und zuerst wird ohne Permutation im Merkle-Hellman Verfahren interpretiert.

Wir hoffen, dass mit einer rationalen Zahl V und dem öffentlichen Schlüssel $(b_1, \ldots, b_n)$ durch

$$r_i = b_i V \bmod 1 \tag{4.1}$$

eine stark wachsende Folge $(r_1, \ldots, r_n)$ bekommen. Falls solches V gefunden wird, dann kann der Angreifer eigentlich wie der Empfänger entschlüsseln [23] und zwar

$$yV = \sum_{i=1}^{n} x_i b_i V \bmod 1 = \sum_{i=1}^{n} x_i r_i \tag{4.2}$$

Jetzt kann man wie Alice alle x_i berechnen.

Falls Alice noch eine Permutation π hat, dann gilt

$$r_i = b_{\pi(i)} V \bmod 1 \tag{4.3}$$

$$yV = \sum_{i=1}^{n} x_{\pi(i)} b_{\pi(i)} V \bmod 1 = \sum_{i=1}^{n} x_{\pi(i)} r_i \tag{4.4}$$

Für die Permutation π kann man alle Möglichkeiten der Reihenfolge von $(b_{\pi(1)}, \ldots, b_{\pi(n)})$ durch testen und es garantiert eine richtige Permutation zu finden.

Jetzt müssen wird die folgenden Fragen überlegen:

1.Wie kann man das richtige V finden?

2. Kann man das V direkt finden, oder (nur) approximieren?

3. Ist V überhaupt relevant zu M, W bzw. U?

Wir werden zuerst die 3. Frage beantworten und dann sehen die Antworte von 1. sowie 2. Frage.

Die wichtigste Idee ist immer noch die Eigenschaft einer stark wachsenden Folge. Zurück zum Entschlüsseln bei Alice: wir raten, dass M_0 der Modulus bei der Erzeugung von öffentlichen Schlüssel aber für den Angreifer unbekannt ist. Und sei W_0 analog die unbekannte Inverse. Es kann sein, dass $M_0 = M$, $W_0 = W$ aber nicht unbedingt. Das Bild der Funktion beim Entschlüsseln für ein festes b_i ist f_i: $[0, M_0) \to [0, M_0)$ mit

$$f_i(W_0) = W_0 b_i \bmod M_0 \tag{4.5}$$

ist in der Sägezahn-Form in Abb. 4.1:

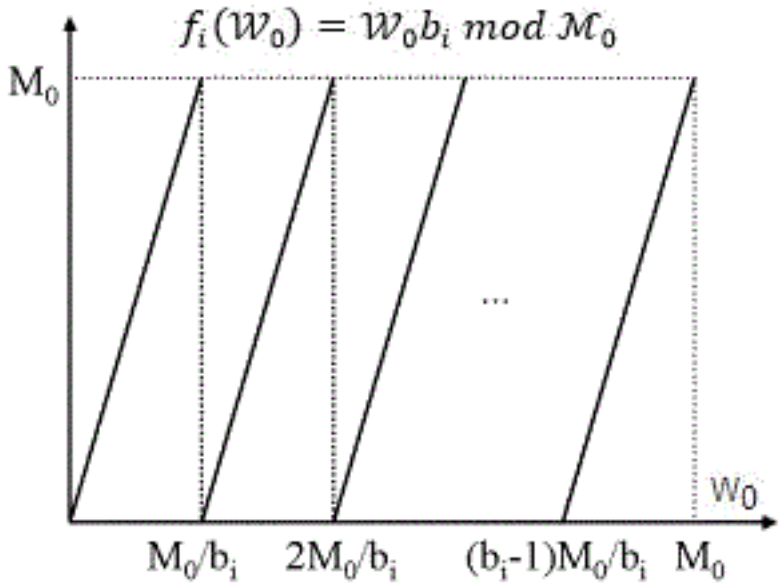

Abbildung 4.1: $f_i(W)$
[24]

Wir können sehen:

Es gibt genau b_i Linien, mit ihren Nullstellen von 0 bis $\frac{b_i-1}{b_i}M_0$.
f_i ist stetig bis auf alle Sprungstellen.
Jede Steigung ist b_i, bis auf alle Sprungstellen.
Die Steigung b_i ist unabhängig von M_0

Außerdem wissen wir im Merkle-Hellman Verfahren schon

$$Wb_i \bmod M \ = \ a_i \tag{4.6}$$

und für ein passendes W_0 bilden dann alle $f_i(W_0) = a_i$, $1 \leq i \leq n$ eine stark wachsende Folge.

Gleichzeitig können wir aus (4.1) für V eine Klasse der Funktionen $f_i^{'} : [0, 1) \to [0, 1)$ definieren, mit

$$f_i^{'}(V) = Vb_i \bmod 1 \tag{4.7}$$

Für ein festes b_i ist auch ein Graph in der Sägezahn-Form in Abb. 4.2 zu erwarten:

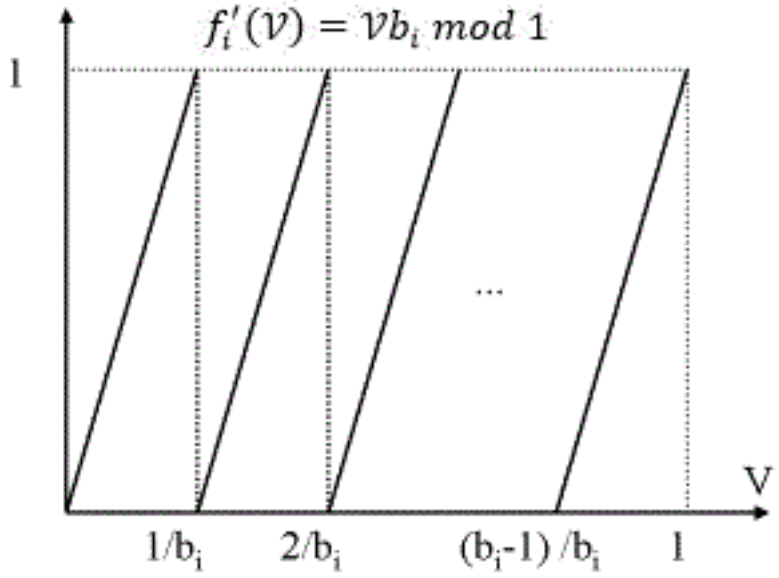

Abbildung 4.2: $f_i^{'}(V)$
[24]

Es ist auch leicht zu finden:

Es gibt auch genau b_i Linien, mit ihren Nullstellen von $\frac{0}{b_i}$ bis $\frac{b_i-1}{b_i}$

$f_i^{'}$ ist stetig bis auf alle Sprungstellen.

Jede Steigung ist b_i und unabhängig von anderen Variablen.

Und es existiert auch ein V_0, damit

$$f_i^{'}(V_0) = V_0 b_i \,\mathsf{mod}\, 1 = r_i$$

eine stark wachsende Folge $(r_1, \ldots, r_n)$ liefern kann.

Wir können noch sehen, dass beide Bilder fast identisch sind, nur die horizontalen-und vertikalen Achsen unterscheiden sich. Und die Umwandlung vonf_i zu $f_i^{'}$ ist auch einfach, dass man f_i durch den Modulus M_0 teilt. D.h.:

$$\frac{f_i}{M_0} = \frac{W_0}{M_0} \,\mathsf{mod}\, 1 = f_i(\frac{W_0}{M_0}) \tag{4.8}$$

Ersetze $\frac{W_0}{M_0}$ durch V, dann stimmen $f_i(V)$ und $f_i^{'}(V)$ überein. Und folglich ist

$$V = \frac{W_0}{M_0} \tag{4.9}$$

$$r_i = \frac{a_i}{M_0} \tag{4.10}$$

Ferner bemerkt man, dass

$$\sum_{i=1}^{n} r_i < 1 \tag{4.11}$$

wenn man $M_0 \approx M$ sieht.

Danach werden die Funktionen $f_i^{'}(V)$ und $f_i(V)$ als dieselbe betrachtet.

Falls noch eine Permutation π eingesetzt wurde, liefert $(r_{\pi(1)}, \ldots, r_{\pi(n)})$ die gleiche Reihenfolge wie $(a_{\pi(1)}, \ldots, a_{\pi(n)})$ mit

$$f_{\pi(i)}(W_0) = W_0 b_{\pi(i)} \bmod M_0 \; = \; a_{\pi(i)}$$

$$f_{\pi(i)}(V_0) = V_0 b_{\pi(i)} \bmod 1 \; = \; r_{\pi(i)}$$

$$r_{\pi(i)} = \frac{a_{\pi(i)}}{M_0} \tag{4.12}$$

Das gesuchte V ist dann in der Berechnung eigentlich unabhängig von M usw.. Damit haben wir jetzt die Antwort der Frage 3. Und das bringt uns den Vorteil, dass wir nur einen Variabel V berechnen müssen, statt gegen zwei Unbekannten M und W bekämpfen.

Gleichzeitig sehen wir, dass es für ein festes $f_i(V)$ genau b_i mögliche Linien gibt und es für ein festes V auch n Stücke von $f_1(V)$, ..., $f_n(V)$ gibt, die zu berechnen von V dienen. Der nächste Schritt ist, wie man das richtige V finden bzw. in einem kleineren Bereich begrenzen kann. Betrachte, dass jedes b_i ungleich zu einander ist. Wenn man einige unterschiedliche Bilder aus f_i zusammenlegen, dann kann man in manchen Stellen von V, dass $f_i(V) = r_i$, $f_j(V) = r_j$, usw., $1 \leq i < j < \cdots \leq n$, eine monoton steigende (und sogar „stark wachsende“) Folge liefert. Idealerweise soll ein solches V für alle i in der Abbildung gelten, wenn alles zusammengelegt wird. Mit diesem Überlegen können wir später für die Suche nach V helfen.

Und wahrscheinlich kann man mit Glück sogar mehrere disjunkte Intervalle der Vs finden, da jedes f_i periodisch ist. Obwohl diese Überlegung ungenau aussieht, ist aber auch unwichtig, da der Existenz von V bekannt ist. Wenn ein gefundenes V eine stark wachsende Folge liefert, so hat man die wichtigste Information vom Merkle-Hellman Kryptosystem.

4.2 Anfangen mit der Anfangsfolge

Um die 1. Frage zu beantworten, wollen wir zuerst mit einigen b_i bearbeiten.

Wurde eine Permutation π eingesetzt, ist $n!$ Durchläufe zu erwarten. Auf einer Seite ist das nicht in polynomialer Zeit, was sehr ineffizient ist, auf der anderen Seite ist ein richtiges V für alle b_i gleichzeitig bzw. einmalig zu bestimmen auch unrealistisch. Wir können aber die Anfangsfolge mit „*zum Beispiel*“ $l = 4$ Elementen testen. Ist ein passenden Wert oder Intervall von V für die ersten vier Elemente mit der stark wachsenden Folge gefunden, dann kann dieser Wert oder Intervall von V für weitere Elemente die genaue Suche nach V sowie die Anzahl der mögliche Permutationen ggf. minimieren, was wir später sehen werden. Allerdings muss man zuerst maximal $\mathcal{O}(n^4)$ Male für die Anfangsfolge $(\pi(1), \ldots, \pi(4))$ probieren.

Seien wir in der richtigen Phase mit der Anfangsfolge, nämlich $\pi(j) = j$, $j \in \{1, \ldots, l\}$, mit $l = 4$, also sind die vier Elemente $\{b_{\pi(1)}, \ldots, b_{\pi(4)}\}$ des öffentlichen Schlüssels genau $\{b_1, \ldots, b_4\}$, die aus $\{a_1, \ldots, a_4\}$, erzeugt werden.

Nun schauen wir die Abb. von f_i. Sein das gesuchte V liegt irgendwo zwischen $\frac{c_i}{b_i}$ und $\frac{c_i+1}{b_i}$, c_i ist jetzt aber noch unbekannt, also

$$\frac{c_i}{b_i} \leq V < \frac{c_i + 1}{b_i} \tag{4.13}$$

Und setze

$$V = \frac{c_i}{b_i} + \varepsilon_i \tag{4.14}$$

wobei $0 \leq \varepsilon_i < \frac{1}{b_i}$ ist.

Dann gilt

$$b_i V = c_i + b_i \varepsilon_i \tag{4.15}$$

und wegen $0 \leq \varepsilon_i < \frac{1}{b_i}$ gilt

$$b_i V \bmod 1 = b_i \varepsilon_i = \frac{a_i}{M^*} \tag{4.16}$$

wobei M^*eigentlich der aproximierte Wert von M ist. Man kann zuerst $M \approx M^*$betracheten. Ferner ist

$$\varepsilon_i = \frac{a_i}{b_i M^*} \tag{4.17}$$

Analog liegt V zwischen $\frac{c_j}{b_j}$ und $\frac{c_j+1}{b_j}$, $i,\, j \in \{1,\, 2,\, 3,\, 4\}$, $i \neq j$. Also

$$V = \frac{c_j}{b_j} + \varepsilon_j \tag{4.18}$$

Mit (4.17), (4.18) und $i \neq j$ folgt

$$\frac{c_j}{b_i} - \frac{c_i}{b_j} = \varepsilon_i - \varepsilon_j = \frac{a_i}{b_i M^*} - \frac{a_j}{b_j M^*} \tag{4.19}$$

Und nach einer kleinen Umrechnung ist

$$c_j b_i - c_i b_j = \frac{1}{M^*}(a_i b_j - a_j b_i) \tag{4.20}$$

Und dann

$$c_j b_i - c_i b_j \leq \frac{1}{M^*} a_i b_j$$

$$c_i b_j - c_j b_i \leq \frac{1}{M^*} a_j b_i \tag{4.21}$$

Betrachte $M \approx M^* > \sum_{i=1}^{n} a_i$, daher ist

$$a_i < 2^{-n+i} M \tag{4.22}$$

Noch mit (4.21) gilt

$$-2^{-n+j} b_i < c_j b_i - c_i b_j < 2^{-n+i} b_j \tag{4.23}$$

Noch übersichtlicher ist

$$|c_j b_i - c_i b_j| < 2^{-n+4} b_{max} \tag{4.24}$$

was ein bisschen grob abgeschätzt wird.

Um das richtige V zu finden, kann man zuerst alle c_i bestimmen, $i \in \{1, 2, 3, 4\}$. Mit (4.21) oder (4.22) sehen wir, dass man für beliebige $i, j \in \{1, 2, 3, 4\}$, $i \neq j$ zwei Ungleichungen zur Verfügung stehen kann. Offensichtlich ist $0 \leq c_i < b_i$ für alle $1 \leq i \leq n$. Und mit den vier Unbekannten c_i hat man ein diophantisches Ungleichungssystem

$$c_j b_i - c_i b_j < 2^{-n+i} b_j$$

$$c_i b_j - c_j b_i < 2^{-n+j} b_i$$

$$0 \leq c_i,\ c_j < b_j \tag{4.25}$$

für alle $i, j \in \{1, 2, 3, 4\}$ und $j \neq i$.

Wegen der Existenz von V soll das Ungleichungssystem eine Lösung(-smenge) der ganzzahligen Zahlen besitzen. Außerdem kann das Ungleichungssystem zu einem Integer-Programming-Problem umwandeln kann und das verspricht auch eine ganzzahlige Lösung zu bekommen[25].

4.3 Wahrscheinlichkeit, um V zu finden

Wir müssen jetzt das Ungleichungssystem als Integer-Programming-Problem lösen, um das V zu finden.

Zurück zur Auswahl von l, dass $l = 4$ genommen wird. Aber warum die Anfangsfolge genau 4 Elementen ist? Grob erklärt ist für die Berechnung von V mit 4 Unbekannten im Ungleichungssystem durch Integer-Programming gut geeignet. Weniger als 4 Unbekannte bringt es zu viele Lösungen, die meisten davon sind aber unbrauchbar, da so viele möglichen Stelle von V unrealistisch sind. Für $l \geq 5$ bringt im gegen Satz dazu viel mehr Aufwand zu rechnen.

Zudem hat Shamir noch bewiesen, dass zwischen l und dem Expansionsfaktor d folgendes Verhältnis gelten muss

$$l > d + 1$$

Bei unserem Fall ist $d = 2$, also l muss mindestens 4 sein. Für einen höheren Wert von d muss man noch mehr Unbekannten im Integer-Programming-Problem bringen: z.B.: wenn $d = 3$ ist, muss man 5 Variablen nehmen.[25]

Zur genaueren Analyse für $l = 4$: wie wahrscheinlich bzw. ob überhaupt eine ganzzahlige Lösung gefunden werden kann, die das Integer-Programming erfüllt? Falls viele Lösungen vorkommen können, dann ist ein Durchtesten der langen Liste sehr mühsam. Andererseits, falls häufig keine Lösung gefunden wird, ist der Angriff gescheitert.

Erste Behauptung:

Die Wahrscheinlichkeit, überhaupt eine Lösung zu finden, ist gering.

Sei $\Gamma = \{1, 2, 3, 4\}$.

Mit (4.17) und (4.22) bekommen wir für alle $i \in \Gamma$

$$\varepsilon_i < \frac{2^{-n+i}}{b_i} \tag{4.26}$$

Sei

$$b_i \geq 2^{2n+i-\lambda_i}, i \in \Gamma \tag{4.27}$$

Da $b_i \leq 2^{2n}$, folgt

$$0 < 2n + i - \lambda_i < 2n$$

nämlich

$$i < \lambda_i < 2n + i \tag{4.28}$$

Wegen $b_i = a_i U \bmod M$, ist b_i mit ungefähr gleicher Größenordnung von M. Also kann man sehen, $\lambda_i \approx i$, und im schlimmsten Fall gilt $\lambda_i \in \mathcal{O}(n)$, wenn b_i sehr klein ist. Da i eigentlich auf Konstant beschränkt wird, ist λ_i auch so, und zwar $\lambda_i \in \mathcal{O}(1)$, für alle $i \in \Gamma$, λ_i ist maximal sublinear zu n.[23]

Sei $\lambda = \lambda_{max}$, $\varepsilon = \varepsilon_{max}$ für alle $i \in \Gamma$, dann ist

$$\varepsilon < 2^{-3n+\lambda} \tag{4.29}$$

Nun nehmen wir an, dass die Zahlen b_i für alle $i \in \Gamma$ unabhängige Zufallsvariablen sind. Betrachte die Stelle $0 \leq \frac{c_j}{b_j} < 1$, neben der das gesuchte V liegt.

Die Wahrscheinlichkeit, dass für den Durchschnitt des Intervalls $[\frac{c_j}{b_j}, \frac{c_j}{b_j} + \varepsilon_j]$ mit jeder Positionsmenge $\frac{\mathbb{N}}{b_l}$ für alle $l \in \Gamma$, $l \neq j$, nicht leer ist, höchstens

$$(2^{-n+\lambda})^3 = 2^{-3n+3\lambda} \tag{4.30}$$

Wenn alle möglichen c_l summiert werden, kann die Wahrscheinlichkeit unter der oberen Annahme, überhaupt eine Lösung zu finden, weniger als $2^{-n+3\lambda}$. Laut der Annahme „λ sublinear zu n“ ist die obere Wahrscheinlichkeit gegen null, wenn n groß genug ist. Und unter der Annahme „λ ist auf Konstant beschränkt“ strebt die Wahrscheinlichkeit noch schneller, also exponentiell gegen null. Deswegen soll man keine Sorge haben, dass man zu viele Lösungen finden und mühsam prüfen muss.

Zweite Behauptung:

In fast allen Fällen existiert mindestens eine Lösung.

Wir wissen noch, dass alle b_i eigentlich nicht unabhängig sind. Deshalb kann man vorstellen, dass die obere Wahrscheinlichkeit tatsächlich größer ist, als was wir abgeschätzt haben.

Mann kann auch vorstellen, dass wegen der Existenz von $V \approx \frac{W}{M}$ das V nähe genug von mehreren a_j liegen soll, die wir wohl bestimmen können.

Außerdem hat Shamir eine Annahme benutzt und zwar

***Hypothese*[22]**

Das Integer-Programming-Problem von (4.25) bezüglich dem Rucksackproblem von Merkle-Hellman hat bis einem multiplikativen Konstant abhängig von l, mit $l = |\Gamma|$, und die erwartete Anzahl der Lösungen für das obere quasi-zufällige Integer-Programming-Problem ist mindestens 1.

Äquivalent kann es gesagt werden, dass für das obere Integer-Programming-Problem, die Klasse von Mapping von den Folgen $(W, a_1, \ldots, a_n)$ an ein festes b_1, die Bilder von $(a_1, \ldots, a_l)$ hat, was fast immer mindestens eine positive Fraktion in Abhängigkeit von l schlägt.

Das wiederum ist aus dem folgenden Lemma:

***Lemma*:[22]**

Sei b_1 *fest mit* $b_1 = \mu M$, $\frac{1}{n^2} \leq \mu < 1$. *Die Anzahl der verschieden* $(b_2, \ldots, b_l) \bmod b_1$, *die die Bilder von etwa*$(W, a_1, \ldots, a_l,)$ *mit* $Wb_1 \equiv a_1 \bmod M$ *sind, ist entweder* 0 *oder mindestens ein Konstant abhängig von* $l\mu^l 2^{(l-1)dn}$.

Hier gibt es eine Annahme, dass$\frac{M}{n^2} \leq b_i < M$. Da alle b_i eigentlich etwas gleich groß wie M ist, kann diese Annahme fast immer gelten.

Nun werden solche Aussagen ohne Beweis gestellt. Daher ist eine Lösung von (4.25) (*fast*) immer garantiert.

Darüber hinaus kann man dieses Ungleichungssystem durch die diophantische Approximation analysieren, und es wird zeigt, dass sogar auch für mehrfach permutierte Version von Merkle-Hellman fast immer eine Lösung zu erwarten ist [6]. Obwohl wir hier die diophantische Approximation nicht anwenden, ist eine ganzzahlige Lösung im Integer-Programming-Problem auch zu erwarten.

4.4 Das Simplex Verfahren

Um dieses Integer-Programming-Problem zu lösen, haben wir uns das Simplex Verfahren entschieden. Das Simplex-Verfahren ist eine der bekanntesten Algorithmen, um lineare Integer-Programmen effizient zu lösen. Wie im Buch von George B. Dantzig[26] erklärt wird, hat dieser Algorithmus normalerweise polynomiale Laufzeit, nur im extremen Fall kann exponentiell laufen. Das Verfahren kann mehrere Variante und gleichzeitig mit vielen Techniken haben, um die Lösung schneller zu liefern.

Im Allgemeinen dient das Simplex zur linearen Programme der Form

$$(LP) \quad max \left\{ c^T x | Ax \leq b,\ x \geq 0 \right\}$$

wobei $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ eine Matrix mit reellen Einträgen, $c \in \mathbb{R}^n$ der Zielfunktionsvektor und $b \in \mathbb{R}^m$ der Vektor von Beschränkungen ist.

Ein Punkt bzw. Vektor x ist zu finden, hier ist x mit ganzzahligen Einträgen, um das lineare Gleichungssystem zu erfüllen und einen möglichst hohen Zielfunktionswert $F(x) = c^T x$ zu haben.

Falls ein lineares Programm nicht wie die Standardform von oben ist, wird eine Umformung benötigt.

Für die Menge der zulässigen Lösungen gibt es drei Möglichkeiten:

1. das LP besitzt keine zulässigen Lösungen, d. h., das Polyeder ist leer.

3. das LP ist unbeschränkt, es gibt Lösungen mit beliebig hohem Zielfunktionswert.

3. es gibt genau eine oder unendliche viele Optimallösungen.

Das Simplex-Verfahren setzt sich aus zwei Phasen zusammen:

Phase I bestimmt eine zulässige Startlösung oder stellt fest, dass das Problem keine Lösung besitzt.

Phase II verbessert eine bestehende Lösung immer weiter, bis keine Verbesserung der Zielfunktion mehr möglich ist oder die Unbeschränktheit des Problems festgestellt wird.

Laufzeit:

Aus theoretischer Sicht ist das Simplex-Verfahren daher beispielsweise den Innere-Punkte-Verfahren mit polynomialer Laufzeit unterlegen. Aus praktischer Sicht hat es sich aber in vielen Fällen als schneller erwiesen. Der größte Vorteil des Simplex-Algorithmus gegenüber anderen Verfahren liegt jedoch darin, dass es bei kleinen Änderungen der Eingabedaten im Laufe des Algorithmus einen „Warmstart“ erlaubt, also die letzte berechnete Basis als Ausgangspunkt für wenige weitere (primale oder duale) Iterationen nehmen kann, während beispielsweise Innere-Punkte-Verfahren in solch einem Fall von vorne anfangen müssen. Dieser Fall tritt sehr häufig auf, wenn sehr viele ähnliche lineare Programme in Folge gelöst werden müssen, beispielsweise im Rahmen von *Schnittebenenverfahren*, *Branch-and-Bound* oder *Branch-and-Cut*.

Wir haben leider kein passendes Tool bzw. API von Simplex gefunden. Das beste Tool heutzutage „lp_solve“ berechnet bei unserem Testfall keine richtige Lösung und daher wird ein natives Verfahren ersetzt. Obwohl das Verfahren nicht effizient ist, liefer das immer richtige Lösung an.

4.5 Bestimmen von V

Ob das V genau gleich $\frac{W}{M}$ sein muss, ist keins von unserer Interesse, da eigentlich nur ein approximierter Wert von $\frac{W}{M}$ ausreicht sein kann, nämlich $\frac{W^*}{M^*}$ in unserem Gedanke. Das liegt daran, dass $\{r_i = \frac{b_i}{M^*} | 1 \leq i \leq n\}$ und $\{b_i | 1 \leq i \leq n\}$ zwar mit gleicher Eigenschaft sind, aber ob $r_i = \frac{b_i}{M}$ unbedingt ist, muss eigentlich nicht sein, weil nur diese „stark wachsende“ Eigenschaft beim Verschlüsseln schon reicht. Tatsächlich kann man vorstellen,

dass in einem kleinen rationalen Intervall (δ, η) alle Punkte die obere Eigenschaft erfüllen können. D.h., ein kleines „Rauschen“ ist erlaubt und wir müssen nur eigentlich den Wert V gut genug approximieren. Damit wird jetzt die Frage 2 im ersten Schritt beantwortet.

Wenn eine Lösung vom oben genannten Integer-Programming-Problem gefunden wird, also die Menge $\{c_{\pi(j)}\}$ mit Annahme von $\pi(j) = j$, $j \in \Gamma$, dann testen wir gleich, ob die das richtige V liefert. Falls wir mehrere Lösungen finden, dann testen wir alle $\{c^{(1)}_{\pi(j)} | j \in \Gamma\}$, $\{c^{(2)}_{\pi(j)} | j \in \Gamma\}$, $\cdots$ durch. Da wir in (4.25) nur konstante Unbekannte haben und positive ganzzahlige Lösungen finden wollen, ist diese Liste endlich.[23]

Setze $c_j = c_{\pi(j)}$, aus (4.18) und (4.29) setze noch $\alpha = \frac{c_j}{b_j}$, $\varepsilon = 2^{-3n+\lambda}$ und folgt $V \in [\alpha, \alpha+\varepsilon]$. Offensichtlich darf V sich nicht am Rand befinden, dann ist $V \in (\alpha, \alpha + \varepsilon)$.

Fixiere dieses Paar $\{c_j, b_j\}$, und betrachte alle anderen Paare $\{c_i, b_i\}$ mit $1 \leq i \leq n$, $i \neq j$. Der Abstand $\frac{1}{b_i}$ zwischen zwei Minimalen ist (fast) immer größer als ε, weil $b_i < 2^{2n}$. Da ε normalerweise auf Konstant beschränkt ist, kann man hier so mutig behaupten. Daraus folgt, dass im Intervall $(\alpha, \alpha + \varepsilon)$ bezüglich j, maximal einen Punkt von p/b_i mit $p < b_i$, $i \leq n$, $i \neq j$. Das Bild der Funktion f_i mit $f_i(V) = b_i V \bmod 1$, $i \neq j$, kann das Intervall $(\alpha, \alpha+\varepsilon)$ maximal zu zwei Abschnitten teilen. Mit den ersten vier c_j berechneten bekommt man maximal 4^2 Schnittpunkte von den vier Linien bezüglich c_j und b_j mit $1 \leq j \leq 4$. Diese Schnittpunkte kann man leicht berechnen. Und wenn man alle n Linien bezüglich c_i und b_i mit$1 \leq i \leq n$ sieht, ist dann maximal $\mathcal{O}(n^2)$ Schnittpunkte zu erwarten. Durch je zwei benachbarte Schnittpunkte wird das Intervall $(\alpha, \alpha+\varepsilon)$ zu höchstens $\mathcal{O}(n^2)$ Teilintervallen unterteilt. Betrachte jedes Teilintervall $[\beta, \beta + \zeta]$ ausgenommen den Rand $\{\alpha, \alpha+\varepsilon\}$, dass das keine Position c_i/b_i mit $1 \leq i \leq n$ enthält, damit wir prüfen können, ob eine Stelle V^* mit der Rechnung (4.8) eine stark wachsende Folge etwa wie $(\frac{a_1}{M}, \ldots, \frac{a_n}{M}, 1)$ liefert. D.h., wir können mit $\{c_j | 1 \leq j \leq l, l = 4\}$ einige kleine Intervalle bestimmen, um das gesuchte V zu approximieren.

Was man aber noch diskutieren muss, ist der Sonderfall von$(0, \varepsilon)$ bzw. $(0, \zeta)$, nämlich, wenn $b_j = 0$ ist. Offensichtlich sind alle anderen Positionen bzgl. b_i mit$1 \leq i \leq n$ entweder auch 0, oder liegen am rechts von 0. Allerdings wissen wir schon, dass der Abstand $\frac{1}{b_i}$ ist fast immer größer als ε, dann müssen alle Positionen genau 0 sein, da alle Rechts $\frac{c_j}{b_j}$, $c_j > 0$ außerhalb vom Intervall $(0, \varepsilon)$ liegt. Zudem sieht man noch, dass das Intervall $(0, \varepsilon)$ gar nicht unterteilt wird und wir nur ein einziges Intervall haben. Daher sollen alle $c_i = 0$, $i \in n$ sein. Insbesondere ist $(0, 0, 0, 0)$ auch eine mögliche Lösung zum oben genannten Ungleichungssystem. In diesem Fall soll ein gewünschtes V nahe 0 sein, damit die Folge $(b_{\pi(1)}, \ldots, b_{\pi(n)})$ streng monoton wachsend ist. Dieser Sonderfall hat die Permutation schon fest gelegt und man behandelt mit anderen Permutationen nicht mehr. Falls die

Folge des öffentlichen Schlüssels nicht permutiert ist, bedeutet das, dass die stark wachsende $\{a_i | 1 \leq i \leq n\}$, durch die Rechnung $b_i = a_i U \bmod M$, auch eine streng monoton wachsend Folge $(b_1, \ldots, b_n)$ erzeugt. Das ist aber kaum möglich. Für solche Fälle kann man separat behandeln.

Andererseits kann b_j, $j \in \Gamma$ im extremen Fall sehr klein sein, damit das zugehörige λ den maximalen Wert erreicht. Das führt dazu, dass $\varepsilon = 2^{-3n+\lambda}$ größer als der Abstand $\frac{1}{b_i}$ ist. Infolgedessen dürfen andere Positionen $\frac{c_i}{b_i}$ mit $i \neq j$ innerhalb vom Intervall $(0, \varepsilon)$ liegen. Aber dieser Sonderfall ist wegen des Analysis von λ auch sehr unwahrscheinlich vorzukommen, damit man nicht viel darauf beachten soll.

Zusammengefasst ist der Sonderfall $\{c_j = 0 | j \in \Gamma\}$ Shamir-Algorithmus nicht zu schmälern.

Nun werden alle $\mathcal{O}(n^2)$ Schnittpunkte von n Linien berechnet. Die genauere Selektion der Teilintervalle kann man mit $(c_1, \ldots, c_4)$ und ε zuerst für $(\pi(1), \ldots, \pi(4))$ vorwählen. Man kann jetzt sehen, dass in diesem Durchschnitt alle Linien von unten nach oben „richtig“ angeordnet werden soll, nämlich, innerhalb eines Teilintervalls $(\beta, \beta + \zeta) \subseteq [\alpha, \alpha + \varepsilon)$ alle Punkte V davon bringen die Monoton-Eigenschaft bezüglich der Permutation π

$$0 < b_{\pi(1)} V \bmod 1 < \cdots < b_{\pi(4)} V \bmod 1 < 1 \tag{4.31}$$

Danach müssen in diesem selektierten Teilintervall insbesondere alle anderen $b_{\pi(i)}$, $5 \leq i \leq n$ für (4.31) noch weiter gelten

$$0 < b_{\pi(1)} V \bmod 1 < \cdots < b_{\pi(n)} V \bmod 1 < 1 \tag{4.32}$$

Diese Ungleichungen stellen gleichzeitig die Permutation π fest, die möglicherweise mit der originalen Permutation Π überein stimmen kann. Das führt dazu, dass wir für andere Permutationen von $(\pi(5), \ldots, \pi(n))$ statt $\mathcal{O}(n!)$ auf $\mathcal{O}(n^2)$ begrenzt. Außerdem ist das Raten von $(\pi(1), \ldots, \pi(4))$ maximal $\mathcal{O}(n^4)$, somit müssen wir fürs Raten aller Permutationen $(\pi(1), \ldots, \pi(n))$ maximal $\mathcal{O}(n^6)$ belasten.

Darüber hinaus wissen wir, dass es in einem Teilintervall mindestens ein V existiert, mit dem eine stark wachsende Folge erzeugt wird, was wir gleich prüfen werden.

Dafür definieren wir natürliche Zahlen $s_i = \lfloor b_i \beta \rfloor 1 \leq i \leq n$, nämlich den ganzzahligen Teil davon und damit

$$b_i V - s_i = b_i V \bmod 1 = r_i \tag{4.33}$$

Zusammen mit dem Unbekannten V haben wir für jedes richtig selektiertes Teilintervall $[\beta,\, \beta + \zeta]$ ein Ungleichungssystem

$$\beta < V < \beta + \zeta$$

$$\sum_{j=1}^{i-1}(b_j V - s_j) < b_i V - s_i,\ 2 \leq i \leq n$$

$$\sum_{i=1}^{n}(b_i V - s_i) < 1 \tag{4.34}$$

Setze $t_i = b_i V - s_i$, $1 \leq i \leq n$, so erhalten wir noch das in noch übersichtlicherer Form:

$$\beta < V < \beta + \zeta$$

$$\sum_{j=1}^{i-1} t_j < t_i,\ 2 \leq i \leq n$$

$$\sum_{i=1}^{n} t_i < 1 \tag{4.35}$$

Theoretisch kann man diese Ungleichungen lösen und ein Teilintervall $[L,\, R] \subseteq [\beta,\, \beta + \zeta]$, $0 \leq L,\, R \leq 1$ nehmen, damit für alle $V \in [L,\, R]$ das oben genannte Ungleichungssystem füllt, so dass man $V \approx \frac{W}{M}$ finden kann, mit dem eine stark wachsende Folge erzeugt wird. Infolgedessen ist unser Ziel des Angriffs erreicht.

4.6 Diskretes Approximieren zu V

Wir können insgesamt höchstens $\mathcal{O}(n^2)$ Teilintervalle $[\beta,\, \beta + \zeta]$ selektieren, aber eine direkte Berechnung für solche Teilintervalle $[L,\, R]$ aus (4.34) in einem $[\beta,\, \beta + \zeta]$ ist in Praxis nicht einfach. Alternativ kann man mit diskreten Punkten in einem Teilintervall $[\beta,\, \beta + \zeta]$ das gewünschte V approximieren. Da nur ein gewünschter Wert von V statt genau $\frac{W}{M}$ oder ein Teilintervall mit aller möglichen Werten ausreicht, darf man in einem gesamten Teilintervall Schritt für Schritt diskret raten und prüfen. Wenn einen richtigen Wert gefunden wird, beendet man diese Suche nach V.

Jedoch muss man solche diskreten Punkte sorgfältig definieren. Sei $V^* = W^*/M^*$ eine selektierte diskrete Stelle. Der kleinste Abstand zwischen V^*und $\frac{W}{M}$ soll nicht größer als ein bestimmter Wert. Um diese Grenze festzustellen, schauen wir noch die ursprüngliche stark wachsenden Folge $\{a_i|1 \leq i \leq n\}$. Sei nun $M^* = \varkappa M$ und $W^* = \varkappa(W + \xi)$. Dann gilt

$$\left|\frac{W^*}{M^*} - \frac{W}{M}\right| = \left|\frac{\xi}{M}\right| \tag{4.36}$$

Andererseits ist

$$W^*b_i - M^*c_i = \varkappa((Wb_i - Mc_i) - \xi b_i) \tag{4.37}$$

Nach (4.7) ergibt sich

$$Wb_i - Mq_i = a_i,\ q_i \in \mathbb{N} \tag{4.38}$$

Verglichen mit (4.37) und (4.38) hat man

$$W^*b_i - M^*c_i = \varkappa(a_i - \xi b_i) \tag{4.39}$$

Wir sehen aus (4.39), dass die Folge von

$$a_i^* = a_i - \xi b_i \tag{4.40}$$

wird auch stark wachsend, wenn das Rauschen ξb_i gering genug ist, damit die Eigenschaft von $\{a_i|1 \leq i \leq n\}$ nicht zerstört bzw. beeinflusst. Also um den minimalen Abstand $|\xi/M|$ zu bestimmen, ist äquivalent um ξb_i zu berechnen.

Dazu definieren wir noch die Distanz d_i bezüglich a_i und M mit

$$d_i = a_i - \sum_{j=1}^{i-1} a_j,\ \ 1 \leq i \leq n \tag{4.41}$$

Dann folgt

$$\sum_{i=1}^{n} a_i = \sum\nolimits_{i=1}^{n} 2^{n-i} d_i \tag{4.42}$$

Damit ξb_i mit (4.40) die stark wachsende Eigenschaft nicht beeinflusst, sollen alle d_i möglichst „*groß*“ definiert werden. Nun haben wir ein Integer-Programming-Problem mit den Unbekannten d_i

$$M > \sum_{i=1}^{n} 2^{n-i} d_i \tag{4.43}$$

Zudem sei $d_i < D$, für alle $1 \leq i \leq n$, und dann setze

$$d_i^* = d_i - D \tag{4.44}$$

mit $d_i^* > 0$, $1 \leq i \leq n$. Daraus folgt

$$M - (2^n - 1)D > \sum_{i=1}^{n} 2^{n-i} d_i^* \tag{4.45}$$

Sei $S_n(M)$ die Anzahl der ganzzahligen Lösungen von (4.43) und analog sei $S_n(M-(2^n-1)D)$ für (4.45). Es soll gelten

$$S_n(M) \sim S_n(M - (2^n - 1)D)$$

wenn $M = 2^{dn}$, $1 < d < 2$, $n \to \infty$, wobei $\sim$ die Äquivalenz Relation ist, damit die stark wachsende Eigenschaft der Folge bleibt.

Mit der Hilfe von

***Theorem*[22]**:

$$S_n(M) = \frac{2^{-\binom{n}{2}}}{n!} M^n + \mathcal{O}\left[\frac{2^{-\binom{n}{2}}}{n!} M^n \left(\frac{2^n n^4}{M} \right) \right]$$

wobei $n \to \infty$ und M groß genug ist.

Man kann für $S_n(M - (2^n - 1)D)$ etwa wie

$$D = n^{-2} 2^{-n} M \tag{4.46}$$

nehmen, damit die Äquivalenz Relation zwischen $S_n(M)$ und $S_n(M-(2^n-1)D)$ tatsächlich gilt.

Mit (4.46) kann man ferner ξb_i aus (4.40) setzen, beispielsweise als

$$\xi b_i < \xi M \leq n^{-5} 2^{-n} M \tag{4.47}$$

damit das Rauschen ξb_i gering genug wird.

Infolgedessen beträgt der minimale Abstand nach (4.36) maximal

$$\left| \frac{\xi}{M} \right| \leq \frac{1}{n^5 2^n M} \tag{4.48}$$

Nach (4.18) und (4.36) hat man

$$0 < \frac{W}{M} - \frac{c_i}{b_i} < \frac{1}{2^{n-i} b_i}$$

Also die Breite des gesamten Wertbereichs von V bzgl. b_i und c_i ist kleiner als $\frac{1}{2^{n-i} b_i}$.

Die Größenabschätzung von M ist

$$M < 2^d n, \; d = 2$$

somit wir für höchstens

$$\frac{n^5 2^n M}{2^{n-i} b_i} \leq \left\lceil \frac{2^{2n}}{b_i} \right\rceil 2^i n^5$$

Punkte testen müssen, für jedes festes b_i.

Setze nun

$$\varrho = \left\lceil \frac{2^{2n}}{b_i} \right\rceil$$

Da wir am Anfang b_i, $1 \leq i \leq 4$ fest genommen haben, sind 2^i und ϱ eigentlich konstant, und somit ist das Testen für das ganze Intervall $\mathcal{O}(n^5)$ zu erwarten. Wie vorher besprochen wird, berechnet man die $\mathcal{O}(n^2)$ Abschnitte zuerst und selektiert die Teilintervalle $[\beta, \beta + \zeta]$, und danach testet schrittweise in Weite von $1/\varrho 2^{3n} n^5$ und sieht, ob in h-tem Schritt

$$V^* = \beta + \frac{1}{\varrho 2^{3n} n^5} h \tag{4.49}$$

eine stark wachsende Folge durch $V^* b_i \bmod 1$, $1 \leq i \leq n$ erzeugt. Die proportionale Anzahl der Punkte zu testen ist durchschnittlich $\mathcal{O}(n^3)$.

Somit wird die 1. sowie 2. Frage im Abschnitt 4.1 auch beantwortet.

4.7 Permutation und Zeitkomplexität

Falls das Merkle-Hellman Kryptosystem ohne Permutation eingesetzt wird, muss man nicht raten. Der Zeitaufwand vom Integer-Programming-Problem, um $\{c_j | 1 \leq j \leq 4\}$ zu bestimmen, ist in Praxis polynomial[9, 25], da es nur 4 Unbekannte im Integer-Programming-Problem gibt. Und die durchschnittliche erwartete Anzahl der Lösungen ist wenig. Das schrittweise Testen im richtigen Teilintervall $[\beta, \beta + \zeta]$, um V zu approximieren, ist durchschnittlich $\mathcal{O}(n^3)$, maximal $\mathcal{O}(n^5)$.

Wenn eine Permutation in den öffentlichen Schlüssel eingesetzt wird, wissen wir nicht, ob wir uns in der richtigen Permutation befinden. Wir können aber raten und wegen der Existenz der Lösung prüfen wir jede Möglichkeit, ob die die gewünschte Eigenschaft erfüllt. Dazu raten wir zuerst $(\pi(1), \ldots, \pi(4))$ für $\mathcal{O}(n^4)$ mal. Jedes Mal lösen wir das Integer-Programming-Problem. Wir wissen noch, dass das Selektieren der Teilintervalle eigentlich auch für eine richtige Wahl der Permutation π tut. Bestimmt eine Anordnung von unten nach oben bezüglich den $\pi(i)$ Linienabschnitten eine Permutation $(b_{\pi(5)}, \ldots, b_{\pi(n)})$, startet man das schrittweise Testen im zugehörigen Teilintervall $[\beta, \beta + \zeta]$. [23] Also für ein festes „Raten“ von $(\pi(1), \ldots, \pi(4))$, ist das gesamte Testen mit Berücksichtigung aller Permutationen der restlichen $b_{\pi(i)}$, $5 \leq i \leq n$, maximal $\mathcal{O}(n^5)$ Schritte zu erwarten. Im Vergleich zur gesamten Schritten in der Testphase ohne Permutation haben wir trotz der Permutation den gesamten Aufwand auch auf $\mathcal{O}(n^5)$ begrenzt.

Wenn ein passendes V gefunden wird, wissen wir gleichzeitig die zugehörige Permutation noch. Danach können wir wie Alice alle Ciphertexte erkennen. Somit ist alles in Polynomialzeit berechenbar, und deshalb ist der Angriff von Shamir erfolgreich und auch mit einer Permutation ist das Merkle-Hellman Kryptosystem untauglich.

Kapitel 5

Implementierung von Shamirs Algorithmus

Nach der theoretischen Beschreibung von Merkle-Hellman Kryptosystem und Shamirs Angriff wollen wir jetzt die Verfahren implementieren, damit man die praktische Performance sehen kann.

5.1 Implementierungsumgebung

Hardwarekonfiguration:

Laptop Lenovo ThinkPad X60s
Intel Core Duo L2300 1,5Ghz / 2MB L2 Cache
3GB DDR2-667 RAM
160GB HDD 7200rpm

Betriebssystem:

Microsoft Windows 7 Professional SP1

Softwareplattform:

Development Environment:	Eclipse IDE for Java Developers
Version:	Indigo Service Release 2
Programmiersprache:	Java

5.2 Überblick zur Programmierung

Datenstruktur

Einerseits ist der öffentliche und private Schlüssel die natürliche Zahle mit Bitlänge von ca. $2n$. Im Vergleich zum typischen Datenstruktur von Integer (32Bits) oder Double (ca. 50Bits Mantisse) der Programmiersprachen Java, C++ usw. sind $2n$-Bits für großes n zu lang. Aber Java bietet noch einen schon vorhandenen Datentyp „*BigInteger*“, mit dem man natürliche Zahlen beliebiger Längen bearbeiten kann. Anderseits müssen wir in der Berechnung von rationaler Zahl V möglichst genaue Werte haben, im Gegensatz dazu kann man mit Fließkomma-Zahl das Runden nicht kontrollieren. Deswegen habe ich selbst ein neues Datentyp „*BigRational*“ erfunden, damit man rationale Zahle auch wie „*BigInteger*“ mit typischer Operationen bearbeiten kann. Hier heißt „*Big*“ eigentlich, dass der Teiler und der Nenner „*BigInteger*“ und/oder normale Integer sein können. Mit diesem „*BigRational*“ kann man rationale Zahlen ohne Rundungsfehler beliebig genau berechnen, sodass die Rechnung von V sowohl theoretisch als auch praktisch möglich wird.

Klassifikation

Das gesamte Package „*Kryptosystem*“ besteht für die Implementierung von Merkle-Hellman Kryptosystem und Shamir-Angriff aus folgenden Klassen:

KryptoSystem

Diese Klasse steuert den Ablauf von Verschlüsseln, Entschlüsseln, Angriff usw. Am Ende wird Ergebnisse ausgegeben, damit man sieht, ob der Angriff richtig tut.

PlainText

Diese Klasse dient eine Binärcodierung der originalen Klartext.

MHKey, MHPrivateKey und *MHPublicKey*

Diese drei Klassen sind die Definition und Erzeugung vom öffentlichen und privaten Schlüssel in Merkle-Hellman Kryptosystem.

MerkleHellmann

In dieser Klasse werden Nachrichten nach Merkle-Hellmann Verfahren verschlüsselt sowie entschlüsselt. Eine Permutation ist auch eingesetzt.

ShamirsAttack

Diese Klasse beinhaltet Hauptfunktionen des Shamir-Angriffs. Insbesondere steuert die das Raten der Anfangsfolge und die Durchprüfung möglicher Lösungsliste aus Integer-Programming-Problem. Am Ende werden Nachrichten auch entschlüsselt.

IntegerProgramming

Diese Klasse löst das lineare Ungleichungsproblem und zeigt alle möglichen Lösungen, die die Position vom gesuchten V ggf. bestimmen.

SuperIncreasingSequence

Stark wachsende Folge wird mit $f(V) = b_i V \bmod 1$ hier berechnet und geprüft. Dafür werden zuerst die möglichen Teilintervalle selektiert und die Permutation festgestellt.

5.3 Bemerkungen und Performance des Programms

Bitlänge:

Da die Codierung binär und je Einheit(Byte) 8 Bits lang ist, haben wir mit $n = 8, 16, \ldots$ angefangen. Also sind M, W usw. 16 Bits für $n = 8$, oder 32 Bits für $n = 16$. Aber wenn n noch größer wird, ist der Zeitbedarf extrem hoch. Für $n = 8$ dauert ein kompletter Durchlauf für den Angriff je nach der Anzahl der Lösungen von Integer-Programming-Problem einige Minuten bis ca. Halbstunde. Aber schon bei $n = 16$, muss es normalerweise eine Stunde oder mehr dauern. Und für $n = 32$ oder noch größer kann das am ganzen Tag nicht fertig sein. Natürlich kann ein Angriff nicht mit so schnell sein und Personal Computers sind auch nicht mit einem großen Server oder Cluster vergleichbar. Jedoch kann man in der industriellen Anwendung erst bei der Angabe des öffentlichen Schlüssel den Angriff schon richtig starten, weil die zu sendenden Nachrichten nicht für den Angriff benötigt werden, und außerdem sind die leistungsstärkeren Server die Rechnungsgeschwindigkeit wesentlich zu beschleunigen.

Integer-Programming

Um das ganzzahlige Ungleichungssystem zu lösen, haben wir bei der Nutzung vom berühmten Hilfstool „*Ip_Solve*“ gefunden, dass dieses Simplex-Program für unseres Ungleichungssystem nicht richtig funktioniert sowie ungeeignet. „*Ip_Solve*“ wird von C/C++ implementiert und hat „*double*“ als Datentype benutzt. Das „*double*“ soll eigentlich für Zahlen ca. 50-Bits groß noch in Ordnung sein, aber „*Ip_Solve*“ gibt bei $n = 8$ oder $n = 16$ keine richtige Lösungen aus. Das kann daran liegen, dass „*double*“ Fließkommazahl ist und mit Rundungsfehlern Schritt für Schritt den Fehler der ganzzahligen Lösungen führen. Deswegen sind vielleicht nur richtige „*Rationale*“-Typen geeignet. Leider findet man zurzeit keine passenden Hilfs-Programme bzw. Software und außerdem ist eine richtige Implementierung nicht einfacher Version von Simplex sehr aufwendig. Daher haben wir das

Tool von Simplex verzichten und eine native Methode implementiert, mit der wir unseres Integer-Programming ohne Fehler rechnen können. Die Zeitkomplexität ist aber viel mehr als polynomial. Deswegen sind für $n > 16$ deutlich langsamer. Jedoch ist diese Methode ziemlich sicher und keine richtigen Lösungen sind zu verpassen und zwar c_1 ist von 1 bis $b1 - 1$, und für jedes feste c_1, rechne jeweils c_2, c_3 und c_4. Man muss selbstverständlich für c_i, $2 \leq i \leq 4$ von 1 bis $b_1 - 1$ jeden Wert prüfen. Es gibt intelligente Strategie, damit wir die Berechnung für c_i, $2 \leq i \leq 4$ viel sparen und keine exponentielle Zeit zu haben ist. Trotzdem muss man für c_1 immer noch $\mathcal{O}(2^n)$ Schritte benötigen. Dann ist dieses native Verfahren für Integer-Programming-Problem leider nicht in Polynomialzeit berechenbar.

Verfeinerung für die Ungleichungen

Für die Ungleichungen als Integer-Programming habe ich etwas verfeinert und zwar die Rechtsseite einer Ungleichung ist mir mit jedes c_j und jedes b_i abhängig und damit wird die Suche nach c_j, $1 \leq i \leq 4$ noch schneller und genauer. Ich sehe in vielen Literaturen, dass vermutlich wegen der Übersichtlichkeit dieses Integer-Programming-Problems immer feste Werte für die 6 Ungleichungen genommen werden. Das ist zwar auch richtig, aber in meinem Testen der Implementierung kam manchmal von, dass für ein festes c_1, mehrere möglichen Werte von c_i, $2 \leq i \leq 4$ vorkommen. Das führt zu noch einem kleineren Aufwand mehr. Mit meiner kleinen Verfeinerung wird solcher Sonderfall grundsätzlich vermieden.

Anzahl der Lösungen

In einigen Fällen im praktischen Testen kommen nicht nur eine Lösung von gesuchtem V vor, mit dem man die Nachrichten entschlüsseln kann. Aber nicht jede Lösung von $\{c_j | 1 \leq j \leq 4\}$ aus dem Integer-Programming kann eine richtige stark wachsende Folge bringen. Also ein Testen dafür ist noch notwendig.

Kapitel 6

Fazit

6.1 Zusammenfassung

Die Aufgabenstellung dieser Arbeit ist, um das Merkle-Hellman Kryptosystem und seine Varianten sowie Angriffsverfahren, insbesondere den Shamir-Algorithmus zu analysieren und zu implementieren. Der Shamir-Angriff wurde ausführlich behandelt, wobei es klar ist, dass der Angriff gegen der Lücke vom Merkle-Hellman Kryptosystem mit der stark wachsenden Folge erfolgreich ist. In manchen Stellen habe ich das Verfahren von Shamir verfeinert. In der praktischen Implementierung hat man auch bemerkt, dass der Angriff auch immer funktioniert und manchmal kommen mehrere Lösungen vor, die äquivalent zu entschlüsseln sind. Das alles kann uns in der Zukunft helfen, eine bessere Variante vom Merkle-Hellman Kryptosystem zu erfinden und ggf. anzugreifen.

6.2 Diskussion und offenes Thema

In der Analysis vom Abschnitt 4.6 haben wir gesehen, dass eine stark wachsende Folge mit einem bestimmten Rauschen auch eine stark wachsende Folge erzeugen kann.[22] Beispielsweise haben wir im Shamir-Angriff aus der Gleichung 4.40 und der Ungleichung 4.47

$$a_i^* = a_i - \xi b_i$$

$$\xi b_i < n^{-5} 2^{-n} M$$

Wenn das Rauschen ξb_i so klein genug ist, kann die Folge von a_i^*, $1 \leq i \leq n$ auch stark

wachsend sein.

Was zu beachten ist, dass hier a_i^* eine rationale Zahl ist und nicht immer ganze Zahl sein kann. Aber wenn man eine Kollision der öffentlichen Schlüssel im Merkle-Hellman Kryptosystem haben will, soll die Folge aus allen a_i^*mit ganzzahligen Einträgen bestehen. Wir müssen ξb_i mit $\xi b_i < n^{-5}2^{-n}M$ sorgfältig selektieren, sodass ξb_i auch positive ganze Zahl ist, für alle $1 \leq i \leq n$. Im Allgemeinen ist M $2n$-Bits lang, also $M \geq 2^{2n-1}$. Dann ist $n^{-5}2^{-n}M \geq n^{-5}2^{n-1}$. Falls n groß genug ist, kann $\left\lfloor \frac{2^{n-1}}{n^5} \right\rfloor$ eine positive ganze Zahl sein. Somit ist die Folge von a_i^*, $1 \leq i \leq n$ äquivalent zur ursprünglichen Folge $(a_1, \ldots, a_n)$.

Ferner können die anderen Teile des privaten Schlüssels, nämlich M, U und W, mit allen a_i^*, $1 \leq i \leq n$ einen äquivalenten öffentlichen Schlüssel zur Kollision bringen, da $M > \sum_{i=1}^{n} a_i^*$ gelten soll.

Wir sehen jetzt dann, wann eine Kollision des öffentlichen Schlüssels vorkommen kann. Hierzu bleibt noch die Frage, wie wahrscheinlich diese Kollision vorkommt? Wegen der Zeitbeschränkung habe ich das noch nicht analysiert. Das kann man in der Zukunft noch genauer erforschen.

Literaturverzeichnis

[1] Ralph C. Merkle, Student Member, Ieee, Martin E. Hellman, and Senior Member. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions On Information Theory*, 24:525–530, 1978.

[2] Frieder Knueppel. Asymmetrische Kryptosysteme, 2013.

[3] Adi Shamir. A Polynomial-Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem, 1982.

[4] Adi Shamir. A Polynomial-Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem. In *IEEE Transcations On Inofmation Theory, Vol. IT-30, No. 5, September 1984*, 1984.

[5] Rajib Biswas. BITS Pilani. Asymmetric Cryptosystem.

[6] J. C. Lagarias. Knapsack Public Key Cryptosystems and Diophantine Approximation. In David Chaum, editor, *CRYPTO*, pages 3–23. Plenum Press, New York, 1983.

[7] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. In *in Proceedings of 24rd Annu. Symp. Foundations of comput. Sci*, 1983.

[8] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer national Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.

[9] Lenstra A.K. LovÃ¡sz L. Lenstra, H.W. jr. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[10] Henri Cohen. *A course in computational algebraic number theory*. Springer, 2000.

[11] E. F. Brickell and A. M. Odlyzko. Cryptanalysis: a survey of recent results. In *Proc.IEEE 76*, 1988.

[12] Keiji Oomura and Keisuke Tanaka. Density Attack on the Knapsack Cryptosystems with Enumerative Source (extended abstract), 2003.

[13] Matthijs J. Coster, Antoine Joux, Brian A. Lamacchia, Andrew M. Odlyzko, Claus peter Schnorr, and Jacques Stern. An improved low-density subset sum algorithm. In *in Advances in Cryptology: Proceedings of Eurocrypt '91*, 1991.

[14] Claude E. Shannon. Communication Theory of Secrecy Systems. In *Bell System Technical Journal, vol.28-4*, 1949.

[15] Benny Chor and Ronald L. Rivest. A Knapsack Type Public Key Cryptosystem Based on Arithmetic in Finite Fields. *IEEE Trans. Inform. Theory*, 34:54–65, 1988.

[16] Serge Vaudenay. Cryptanalysis of the Chor-Rivest Cryptosystem. In *CRYPTO '98*, pages 243–256. Springer-Verlag, 1998.

[17] Hendrik W. Lenstra Jr. On the Chor-Rivest Knapsack Cryptosystem. *J. Cryptology*, 3(3):149–155, 1991.

[18] Bao-cang and Hu Yu-pu. Public Key Cryptosystem using Random Knapsacks. *Journal of Electronics and Information Technology*, 2010.

[19] Fei Xiang dong; Ding Yan-yan and Pan Yu. Analysis and improvement of a knapsack Public-key cryptosystem, 2011.

[20] Min-Shiang, Cheng-Chi Lee, , and Shiang-Feng Tzeng. A New Knapsack Public-key Cryptosystem Based on Permutation Combination Algorithm. 3(9):855 – 861, 2009.

[21] Hamid R. Dalili Rastaghi, Roohallah; Oskouei. Cryptanalysis of a Public-key Cryptosystem using Lattice Basis Reduction Algorithm. In *International Journal of Computer Science Issues (IJCSI) . Sep2012, Vol. 9 Issue 5, p110-117. 8p*, 2012.

[22] J. C. Lagarias. Performance Analysis of Shamir's Attack on the Basic Merkle-Hellman Knapsack Cryptosystem, 1984.

[23] Volker Diekert, Manfred Kufleitner, and Gerhard Rosenberger. *Elemente der Diskreten Mathematik*. Walter de Gruyter, 2013.

[24] Eberhard Karls University Tuebingen Informatilk. Interaktives Kryptologie Skript, Shamirs Algorithmus, 2013.

[25] Jr. H. W. Lenstra. Integer Programming and Cryptography, 1984.

[26] George B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.