Über Schnorr's Preprocessing für Diskrete LOG Unterschriften

Diplomarbeit

von

Johannes Merkle

Fachbereich Mathematik

Johann–Wolfgang–Goethe–Universität
Frankfurt am Main

19. September 1995

Vorwort

In einer Gesellschaft, in der die Bedeutung des Informationsaustausches über elektronische Medien ständig zunimmt, sind digitale Public–Key Authentikations– und Unterschriften–Schemata wichtig. Durch sie lassen sich in einem Kommunikationsnetz die Zugriffsberechtigung eines Teilnehmers kontrollieren (Authentikation) bzw. die Herkunft einer Nachricht überprüfen (Unterschrift). Public–Key bedeutet, daß sich der digitale Schlüssel des Anwenders aus einem geheimen und einem öffentlichem Teil zusammensetzt. Seit der Erfindung des RSA–Schemas [5] wurden in der Kryptographie große Anstrengungen unternommen, um die Effizienz dieser Schemata zu verbessern. Viele dieser Schemata basieren auf dem diskreten Logarithmus, so z.B. die Schemata von ElGamal [2], Beth [1] und Günther [3]. Die Berechnung des diskreten Logarithmus gilt als schwierig. Trotz großer Anstrengungen wurde bisher kein effizienter Algorithmus dazu gefunden. Die Umkehrfunktion hingegen — die Exponentiation in einem Modul — kann wesentlich schneller berechnet werden. Dies nutzt ein auf dem diskreten Logarithmus basierendes Public–Key Schema aus. Aus dem geheimen Schlüssel und der Nachricht wird durch Exponentiation eine Unterschrift erzeugt. Diese kann öffentlich gemacht werden. Der geheime Schlüssel läßt sich daraus jedoch in vernünftigem Zeitaufwand nicht mehr ermitteln. Die Sicherheit eines solchen Schemas ist also immer von der Berechenbarkeit des diskreten Logarithmus abhängig.

Schnorr stellte 1989 ein Public–Key Shema vor, welches auf dem diskreten Logarithmus basiert und besonders effizient beim Erzeugen der Authentikation bzw. der Unterschrift ist [8]. Dies macht es besonders geeignet für die Interaktion zwischen Smart–Cards (z.B. Telefonkarten) und einem Terminal. Die hohe Effizienz in der Authentikations– bzw. Unterschriftenerzeugung wird durch die Beschränkung dieser Prozesse auf im wesentlichen eine Exponentiation und durch die Verwendung eines Preprocessings zur Beschleunigung derselben erreicht. Außerdem kann diese Exponentiation vor bzw. zwischen den Interaktionen durchgeführt werden, da sie von der Interaktion und der Nachricht unabhängig ist. Das Schema ist sicher, sofern das Preprocessing sicher ist. Dabei sehen wir ein Verfahren als sicher an, wenn es es keine Attacke mit einer Workload $< 2^{72}$ gibt. Unter der Workload einer Attacke versteht man die zu erwartende Schrittzahl zu ihrer erfolgreichen Durchführung dividiert durch die Erfolgsquote.

Das Preprocessing ist vom Rest des Schemas unabhängig und kann daher auch in anderen Schemata verwendet werden, die auf dem diskreten Logarithmus basieren. Da das Schema selbst nicht vom Preprocessing abhängt, bleibt es letztendlich dem Anwender überlassen, welche Art von Preprocessing er verwendet. Es ist aber wichtig, zumindest ein sicheres Preprocessing zu kennen. Schnorr gab

ein Beispiel für das Preprocessing an. Durch dieses wurde die Anzahl der zur Authentikations– bzw. Unterschriftenerzeugung benötigten Multiplikationen von 210 auf 12 gesenkt [8]. Das angegebene Preprocessing wurde jedoch schon bald erfolgreich von de Rooij angegriffen [6]. Auch ein von Schnorr verbesserter Algorithmus [9] wurde von de Rooij gebrochen [7]. Daraufhin änderte Schnorr das Preprocessing erneut ab, ohne es jedoch zu veröffentlichen. Das Ziel dieser Arbeit war es, auf dieser Grundlage ein sicheres Preprocessing zu entwickeln. Wir glauben, daß dies gelungen ist, auch wenn sich die Sicherheit letztendlich nicht streng mathematisch beweisen ließ.

Die Gliederung dieser Arbeit stellt in etwa die historische Entwicklung vom ersten Preprocessing in [8] zu der am Ende der Arbeit stehenden Form dar. Wir sind jedoch noch zu keinem endgültigen Algorithmus für das Preprocessing gekommen. Vielmehr geben wir am Ende eine Grundform mit Empfehlungen für die Parameter und verschiedenen Variationsmöglichkeiten an. Die Untersuchungen über diese und mögliche andere Variationen sind noch nicht abgeschlossen.

In Kapitel 1 legen wir die historische Entwicklung bis zum Ausgangspunkt unserer eigenen Überlegungen dar. Diese besteht aus den Veröffentlichungen von de Rooij [6],[7] und Schnorr [8],[9].

In Kapitel 2 stellen wir einige einfache Attacken gegen das verbesserte, unveröffentlichte Preprocessing von Schnorr vor. Im Zuge dieser Überlegungen modifizieren wir das Preprocessing, um die angegebenen Attacken abzuwehren.

Ab Kapitel 3 behandeln wir das Preprocessing mit den Methoden der linearen Algebra. Dies scheint uns der angemessene Weg, um systematisch nach Attacken zu suchen. Die dafür notwendigen Begriffe und Definitionen stellen wir — ebenso wie einfache Zusammenhänge zwischen ihnen — in Kapitel 3 zur Verfügung. Vor allem stellen wir die das Preprocessing repräsentierenden Gleichungen als Matrix dar. Damit führen wir alle von uns betrachteten Attacken auf n–Zeilen–Versuche — Linearkombinationen von n Zeilen dieser Matrix — zurück.

Theoretische Ergebnisse über n–Zeilen–Versuche zeigen wir in den Kapiteln 4 und 6. Unter anderem zeigen wir in Kapitel 4, daß aus Linearkombinationen von weniger als 5 Zeilen keine erfolgreichen Attacken hervorgehen.

Es zeigt sich schnell, daß wir mit theoretischen Überlegungen allein die Sicherheit des Preprocessings — d.h. die minimale Workload einer Attacke — nicht bestimmen können. Deshalb entwickeln wir in den Kapiteln 5 und 8 ein Computerprogramm. Mit diesem schätzen wir die Workload von Attacken ab.

Von Kapitel 7 an belegen wir viele unserer Aussagen durch statistische Tests. Dies geschieht jedoch nur dann, wenn wir keine theoretische Argumentation gefunden haben. Die Komplexität der Gleichungen läßt diese oftmals nicht mehr zu.

Die ersten Ergebnisse des Programms führen dann zur weiteren Modifikation des Preprocessings in Kapitel 7. Es zeigt sich, daß sich die Sicherheit durch diese Modifikation beträchtlich erhöht.

In den Kapiteln 8, 9 und 10 betrachten wir nacheinander 5–, 6– und 7–Zeilen–Versuche. Es zeigt sich, daß die Workload von Attacken, die aus solchen Linearkombinationen hervorgehen, mit der Anzahl der verwendeten Zeilen — d.h. mit n — ansteigt. Wir sehen darin ein starkes Indiz für eine allgemeine Tendenz. Damit folgt dann, daß die Attacken mit der kleinsten Workload aus 5–Zeilen–Versuchen hervorgehen. Die von uns abgeschätzte Sicherheit des Preprocessings geben wir für verschiedene Parameter am Ende von Kapitel 10 an.

In Kapitel 11 zeigen wir noch einen ganz anderen Ansatz für eine Attacke, der mit den Methoden der Gittertheorie arbeitet. Eine Analyse zeigt aber, daß dieser Ansatz zu keiner effizienten Attacke führt.

In Kapitel 12 diskutieren wir noch einige Variationen des Preprocessings, die geeignet erscheinen, die Sicherheit zu erhöhen. Zu manchen können wir allerdings noch keine Aussagen treffen, da unsere Untersuchungen noch nicht abgeschlossen sind. Eine Zusammenfassung erfolgt in Kapitel 13.

Das von Seiten der Praxis geäußerte Interesse an dem Preprocessing sowie die gleichzeitige Forschung französischer Mathematiker auf demselben Gebiet unterstreicht die Bedeutung des Preprocessings. Es ist deshalb mit einer weiteren Entwicklung auf diesem Gebiet zu rechnen.

Ich danke Claus Peter Schnorr für eine fruchtbare und wertvolle Dikussion zu Inhalt und Form der Arbeit sowie Harald Ritter für seine geduldige Hilfestellung bei meiner schwierigen Einarbeitung in das UNIX–Betriebssystem.

Inhaltsverzeichnis

1 Das alte Preprocessing

Auf der CRYPTO'89–Tagung veröffentlichte Schnorr [8] ein neues Public–Key–Unterschriften–Schema mit dazugehörigem Authentikations–Schema, das auf dem diskreten Logarithmus basiert und speziell für die Interaktion zwischen Smart Card und Terminal geeignet ist.

Zum Schnorr–Schema gehört ein Preprocessing, das die Exponentiation $r \rightarrow \alpha^r \bmod p$ einer Zufallszahl r simuliert und dadurch die Anzahl der Rechenschritte zur Unterschriftenerzeugung erheblich reduziert. Damit wird das Schema der geringen Rechenleistung von Smart Cards gerecht. Da das Preprocessing unabhängig vom Rest des Schemas ist, kann es auch für andere Schemata verwendet werden, die auf dem diskreten Logarithmus basieren, so z.B. die Schemata von ElGamal [2], Beth [1] und Günther [3]. Wir geben zuerst eine kurze Darstellung des Schnorr–Schemas:

1.1 Das Authentikations– und Unterschriften–Schema

Sei n eine natürliche Zahl. Dann identifizieren wir $\mathbf{Z}/n\mathbf{Z}$ mit $\{0, 1, \ldots, n-1\}$, d.h. die Restklassen mod$n$ werden durch ihre Residien in $\{0, 1, \ldots, n-1\}$ repräsentiert. Seien p und q prim mit $q > 2^{140},\ p > 2^{512}$ und $\alpha \in \mathbf{Z}_p$ mit $\mathrm{ord}(\alpha) = q$. Jeder User wählt einen privaten Schlüssel $s \in \{1, \ldots, q\}$ und berechnet den öffentlichen Schlüssel $v = \alpha^{-s} \bmod p$. Dieser kann in einem öffentlichen File von jedem User nachgelesen oder direkt vom KAC (Key Authentication Center) erfragt werden. v ist also jedem User bekannt. $t < \log_2 q$ ist ein Sicherheitsparameter des Schemas.

Das Authentikations–Protokoll (A beweist B gegenüber seine Identität)

1. A wählt ein zufälliges $r \in \{0, \ldots, q-1\}$ und sendet $x = \alpha^r \bmod p$ an B.
2. B sendet ein zufälliges $e \in \{0, \ldots, 2^t - 1\}$ an A.
3. A sendet $y := r + se \bmod q$ an B.
4. B überprüft, ob $x = \alpha^y v^e \bmod p$ gilt.

Wenn A und B das Protokoll befolgen, akzeptiert B den Identitäts–Beweis von A. Ein betrügerischer A, der den geheimen Schlüssel s nicht kennt, kann e erraten und von B akzeptiert werden. Die Erfolgswahrscheinlichkeit für diesen Betrugsversuch ist 2^{-t}. Es gibt keinen Betrugsversuch mit größerer Erfolgswahrscheinlichkeit, sofern nicht das Berechnen von $\log_\alpha(v)$ leicht ist ([9]).

Das Unterschriften–Protokoll (A beweist B, daß die Nachricht m von ihm kommt)
Hier verwenden A und B eine öffentlich bekannte Hash–Funktion h.

1. A wählt ein zufälliges $r \in \{1, \ldots, q\}$ und bildet $x := \alpha^r \bmod p$
2. A berechnet $e := h(x, m) \in \{0, \ldots, 2^t - 1\}$
3. A berechnet $y := r + se \bmod q$. (e, y) ist die Unterschrift der Nachricht m.
4. B berechnet $\bar{x} := \alpha^y v^e \bmod q$ und testet, ob $h(\bar{x}, m) = e$ gilt.

Die Sicherheit des Unterschriften–Protokolles hängt auch von der gewählten Hash–Funktion h ab. Deshalb können wir hier keine so konkreten Aussagen wie über die Sicherheit des Authentikations–Protokolles machen. Es ist uns jedoch keine Attacke gegen das Unterschriften–Protokoll bekannt, die nicht auch gegen das Authentikations–Protokoll angeführt werden könnte.

Unabhängig davon, ob es sich um das Authentikations–Protokoll oder um das Unterschriften–Protokoll handelt, nennen wir das Paar (e, y) die **Signatur**.

Beim Erzeugen einer Signatur wird der weitaus größte Teil der Rechenzeit zum Bilden von $x = \alpha^r \bmod p$ benötigt. Ist q 140 Bits lang, so benötigt diese Exponentiation bis zu 210 Multiplikationen. Durch das Preprocessing soll dieser Rechenaufwand erheblich reduziert werden, ohne daß das Schema an Sicherheit einbüßt.

Schnorr [8] gab folgendes Beispiel für das Preprocessing an:

1.2 Das Preprocessing

Die Smart Card speichert k zufällig und unabhängig voneinander gewählte Paare (r_i, x_i) mit $x_i = \alpha^{r_i} \bmod p$ für $i = 0, \ldots, k-1$. Zu Anfang können diese Paare vom Key–Authentication–Center (KAC) errechnet werden. Für jede Signatur erzeugt die Smart Card aus den gespeicherten Paaren ein neues (r, x) und ersetzt anschließend ein Paar (r_i, x_i) durch eine Kombination $(\sum c_i r_i \bmod q, \prod x_i^{c_i} \bmod p)$, mit zufällig gewählten ganzen Zahlen $c_0, \ldots, c_{k-1}$.

1.2.1 Der Preprocessing–Algorithmus

Initiation: lade k Paare $(r_0, x_0), \ldots, (r_{k-1}, x_{k-1})$ mit $x_i = \alpha^{r_i} \bmod q$ und zufälligen, unabhängigen $r_0, \ldots, k_{k-1}$ in $\mathbf{Z}_q$.
$\nu := k$

1. wähle zufällige $a(0,\nu), \ldots, a(d-3,\nu) \in \{\nu - k, \ldots, \nu - 1\}$
$a(d-2,\nu) := \nu - 1,\ a(d-1,\nu) := \nu - k,\ a(d,\nu) := \nu - 1$

2. $r_\nu := \sum_{i=0}^{d} 2^i r_{a(i,\nu)} \bmod q$ $\qquad$ $x_\nu = \prod_{i=0}^{d} x_{a(i,\nu)}^{2^i} \bmod p$
$r_\nu^* := r_{\nu-k} + 2r_{\nu-1} \bmod q$ $\qquad$ $x_\nu^* := x_{\nu-k} x_{\nu-1}^2 \bmod p$

3. verwende (r_ν^*, x_ν^*) für die nächste Signatur (e_ν, y_ν) gemäß $y_\nu = r_\nu^* + se_\nu \bmod q$ und ersetze $(r_{\nu-k}, x_{\nu-k})$ durch (r_ν, x_ν).

4. $\nu := \nu + 1$
GOTO 1. für die nächste Signatur.

d ist hier ein Sicherheits–Parameter.

Bemerkung: Unsere Notation entspricht der von de Rooij [6]. Sie unterscheidet sich geringfügig von der von Schnorr [8] gewählten, stellt aber das gleiche Verfahren dar.

Während bei $q > 2^{140}$ die Exponentiation $r \to \alpha^r \bmod p$ über 210 Multiplikationen mod$p$ erfordern kann, kommt das Preprocessing mit $2d + 2$ Multiplikationen aus. Schnorr [8] schlug als Parameter $k = 8$ und $d = 6$ vor, um zu gewährleisten, daß erfolgreiche Attacken eine Workload von mindestens 2^{72} haben. Diese Forderung erscheint dem Stand der Technik angemessen. Wie sich bald zeigte, erfüllt das Preprocessing in dieser Form diese Anforderung nicht.

1.2.2 Ziel und Form der Attacken

Wir untersuchen Attacken, welche den geheimen Schlüssel s aus der Folge der Signaturen (e_ν, y_ν) für $\nu = k, k+1, \ldots$ ermitteln. Die Folge der Signaturen wird als bekannt vorausgesetzt. Wir beschränken uns auf Attacken, welche s aus den

linearen Gleichungen $r_\nu = \sum_{i=0}^{d} 2^i r_{a(i,\nu)} \bmod q$, $r_\nu^* = r_{\nu-k} + 2r_{\nu-1} \bmod q$, und $y_\nu = r_\nu^* + se_\nu \bmod q$ für $\nu = k, k+1, \ldots$ bestimmen. Die x_ν–Werte und die multiplikativen Gleichungen modulo p bleiben außer Betracht.

Eine erfolgreiche Attacke besteht aus Bedingungen an die Werte $a(i,\nu)$ derart, daß durch Erraten einiger Werte $a(i,\nu)$ die Bestimmung von s aus den linearen Gleichungen möglich ist.

Erfolgreiche Attacken sind mit $k+1$ beliebigen Signaturen möglich. Es genügen nämlich zur Bestimmung der $k+1$ Unbestimmten $r_0, \ldots, r_{k-1}$ (die Preprocessing–Eingabe) und s die Kenntnis von $k+1$ linearen Gleichungen in diesen Unbestimmten.

Unter der **Workload** einer Attacke verstehen wir den Erwartungswert der Schrittzahl für ihre erfolgreiche Durchführung. Dabei setzen wir für einfache Standardaufgaben, wie dem Lösen eines kleinen linearen Gleichungssystems, die Schrittzahl 1 an. Damit setzt sich die Workload aus den beiden folgenden Faktoren zusammen:

1. Dem Kehrwert der Wahrscheinlichkeit, daß die zufälligen Werte $a(i,\nu), b(i,\nu)$ die geforderten Bedingungen der Attacke erfüllen.

2. Dem Erwartungswert der Schrittzahl zum Erraten der benötigten zufälligen Werte $a(i,\nu)$.

Weil alle übrigen Beiträge zur Schrittzahl vernachlässigt werden, stellt diese Workload eine untere Schranke für die tatsächlich zu erwartende Schrittzahl für die erfolgreiche Durchführung der Attacke dar. Für den Erwartungswert der Schrittzahl zum Erraten der benötigten zufälligen Werte $a(i,\nu)$ nehmen wir stets die Anzahl der Wertekombinationen, welche die Bedingungen der Attacke erfüllen.

1.3 Die Attacken von de Rooij

1.3.1 Die erste Attacke

De Rooij [6] stellte auf der EUROCRYPT–Tagung '91 folgende Attacke gegen das Preprocessing vor:

Seien $i,\ l,\ c,\ a,\ b \in \mathbf{N},\ a < b$. Durch die Gleichungen $r_j^* = r_{j-k} + 2r_{j-1} \bmod q$ und $y_j = r_j^* + se_j \bmod q$ für $j = l+k, l+2k-1, \ldots, l+c\cdot(k-1)+1$ erhält man

eine Gleichung nur in $r_l, r_{l+c(k-1)}$ und s:

$$r_{l+c(k-1)} = (-2)^{-c} r_l - \sum_{j=1}^{c} (-2)^{j-c-1} (y_{l+j(k-1)+1} - s e_{l+j(k-1)+1}) \bmod q \qquad (1)$$

Gilt für $\nu = i,\ i + a \cdot (k-1),\ i + b \cdot (k-1)$ und für alle $j = 1, \ldots, d-3$

$$a(j,\nu) = \nu - 1 \ \vee \ a(j,\nu) = \nu - k$$

so ergibt sich, wenn wir die rechte Seite der Gleichung $r_\nu := \sum_{i=1}^{d} 2^i r_{a(i,\nu)} \bmod q$ nach $r_{\nu-1}$ und $r_{\nu-k}$ sortieren:

$$\begin{aligned} r_i &= \lambda r_{i-1} + \mu r_{i-k} \bmod q \\ r_{i+a(k-1)} &= \lambda' r_{i-1+a(k-1)} + \mu' r_{i-k+a(k-1)} \bmod q \\ r_{i+b(k-1)} &= \lambda'' r_{i-1+b(k-1)} + \mu'' r_{i-k+b(k-1)} \bmod q \end{aligned} \qquad (2)$$

Die Koeffizienten $\lambda, \lambda', \lambda'', \mu, \mu', \mu''$ hängen von den Werten $a(j,\nu)$ ab. Diese $a(j,\nu)$ sind zu erraten. Wenden wir die Gleichung (1) mit $l = i, i-1$ und $c = a, b, a-1, b-1$ an, so erhalten wir daraus 3 Gleichungen in r_i, r_{i-1} und s und können s ermitteln.

Die Attacke:

Bedingung: Für feste $a, b \in \mathbf{N},\ \nu = i,\ i + a \cdot (k-1),\ i + b \cdot (k-1)$ gelte für $j = 0, \ldots, d-3$, daß $a(j,\nu) \in \{\nu - 1, \nu - k\}$.

Durchführung: Berechne aus den $a(j,\nu)$ die Koeffizienten $\lambda, \lambda', \lambda'', \mu, \mu', \mu''$, forme die Gleichungen (2) mit (1) zu 3 Gleichungen in r_i, r_{i-1} und s um, und bestimme daraus s.

Die Workload ist die Anzahl der Möglichkeiten für die $3(d-2)$ Werte $a(j,\nu)$, also

$$\text{Workload} = k^{3(d-2)} \approx \ 2^{36}$$

für $k = 8$ und $d = 6$.

1.3.2 Das modifizierte Preprocessing

Nach der erfolgreichen Attacke von de Rooij [6] änderte Schnorr [9] die Schritte 1. und 2. des Preprocessings geringfügig ab:

1. wähle eine zufällige Permutation $a(\cdot,\nu)$ von $\{\nu - k, \ldots, \nu - 1\}$
$a(k+1,\nu) := \nu - k,\ a(k+2,\nu) := \nu - 1$

2. $r_\nu := \sum_{i=1}^{k+2} 2^{i-1} r_{a(i,\nu)} \bmod q$ $\qquad$ $x_\nu := \prod_{i=1}^{k+2} x_{a(i,\nu)}^{2^{i-1}} \bmod p$

$r^*_\nu := r_{\nu-k} + 2r_{\nu-1} \bmod q$ $\qquad$ $x^*_\nu := x_{\nu-k} x^2_{\nu-1} \bmod p$

Das neue Preprocessing erforderte $2k+2$ Multiplikationen pro Runde. Wieder schlug Schnorr [9] $k=8$ vor. Die Attacke von de Rooij [7] war durch die Forderung, daß $(a(1,\nu),\ldots,a(k,\nu))$ eine Permutation von $(\nu-k,\ldots,\nu-1)$ ist, abgewehrt.

1.3.3 Die zweite Attacke

1993 veröffentlichte de Rooij [7] eine neue Attacke gegen das verbesserte Preprocessing:

Wieder gilt für alle l, c die Gleichung (1). Wir ersetzen in der Gleichung

$$r_\nu := \sum_{i=1}^{k+2} 2^{i-1} r_{a(i,\nu)} \bmod q \tag{3}$$

für $j = \nu, a(i,\nu)$ die r_j durch (1) (mit $j = l + c(k-1)$ und $0 \le l \le k-2$). Fassen wir alle y_i in Y_ν und alle e_i in E_ν zusammen, so erhalten wir:

$$Y_\nu = \sum_{j=1}^{k-2} m(j,\nu) r_j + sE_\nu \bmod q \tag{4}$$

Hierbei hängt E_ν von den e_i, von ν und von der Permutation $a(\cdot,\nu)$ ab, Y_ν hängt von den y_i, von ν und von der Permutation $a(\cdot,\nu)$ ab und die $m(j,\nu)$ hängen von j, ν und von der Permutation $a(\cdot,\nu)$ ab.

Gilt nun für $i_1 < i_2$ mit $i_1 \equiv i_2 \bmod k-1$

$$a(\cdot, i_1) = a(\cdot, i_2)$$

so erhalten wir bei allen Ersetzungen der r_j in (3) durch (1) im Fall $\nu = i_1$ dasselbe l und ein um die Konstante $d := (i_2 - i_1)/(k-1)$ größeres c als im Fall $\nu = i_2$. Dieser zusätzliche Faktor 2^{-d} steht im Fall $\nu = i_2$ in allen Summanden der rechten Seite der Gleichung (1) und damit auch in allen Summanden der Gleichung (4). Wir können also im Fall $\nu = i_2$ diesen Faktor 2^{-d} aus (4) herauskürzen, und erhalten dann wegen $a(\cdot, i_1) = a(\cdot, i_2)$

$$m(j, i_1) = m(j, i_2) \bmod q$$

für alle $j = 1, \ldots, k-2$. Daraus folgt

$$s(E_{i_1} - E_{i_2}) = Y_{i_1} - Y_{i_2} \bmod q$$

Es gilt auch umgekehrt: $m(j, i_1) = m(j, i_2)$ für $j = 1, \ldots, k-2$ nur wenn

1. $i_1 \equiv i_2 \bmod k-1$ und
2. $a(\cdot, i_1) = a(\cdot, i_2)$

gilt. Wir verzichten auf den einfachen Beweis.

Da die E_ν nicht nur von ν und $a(\cdot, \nu)$, sondern auch von den e_j abhängen, gilt mit sehr großer Wahrscheinlichkeit $E_{i_1} \neq E_{i_2} \bmod q$.

Die Attacke:

Bedingung: Für feste $i_1 < i_2$ mit $i_1 \equiv i_2 \bmod k-1$ gelte $a(\cdot, i_1) = a(\cdot, i_2)$
Durchführung:
1. Errate $a(\cdot, i_1)$
2. Errechne aus $i_1,\ i_2,\ a(\cdot, i_1)$ und den Signaturen (e_ν, y_ν) für $\nu = 1, \ldots, i_2$ die Werte $E_{i_1},\ E_{i_2},\ Y_{i_1}$ und Y_{i_2}
3. Im Falle $E_{i_1} \neq E_{i_2} \bmod q$ gilt $s = (Y_{i_1} - Y_{i_2})/(E_{i_1} - E_{i_2}) \bmod q$

Im Folgenden steht Pr für Wahrscheinlichkeit

Die Workload ist das Produkt der folgenden Faktoren:

1. $\Pr[a(\cdot, i_1) = a(\cdot, i_2)]^{-1}$	$k!$
2. Anzahl der Permutationen $a(\cdot, i_1)$	$k!$
3. $\Pr[E_{i_1} \neq E_{i_2} \bmod q]^{-1}$	≈ 1

Insgesamt gilt also:

$$\text{Workload} \approx (k!)^2 \overset{k=8}{\approx} 2^{31}$$

1.3.4 Schlußfolgerung

Die beiden Preprocessing–Beispiele in [8] und [9] sind so konzipiert, daß beliebige k aufeinanderfolgende Werte $r^*_\nu, \ldots, r^*_{\nu+k-1}$ statistisch unabhängig sind. Diese Unabhängigkeit wird durch die Gleichung $r^*_\nu = r_{\nu-k} + 2r_{\nu-1} \bmod q$ erreicht. Nach de Rooij ermöglicht gerade diese nicht randomisierte Gleichung effiziente Attacken. Im folgenden wird diese Gleichung und damit die statistische Unabhängigkeit von k aufeinanderfolgenden Signaturen aufgegeben.

2 Eine neue Grundform des Preprocessings

Wir stellen eine neue Grundform für das Preprocessing vor. Zu dieser Grundform stellen wir dann effiziente Attacken zusammen. Das neue Preprocessing entwickeln wir aus der Grundform dadurch, daß mögliche effiziente Attacken durch Stellen geeigneter Bedingungen an die zu wählenden Zufallswerte verhindert werden.

In der neuen Grundform des Preprocessings wird — zur Verwendung in der Signatur — ein zusätzlicher Wert r_k mitgeführt. Dieser Wert r_k wird in jeder Runde gemäß einer randomisierten linearen Gleichung

$$r_k := \sum_{i=1}^{l} 2^{i-1} r_{a(i,\nu)} \bmod q$$

transformiert. Diese randomisierte Gleichung ersetzt nun die feste Gleichung $r^*_\nu = r_{\nu-k} + 2r_{\nu-1} \bmod q$ des alten Preprocessings. In jeder Runde ν werden die Werte r_k und $r_{\nu \bmod k}$ mittels zufälliger Werte $a(1,\nu), \ldots, a(l,\nu)$ und $b(1,\nu), \ldots, b(l,\nu)$ aus $\{0, \ldots, k\}$ linear transformiert. Dabei steht $\nu \bmod k$ für das Residuum von ν in $\{0, \ldots, k-1\}$. Sicherheitsparameter für die neue Grundform ist eine ganze Zahl l mit $1 \leq l \leq k$.

2.1 Der neue Preprocessing–Algorithmus

Initiation: lade $k+1$ Paare $(r_0, x_0) \ldots, (r_k, x_k)$ mit $x_i = \alpha^{r_i} \bmod p$ und zufälligen, unabhängigen $r_o, \ldots, r_k$ in $\mathbf{Z}_q$.
$\nu := 1$. ν ist die Rundennummer

1. wähle l verschiedene Zufallszahlen $a(1,\nu), \ldots, a(l,\nu) \in \{0, \ldots, k\}$ mit $k \in \{a(1,\nu), \ldots, a(l,\nu)\}$

$$r_k := \sum_{i=1}^{l} 2^{i-1} r_{a(i,\nu)} \bmod q \qquad x_k = \prod_{i=1}^{l} x_{a(i,\nu)}^{2^{i-1}} \bmod p$$

2. wähle l verschiedene Zufallszahlen $b(1,\nu), \ldots, b(l,\nu) \in \{0, \ldots, k\}$ mit $\nu \in \{b(1,\nu), \ldots b(l,\nu)\}$

$$r_{\nu \bmod k} := \sum_{i=1}^{l} 2^{i-1} r_{b(i,\nu)} \bmod q \qquad x_{\nu \bmod k} = \prod_{i=1}^{l} x_{b(i,\nu)}^{2^{i-1}} \bmod p$$

3. verwende (r_k, x_k) für die ν–te Signatur (e_ν, y_ν) gemäß
$y_\nu = r_k + se_\nu \bmod q$

4. $\nu := \nu + 1$
GOTO 1.

Die Zufallszahlen $a(1,\nu), \ldots, a(l,\nu)$ und $b(1,\nu), \ldots, b(l,\nu)$ werden unabhängig gewählt, so daß die angegebenen Bedingungen erfüllt sind. Das Verfahren benötigt $4l-4$ Multiplikationen pro Runde

Notation: r_i^ν bezeichne den Wert r_i am Ende von Runde ν. $r_0^0, \ldots, r_k^0$ bezeichnen die zu Anfang geladenen Werte $r_0, \ldots, r_k$. Damit gilt $y_\nu = r_k^\nu + se_\nu$ für alle $\nu = 1, 2, \ldots$.

Bemerkung: Die Bedingungen $k \in \{a(1,\nu), \ldots, a(l,\nu)\}$ und $\nu \in \{b(1,\nu), \ldots, b(l,\nu)\}$ sichern, daß für jedes $\nu \geq 1$ und für jede Wahl von $a(1,\nu), \ldots, a(l,\nu), b(1,\nu), \ldots b(l,\nu)$ die lineare Transformation

$$(r_0^{\nu-1}, \ldots, r_k^{\nu-1}) \longrightarrow (r_0^\nu, \ldots, r_k^\nu)$$

regulär ist. Es gibt also Matrizen $T^{(\nu)} \in I\!M_{k+1,k+1}(\mathbf{Z}_q)$ für $\nu = 1, 2, \ldots$ mit $(r_0^\nu, \ldots, r_k^\nu) = (r_0^0, \ldots, r_k^0)T^{(\nu)}$. $T^{(\nu)}$ hängt nur von $a(\cdot, i), b(\cdot, i)$ für $i = 1, \ldots, \nu$ ab und ist stets regulär. Insbesondere ist damit $(r_0^\nu, \ldots, r_k^\nu)$ stets gleichmäßig in $\mathbf{Z}_q^{k+1}$ verteilt, weil dies für die Eingabe $(r_0, \ldots, r_k)$ gilt.

Die Gleichungen modulo q ergeben folgendes System für $\nu = 1, 2, \ldots$:

$$r_k^\nu = \sum_{i=1}^{l} 2^{i-1} r_{a(i,\nu)}^{\nu-1} \bmod q \tag{5}$$

$$r_{\nu \bmod k}^\nu = \sum_{i=1}^{l} 2^{i-1} r_{b(i,\nu)}^{\tilde{\nu}-1} \bmod q \qquad \text{mit } \tilde{\nu} = \begin{cases} \nu+1 & \text{für } b(i,\nu) = k \\ \nu & \text{sonst} \end{cases} \tag{6}$$

$$y_\nu = r_k^\nu + se_\nu \bmod q \tag{7}$$

$$r_j^\nu = r_j^{\nu-1} \qquad \text{für } j \neq (\nu \bmod k), k \tag{8}$$

Für alle $\mu \neq k$ und $\mu \leq \nu \leq \mu + k - 1$ identifizieren wir wegen (8) von nun an r_μ^ν mit r_μ^μ. Dafür lassen wir die Gleichungen (8) aus unserem System heraus. Außerdem unterdrücken wir in allen Gleichungen modulo q den Zusatz mod$q$.

2.2 Die Form der Attacken

Eine Attacke besteht aus Bedingungen an die Werte $a(i,\nu)$ und $b(i,\nu)$ derart, daß durch Erraten einiger dieser Werte die Bestimmung des geheimen Schlüssels s aus den obigen linearen Gleichungen mittels der Folge von Signaturen (e_ν, y_ν) mit $\nu = 1, 2, \ldots$ möglich ist.

Erfolgreiche Attacken sind mit beliebigen $k+2$ Signaturen möglich. Zur Bestimmung der $k+2$ Unbekannten $s, r_0^0, \ldots, r_k^0$ genügen nämlich $k+2$ Gleichungen in diesen Unbekannten. Wir beschränken uns auf Attacken, die nur die ersten $k+2$ Signaturen verwenden. Für die neue Grundform des Preprocessings erscheinen nämlich Attacken mittels $k+2$ aufeinanderfolgenden Signaturen die effizientesten Attacken zu sein. Dies werden wir in den Abschnitten 6,7 und 10 sehen.

Eine Attacke läuft darauf hinaus, aus den Gleichungen (5) und (6) für $\nu = 1, 2, \ldots$ eine lineare Gleichung in $r_k^1, r_k^2, \ldots$ zu erzeugen. Mit den Signaturen (y_ν, e_ν) und den Gleichungen (7) für $\nu = 1, 2, \ldots$ kann man dann s bestimmen.

2.3 Beispiele von Attacken

Es liegen folgende $3k+3$ Gleichungen in den $3k+3$ Unbekannten $r_1^1, r_2^1 \ldots, r_{k-1}^1, r_0^1$, $r_2^2, r_3^3 \ldots, r_{k-1}^{k-1}, r_0^k, r_1^{k+1}, r_k^1, r_k^2, \ldots, r_k^{k+2}, s$ vor:

Gleichung (5) für $\nu = 2, \ldots, k+2$
Gleichung (6) für $\nu = 2, \ldots, k+1$
Gleichung (7) für $\nu = 1, \ldots, k+2$

Damit läßt sich durch Raten der $a(i,\nu)$ für $\nu = 1, \ldots, k+1$ und $b(i,\nu)$ für $\nu = 1, \ldots, k$ dieses Gleichungssystem lösen und s ermitteln. Die Workload hierzu ist

$$\left(\frac{k!\,l}{(k+1-l)!}\right)^{2k+1} \overset{k=8,l=4}{\approx} 2^{187}$$

Hierbei werden die ersten $k+2$ Signaturen verwendet.

Wir suchen nun nach Attacken mit einer Workload $< 2^{72}$. ν ist stets die Rundennummer, d.h. es gilt $\nu \geq 1$.

Attacke 1: $l = 1$ Wegen $k \in \{a(1,\nu), \ldots, a(l,\nu)\}$ gilt stets $r_k^\nu = r_k^{\nu+1}$. Die Attacke hat die Workload 1.

Atttacke 2: Sei $l = 2$.
Bedingung: Für ein festes $\nu \geq 2$ gelte

$$\begin{array}{lrcl} 1. & \{k, \nu{-}1\} & = & \{a(1, \nu{-}1),\, a(2, \nu{-}1)\} \\ 2. & \{k, \nu{-}1\} & = & \{a(1, \nu),\, a(2, \nu)\} \end{array}$$

Dann gilt:

$$\begin{array}{rcccc} r_k^{\nu} & = & \alpha_1 r_k^{\nu-1} & + & \alpha_2 r_{\nu-1}^{\nu-1} \\ r_k^{\nu+1} & = & \beta_1 r_k^{\nu} & + & \beta_2 r_{\nu-1}^{\nu} \end{array}$$

mit $\{\alpha_1, \alpha_2\} = \{\beta_1, \beta_2\} = \{1, 2\}$. Dabei gilt $r_{\nu-1}^{\nu-1} = r_{\nu-1}^{\nu}$.

Aus den Gleichungen läßt sich nach Erraten der Koeffizienten eine Gleichung nur in $r_k^{\nu-1}, r_k^{\nu}, r_k^{\nu+1}$ bilden.

Die Workload besteht aus folgenden Faktoren:

1/Pr für 1. bzw. 2.: wegen $k \in \{a(1, \nu), a(2, \nu)\}$ jeweils	k
$\alpha_1, \ldots, \delta_2$ raten	2^2

Insgesamt gilt also:

$$\text{Workload } = k^2 \cdot 2^2 \overset{k=8}{=} 2^8$$

Attacke 3: Sei $l = 3$ und $k \geq 4$
Bedingung: Für festes $\nu \geq 2$ gelte

$$\begin{array}{lrcl} 1. & \{k, \nu{-}2, \nu{-}1\} & = & \{a(1, \nu), a(2, \nu), a(3, \nu)\} \\ 2. & \{k, \nu{-}2, \nu{-}1\} & = & \{a(1, \nu{+}1), a(2, \nu{+}1), a(3, \nu{+}1)\} \\ 3. & \{k, \nu{-}2, \nu{-}1\} & = & \{a(1, \nu{+}2), a(2, \nu{+}2), a(3, \nu{+}2)\} \end{array}$$

Dann folgt

$$\begin{array}{lrcccccc} 1. & r_k^{\nu} & = & \alpha_1 r_k^{\nu-1} & + & \alpha_2 r_{\nu-2}^{\nu-1} & + & \alpha_3 r_{\nu-1}^{\nu-1} \\ 2. & r_k^{\nu+1} & = & \beta_1 r_k^{\nu} & + & \beta_2 r_{\nu-2}^{\nu} & + & \beta_3 r_{\nu-1}^{\nu} \\ 3. & r_k^{\nu+2} & = & \gamma_1 r_k^{\nu+1} & + & \gamma_2 r_{\nu-2}^{\nu+1} & + & \gamma_3 r_{\nu-1}^{\nu+1} \end{array}$$

mit $\{\alpha_1, \alpha_2, \alpha_3\} = \ldots = \{\gamma_1, \gamma_2, \gamma_3\} = \{1, 2, 4\}$.

Wegen $k \geq 4$ gilt $r_{\nu-2}^{\nu-1} = r_{\nu-2}^{\nu} = r_{\nu-2}^{\nu+1}$ und
$r_{\nu-1}^{\nu-1} = r_{\nu-1}^{\nu} = r_{\nu-1}^{\nu+1}$.

Die Workload besteht aus folgenden Faktoren:

1/Pr für 1.–3.: jeweils	$\binom{k}{2}$
$\alpha_1, \ldots, \delta_3$ raten	$(3!)^3$

Insgesamt gilt also:

$$\text{Workload } = \binom{k}{2}^3 (3!)^3 \overset{k=8}{\approx} 2^{22}$$

Attacke 4: Sei $l = 4$ und $k \geq 6$
Bedingung: Für festes $\nu \geq 2$ gelte

$$\begin{array}{lll}
1. & \{k, \nu{-}3, \nu{-}2, \nu{-}1\} & = \{a(1,\nu), a(2,\nu), a(3,\nu), a(4,\nu)\} \\
2. & \{k, \nu{-}3, \nu{-}2, \nu{-}1\} & = \{a(1,\nu{+}1), a(2,\nu{+}1), a(3,\nu{+}1), a(4,\nu{+}1)\} \\
3. & \{k, \nu{-}3, \nu{-}2, \nu{-}1\} & = \{a(1,\nu{+}2), a(2,\nu{+}2), a(3,\nu{+}2), a(4,\nu{+}2)\} \\
4. & \{k, \nu{-}3, \nu{-}2, \nu{-}1\} & = \{a(1,\nu{+}3), a(2,\nu{+}3), a(3,\nu{+}3), a(4,\nu{+}3)\}
\end{array}$$

Dann folgt

$$\begin{array}{llllllllll}
1. & r_k^{\nu} & = & \alpha_1 r_k^{\nu-1} & + & \alpha_2 r_{\nu-3}^{\nu-1} & + & \alpha_3 r_{\nu-2}^{\nu-1} & + & \alpha_4 r_{\nu-1}^{\nu-1} \\
2. & r_k^{\nu+1} & = & \beta_1 r_k^{\nu} & + & \beta_2 r_{\nu-3}^{\nu} & + & \beta_3 r_{\nu-2}^{\nu} & + & \beta_4 r_{\nu-1}^{\nu} \\
3. & r_k^{\nu+2} & = & \gamma_1 r_k^{\nu+1} & + & \gamma_2 r_{\nu-3}^{\nu+1} & + & \gamma_3 r_{\nu-2}^{\nu+1} & + & \gamma_4 r_{\nu-1}^{\nu+1} \\
4. & r_k^{\nu+3} & = & \delta_1 r_k^{\nu+2} & + & \delta_2 r_{\nu-3}^{\nu+2} & + & \delta_3 r_{\nu-2}^{\nu+2} & + & \delta_4 r_{\nu-1}^{\nu+2}
\end{array}$$

mit $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \ldots = \{\delta_1, \delta_2, \delta_3, \delta_4\} = \{1, 2, 4, 8\}$.
Wegen $k \geq 6$ gilt $r_{\nu-3}^{\nu-1} = r_{\nu-3}^{\nu} = r_{\nu-3}^{\nu+1} = r_{\nu-3}^{\nu+2}$,
$r_{\nu-2}^{\nu-1} = r_{\nu-2}^{\nu} = r_{\nu-2}^{\nu+1} = r_{\nu-2}^{\nu+2}$ und
$r_{\nu-1}^{\nu-1} = r_{\nu-1}^{\nu} = r_{\nu-1}^{\nu+1} = r_{\nu-1}^{\nu+2}$.

Die Workload besteht aus folgenden Faktoren:

1/Pr für 1.–4.: jeweils	$\binom{k}{3}$
Koeffizienten raten	$(4!)^4$

Insgesamt gilt also:

$$\text{Workload } = \binom{k}{3}^4 (4!)^4 \overset{k=8}{\approx} 2^{42}$$

Die Attacke kann durch das Stellen einer der folgenden Bedingungen verhindert werden:

a) $\nu \in \{a(i,\nu)\}$
b) $\nu - 1 \in \{a(i,\nu)\}$

Attacke 5:
Bedingung: Für festes ν gelte $a(i,\nu) = a(i,\nu+1) \neq \nu$ für alle i.
Durchführung: Rate j mit $a(j,\nu) = a(j,\nu+1) = k$. Dann ergibt sich

$$\begin{aligned} r_k^{\nu+1} &= \sum_{i=1}^{l} 2^{i-1} r^{\nu}_{a(i,\nu+1)} \\ &= 2^{j-1} r_k^{\nu} + \sum_{i \neq j} 2^{i-1} r^{\nu-1}_{a(i,\nu+1)} \\ &= 2^{j-1}(r_k^{\nu} - r_k^{\nu-1}) + r_k^{\nu} \end{aligned}$$

Diese Attacke läßt sich verallgemeinern:

Sei $\mathbf{t < k - l}$:
Bedingung: Für festes ν gelte

$$a(i,\nu) = a(i,\nu+t) \neq \nu, \nu+1, \ldots, \nu+t-1 \text{ für alle } i \tag{9}$$

Rate j mit $a(j,\nu) = a(j,\nu+t) = k$. Im obigen Spezialfall galt $t = 1$. Analog dazu ergibt sich

$$r_k^{\nu+t} = 2^{j-1}(r_k^{\nu+t-1} - r_k^{\nu-1}) + r_k^{\nu}$$

Die Workload berechnet man folgendermaßen:

1. $\#\{(a(\cdot,\nu), a(\cdot,\nu+t)) \,|\, a(i,\nu) = a(i,\nu+t) \neq \nu, \nu+1, \ldots, \nu+t-1\}$ $= \frac{l(k-t)!}{(k-t+1-l)!}$
2. $\#\{(a(\cdot,\nu), a(\cdot,\nu+t))\}$ $= \left(\frac{l\,k!}{(k+1-l)!}\right)^2$
3. j raten l

Insgesamt gilt also für $k = 8$ und $l = 4$:

$$\text{Workload } = \left(\frac{l\,k!}{(k+1-l)!}\right)^2 \left(\frac{(k+t+1-l)!}{l(k-t)!}\right) l \quad \begin{array}{ll} \overset{t=1}{\approx} & 2^{13} \\ \overset{t=2}{\approx} & 2^{14} \\ \overset{t=3}{\approx} & 2^{15} \end{array}$$

Die Attacke kann verhindert werden, durch das Stellen einer der folgenden Bedingungen:

a) $\nu \in \{a(i,\nu)\}$ ((9) wird unmöglich)
b) $\nu - 1 \in \{a(i,\nu)\}$ ((9) wird unmöglich)

Bemerkung: Die Attacke könnte auch durch

$$\text{e) } r_k^\nu = r_\nu^{\nu-1} + \sum_{i=1}^{l} 2^{i-1} r_{a(i,\nu)}^{\nu-1}$$

abgewehrt werden. Dadurch kommt für jedes ν ein anderer Summand in der Gleichung für r_k^ν hinzu, so daß sich r_k^ν und r_k^μ für $\nu \neq \mu$ in zumindest diesem Summanden unterscheiden.

Notation zu den Attacken 6.1 und 6.2: Für $j = 1, \ldots, l$ sei

$$\nu_j = \begin{cases} a(j, k+1) & \text{für } a(j, k+1) \neq 0 \\ k & \text{für } a(j, k+1) = 0 \end{cases}$$

Attacke 6.1: **Bedingung:** Für festes ν gelte $\nu_1 = k$, $\nu_2, \ldots, \nu_l \geq 2$ und $a(i, \nu_j) = b(i, \nu_j)$ für $j \geq 2$ und alle i.
Durchführung: Rate $\nu_2, \ldots, \nu_l$ und $i_2, \ldots, i_l$ mit
$a(i_j + 1, \nu_j) = k$. Dann ergibt sich für alle $j \geq 2$

$$\begin{aligned} r_k^{\nu_j} &= \sum_{i=1}^{l} 2^{i-1} r_{a(i,\nu_j)}^{\nu_j - 1} \\ &= \sum_{i=1}^{l} 2^{i-1} r_{b(i,\nu_j)}^{\nu_j - 1} \\ &= r_{\nu_j \bmod k}^{\nu_j} + 2^{i_j} \left(r_k^{\nu_j - 1} - r_k^{\nu_j} \right) \\ &= r_{\nu_j \bmod k}^{k} + 2^{i_j} \left(r_k^{\nu_j - 1} - r_k^{\nu_j} \right) \end{aligned}$$

Damit erhalten wir

$$\begin{aligned} r_k^{k+1} &= \sum_{j=1}^{l} 2^{j-1} r_{\nu_j \bmod k}^{k} \\ &= \sum_{j=2}^{l} 2^{j-1} \left(r_k^{\nu_j} - 2^{i_j} \left(r_k^{\nu_j - 1} - r_k^{\nu_j} \right) \right) + r_k^k \end{aligned}$$

Man beachte, daß man anstatt $\nu_1 = k$ auch $\nu_{j_0} = k$ für $j_0 \neq 1$ annehmen kann. Dabei erhält man einen zusätzlichen Faktor $2^{j_0 - 1}$ vor r_k^k. Die Workload erhöht sich durch diese Annahme nicht.

Die Workload besteht aus folgenden Faktoren:

1. $a(i,\nu_j) = b(i,\nu_j)$ für $j \geq 2$ und alle i	$\left(\frac{(k+1)!}{(k+1-l)!}\right)^l$
2. $\nu_1 = k$ 3. $\nu_2, \ldots, \nu_l$ raten	$\frac{(k+1)!}{(k+1-l)!}$
4. $i_2, \ldots, i_l$ raten	l^{l-1}

Insgesamt gilt also für $k = 8$ und $l = 4$:

$$\text{Workload } = l^{\,l-1} \left(\frac{(k+1)!}{(k+1-l)!}\right)^{l-1} \frac{(k+1)!}{(k+1-l)!} \approx 2^{52}$$

Bemerkung: Forderte man die Bedingung a), so verringerte sich die Workload auf

$$l^{\,l-1} \left(\frac{l\,k!}{(k+1-l)!}\right)^{l-1} \frac{l\,k!}{(k+1-l)!} \overset{k=8,l=4}{\approx} 2^{48}$$

Die Attacke kann abgewehrt werden durch Stellen einer der zusätzlichen Bedingungen

c) $k \notin \{b(i,\nu)\}$
d) $\forall j: \quad (a(1,\nu), \ldots, (a(l,\nu)) \neq (b(1,\nu), \ldots, b(l,\nu))$

Durch jede dieser Bedingungen wird wieder eine Unsymmetrie zwischen den Gleichungen für r_k^ν und r_ν^ν erzeugt. Durch diese wird die Attacke verhindert.

Im folgenden fordern wir stets a), b) und c), weil erstens dadurch alle Attacken, die wir bisher vorgestellt haben, abgewehrt werden und zweitens damit — wie wir in Abschnitt 4 sehen werden — allgemein starke Einschränkungen für Attacken gelten.

Attacke 6.2:

Bedingungen: Für festes ν gelte

1. $\underset{j=1,\ldots,l-1}{\forall}$ $a(j,k+1) = j-1$ und $a(l,k+1) = k$

2. $\underset{j=1,\ldots,l-1}{\forall}$ $a(1,\nu_j) = k$

3. $\underset{j=1,\ldots,l-1}{\forall}$ $a(i,\nu_j) = b(i,\nu_j)$

4. $\underset{j=1,\ldots,l-1}{\forall}$ $a(j,k) = b(1,\nu_j)$

5. $\underset{j=1,\ldots,l-1}{\forall}$ $b(1,\nu_j) = \nu_j - 1$

Dann sind a), b) und c) erfüllt und es gilt für alle $j \neq l$

$$r^{\nu_j-1}_{b(1,\nu_j)} \overset{5.}{=} r^{\nu_j-1}_{\nu_j-1} \overset{\nu_j\leq k}{=} r^{k-1}_{\nu_j-1} \overset{5.}{=} r^{k-1}_{b(1,\nu_j)} \overset{4.}{=} r^{k-1}_{a(j,k)} \tag{10}$$

Wegen $1 \leq \nu_j \leq k$ gilt $r^k_{\nu_j \bmod k} = r^{\nu_j}_{\nu_j \bmod k}$ und damit folgt für $j = 1, \ldots, l-1$

$$\begin{aligned} r^k_{\nu_j \bmod k} &= \sum_{i=1}^{l} 2^{i-1} r^{\nu_j-1}_{b(i,\nu_j)} \\ &\overset{3.}{=} r^{\nu_j-1}_{b(1,\nu_j)} + \sum_{i=2}^{l} 2^{i-1} r^{\nu_j-1}_{a(i,\nu_j)} \\ &\overset{(10),2.}{=} r^{k-1}_{a(j,k)} + r^{\nu_j}_k - r^{\nu_j-1}_k \end{aligned} \tag{11}$$

Aus 4. und c) ergibt sich $a(j,k) \neq k$ für alle $j = 1, \ldots, l-1$ und mit a) folgt

$$a(l,k) = k \tag{12}$$

Schließlich erhalten wir

$$\begin{aligned} r^{k+1}_k &= \sum_{j=1}^{l} 2^{j-1} r^k_{\nu_j \bmod k} \\ &\overset{1.}{=} 2^{l-1} r^k_k + \sum_{j=1}^{l-1} 2^{j-1} r^k_{\nu_j \bmod k} \\ &\overset{(11)}{=} 2^{l-1} r^k_k + \sum_{j=1}^{l-1} 2^{j-1} \left(r^{\nu_j \bmod k}_k - r^{\nu_j-1}_k \right) + \sum_{j=1}^{l-1} 2^{j-1} r^{k-1}_{a(j,k)} \end{aligned}$$

$$\stackrel{(12)}{=} \quad 2^{l-1} r_k^k + \sum_{j=1}^{l-1} 2^{j-1} \left(r_k^{\nu_j} - r_k^{\nu_j - 1} \right) + r_k^k - 2^{l-1} r_k^{k-1}$$

also wieder eine Gleichung nur in den r_k's.

Die Workload besteht aus folgenden Faktoren:

1. $\dfrac{l!\,(k-2)!}{(l-3)!\,(k+1-l)!}$
2. $l^{\,l-1}$
3. $\wedge$ 5. $\left(\dfrac{(k-1)!}{(k-l)!} \right)^{l-1}$
4. $\dfrac{l!\,(k-2)!}{(l-3)!\,(k+1-l)!}$, denn dadurch sind wegen (12) alle $a(j, k-1)$ festgelegt

Insgesamt gilt also für $k = 8$ und $l = 4$:

$$\text{Workload} \approx 2^{44}$$

2.4 Die 2er–Potenz–Form des Preprocessings

Um die Workload der Attacke 6.2 zu erhöhen, führen wir zusätzliche Zufallswerte $f(i,\nu)$ und $g(i,\nu)$ ein. Es seien $l_1, l_0 \leq k$ und $d_1, d_0 \geq 2$. Dabei sind k, l_1, l_0, d_1, d_0 Sicherheitsparameter. Für die 2er–Potenz–Form des Preprocessings gelte:

$$r_k^\nu = \sum_{i=1}^{l_1} 2^{f(i,\nu)} r_{a(i,\nu)}^{\nu-1} \qquad \text{mit } f(i,\nu) \text{ zufällig aus } \{0, \ldots, d_1 - 1\}$$

$$r_{\nu \bmod k}^\nu = \sum_{i=1}^{l_0} 2^{g(i,\nu)} r_{b(i,\nu)}^{\nu-1} \qquad \text{mit } g(i,\nu) \text{ zufällig aus } \{0, \ldots, d_0 - 1\}$$

Die Werte $a(1,\nu), \ldots, a(l_1,\nu)$ sind verschieden aus $\{0, \ldots, k\}$ mit
1. $\nu \in \{a(i,\nu)\}$
2. $\nu - 1 \in \{a(i,\nu)\}$
3. $k \in \{a(i,\nu)\}$.

Die Werte $b(1,\nu), \ldots, b(l_0,\nu)$ sind verschieden aus $\{0, \ldots, k\}$ mit
4. $\nu \in \{b(i,\nu)\}$
5. $k \notin \{b(i,\nu)\}$

Dabei sind die Exponenten $f(i,\nu)$ und $g(i,\nu)$ der Koeffizienten unabhängig von den Werten $a(i,\nu)$ und $b(i,\nu)$ gewählt. Die Gleichungen sind also noch mehr randomisiert und ein Angreifer muß noch mehr Werte erraten.
Diese neue Form des Preprocessings nennen wir die **2er–Potenz–Form**.

2.4.1 Eine Attacke gegen die 2er–Potenz–Form des Preprocessings

Für $l_0 \neq l_1$ ist die Attacke 6.2 offensichtlich nicht mehr durchführbar. Wir zeigen nun, daß sie auch für $l_0 = l_1 = l$ nicht mehr effektiv ist. Zuerst muß die Attacke dem neuen Preprocessing angepaßt werden. Sei $S_{\{1,\ldots,n\}}$ die Menge der Permutationen der Menge $\{1,\ldots,n\}$. Durch die freie Wahl der Exponenten in der Gleichung für $r_k^{\nu+1}$ können verschiedene Tupel $(a(1,\nu),\ldots,a(l,\nu),$
$f(1,\nu),\ldots,f(l,\nu))$ und $(a'(1,\nu),\ldots,a'(l,\nu),f'(1,\nu),\ldots,f'(l,\nu))$ dieselbe Gleichung für $r_k^{\nu+1}$ erzeugen. Das ist genau dann der Fall, wenn es eine Permutation $\sigma \in S_{\{1,\ldots,l\}}$ der gibt, so daß $a(i,\nu) = a'(\sigma(i),\nu)$ und $f(i,\nu) = f'(\sigma(i),\nu)$ für alle $i = 1,\ldots,l$ gilt. Analoges gilt für die Tupel $(b(1,\nu),\ldots,b(l,\nu),g(1,\nu),\ldots,$
$g(l,\nu))$ und die Gleichung für $r_{\nu \bmod k}^{\nu+1}$. Die Attacke kann dann folgendermaßen formuliert werden:

Bedingungen:

1. $j = 1,\ldots,l-1:$ $\quad a(j,k+1) = j-1$ und $a(l,k+1) = k$

2. $j = 1,\ldots,l-1:$ $\quad a(1,\nu_j) = k \quad \wedge \quad f(1,\nu_j) = 0 \quad \wedge \quad f(j,k+1) = 0$

3. $j = 1,\ldots,l-1:$ $\quad \underset{\pi \in S_{\{2,\ldots,l\}}}{\exists} f(i,\nu_j) = g(\pi(i),\nu_j) \quad \wedge \quad a(i,\nu_j) = b(\pi(i),\nu_j)$

4. $j = 1,\ldots,l-1:$ $\quad \underset{\sigma \in S_{\{2,\ldots,l\}}}{\exists} b(1,\nu_j) = a(\sigma(j),k) \quad \wedge \quad g(1,\nu_j) = f(\sigma(j),k)$

5′. $j = 1,\ldots,l-1:$ $\quad b(1,\nu_j) = \nu_j - 1$

Damit ergibt sich bei $k = 8$ und $l = 4$ eine Workload von $\approx 2^{79}$ (ohne Beweis).

2.4.2 Schlußfolgerung

Bei allen bisher behandelten Attacken führt die 2er–Potenz–Form des Preprocessings zu höheren Workloads. Umgekehrt ist uns keine Attacke gegen die 2er–Potenz–Form bekannt, die gegen die alte Form des Preprocessings weniger erfolgreich wäre. Von nun an gehen wir von der 2er–Potenz–Form aus.

Bemerkung: Die 2er–Potenz–Form des Preprocessings erfordert $l_0 + l_1 + \max_\nu(f(i,\nu)) + \max_\nu(g(i,\nu)) - 2$ Multiplikationen pro Signatur, also im Fall $d_0 = d_1 = l_0 = l_1$ höchstens so viele wie die alte Form bei den entsprechenden Parametern.

Wie man am Beispiel der letzten Attacke sehen kann, ist unsere bisherige Notation für die 2er–Potenz–Form nicht sehr günstig. So führt z.B. eine Permutation der $a(i,\nu)$ und der zugehörigen $f(i,\nu)$ zu den selben Gleichungen. Das ist ein Grund, warum wir im nächsten Kapitel zur Matrixdarstellung der Gleichungen übergehen werden. Außerdem ermöglicht es uns die Matrixdarstellung, das Preprocessing mit den Methoden der linearen Algebra auf Attacken zu untersuchen.

3 Die 2er–Potenzform des Preprocessings

3.1 Der neue Preprocessing–Algorithmus

Wir gehen nun von der allgemeinen Form des Preprocessings aus, wie wir sie in Abschnit 2.3 vorgestellt haben.

k, l_0, l_1, d_0, d_1 sind Sicherheitsparameter

Initiation: lade $k+1$ Paare $(r_1^0, x_1^0) \ldots, (r_k^0, x_k^0)$ mit $x_i^0 = \alpha^{r_i^0} \bmod p$.
$\nu := 1$. ν ist die Rundennummer

1. wähle l_1 verschiedene Zufallszahlen $a(1,\nu), \ldots, a(l_1,\nu) \in \{0, \ldots, k\}$ mit $k, \nu, \nu-1 \in \{a(1,\nu), \ldots, a(l_1,\nu)\}$ und l_1 verschiedene Zufallszahlen $f(1,\nu), \ldots, f(l_1,\nu) \in \{0, \ldots, d_1 - 1\}$

$$r_k^\nu := \sum_{i=1}^{l_1} 2^{f(i,\nu)} r_{a(i,\nu)}^{\nu-1} \bmod q \qquad x_k = \prod_{i=1}^{l_1} (x_{a(i,\nu)}^{\nu-1})^{2^{f(i,\nu)}} \bmod p$$

2. wähle l_0 verschiedene Zufallszahlen $b(1,\nu), \ldots, b(l_0,\nu) \in \{0, \ldots, k\}$ mit $\nu \in \{b(1,\nu), \ldots b(l_0,\nu)\}$ und $k \notin \{b(1,\nu), \ldots b(l_0,\nu)\}$ und l_0 verschiedene Zufallszahlen $g(1,\nu), \ldots, g(l_0,\nu) \in \{0, \ldots, d_0 - 1\}$

$$r_{\nu \bmod k}^\nu := \sum_{i=1}^{l_0} 2^{g(i,\nu)} r_{b(i,\nu)}^{\nu-1} \bmod q \qquad x_{\nu \bmod k}^\nu = \prod_{i=1}^{l_0} (x_{b(i,\nu)}^{\nu-1})^{2^{g(i,\nu)}} \bmod p$$

3. verwende (r_k^ν, x_k^ν) für die ν–te Signatur (e_ν, y_ν) gemäß
$y_\nu = r_k^\nu + s e_\nu \bmod q$

4. $\nu := \nu + 1$
GOTO 1.

Die Zufallszahlen $a(1,\nu), \ldots, a(l,\nu)$, $b(1,\nu), \ldots, b(l,\nu)$, $f(1,\nu), \ldots, f(l,\nu)$ und $g(1,\nu), \ldots, g(l,\nu)$ werden unabhängig gewählt, so daß die angegebenen Bedingungen erfüllt sind. Das Verfahren benötigt höchstens $l_0 + l_1 + d_0 + d_1 - 4$ viele Multiplikationen pro Runde

3.2 Die Matrixdarstellung

Wir stellen die linearen Gleichungen des Preprocessings als Matrix dar. Diese Matrixdarstellung ermöglicht es uns, Attacken mit den Methoden der linearen

Algebra zu behandeln.

Bemerkung: Eine Attacke läuft daraus hinaus, aus den Gleichungen

$$r_k^\nu := \sum_{i=1}^{l_1} 2^{f(i,\nu)} r_{a(i,\nu)}^{\nu-1} \bmod q \tag{13}$$

$$r_{\nu \bmod k}^\nu := \sum_{i=1}^{l_0} 2^{g(i,\nu)} r_{b(i,\nu)}^{\nu-1} \bmod q \tag{14}$$

für $\nu = 1, 2, \ldots$ eine lineare Gleichung in $r_k^1, r_k^2, \ldots$ zu erzeugen. Mit den Gleichungen

$$y_\nu = r_k^\nu + s e_\nu \bmod q \tag{15}$$

für $\nu = 1, 2, \ldots$ und den Signaturen (y_ν, e_ν) läßt sich dann s bestimmen. In der Gleichung (13) für $\nu = 1$ kommt jedoch auch der Wert r_k^0 vor, der sich nicht mit einer Gleichung (15) eliminieren läßt. Wir betrachten deshalb nur Attacken, die die Gleichungen (13) und (14) für $\nu = 2, 3, \ldots$ verwenden. Aus den in Abschnitt 2.2 aufgeführten Gründen betrachten wir auch nur Attacken, die $k+2$ aufeinanderfolgende Signaturen verwenden. O.B.d.A. untersuchen wir also nur Attacken mittels der folgenden Gleichungen:

Gleichung (13) für $\nu = 2, \ldots, k+2$
Gleichung (14) für $\nu = 2, \ldots, k+1$
Gleichung (15) für $\nu = 1, \ldots, k+2$

Die Gleichungen (13) für $\nu = 2, \ldots, k+2$ und (14) für $\nu = 2, \ldots, k+1$ definieren unter Berücksichtigung der Identität $r_j^\nu = r_j^{\nu-1}$ für $j \neq k, \nu \bmod k$ eine $(2k+1) \times (3k+2)$ Matrix A. Hierbei stellt die Zeile $2i-1$ die Gleichung für r_k^{i+1} und die Zeile $2i$ die Gleichung für $r_{i+1 \bmod k}^{i+1}$ dar und die Spalte j repräsentiert

$$\begin{array}{ll} r_{j+1 \bmod k}^1 & \text{für} \quad 1 \leq j \leq k \\ r_{j-k+1 \bmod k}^{j-k+1} & \text{für} \quad k+1 \leq j \leq 2k \\ r_k^{j-2k} & \text{für} \quad 2k+1 \leq j \leq 3k+2 \end{array}$$

Diese Matrix A ist durch die Menge der Werte $a(i,\nu), f(i,\nu), b(j,\eta)$ und $g(j,\eta)$ für $i = 1, \ldots, l_1$, $j = 1, \ldots, l_0$, $\nu = 2, \ldots, k+2$ und $\eta = 2, \ldots, k+1$ eindeutig festgelegt.

Definition 3.1 *Der* **Ereignisraum** $\mathcal{P}$ *sei die Menge aller Tupel* $(a(i,\nu), b(j,\eta), f(i,\nu), g(j,\eta)) \mid 2 \leq \nu \leq k+2\ ,\ 2 \leq \eta \leq k+1\ ,\ 1 \leq i \leq l_1\ ,\ 1 \leq j \leq l_0\ )$, die

den folgenden Bedingungen genügen:

$$
\begin{array}{l}
a(i,\nu), b(j,\eta) \in \{0,\ldots,k\} \\
\\
\begin{array}{rcl}
i_1 \neq i_2 & \Rightarrow & a(i_1,\nu) \neq a(i_1,\nu) \\
j_1 \neq j_2 & \Rightarrow & b(j_1,\eta) \neq b(j_2,\eta)
\end{array} \\
\\
\begin{array}{rrcl}
1. & \nu & \in & \{a(i,\nu) \mid 1 \leq i \leq l_1\} \\
2. & \nu-1 & \in & \{a(i,\nu) \mid 1 \leq i \leq l_1\} \\
3. & k & \in & \{a(i,\nu) \mid 1 \leq i \leq l_1\} \\
4. & \nu & \in & \{b(j,\eta) \mid 1 \leq j \leq l_0\} \\
5. & k & \notin & \{b(j,\eta) \mid 1 \leq j \leq l_0\}
\end{array} \\
\\
f(i,\nu) \in \{0,\ldots,d_1-1\} \\
g(j,\eta) \in \{0,\ldots,d_0-1\}
\end{array}
$$

Jedes $x \in \mathcal{P}$ definiert eine Matrix A_x in der oben beschriebenen Weise. Es bezeichne $\mathcal{T}$ die Menge dieser Matrizen.

$$\mathcal{T} := \{A_x | x \in \mathcal{P}\}$$

Die Einträge einer Matrix $A \in \mathcal{P}$ sind die Koeffizienten α_μ^ν und β_μ^ν der Gleichungen

$$r_k^\nu = \sum_\mu \alpha_\mu^\nu \, r_\mu^{\nu-1}$$

für $\nu = 2,\ldots,k+2$ und

$$r_{\eta \bmod k}^\eta = \sum_\mu \beta_\mu^\eta \, r_\mu^{\eta-1}$$

für $\eta = 2,\ldots,k+1$. Wir setzen also für $2 \leq \nu \leq k+2,\ 2 \leq \eta \leq k+1,\ 0 \leq \mu \leq k$

$$
\begin{array}{ll}
\alpha_\mu^\nu = 2^n :\Leftrightarrow & \exists\limits_i \ a(i,\nu) = \mu \ \wedge \ f(i,\nu) = n \\
\beta_\mu^\eta = 2^n :\Leftrightarrow & \exists\limits_j \ b(j,\eta) = \mu \ \wedge \ g(j,\eta) = n \\
\alpha_\mu^\nu, \beta_\mu^\eta = 0 & \text{sonst}
\end{array}
$$

Die Bedingungen $1. - 5.$ lassen sich nun wie folgt schreiben:

$$
\left.
\begin{array}{rlcl}
1. & \alpha_{\nu \bmod k}^\nu & \neq & 0 \\
2. & \alpha_{\nu-1 \bmod k}^\nu & \neq & 0 \\
3. & \alpha_k^\nu & \neq & 0 \\
4. & \beta_{\eta \bmod k}^\eta & \neq & 0 \\
5. & \beta_k^\eta & = & 0
\end{array}
\right\} \quad \nu = 2,\ldots,k+2 \quad \eta = 2,\ldots,k+1
$$

Die ungeraden Zeilen nennen wir α–Zeilen, die geraden β–Zeilen.

Das Ziel eines Angreifers ist es, aus den Gleichungen der Matrix $A \in \mathcal{P}$ eine Gleichung in den Werten $r_k^1, \ldots, r_k^{k+2}$ zu bilden. Eine solche Gleichung erhält man stets mit einer Linearkombination von Zeilen aus A, die in den ersten $2k$ Spalten jeweils 0 ergibt. Deshalb betrachten wir die Untermatrix Z der ersten 2k Spalten von A.

Definition 3.2 *Sei $\kappa : \mathbb{M}_{2k+1,3k+2} \to \mathbb{M}_{2k+1,2k}$ die Einschränkung auf die ersten $2k$ Spalten. Dann definieren wir:*

$$\mathcal{R} := \kappa(\mathcal{T})$$

Figur 1 zeigt eine Matrix $A \in \mathcal{T}$ und die Untermatrix $Z \in \mathcal{R}$.

3.2.1 Die induzierten Gleichverteilungen

Die Gleichverteilung auf $\mathcal{P}$ induziert mittels der Konstruktion der Matrix A_x aus einem $x \in \mathcal{P}$ eine Gleichverteilung auf $\mathcal{T}$:

Satz 3.1 *Die Konstruktion der Matrix A definiert eine Surjektion* $\psi : \mathcal{P} \twoheadrightarrow \mathcal{T}$ *mit* $\underset{A\in\mathcal{T}}{\forall} \; |\psi^{-1}(A)| = (l_1!)^{k+1}(l_0!)^k$

Beweis: Eine Matrix $A \in \mathcal{T}$ wird von einem Element $x \in \mathcal{P}$ über die α_μ^ν's und die β_μ^η's definiert. Diese sind für verschiedene ν bzw. η unabhängig voneinander. Für festes ν sind die von zwei Elementen x und $\tilde{x}$ aus $\mathcal{P}$ definierten α_μ^ν und $\tilde{\alpha}_\mu^\nu$ genau dann identisch, wenn die Werte $a(j,\mu)$ und $\tilde{a}(j,\mu)$ durch dieselbe Permutation der j auseinander hervorgehen, wie die Werte $f(j,\nu)$ aus den $\tilde{f}(j,\nu)$'s, d.h. wenn

$$\underset{\sigma\in S_{\{1,\ldots,l_1\}}}{\exists} \quad \underset{i}{\forall} \; a(i,\nu) = \tilde{a}(\sigma(i),\nu) \quad \wedge \quad f(i,\nu) = \tilde{f}(\sigma(i),\nu)$$

gilt. Dafür gibt es $(l_1!)$ viele Möglichkeiten. Für die β–Zeilen gilt analoges mit den Werten $b(j,\eta)$ und $g(j,\eta)$. Hierfür gibt es jeweils $(l_0!)$ viele Möglichkeiten. Da es $k+1$ viele α–Zeilen und k viele β–Zeilen gibt ergibt sich insgesamt eine Anzahl von $(l_0!)^k(l_1!)^{k+1}$ vielen $x \in \mathcal{P}$, die dieselbe Matrix A erzeugen.
Die Surjektivität von ψ ist klar. Damit ist der Satz bewiesen.□

Mittels der Abbildung κ definiert jedes $x \in \mathcal{P}$ eine Matrix $Z_x \in \mathcal{R}$. Damit induziert die Gleichverteilung auf $\mathcal{P}$ eine Gleichverteilung auf $\mathcal{R}$.

Korrolar 3.2 *Die Konstruktion der Matrix Z aus einem Tupel* $x \in \mathcal{P}$ *definiert eine Surjektion* $\varphi : \mathcal{P} \twoheadrightarrow \mathcal{R}$ *mit*

$$\underset{Z\in\mathcal{R}}{\forall} \quad \left|\varphi^{-1}(Z)\right| = (l_1!)^{k+1}(l_0!)^k(d_1)^{k+1}$$

Beweis: φ setzt sich folgendermaßen zusammen: $\varphi : \mathcal{P} \overset{\psi}{\to} \mathcal{T} \overset{\kappa}{\to} \mathcal{R}$
Zwei Matrizen aus $\kappa^{-1}(Z) \cap \mathcal{T}$ unterscheiden sich nur durch die $k+1$ Werte α_k^i. Daher gilt für alle $Z \in \mathcal{R}$:

$$|\kappa^{-1}(Z) \cap \mathcal{T}| = (d_1)^{k+1}$$

Daraus folgt

$$|\varphi^{-1}(Z)| = |\psi^{-1}(\kappa^{-1}(Z) \cap \mathcal{T})| = (l_0!)^k(l_1!)^{k+1}(d_1)^{k+1}\square$$

Jedes $x \in \mathcal{P}$ definiert eine Matrix $A_x \in \mathcal{T}$. In den Spalten $2k+1$ bis $3k+1$ von A_x stehen die Einträge $\alpha_k^2, \ldots, \alpha_k^{k+2}$. Dadurch wird eine Abbildung χ von $\mathcal{P}$ auf die Menge der Tupel $(\alpha_k^2, \ldots, \alpha_k^{k+2}) \in \{2^0, \ldots, 2^{d_1-1}\}^{k+1}$ definiert.

Die Gleichverteilung auf $\mathcal{P}$ induziert auch eine Gleichverteilung auf der Menge der Tupel $(\alpha_k^1, \ldots, \alpha_k^{k+1})$.

Korrolar 3.3 *Die Konstruktion der Matrix A definiert eine Surjektion* $\chi : \mathcal{P} \twoheadrightarrow \{2^0, \ldots, 2^{d_1-1}\}^{k+1}$. *so daß* $|\chi^{-1}(\alpha)|$ *konstant ist für alle* $\alpha = (\alpha_k^2, \ldots, \alpha_k^{k+2}) \in \{2^0, \ldots, 2^{d_1-1}\}^{k+1}$

Beweis: Sei $A \to \bar{A}$ die Einschränkung auf die letzten k+2 Spalten und sei

$$\underset{A,B \in \mathcal{T}}{\forall} \quad A \sim B :\Leftrightarrow \bar{A} = \bar{B}$$

Dann gilt

$$\mathcal{T}/\sim \;\simeq\; \{2^0, \ldots, 2^{d_1-1}\}^{k+1}$$

denn in den Spalten $2k+1$ bis $3k+2$ hängen nur die Einträge $\alpha_k^2, \ldots, \alpha_k^{k+2}$ von $x \in \mathcal{P}$ ab. Außerdem gilt

$$\underset{A \in \mathcal{T}}{\forall} \quad \#\{B | A \sim B\} = |\mathcal{R}|$$

Setze $\chi : \mathcal{P} \overset{\psi}{\twoheadrightarrow} \mathcal{T} \twoheadrightarrow \mathcal{T}/\sim$ $\qquad \square$

3.3 Die Effizienz von Attacken

Ziel eines Angreifers ist es, einen Koeffizientenvektor $(c_i \,|\, 1 \leq i \leq k+2)$ zu finden, so daß $Pr\left[\sum_i c_i r_k^i = 0\right]$ möglichst groß ist. Die Wahrscheinlichkeit bezieht sich auf die Gleichverteilung über $\mathcal{P}$. Die Verteilung der Eingaben $(r_0^0, \ldots r_k^0)$ bleibt unberücksichtigt. Gilt die Gleichung $\sum_i c_i r_k^i = 0$ nämlich nur für einen Teil der Eingaben, so ist diese Wahrscheinlichkeit gleich $1/q$. Weil q mit $q > 2^{140}$ sehr groß ist, lassen wir so kleine Wahrscheinlichkeiten unberücksichtigt und betrachten für Attacken nur Gleichungen $\sum_i c_i r_k^i = 0$, die für alle Eingaben gelten, und schreiben $\sum_i c_i r_k^i \equiv 0$.

Definition 3.3 *Eine* **Attacke** *ist ein Tupel* $\vec{c} \in \mathbf{Z}^{k+2}$. *Die* **Effizienz** $\mathcal{E}(\vec{c})$ *einer Attacke* $\vec{c}$ *ist* $\underset{\mathcal{P}}{Pr}\left[\sum_{i=1}^{k+2} c_i r_k^i \equiv 0\right]$. *Eine Attacke* $\vec{c}$ *heißt* **erfolgreich**, *wenn* $\mathcal{E}(\vec{c}) > 0$ *ist.*

Aus einer Attacke ergibt sich stets ein Angriff auf das Preprocessing:

Bedingung: $\sum c_i r_k^i \equiv 0 \bmod q$

Durchführung: Ermittle s mit den Gleichungen (15) und den Signaturen (y_ν, e_ν).

Definition 3.4 *Die Workload einer Attacke $\vec{c}$ ist $Pr[\sum c_i r_k^i \equiv 0]^{-1}$.*

Die Workload einer Attacke entspricht der zu erwartenden Anzahl von Versuchen zur Erzeugung von $k+2$ aufeinanderfolgende Signaturen $r_k^{e+1}, \ldots, r_k^{e+k+2}$, welche die Gleichung $\sum_{i=1}^{k+2} c_i r_k^{e+i} \equiv 0 \bmod q$ erfüllen.

3.3.1 Das Erzeugen einer Attacke

Sei A_i die i–te Zeile der Matrix A. Die Werte r_k^i mit $1 \le i \le k+2$ hängen allein durch die Gleichungen miteinander zusammen, die die Matrix A repräsentiert. Deshalb gilt für eine Attacke $\vec{c}$

$$\sum_{i=1}^{k+2} c_i r_k^i \equiv 0 \tag{16}$$

genau dann, wenn es ein Tupel $\vec{y} \in \mathbf{Z}^{2k+1}$ gibt, sodaß für $\vec{y} \cdot A = \sum_{i=1}^{2k+1} y_i A_i$ gilt

$$(\vec{y} \cdot A)_j = \begin{cases} 0 & \text{für } 1 \le j \le 2k \\ c_{j-2k} & \text{für } 2k+1 \le j \le 3k+2 \end{cases} \tag{17}$$

Die Spalten $2k+1$ bis $3k+2$ der Matrix A entsprechen nämlich den Werten r_k^1 bis r_k^{k+2}.

Definition 3.5 *Sei $1 \le n \le k+2$. Ein* **n–Zeilen–Versuch** *ist ein Tupel $\vec{y} \in \mathbf{Z}^{2k+1}$ mit $\#\{i \,|y_i \neq 0\} = n$. Wir sagen ein n–Zeilen–Versuch $\vec{y}$* **erzeugt** *eine Attacke $\vec{c}$, wenn es eine Matrix $A \in \mathcal{T}$ gibt, so daß Gleichung (17) gilt.*

Für $A \in \mathcal{T}$, $Z = \kappa(A)$ und einen Versuch $\vec{y}$ gilt:

$$\forall_{j \le 2k} \quad (\vec{y} \cdot A)_j = 0 \quad \Longleftrightarrow \quad \vec{y} \cdot Z = \vec{0}$$

Damit erhalten wir mit (17)

$$\Pr_{\mathcal{P}} \left[\sum_{i=1}^{k+2} c_i r_k^i \equiv 0 \right] = \sum_{\vec{y} \in \mathbf{Z}_q^{2k+1}} \Pr_{\mathcal{P}} \left[\forall_{1 \le j \le k+2} \quad (\vec{y} \cdot A)_{2k+j} = c_j \ \wedge \ \vec{y} \cdot Z = \vec{0} \right]$$

Lemma 3.4 *Der n–Zeilen–Versuch $\vec{y}$ erzeuge $\vec{c}$. Dann gilt:*

$$\Pr_{\mathcal{P}}\left[\forall_{1\le j\le k+2}\ (\vec{y}\cdot A)_{2k+j}=c_j\ \wedge\ \vec{y}\cdot Z=\vec{0}\right]=\Pr_{Z\in\mathcal{R}}\left[\vec{y}\cdot Z=\vec{0}\right]d_1^{-m(\vec{y})}$$

mit $m(\vec{y}):=\#\{i\,|y_i\neq 0\wedge i\equiv 1 \bmod 2\}$.

Beweis: Die ersten $2k$ Spalten einer Matrix $A\in\mathcal{T}$ sind von den Spalten $2k+1$ bis $3k+2$ statistisch unabhängig. Mit Korollar 3.2 folgt also

$$\begin{aligned}&\Pr_{\mathcal{P}}\left[\forall_{1\le j\le k+2}\ (\vec{y}\cdot A)_{2k+j}=c_j\ \wedge\ \vec{y}\cdot Z=\vec{0}\right]\\ =\ &\Pr_{\mathcal{P}}\left[\forall_{1\le j\le k+2}\ (\vec{y}\cdot A)_{2k+j}=c_j\right]\Pr_{\mathcal{R}}\left[\vec{y}\cdot Z=\vec{0}\right]\end{aligned}$$

Es gilt

$$(\vec{y}\cdot A)_{2k+j}=\begin{cases}y_{2i-1}\alpha_k^{i+1} & \text{für } i=1\\ y_{2i-1}\alpha_k^{i+1}-y_{2i-3} & \text{für } 2\le i\le k+1\\ -y_{2i-3} & \text{für } i=k+2\end{cases}$$

Da $\vec{c}$ und $\vec{y}$ festliegen, kann $(\vec{y}\cdot A)_{2k+j}=c_j$ für $j=1,\ldots,k+2$ nur für ein Tupel $(\alpha_k^2,\ldots,\alpha_k^{k+2})$ gelten. Es sind aber nur $m(\vec{y})$ viele y_{2i-1} nicht Null. Daraus folgt mit Korollar 3.3

$$\Pr_{\mathcal{T}}\left[\forall_{1\le j\le k+2}\ (\vec{y}\cdot A)_{2k+j}=c_j\right]=d_1^{m(\vec{y})}\square$$

Folgende Definition liegt nun nahe:

Definition 3.6 *Die* **Effizienz** $\mathcal{E}(\vec{y})$ *eines n–Zeilen–Versuches $\vec{y}$ ist*

$$\Pr_{Z\in\mathcal{R}}\left[\vec{y}\cdot Z=\vec{0}\right]d_1^{-m(\vec{y})}$$

mit $m(\vec{y}):=\#\{i\,|y_i\neq 0\wedge i\equiv 1 \bmod 2\}$. *Ein Versuch $\vec{y}$ heißt* **erfolgreich**, *wenn* $\mathcal{E}(\vec{y})>0$ *ist.*

Damit ergibt sich zusammen mit Lemma 3.4

$$\mathcal{E}(\vec{c})=\sum_{\substack{\vec{y}\in\mathbf{Z}^{2k+1}\\ \vec{y}\text{ erzeugt }\vec{c}}}\mathcal{E}(\vec{y})$$

Es gilt also:

Satz 3.5 *$\vec{y}$ erzeuge $\vec{c}$. Dann ist $\vec{c}$ erfolgreich sofern $\vec{y}$ erfolgreich ist*

Die Umkehrung gilt i.A. nicht!

Ein erfolgreicher Versuch $\vec{y}$ liefert also immer eine erfolgreiche Attacke $\vec{c}$. Gilt jedoch $\vec{c} = 0$, so ergibt sich daraus kein Angriff auf das Preprocessing. Daß dieser Fall nicht eintritt — sofern nicht schon $\vec{y}$ trivial ist — gewährleistet der folgende Satz.

Satz 3.6 *In jeder Matrix $A \in \mathcal{T}$ sind die Zeilen linear unabhängig.*

Beweis: Wir nehmen an, es gäbe eine Linearkombination $\sum_{i=1}^{2k+1} x_i A_i = 0$ mit $x_i \neq 0$ für mindestens ein i. Wegen der Stufenform von A in den Spalten $2k+1$ bis $3k+2$ muß $x_i = 0$ für alle ungeraden i gelten. Es handelt sich also um eine Linearkombination nur in den geraden Zeilen. Diese haben aber in den Spalten 1 bis $2k$ eine Stufenform, so daß sie nicht linear abhängig sein können. Das ist ein Widerspruch zur Annahme und der Satz ist bewiesen.□

Daraus folgt :

Korrolar 3.7 *Der Versuch $\vec{y} \neq 0$ erzeuge die Attacke $\vec{c}$. Dann ist $\vec{c} \neq \vec{0}$.*

Dieses Ergebnis ist wichtig. Die Existenz eines erfolgreichen und nicht–trivialen Versuches garantiert also schon die Existenz einer erfolgreichen und nicht–trivialen Attacke. Diese läßt sich durch Erraten der Werte $\alpha_k^1, \ldots \alpha_k^{k+1}$ aus dem Versuch ermitteln.

3.3.2 Die Auswahl der Zeilen

Wenn wir Aussagen über die Effizienz eines Versuches machen wollen, müssen wir nur jene Zeilen j betrachten, für die $y_j \neq 0$ gilt. Deshalb ist folgende Definition nützlich:

Definition 3.7 *Sei $\vec{y}$ ein n–Zeilen–Versuch.Dann gibt es genau eine Injektion $\sigma : \{1 \ldots, n\} \to \{1, \ldots, 2k+1\}$ mit*

$$(\; i < j \quad \Rightarrow \quad \sigma(i) < \sigma(j) \;)$$

und $y_{\sigma(i)} \neq 0$ *für alle* $i = 1, \ldots, n$.σ *heißt die* **Zuordnung des Versuches** $\vec{y}$. *Eine* **n–Konfiguration** *ist eine Matrix* $B \in \mathbb{M}_{n,2k}$, *so daß es eine Injektion* $\sigma : \{1, \ldots, n\} \to \{1, \ldots, 2k\}$ *und eine Matrix* $Z \in \mathcal{R}$ *gibt mit*

$$\underset{1 \leq i \leq n}{\forall} \quad Z_{\sigma(i)} = B_i \quad \wedge \quad (i < j \Rightarrow \sigma(i) < \sigma(j))$$

(σ *ist wegen der Stufenform von* $Z \in \mathcal{R}$ *eindeutig bestimmt.)*
σ *heißt die* **Zuordnung der n–Konfiguration B**. *Ein n–Zeilen–Versuch* $\vec{y}$ **annulliert** *die n–Konfiguration B, wenn* $\vec{y}$ *und B dieselbe Zuordnung* σ *haben und* $\sum_{i=1}^{n} y_{\sigma(i)} B_i = 0$ *gilt.*

Veranschaulichung: Gilt $\sum_{i=1}^{2k+1} y_i Z_i = 0$ für einen n–Zeilen–Versuch $\vec{y}$ und eine Matrix $Z \in \mathcal{R}$, so interessieren uns nur die Zeilen i aus Z mit $y_i \neq 0$. Diese werden durch eine n–Konfiguration B dargestellt. B wird dann von dem Versuch $\vec{y}$ annulliert.
Uns interessiert hier vor allem zu einem fest gewählten n–Zeilen–Versuch $\vec{y}$ die Anzahl der n–Konfigurationen, die von $\vec{y}$ annulliert werden.

Sei n fest gewählt. Wir definieren auf der Menge der n–Konfigurationen eine Äquivalenzrelation:

$$B \sim \bar{B} : \quad \Longleftrightarrow \quad (\sigma = \bar{\sigma} \quad \wedge \quad \underset{i,j}{\forall} \; B_{i,j} \neq 0 \Leftrightarrow \bar{B}_{i,j} \neq 0) \tag{18}$$

Definition 3.8 *Eine* **n–Positionierung** *P ist eine Äquivalenzklasse bzgl. der durch* (18) *definierten Äquivalenzrelation. Die Zuordnung* σ *der n–Konfigurationen aus P nennen wir die* **Zuordnung der n–Positionierung** *P.*

Veranschaulichung: Eine n–Positionierung gibt für eine Matrix $Z \in \mathcal{R}$ die Positionen der Einträge $\neq 0$ in n fest gewählten Zeilen an.

Notation: Unter einem **Eintrag** einer Matrix $Z \in \mathcal{R}$ oder $A \in \mathcal{T}$ verstehen wir stets ein Z_{ij} bzw. A_{ij}, das $\neq 0$ ist. Unter einem **Pflichteintrag** verstehen wir einen Eintrag der -1 ist oder ein $\alpha_\nu^\nu, \alpha_{\nu-1}^\nu, \alpha_k^\nu$ oder β_ν^ν, d.h. einen Eintrag, der durch eine der Bedingungen 1.)–5.) als $\neq 0$ festgelegt ist. Unter einem **Wahleintrag** verstehen wir einen Eintrag, der kein Pflichteintrag ist.

Wenn n feststeht, reden wir auch einfach von einem Versuch, Konfiguration oder Positionierung.

Bemerkung: Man kann alle Definitionen dieses Kapitels auch für andere Bedingungen als 1.)–5.) dementsprechend definieren.In den Definitionen von P würden

dann z.B. statt 1.)–5.) die dementsprechenden Bedingungen gefordert.
Im folgenden gehen wir aber zunächst immer von 1.)–5.) aus.

Definition 3.9 *Sei $\vec{y}$ ein n–Zeilen–Versuch mit Zuordnung $\sigma_{\vec{y}}$. Dann sei*

$$\begin{aligned} N(\vec{y}) &:= \{n\text{–Konfigurationen } B \mid \vec{y} \text{ annulliert } B\} \\ |\vec{y}| &:= \#\{n\text{–Konfigurationen } B \mid \sigma_{\vec{y}} \text{ ist Zuordnung von } B\} \end{aligned}$$

Für jede α–Zeile einer Matrix $Z \in \mathcal{R}$ gibt es $\binom{k-2}{l_1-3} d_1^{l_1-1}$ viele Möglichkeiten für Positionierung und Wert der Einträge, für jede β–Zeile $\binom{k-1}{l_0-1} d_0^{l_0}$ viele. Daraus ergibt sich

$$|\vec{y}| = \left[\binom{k-2}{l_1-3} d_1^{l_1-1}\right]^{m(\vec{y})} \left[\binom{k-1}{l_0-1} d_0^{l_0}\right]^{n-m(\vec{y})} \tag{19}$$

Es gilt:

$$\mathcal{E}(\vec{y}) = \Pr_{Z\in\mathcal{R}}\left[\sum_{i=1}^{n} y_{\sigma(i)} Z_{\sigma(i)} = 0\right] d_1^{-m(\vec{y})} \tag{20}$$

Die Wahrscheinlichkeit kann auch über die Menge der n–Konfigurationen mit Zuordnung σ genommen werden. Wir erhalten also

$$\begin{aligned} \mathcal{E}(\vec{y}) &= \frac{|N(\vec{y})|}{|\vec{y}| {d_1}^{m(\vec{y})}} \\ &= \frac{|N(\vec{y})|}{\binom{k-2}{l_1-3}^{m(\vec{y})} \binom{k-1}{l_0-1}^{n-m(\vec{y})} {d_1}^{l_1 m(\vec{y})} {d_0}^{l_0(n-m(\vec{y}))}} \end{aligned} \tag{21}$$

In Abschnitt 5 werden wir ein Computer–Programm vorstellen, das $\mathcal{E}(\vec{y})$ mittels $|N(\vec{y})|$ berechnet. In den darauf folgenden Abschnitten legen wir das Preprocessing so fest, daß $\mathcal{E}(\vec{y}) < C \leq 2^{-64}$ für jeden Versuch $\vec{y}$ gilt. Erst dann werden wir versuchen, daraus eine Schranke für die Effizienz von Attacken $\vec{c}$ abzuschätzen.

4 Versuche mit bis zu vier Zeilen

In diesem Abschitt zeigen wir daß es keine erfolgreichen 4–Zeilen–Versuche gibt. Wir zeigen außerdem, daß die Bedingungen 1. bis 5. dafür nicht nur eine hinreichende, sondern auch eine notwendige Bedingung darstellen. Alle Argumentationen dieses Abschnitts macht man sich am besten anhand der Figur 2 klar.

Wir gehen von nun an **immer** von $k \geq 8$, $4 \leq l_1 \leq l_0 \leq k-2$ und $d_0, d_1 \geq 4$ aus.

Satz 4.1 *Für alle erfolgreichen n–Zeilen–Versuche gilt $n \geq 5$, d.h. weniger als 5 Zeilen einer Matrix $Z \in \mathcal{R}$ sind immer linear unabhängig.*

Beweis: Wir nehmen an, es gäbe weniger als 5 Zeilen, die linear abhängig sind. Diese Annahme führen wir zu einem Widerspruch.

Wegen 3. stehen in jeder α–Zeile $l_1 - 1$ viele Einträge, wegen 5. in jeder β–Zeile l_0 viele.

3 Zeilen: Die Zeilen müssen gemeinsam links abschließen. Wegen 1. und 4. müssen die ersten 2 Zeilen eine α–Zeile und eine β–Zeile sein, die direkt aufeinanderfolgen. Die 3. Zeile muß wegen 2. direkt auf die 2. folgen, um mit ihr gemeinsam rechts abzuschließen. Weil sich die -1 aus der 2. Zeile mit einem Eintrag aus der 3. Zeile wegkürzen muß, gilt $\text{sgn}(y_{\sigma(2)}) = \text{sgn}(y_{\sigma(3)})$. Wegen 5. ergeben diese beiden Zeilen in der Summe in mindestens l_0 vielen Spalten nicht 0, die 1. Zeile hat aber nur $l_1 - 1$ viele Einträge. Es können sich also nicht alle Einträge wegkürzen.

4 Zeilen: wieder gilt o.B.d.A. wegen 1. und 4., daß die 1. Zeile eine α–Zeile und die 2. die darauffolgende β–Zeile ist. Da die 4 Zeilen gemeinsam rechts abschließen müssen, sind wegen 2. die letzten beiden Zeilen eine β–Zeile und die darauffolgende α–Zeile. Wegen der -1 in der 3. Zeile gilt $\text{sgn}(y_{\sigma(3)}) = \text{sgn}(y_{\sigma(4)})$ und wegen der -1 in der 2. Zeile gilt $\text{sgn}(y_{\sigma(2)}) = \text{sgn}(y_{\sigma(3)}) = \text{sgn}(y_{\sigma(4)})$. Diese 3 Zeilen ergeben in der Summe in mindestens l_0 vielen Spalten nicht 0 (allein in den Spalten, in denen in der 2. Zeile die positiven Einträge stehen), die erste Zeile hat aber nur $l_1 - 1$ viele Einträge. Wieder können sich nicht alle Einträge wegkürzen. □

Satz 4.2 *Gilt eine der Bedingungen* 1., 2. *oder* 4. *nicht, so gibt es erfolgreiche* 4*–Zeilen–Versuche.*

Beweis: Wir zeigen jeweils ein Gegenbeispiel. Dabei sei $k = 8$, $l_1 = 4$ und d_0, d_1 beliebig. Wir geben jeweils eine Nullkonfiguration und den Versuch an, der sie annulliert. Aus diesen Konfigurationen lassen sich leicht Nulkonfigurationen für andere Parameter ableiten.

Es gelte 1. nicht:
$y_2 = y_3 = 1$, $y_4 = y_5 = -1$:

$l_0 = l_1$

0	4	0	8	1	2	0	0	-1	
	4	8	0	0	0	0	0	1	
	8	0	4	1	2	0	0	0	-1
		8	4	0	0	0	0	0	1

$l_0 = l_1 + 1$

0	4	0	8	1	2	1	0	-1	
	4	8	0	0	0	0	0	1	
	8	0	4	1	2	1	0	0	-1
		8	4	0	0	0	0	0	1

Es gelte 2. nicht:
$y_1 = y_5 = 1$, $y_2 = y_3 = -1$:

$l_0 = l_1$

1	0	4	8	0	0	0	0		
1	2	4	8	0	0	0	0	-1	
	2	4	8	0	0	0	0	0	
		4	8	0	0	0	0	1	0

$l_0 = l_1 + 1$

1	0	4	4	0	0	0	0		
1	2	8	8	4	0	0	0	-1	
	2	0	0	4	0	0	0	1	
		4	4	8	0	0	0	0	0

Es gelte 4. nicht:
$y_1 = y_3 = 1$, $y_4 = y_5 = -1$:

$l_0 = l_1$

0	0	0	8	2	0	0	4		
	4	8	0	0	0	0	0	1	
	4	0	8	2	0	0	0	1	-1
		8	0	0	0	0	4	0	1

$l_0 = l_1 + 1$

0	0	0	8	2	0	0	1		
	1	8	0	0	0	0	0	1	
	1	4	8	2	0	0	0	1	-1
		4	0	0	0	0	1	0	1

Satz 4.3 *Gilt* $l_0 = l_1 \geq 5$ *und gilt* 3. *oder* 5. *nicht, so gibt es erfolgreiche* 4*–Zeilen–Versuche.*

Beweis: Wir geben jeweils ein Gegenbeispiel an. Dabei sei wieder $k = 8$, $l_0 = l_1 = 5$ und d_0, d_1 beliebig.

Es gelte 3. nicht

$y_1 = y_3 = 1\ ,\ y_4 = y_5 = -1$

1	2	4	4	0	0	0	4		
1	1	2	2	0	0	0	1	1	
	1	1	1	0	0	0	2	1	-1
		1	1	0	0	0	1	0	1

Es gelte 5. nicht

$y_4 = y_5 = 1, y_1 = -1, y_2 = 2$

2	4	4	0	0	0	0	4		
1	1	1	0	0	0	0	1	-1	
	2	1	0	0	0	0	2	1	-1
		1	0	0	0	0	1	1	1

Bemerkung: Satz 4.1 bleibt für $l_0 > l_1$ auch dann wahr, wenn eine der Bedingungen 3. und 5. nicht gilt. Wir werden aber in Abschnitt 6 sehen, daß es trotzdem sinnvoll ist, beides zu fordern.

5 Das Programm zur Berechnung der Effizienz

Die Effizienz eines Versuches $\vec{y}$ berechnen wir mit einem Computer–Programm. Dieses errechnet zuerst $|N(\vec{y})|$ und dann $\mathcal{E}(\vec{y})$ nach (21).

5.1 Die Berechnung von $N(\vec{y})$

Sei $\vec{y}$ ein fest gewählter erfolgreicher n–Zeilen–Versuch und $\sigma_{\vec{y}} : \{1, \ldots, n\} \to \{1, \ldots, 2k+1\}$ seine Zuordnung.

Definition 5.1 *Sei* $M(\vec{y}) := \{n\text{–Positionierung } P \mid \sigma_{\vec{y}} \text{ ist Zuordnung von } P\}$

Es gilt:

$$N(\vec{y}) = \dot{\bigcup_{P \in M(\vec{y})}} P \cap N(\vec{y})$$

Die Vereinigung ist disjunkt, da jede Konfiguration in nur einer Positionierung enthalten ist. Daraus folgt

$$|N(\vec{y})| = \sum_{P \in M(\vec{y})} |P \cap N(\vec{y})|$$

Definition 5.2 *Sei P eine n–Positionierung mit Zuordnung σ und $1 \leq j \leq 2k$. Dann sei*

$$A(P, j) := \{(x_1, \ldots, x_n) \in \mathbf{Z}^n \mid \sum_{i=1}^{n} y_{\sigma(i)} x_i = 0 \ \wedge \ \exists_{B \in P} \ B_j^T = (x_1, \ldots, x_n)\}$$

Sei $\vartheta_P : \{1, \ldots, 2k\} \longrightarrow \mathbf{Z}_2^n \simeq \{0, \ldots, 2^n - 1\}$ *folgendemaßen definiert:*

$$\vartheta_P(j)_i = 1 \quad :\Longleftrightarrow \quad \forall_{B \in P} \ B_{ij} > 0$$

heißt die Positionierung der Spalte j.

Im folgenden fassen wir stets $\vartheta_P(j)$ als Element aus $\{0, \ldots, 2^n - 1\}$ auf. Seien $v(1), \ldots, v(u)$ die Zeilen, in denen die Positionierung P in der Spalte j einen Eintrag > 0 hat. Dann gilt $\vartheta_P(j) = \sum_{i=1}^{u} 2^{v(i)-1}$.

Veranschaulichung: In allen Konfigurationen aus einer fest gewählten Positionierung stehen die Einträge an denselben Stellen. $\vartheta_P(j)$ gibt an, in welchen Zeilen in der j-ten Spalte Einträge stehen.
$A(P,j)$ ist die Menge aller j-ten Spalten $(x_1,\ldots,x_n)$ von Konfigurationen aus P, für die $\sum_{i=1}^{n} y_{\sigma(i)}x_i = 0$ gilt. Wir sagen, die Spalte wird von $\vec{y}$ **annulliert**.

Bemerkung: Da $\vec{y}$ fest gewählt ist, hängt $A(P,j)$ nur von j und $\vartheta_P(j)$ ab.

Die Werte $a(i,\nu)$ und $b(i,\nu)$ legen fest, an welchen Stellen in einer Matrix $Z \in \mathcal{R}$ Einträge stehen. Durch die Werte $a(i,\nu)$ und $b(i,\nu)$ und eine Zuordnung σ ist die die Positionierung bereits festgelegt. Sei eine Positionierung P fest gewählt. Welche Werte die Einträge in einer Konfiguration $K \in P$ haben, wird nur durch die Werte $f(i,\nu)$ und $g(i,\nu)$ bestimmt. Diese sind voneinander unabhängig gewählt. Also sind auch die Werte der Einträge einer Konfiguration $K \in P$ voneinander unabhängig. Daraus folgt, daß man eine Konfiguration aus P auch "spaltenweise zusammensetzen" kann.

Für jede Positionierung P gilt:
Jedes Tupel $(\vec{x}^1,\ldots,\vec{x}^{2k})$ mit $\vec{x}^i \in \mathbf{Z}^n$, für das gilt

$$\vec{x}^i \textit{ ist die i-te Spalte einer Konfiguration } K_i \in P$$

definiert eine Konfiguration $K = [\vec{x}^1,\ldots,\vec{x}^{2k}] \in P$.

Wird dabei jede Spalte $\vec{x}^i$ von $\vec{y}$ annulliert, so wird auch K von $\vec{y}$ annulliert.
Umgekehrt gilt aber auch: Wird eine Konfiguration von $\vec{y}$ annulliert, so wird auch jede ihrer Spalten von $\vec{y}$ annulliert.

Es gilt also:

$$\underset{P}{\forall}\left(Z \in P \cap N(\vec{y}) \quad\Longleftrightarrow\quad \underset{1\le j\le 2k}{\forall} Z_j^T \in A(P,j)\right)$$

Daraus ergibt sich

$$|P \cap N(\vec{y})| = \prod_{j=1}^{2k} |A(P,j)|$$

und damit

$$|N(\vec{y})| = \sum_{P\in M(\vec{y})} \prod_{j=1}^{2k} |A(P,j)|$$

Bemerkung: Wir haben $\vec{y}$ als erfolgreich vorrausgesetzt. Da ein Versuch nur

dann erfolgreich ist, wenn die beteiligten Zeilen rechts und links gemeinsam abschließen, gilt $\max(i \mid y_i \neq 0) \equiv 1 \bmod 2$. Das macht man sich leicht anhand der Figur 1 klar

Sei $2t+1 = \max(i \mid y_i \neq 0)$ mit $1 \leq t \leq k$ und σ die Zuordnung von $\vec{y}$. Dann sind alle Spalten j einer Konfiguration mit Zuordnung σ mit $j > k+t$ Null.

$$\bigvee_{P \in M(\vec{y})} \bigvee_{\substack{B \in P \\ k+t<j<2k+1}} B_j^T = (0, \ldots, 0)$$

Nach Definition von $A(P,j)$ folgt daraus

$$\bigvee_{\substack{P \in M(\vec{y}) \\ k+t<j<2k+1}} A(P,j) = \{(0, \ldots, 0)\}$$

und damit

$$|N(\vec{y})| = \sum_{P \in M(\vec{y})} \prod_{j=1}^{k+t} |A(P,j)|$$

Die Spalten von $B \in P \in M(\vec{y})$, in denen eine -1 steht, sind durch $\vec{y}$ festgelegt. Deshalb ist folgende Definition sinnvoll:

Definition 5.3 *Sei ein Versuch $\vec{y}$ gegeben. Dann heißt i $(1 \leq i \leq 2k)$* **positiv***, wenn für $B \in P \in M(\vec{y})$ die Spalte i von B keine -1 enthält. Dies ist von der Wahl von B und P unabhängig.*

Es gilt:

$$\bigvee_{\substack{P,Q \in M(\vec{y}) \\ i,j \text{ positiv}}} \vartheta_P(i) = \vartheta_Q(j) \quad \Longrightarrow \quad A(P,i) = A(Q,j)$$

Veranschaulichung: Die Berechnung von $|A(P,j)|$ ist sehr zeitaufwendig. Bei u positiven Einträgen in der Spalte j müssen wegen $d_1, d_0 \geq 4$ mehr als 4^u Fälle durchgerechnet werden. Die Berechnung von $\vartheta_P(j)$ benötigt dagegen nur $O(n)$ viele Schritte. Da $\vartheta_P(j)$ aus $\{0, \ldots, 2^n - 1\}$ ist, müssen insgesamt nicht mehr als 2^n viele Werte $|A(P,j)|$ für positive Spalten j berechnet werden.
Für die nicht–positiven j hängt $|A(P,j)|$ außer von $\vartheta_P(j)$ noch von j selbst ab. Für jedes nicht–positive j müssen bis zu 2^{n-1} Werte $|A(P,j)|$ berechnet werden. Das Programm durchläuft alle Positionierungen $P \in M(\vec{y})$. Für jedes P berechnet es $\vartheta_P(j)$ für alle j. Dann berechnet es $F(\vartheta_P(j)) := |A(P,j)|$ für positive j und $G(\vartheta_P(j), j) := |A(P,j)|$ für nicht–positive j.
Da das Programm die bereits berechneten F– und G–Werte speichert, müssen

also insgesamt nicht mehr als $2^n + c2^{n-1}$ viele Werte $|A(P,j)|$ wirklich berechnet werden. Hierbei ist c die Anzahl der nicht–positiven j ($1 \leq j \leq k+m$). Bei $n = 5$ ist z.B. $c = 2$.
Die F– und G–Werte werden zu Anfang alle auf -1 gesetzt. Dadurch wird gekennzeichnet, daß ein Wert noch nicht berechnet wurde.

Die Anzahl der Einträge in jeder Zeile einer Konfiguration $B \in P \in M(\vec{y})$ ist durch σ festgelegt. In der Zeile i stehen

$$c_i := \begin{cases} l_0 & \text{falls } \sigma(i) \equiv 0 \bmod 2 \\ l_1 - 1 & \text{falls } \sigma(i) \equiv 1 \bmod 2 \end{cases}$$

viele Einträge. Deshalb ist es sinnvoll, die Positionierungen "zeilenweise zu erzeugen". Dazu benötigen wir zunächst eine

Definition 5.4 *Sei P eine Positionierung. Dann gibt es für $1 \leq i \leq n$ genau ein $\vec{x}^{\,i}(P) = (x^i_1, \ldots, x^i_{c_i}) \in \{1, \ldots, 2k+1\}^{c_i}$ mit:*

$$a < b \Rightarrow x^i_a < x^i_b \quad \textit{und} \quad \underset{\substack{1 \leq a \leq c_i \\ B \in P}}{\forall} B_{i,x^i_a} \neq 0$$

$\vec{x}^{\,i}(P)$ heißt die **Positionierung der Zeile i in P**.

Für einen n–Zeilen–Versuch $\vec{y}$ sei

$$M^i(\vec{y}) := \{\vec{x} \in \{1, \ldots, 2k+1\}^{c_i} | \underset{P \in M(\vec{y})}{\exists} \vec{x} = \vec{x}^{\,i}(P)\}$$

Veranschaulichung: Für $\vec{x}^{\,i}(P) = (x^i_1, \ldots, x^i_{c_i})$ gibt x^i_u die Spalte von $B \in P$ an, in der in der i-ten Zeile der u-te Eintrag von vorne steht. Dies ist von der Wahl von B unabhängig.

Die Werte $a(i,\nu)$ bzw. $b(i,\nu)$ sind für verschiedene ν voneinander unabhängig. Außerdem sind die Werte $a(i,\nu)$ unabhängig von den Werten $b(i,\nu)$. Die Positionierung einer Zeile ist aber gerade durch ein Tupel $(a(i,\nu))_{1 \leq i \leq l_1}$ bzw. $(b(i,\nu))_{1 \leq i \leq l_0}$ gegeben. Deshalb definiert jedes Tupel $(\vec{x}^{\,1}, \ldots, \vec{x}^{\,n}) \in M^1(\vec{y}) \times \ldots \times M^n(\vec{y})$ genau eine Positionierung $P \in M(\vec{y})$ und umgekehrt. Es gilt also:

$$M(\vec{y}) \simeq M^1(\vec{y}) \times \ldots \times M^n(\vec{y})$$

Offensichtlich gilt für die Zuordnung σ von $\vec{y}$

$$\underset{i,u}{\forall} \left\lfloor \frac{\sigma(i)+1}{2} \right\rfloor \leq x^i_u \leq \left\lfloor \frac{\sigma(i)+1}{2} \right\rfloor + k - 1$$

Definition 5.5 *Wir setzen* $anfang(i) := \left\lfloor \frac{\sigma(i)+1}{2} \right\rfloor$ *und* $ende(i) := \left\lfloor \frac{\sigma(i)+1}{2} \right\rfloor + k - 1$

Die Bedingungen 1.)–3.) können wir jetzt folgendermaßen schreiben:

$$\begin{array}{lll} 1.) & x_1^i = anfang(i) & \text{für } \sigma(i) \equiv 0 \bmod 2 \\ 2.) & x_1^i = anfang(i) & \text{für } \sigma(i) \equiv 1 \bmod 2 \\ 3.) & x_{c_i}^i = ende(i) & \text{für } \sigma(i) \equiv 1 \bmod 2 \end{array}$$

Es sind also nur noch $\tilde{c}_i := \begin{cases} c_i - 1 = l_0 - 1 & \text{für } \sigma(i) \equiv 0 \bmod 2 \\ c_i - 2 = l_1 - 3 & \text{für } \sigma(i) \equiv 1 \bmod 2 \end{cases}$ viele x_u^i in jeder Zeile i unbestimmt. Diese unbestimmten x_u^i werden in der Unterroutine Schleife festgelegt.

5.2 Die Algorithmen

5.2.1 Skizzierung der Routine Schleife

Initialisierung im Hauptprogramm:

FOR $i = 1$ TO n DO
    $x_1^i := anfang(i)$
    $\vartheta_{_P}(x_1^i) := 2^{i-1}$
    IF $\sigma(i) \equiv 1 \bmod 2$ THEN DO
        $x_{c_i}^i :=$ ende(i)
        $\vartheta_{_P}(x_{c_i}^i) := \vartheta_{_P}(x_{c_i}^i) + 2^{i-1}$
u=2
i=1

Schleife:

FOR $x_u^i = x_{u-1}^i$ TO $ende(i) - c_i + u$ DO
    $\vartheta_{_P}(x_u^i) := \vartheta_{_P}(x_u^i) + 2^{i-1}$
    IF $u < \tilde{c}_i$ THEN DO
        $u := u + 1$

Schleife

$u := u - 1$

ELSE IF $i < n$ THEN DO

$u := 2$

$i := i + 1$

Schleife

$i := i - 1$

$u := \tilde{c}_i + 1$

ELSE DO

berechne $|N(\vec{y}) \cap P|$

$gesamt := gesamt + |N(\vec{y}) \cap P|$

$\vartheta_P(x_u^i) := \vartheta_P(x_u^i) - 2^{i-1}$

Durch diesen Aufbau durchläuft jede Positionierung $P \in M(\vec{y})$ genau einmal die ELSE–Anweisung. Für x_u^i ist $ende(i) - c_i + u$ der maximale Wert, da $x_u^i < x_{u+1}^i < \ldots < x_{c_i}^i \leq ende(i)$ für $u < c_i$ gilt.

5.2.2 Die Berechnung von $|\mathbf{N}(\vec{\mathbf{y}}) \cap \mathbf{P}|$

Initialisierung im Hauptprogramm:

$F(0) := 1$

FOR $i = 1$ TO 2^{n-1} DO

$F(i) := -1$

FOR $j = 1$ TO $2k + 1$ DO

$G(0, j) := 0$

FOR $i = 1$ TO 2^{n-1} DO

$G(i, j) := -1$

Berechnung in Routine Schleife:

faktor:= 1

FOR $j = 1$ TO $2k + 1$ DO

IF j positiv THEN DO

IF $F(\vartheta_P(j)) = -1$ THEN DO

berechne $|A(P, j)|$

$F(\vartheta_P(j)) := |A(P, j)|$

$faktor := faktor \cdot F(\vartheta_P(j))$

ELSE DO

IF $G(\vartheta_P(j), j) = -1$ THEN DO

berechne $|A(P, j)|$

$G(\vartheta_P(j), j) := |A(P, j)|$

$faktor := faktor \cdot G(\vartheta_P(j), j)$

$|N(\vec{y}) \cap P| := faktor$

5.2.3 Die Berechnung von $|A(P, j)|$

Die Berechnung von $|A(P, j)|$ erfolgt in 2 Schritten:

1. Berechnung von u und $v(1) < \ldots < v(u)$ mit $\sum_{i=1}^{u} 2^{v(i)-1} = \vartheta_P(j)$ aus $\vartheta_P(j)$. u ist die Anzahl der Einträge in der Spalte j.

2. Durchlaufen aller Tupel $(e(1), \ldots, e(u))$ mit $e(i) \in \{0, \ldots, d_{v(i) \bmod 2} - 1\}$ und Test, ob

$$\sum_{i=1}^{u} y_{\sigma(v(i))} 2^{e(i)} = \begin{cases} 0 & \text{falls j positiv} \\ y_{2(j-k)} & \text{sonst} \end{cases}$$

Die Anzahl der Tupel, für die das gilt, ist $|A(P, j)|$.

Bemerkung: Für Spalte 1 ist $A_1 := |A(P, 1)|$ unabhängig von der Positionierung. Deshalb muß $A(P, 1)$ nicht für jedes P neu errechnet werden. Vielmehr kann A_1 im Hauptprogramm einmal errechnet und am Ende auf *gesamt* aufmultipliziert werden.

$$\sum_{P \in M(\vec{y})} |P \cap N(\vec{y})| = A_1 \sum_{P \in M(\vec{y})} \prod_{j=2}^{t+k} |A(P, j)|$$

Das Programm errechnet dann für jedes P statt $|P \cap N(\vec{y})|$ nur $\prod_{j=2}^{t+k} |A(P, j)|$.

6 Zuordnung und Effizienz der Versuche mit fünf Zeilen

Die Laufzeiten des Programms steigen mit n exponentiell an. Um umfangreiche Tests durchführen zu können, konzentrieren wir uns deshalb zunächst auf 5–Zeilen–Versuche. In diesem Abschnitt untersuchen wir, für welche Zuordnungen die 5–Zeilen-Versuchen am effizientesten sind. Es zeigt sich, daß die Versuche am effizientesten sind, bei denen die Zeilen dicht aufeinander folgen.

6.1 Die Gestalt der 5–Zeilen–Versuche

Satz 6.1 *Für einen erfolgreichen 5–Zeilen–Versuch gilt stets $m(\vec{y}) = 3$. D.h. es werden stets drei α–Zeilen und zwei β–Zeilen verwendet.*

Beweis: Wegen 1. und 4. sei o.B.d.A. $\sigma(1) = 1$ und $\sigma(2) = 2$. Die letzten beiden Zeilen ($Z_{\sigma(4)}$ und $Z_{\sigma(5)}$) müssen rechts gemeinsam abschließen, also gilt wegen 2. $\sigma(4) = 2t$ und $\sigma(5) = 2t + 1$ für ein $t \in \mathbf{N}$. Weil sich der letzte Eintrag von Z_{2t+1} mit der -1 aus Z_{2t} wegkürzen muß, gilt sgn(y_{2t}) =sgn(y_{2t+1}). Wir setzen $x = \sigma(3)$. Nun ist nur noch zu zeigen, daß x ungerade ist.

Annahme: x ist gerade. Da sich die -1 aus Z_x nur mit den Einträgen aus Z_{2t} und Z_{2t+1} wegkürzen kann, gilt sgn(y_x) =sgn(y_{2t}) =sgn(y_{2t+1}). Mit dem gleichen Argument sieht man sgn(y_2) =sgn(y_x) = ... In der Summe haben diese 4 Zeilen mindestens l_0 viele Einträge (allein in Z_2), Z_1 aber nur $l_1 - 1$ viele. □

Korrolar 6.2 *Für einen 5–Zeilen–Versuch gilt o.B.d.A.:*
$\sigma(1) = 1 \quad \sigma(2) = 2 \quad \sigma(3) = x \quad \sigma(4) = 2t$ *und* $\sigma(5) = 2t + 1$ *mit* $2 < x < 2t, \quad x = 1 \bmod 2$ *und* $2 \leq t \leq k$

Definition 6.1 *Das Vorzeichen eines n–Zeilen–Versuches sei $(sgn(y_{\sigma(i)}))_{1 \leq i \leq n}$. Wir schreiben statt $+1$ und -1 nur $+$ und $-$.*

Bemerkung: Von jetzt an gehen wir, wenn wir von einem erfolgreichen 5–Zeilen–Versuch sprechen, o.B.d.A. von $\sigma(1) = 1, \sigma(2) = 2, \sigma(3) = x, \sigma(4) = 2t, \sigma(5) = 2t + 1$ und $y_1 > 0$ aus.

Satz 6.3 *Jeder erfolgreiche 5–Zeilen–Versuch $\vec{y}$ besitzt entweder das Vorzeichen* $(+,-,-,+,+)$ *oder* $(+,-,+,-,-)$.

Beweis: Wie wir gesehen haben, gilt $\text{sgn}(y_{2t}) = \text{sgn}(y_{2t+1})$ und wegen Spalte 1 ist $y_2 < 0$.
Annahme: $\text{sgn}(y_x) = \text{sgn}(y_{2t})$.
Wegen der -1 in Z_2 gilt $\text{sgn}(y_2) = \text{sgn}(y_x) = \text{sgn}(y_{2t}) = \text{sgn}(y_{2t+1})$. Diese 4 Zeilen haben in der Summe mehr Einträge als Z_1. Das ist ein Widerspruch zur Annahme.
Also gilt: $\text{sgn}(y_x) \neq \text{sgn}(y_{2t}) = \text{sgn}(y_{2t+1})$. $\square$

Definition 6.2 *Zwei n–Zeilen–Versuche $\vec{y}$ mit Zuordnung σ und $\vec{y}'$ mit Zuordnung σ' heißen verwandt, wenn gilt:*

$$\underset{1 \leq i \leq n}{\forall} \quad y_{\sigma(i)} = y'_{\sigma'(i)}$$

Für $l_0 > l_1$ erhalten wir als Ergebnis dieses Abschnittes folgenden Satz:

Satz 6.4
1. *Zu jedem 5–Zeilen–Versuch mit Vorzeichen $+--++$ gibt es einen verwandten Versuch, der mindestens genauso effizient ist und für den* $(x = 3 \wedge t = 2)$ *oder* $(x = 3 \wedge t = 3)$ *gilt.*
2. *Zu jedem 5–Zeilen–Versuch mit Vorzeichen $+-+--$ gibt es einen verwandten Versuch, der mindestens genauso effizient ist und für den* $(x = 3 \wedge t = 2)$, $(x = 3 \wedge t = 3)$, $(x = 5 \wedge t = 3)$ *oder* $(x = 5 \wedge t = 4)$ *gilt.*

Für $l_0 = l_1$ erhalten wir nur für die Vorzeichen $+ - - + +$ ein (schwächeres) Ergebnis.

Notation: Unter einer **Konfiguration der Zeilen** $Z_1, \ldots, Z_n$ verstehen wir eine n-Konfiguration mit Zuordnung σ, so daß $\sigma(i) = Z_i$ für alle $i \leq n$ gilt. Analog dazu ist ein Versuch mit den Zeilen $Z_1, \ldots, Z_n$ zu verstehen.

Wir beweisen Satz 6.4 mittels einiger Zwischenresultate:

6.2 Der Fall $+--++$

Satz 6.5 *Sei $m \in \mathbf{N}, t \geq 2$ und $t+m+1 < k$. Dann gibt es eine Abbildung $\varphi : \mathbb{M}_{5,2k} \to \mathbb{M}_{5,2k}$ mit folgenden Eigenschaften:*

1. *Ist $A \in \mathbb{M}_{5,2k}$ eine Konfiguration der Zeilen $1,\ 2,\ 3{+}2m,\ 2(t{+}m),\ 2(t{+}m){+}1$ und $\vec{y} \in \mathbf{Z}^5$ mit den Vorzeichen $+--++$ mit $\sum_{i=1}^{5} y_i A_i = 0$, so ist $\varphi(A)$ eine Konfiguration der Zeilen $1,\ 2,\ 3,\ 2t,\ 2t+1$ mit $\sum_{i=1}^{5} y_i \varphi(A)_i = 0$.*

2. *φ ist injektiv.*

Beweis: Wir definieren φ folgendermaßen:

$$
\begin{array}{lcll}
\nu = 1,2 & \mu = 1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
& 2 \leq \mu \leq k{-}m{-}1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu+m} \\
& k{-}m \leq \mu \leq k{-}1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu-k+m+2} \\
& \mu \geq k: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
\\
\nu = 3,4,5 & \mu = 1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu+m \bmod 2k} \\
& 2 \leq \mu \leq k{-}m{-}1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu+m} \\
& k{-}m \leq \mu \leq k{-}1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu-k+m+2} \\
& \mu = k,\ k{-}1: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
& \mu \geq k+2: & \varphi(A)_{\nu,\mu} & = A_{\nu,\mu+m \bmod 2k}
\end{array}
$$

Man überzeugt sich leicht davon, daß φ die Einträge der Matrix A permutiert:

$$
\begin{array}{lccc}
\nu = 1,2 & \mu = 1 & \Longleftrightarrow & \mu = 1 \\
& 2 \leq \mu \leq k{-}m{-}1 & \Longleftrightarrow & m{+}2 \leq \mu + m \leq k{-}1 \\
& k{-}m \leq \mu \leq k{-}1 & \Longleftrightarrow & 2 \leq \mu{-}k{+}m{+}2 \leq m{+}1 \\
& \mu \geq k & \Longleftrightarrow & \mu \geq k \\
\\
\nu = 3,4,5 & \mu = 1 & \Longleftrightarrow & \mu{+}m \bmod 2k = m{+}1 \\
& 2 \leq \mu \leq k{-}m{-}1: & \Longleftrightarrow & m{+}2 \leq \mu + m \leq k{-}1 \\
& k{-}m \leq \mu \leq k{-}1: & \Longleftrightarrow & k{+}2 \leq \mu{+}m{+}2 \leq k{+}m{+}1 \\
& \mu = k,\ k{+}1: & \Longleftrightarrow & \mu = k,\ k{+}1 \\
& \mu \geq k+2 & \Longleftrightarrow & \mu{+}m \bmod 2k \geq k{+}m{+}2
\end{array}
$$

Damit ist φ injektiv.

Wir wollen die Abbildung φ an einem Beispiel verdeutlichen:
Sei $k = 12,\ t = 4$ und $m = 3$:

A =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
x											x	-	-	-	-	-	-	-	-	-	-	-	-
x												-1	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	x											x	-	-	-	-	-	-	-	-
-	-	-	-	-	-	x												-1	-	-	-	-	-
-	-	-	-	-	-	-	x											x	-	-	-	-	-

Ein "x" stellt einen festen und Eintrag dar, '-' steht für eine 0.

Da φ nur die Einträge innerhalb einer Zeile permutiert, genügt es, für jedes $\varphi(A)_{i,j}$ das η mit $\varphi(A)_{i,j} = A_{i,\eta}$ anzugeben. Dort, wo φ die Matrix unverändert läßt, wo also $j = \eta$ gilt, schreiben wir der Übersicht wegen keine Zahl.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	5	6	7	8	9	10	11	2	3	4													
	5	6	7	8	9	10	11	2	3	4													
4	5	6	7	8	9	10	11	14	15	16			17	18	19	20	21	22	23	24	1	2	3
4	5	6	7	8	9	10	11	14	15	16			17	18	19	20	21	22	23	24	1	2	3
4	5	6	7	8	9	10	11	14	15	16			17	18	19	20	21	22	23	24	1	2	3

Der Bereich, in dem in der Matrix $\varphi(A)$ Einträge stehen können, ist markiert.

Mit A ist auch $\varphi(A)$ eine Konfiguration der Zeilen Z_1, Z_2, Z_3, Z_{2t} und Z_{2t+1}:
Die Pflichteinträge α_ν^ν und β_ν^ν in den Zeilen 3, 4 und 5 stehen wegen $t+m+1 < k$ in der Matrix A in den Spalten $2+m$ bis k. Deshalb werden sie durch φ um m Spalten nach links verschoben. Die Pflichteinträge $\alpha_{\nu-1}^\nu$ und -1 in den Zeilen 4 und 5 stehen in der Spalte $k+m+t$ und werden deshalb von φ ebenfalls um m Spalten nach links verschoben. Damit Bedingung 2. — $\alpha_{\nu-1}^\nu \neq 0$ — auch in Zeile 3 gilt, muß außerdem $\varphi(A)_{3,k+1} \neq 0$ gelten. Das sieht man folgendermaßen:
Aus $\sum y_i A_i = 0$ folgt für die Spalte $k+1$

$$-y_3 A_{3,k+1} \;=\; -y_2 \;+\; y_4 A_{4,k+1} \;+\; y_5 A_{5,k+1} > 0$$

denn wir hatten $\vec{y} \in \mathbf{Z}^5$ mit den Vorzeichen $+--++$ vorrausgesetzt. Daraus folgt $\varphi(A)_{3,k+1} = A_{3,k+1} \neq 0$.

Es bleibt zu zeigen, daß $\sum_{i=1}^{5} y_i \varphi(A)_i = 0$ gilt:
Für die Spalten $2 \leq \mu \leq k-m-1$ und $\mu = k, k+1$ ist das klar.

Für $k+2 \leq \mu \leq 2k-m$ gilt

$$\left(\sum_{i=1}^{5} y_i \varphi(A)_i \right)_\mu \;=\; \sum_{i=1}^{2} y_i A_{i,\mu} + \sum_{i=3}^{5} y_i A_{i,\mu+m}$$

$$= \quad 0+0$$

Für $\mu \geq 2k-m+1$ erhalten wir

$$\begin{aligned}\left(\sum_{i=1}^{5} y_i\varphi(A)_i\right)_\mu &= \sum_{i=1}^{2} y_i A_{i,\mu} + \sum_{i=3}^{5} y_i A_{i,\mu+m-2k} \\ &= 0+0\end{aligned}$$

da $A_{i,j} = 0$ für $i \geq 3$ und $j \leq m$ gilt.

Sei nun $k-m \leq \mu \leq k-1$.
Wegen $\mu-k+m+2 \leq m+1$ gilt für $i \geq 3:\quad A_{i,\mu-k+m+2} = 0$
Außerdem gilt wegen $\mu+m+2 \geq k+2$ für $i = 1,2:\quad A_{i,\mu+m+2} = 0$
Damit erhalten wir

$$\begin{aligned}\left(\sum_{i=1}^{5} y_i\varphi(A)_i\right)_\mu &= \sum_{i=1}^{2} y_i A_{i,\mu-k+m+2} + \sum_{i=3}^{5} y_i A_{i,\mu+m+2} \\ &= \sum_{i=1}^{5} y_i A_{i,\mu-k+m+2} + \sum_{i=1}^{5} y_i A_{i,\mu+m+2} \\ &= 0+0\end{aligned}$$

□

Wir versuchen nun, die Vorraussetzungen von Satz 6.5 abzuschwächen:

Satz 6.6 *Es gibt keinen erfolgreichen 5–Zeilen–Versuch mit den Vorzeichen* $+--++$ *und* $t \geq k$

Beweis: Wir nehmen an, es gäbe einen 5–Zeilen–Versuch, der eine 5–Konfiguration annulliert. In den Spalten $2 \leq j \leq k-1$ gibt es in den Zeilen mit positiven Koeffizienten höchstens $l_1 - 3$ viele Einträge, in denen mit negativen Koeffizienten dagegen mindestens l_0-2 viele (in Zeile 2). Das ist ein Widerspruch zur Annahme.

□

Satz 6.7 *Sei* $l_1 < l_0$. *Dann gibt es keinen erfolgreichen 5-Zeilen–Versuch mit den Vorzeichen* $+--++$ *und* $t \geq k-1$.

Beweis: Wir nehmen wieder an, es gäbe einen 5–Zeilen–Versuch, der eine 5–Konfiguration annulliert. In den Spalten $2 \leq j \leq k-2$ gibt es in den Zeilen mit positiven Koeffizienten höchstens $l_1 - 3$, in den Zeilen mit negativen Koeffizienten dagegen mindestens $l_0 - 3$ viele Einträge. Das ist ein Widerspruch zur Annahme. □

Wir können Satz 6.5 jetzt auch so formulieren:

Satz: *Sei $m \in \mathbf{N}$ und es gelte $l_1 < l_0$ oder $t + m + 1 \neq k$. Dann ist ein 5–Zeilen–Versuch mit den Zeilen 1, 2, $3+2m$, $2(t+m)$ und $2(t+m)+1$ und den Vorzeichen $+ - - - +$ höchstens so effizient wie der verwandte Versuch mit den Zeilen 1, 2, 3, $2t$ und $2t+1$.*

Wir versuchen nun, den Worst–Case noch weiter einzugrenzen:

Satz 6.8 *Mit $t \geq 3$ gilt: Ein 5–Zeilen–Versuch mit den Zeilen 1, 2, 3, $2t$, $2t+1$ und den Vorzeichen $+ - - - +$ ist höchstens so effizient wie der verwandte Versuch mit den Zeilen 1, 2, 3, 6, 7.*

Beweis: Sei zunächst $t = k - 1$. Es gibt in den Spalten $k + 2 \leq j \leq k + t - 1$ in den Zeilen mit positiven Koeffizienten mindestens 1 (in der Zeile $2t$), in den Zeilen mit negativen Koeffizienten aber keinen Eintrag. Der Versuch ist also nicht erfolgreich.

Sei nun $4 \leq t \leq k-2$. Wir definieren $\psi : \mathbb{M}_{5,2k} \to \mathbb{M}_{5,2k}$ mit $m := t - 3$ wie folgt:

$$
\begin{array}{lrll}
\nu = 1,2,3 & \mu = 1,\ 2: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
& 3 \leq \mu \leq k-m-1: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu+m} \\
& k-m \leq \mu \leq k-1: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu+3+m-k} \\
& \mu \geq k: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
\\
\nu = 4,5 & \mu \leq k-m+1: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu+m} \\
& k-m \leq \mu \leq k-1: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu+m+2} \\
& \mu = k,\ k+1: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu} \\
& \mu \geq k+m+2: & \psi(A)_{\nu,\mu} & = A_{\nu,\mu+m \bmod 2k}
\end{array}
$$

Wieder sieht man, daß ψ die Komponenten der Zeilen von A permutiert und damit injektiv ist. Ist A eine Konfiguration der Zeilen 1, 2, 3, $2t, 2t+1$, so ist $\psi(A)$ eine der Zeilen 1, 2, 3, 6, 7. Falls A von einem 5–Zeilen–Versuch mit Vorzeichen $+ - - + +$ annulliert wird, wird $\psi(A)$ von dem verwandten Versuch annulliert.

□

ψ wirkt folgendermaßen: $(k = 12,\ t = 6\ \Rightarrow\ m = 3)$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
		6	7	8	9	10	11	3	4	5													
		6	7	8	9	10	11	3	4	5													
		6	7	8	9	10	11	3	4	5													
4	5	6	7	8	9	10	11	14	15	16			17	18	19	20	21	22	23	24	1	2	3
4	5	6	7	8	9	10	11	14	15	16			17	18	19	20	21	22	23	24	1	2	3

Wir erhalten also insgesamt:

Für den Worst–Case bei 5–Zeilen–Versuchen mit Vorzeichen $+--++$ gilt:

$$(t = 2 \wedge x = 3) \text{ oder } (t = 3 \wedge x = 3) \qquad \text{für } l_0 > l_1$$

$$(t = 2 \wedge x = 3) \text{ oder } (t = 3 \wedge x = 3),\ (t = k - 1) \qquad \text{für } l_0 = l_1$$

6.3 Der Fall $+-+--$

Im Fall der Vorzeichen $+-+--$ erhalten wir eine nicht ganz so starke Einschränkung für den Worst–Case. Das liegt daran, daß wir hier die Abbildung φ aus Satz 6.5 nicht verwenden können, da jetzt $A_{3,k+1} \neq 0$ i.a. nicht gilt. Dann können wir aber auch nicht $\alpha_\nu^{\nu-1} \neq 0$ (Bedingung 2.) in der 3. Zeile von $\varphi(A)$ folgern.

Wir erhalten aber immerhin:

Satz 6.9 *Sei $m \geq 2,\ t \geq 2$ und $t+m+1 < k$. Dann ist ein 5–Zeilen–Versuch mit den Zeilen $1,\ 2,\ 3+2m,\ 2(t+m)$ und $2(t+m)+1$ höchstens so effizient wie der verwandte Versuch mit den Zeilen $1,\ 2,\ 5,\ 2t+2$ und $2t+3$.*

Beweis: Die injektive Abbildung φ ist hier gegeben durch:

$$\begin{array}{lrcl}
\nu = 1,2 & \mu = 1: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
& 2 \le \mu \le k-m: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu+m-1} \\
& k-m+1 \le \mu \le k-1: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu-k+m+1} \\
& \mu > k: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
 & & & & \\
\nu = 3,4,5 & 1 \le \mu \le k-m: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu+m-1} \\
& k-m+1 \le \mu \le k-1: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu+m+1} \\
& \mu = k,\, k+1: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
& \mu \ge k+2: & \varphi(A)_{\nu,\mu} & = & A_{\nu,\mu+m-1 \bmod 2k}
\end{array}$$

φ wirkt folgendermaßen: $(k = 12,\ t = 4,\ m = 3)$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
	4	5	6	7	8	9	10	11	2	3													
	4	5	6	7	8	9	10	11	2	3													
3	4	5	6	7	8	9	10	11	14	15			16	17	18	19	20	21	22	23	24	1	2
3	4	5	6	7	8	9	10	11	14	15			16	17	18	19	20	21	22	23	24	1	2
3	4	5	6	7	8	9	10	11	14	15			16	17	18	19	20	21	22	23	24	1	2

□

Wieder können wir die Vorraussetzungen abschwächen:

Satz 6.10 *Sei $l_0 > l_1$. Dann gibt es für $t+1 \ge k$ keinen erfolgreichen Versuch mit den Zeilen 1, 2, x, $2t$ und $2t+1$ und den Vorzeichen $+-+--$.*

Beweis: Wir führen die Annahme, es gäbe eine Konfiguration, die von einem solchen Versuch annulliert wird, zu einem Widerspruch.
In den Spalten 1 bis $k-1$ gibt es in den Zeilen mit negativen Koeffizienten mindestens $l_0 - 1$ viele Einträge (alleine in Zeile 2), in Zeile 1 aber nur $l_1 - 2$ viele. In diesen Spalten müssen in der Zeile x also noch mindestens 2 Einträge stehen, so daß in den Spalten $k+2$ bis $k+t-1$ in der Zeile x — und damit in den Zeilen mit positiven Koeffizienten insgesamt — höchstens $l_1 - 3$ viele Einträge stehen. Allein in Zeile $2t$ stehen dort aber mindestens $l_0 - 3$ viele Einträge, da ja in dieser Zeile in den Spalten $j < k-1$ wegen $t \ge k-1$ kein Eintrag steht.

□

Ähnlich wie Satz 6.8 erhalten wir :

Satz 6.11 *Sei* $4 < t < k-1$. *Dann gilt:*
a) Ein 5*–Zeilen–Versuch mit den Zeilen* 1, 2, 5, $2t$ *und* $2t$+1 *und den Vorzeichen* $+-+--$ *ist höchstens so effizient wie der verwandte mit den Zeilen* 1, 2, 5, 8 *und* 9.
b) Ein 5*–Zeilen–Versuch mit den Zeilen* 1, 2, 3, $2t$ *und* $2t$+1 *und den Vorzeichen* $+-+--$ *ist höchstens so effizient wie der verwandte mit den Zeilen* 1, 2, 3, 6 *und* 7.

Beweis: Sei $m := t-4$.

a) Die injektive Abbildung ψ ist hier gegeben durch:

$$\begin{array}{lrcl}
\nu = 1,2,3 & \mu \le 3: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
& 4 \le \mu \le k-m-1: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu+m} \\
& k-m \le \mu \le k-1: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu+m+4-k} \\
& \mu \ge k: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
 & & & & \\
\nu = 4,5 & \mu \le k-m-1: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu+m} \\
& k-m \le \mu \le k-1: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu+m+3} \\
& k \le \mu \le k+2: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu} \\
& \mu \ge k+3: & \psi(A)_{\nu,\mu} & = & A_{\nu,\mu+m \bmod 2k}
\end{array}$$

φ wirkt folgendermaßen: ($k = 12,\ t = 7,\ m = 3$)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
			7	8	9	10	11	4	5	6													
			7	8	9	10	11	4	5	6													
			7	8	9	10	11	4	5	6													
4	5	6	7	8	9	10	11	15	16	17				18	19	20	21	22	23	24	1	2	3
4	5	6	7	8	9	10	11	15	16	17				18	19	20	21	22	23	24	1	2	3

b) Wie Beweis von Satz 6.8

□

Als letztes Ergebnis schließen wir noch für $l_1 < l_0, 5$ den Fall $x = 3 \wedge t = 3$ aus:

Satz 6.12 *Sei* $l_1 < l_0$, $l_1 < 5$ *und* $2 < t < k-1$.*Dann gibt es keinen* 5*–Zeilen–Versuch mit den Zeilen* 1, 2, 3, $2t$, $2t$+1 *und den Vorzeichen* $+-+--$.

Beweis: Wir nehmen wieder an, es gäbe eine Konfiguration der betreffenden Zeilen und einen Versuch der sie annulliert. Wir betrachten die Spalten $t-1 < i < k$ und $k+1 < i < k+t$. In den Zeilen mit positiven Koeffizienten stehen dort höchstens $2(l_1-3)$ viele Einträge. Allein in der Zeile $2t$ stehen in diesen Spalten aber l_0-2 viele Einträge. Nun gilt

$$5 > l_1 \quad \Rightarrow \quad l_1 + 1 > 2l_1 - 4$$

$$\Rightarrow \quad l_0 > 2l_1 - 4 \quad \Rightarrow \quad l_0 - 2 > 2l_1 - 6$$

d.h. allein in Zeile $2t$ stehen mehr Einträge als in den Zeilen mit positiven Koeffizienten zusammen. Das ist ein Widerspruch!

□

Für $l_0 > l_1$ erhalten wir damit:

Der Worst Case mit Vorzeichen $+-+--$ ist einer der folgenden Fälle:

1. $x = 3 \wedge t = 2$
2. $x = 3 \wedge t = 3$ falls $l_1 \geq 5$
3. $x = 5 \wedge t = 3$
4. $x = 5 \wedge 4$

Bei $l_0 = l_1$ kommen noch die Versuche mit $t \geq k - 1$ hinzu.

6.4 Statitik zur Effizienz

In diesem Abschnitt dokumentieren wir erste Ergebnisse der Anwendung des Programms. O.B.d.A. beschränken wir uns auf 5–Zeilen–Versuche $\vec{y}$ mit $\mathrm{ggt}(y_{\sigma(1)}, y_{\sigma(2)}, y_{\sigma(3)}, y_{\sigma(4)}, y_{\sigma(5)}) = 1$.

Beobachtung: Die Versuche mit kleinen $|y_{\sigma(i)}|$ sind die effizientesten

Beobachtung: Die Versuche mit $|y_{\sigma(i)}|$ 2er–Potenzen sind die effizientesten.

Beobachtung: Die Versuche mit Vorzeichen $+--++$ sind die effizientesten

Test: Das Programm durchläuft alle 5–Zeilen–Versuche $\vec{y}$, mit den Parametern $k = 8,\ l_0 = 4,\ l_1 = 4,\ d_0 = 4,\ d_1 = 4,\ x = 3,\ t = 2$ und allen $y_{\sigma(i)}$ mit $|y_{\sigma(i)}| \leq 16$. In Klammern geben wir die Ergebnisse für $l_0 = 5$ und $l_1 = 4$ an.

Ergebnis:
1. Maximale Effizienz : $2^{-34.7}$ $(2^{-35.8})$ bei $(y_{\sigma(1)}, \ldots, y_{\sigma(5)}) = (1, -1, -1, 1, 1)$

2. Wir vergleichen Versuche $\vec{y}$, bei denen nicht alle $|y_{\sigma(i)}|$ 2er–Potenzen sind, mit Versuchen $\vec{z}$, bei denen alle $|z_{\sigma(i)}|$ 2er–Potenzen sind, so daß $\|\vec{z}\|_\infty \geq \|\mathcal{E}(\vec{y})\|_\infty$ gilt. Damit schließen wir einen Einfluß der in Beobachtung 6.4 dargelegten Tendenz aus.
Maximale Effizienz eines Versuches mit Nicht–2er–Potenzen ist $2^{-37.9}$ $(2^{-39.2})$ bei $(1, -1, -3, 1, 1)$ gegenüber $2^{-35.7}$ $(2^{-37.0})$ bei $(4, -4, -2, 2, 1)$ und $2^{-36.9}$ $(2^{-38.1})$ bei $(4, -4, -8, 4, 1)$.

3. Die maximale Effizienz eines Versuches mit Vorzeichen $+ - + - -$ beträgt $2^{38.7}$ $(2^{45.7})$

Test: $x = 3,\ t = 3$, sonst wie Test 6.4

Ergebnis: 1. Maximale Effizienz insgesamt: $2^{-39.7}(2^{-40.3})$ bei $(1, -1, -1, 1, 1)$

2. Maximale Effizienz eines Versuches mit Nicht–2er–Potenzen ist $2^{-40.6}$ bei $(3, -3, -3, 4, 4)$ gegenüber $2^{-40.1}$ bei $(4, -4, -4, 2, 1)$ ($2^{-45.5}$ bei $(2, -2, -3, 1, 1)$ gegenüber z.B. $2^{-41.3}$ bei $(4, -4, -4, 4, 1)$)

3. Die maximale Effizienz eines Versuches mit Vorzeichen $+-+--$ beträgt $2^{42.4}$. Für $l_0 = 5$ existiert nach Satz 6.12 kein erfolgreicher Versuch.

Beobachtung: Für die effizientesten Versuche gilt: $x = 3,\ t = 2$ und Vorzeichen $+ - - + +$

Bemerkung: Für $l_0 > l_1 < 5$ haben wir den Worst–Case schon auf 5 Fälle eingegrenzt. Für $l_0 = l_1$ testen wir 3 zusätzliche Fälle.

Test: $k = 8,\ l_1 = 4,\ d_0 = 4,\ d_1 = 4$. Restliche Parameter siehe Tabelle. Wir testen alle $\vec{y}$ mit $|y_{\sigma(i)}| \leq 16$, bei denen die $|y_{\sigma(i)}|$ alle 2er–Potenzen sind. Wir geben jeweils $-\log_2(\max(\mathcal{E}(\vec{y})))$ an.

Vorzeichen	x	t	$l_0 = 4$	Koeffizienten					$l_0 = 5$	Koeffizienten				
$+--++$	3	2	34.7	1	-1	-1	1	1	35.8	1	-1	-1	1	1
	3	3	39.1	1	-1	-1	1	1	40.3	1	-1	-1	1	1
	5	k-1	46.1	2	-2	-1	1	1	–					
$+-+--$	3	2	38.7	1	-1	1	-1	-1	45.7	2	-1	1	-1	-1
	3	3	42.4	2	-1	1	-1	-1	–					
	5	3	38.1	1	-1	2	-1	-1	43.5	2	-1	2	-1	-1
	5	4	41.2	2	-1	1	-1	-1	47.2	2	-1	2	-1	-1
	5	k-1	41.6	2	-2	2	-1	-1	–					

Unsere Beobachtungen werden hier noch einmal bestätigt. Die Tests wurden mit $k = 8$ durchgeführt, um die Laufzeit des Programms zu beschränken. Wegen der Deutlichkeit der beobachteten Tendenzen haben wir jedoch keinen Zweifel, daß alle Beobachtungen auch für $k > 8$ gelten. Gleiches gilt für den Fall $l_1 = 5$.

7 Das Preprocessing mit Parameter h

7.1 Die Modifikation des Preprocessings

Wie die ersten Ergebnisse des Progamms gezeigt haben, sind die Versuche $\vec{y}$ besonders effizient, bei denen die $|y_{\sigma(i)}|$ klein und 2er–Potenzen sind. Wir ändern daher die erste Preprocessing–Gleichung folgndermaßen ab:

$$r_k^{\nu+1} = r_\nu^\nu + \sum_{i=1}^{l_1} 2^{f(i,\nu)} r_{a(i,\nu)}^\nu$$

mit $\nu, \nu-1, k \in \{a(i,\nu)\}$, $f(j,\nu) \in \{h, \ldots, d_1-1\}$ für $a(j,\nu) = \nu$ und $f(i,\nu) \in \{0, \ldots, d_1-1\}$ sonst. Damit gilt: $\alpha_\nu^\nu = 2^{f(j,\nu)} + 1$. Somit ist ein Versuch $\vec{y}$ nur dann erfolgreich, wenn $y_{\sigma(2)}$ Vielfaches von $2^e + 1$ ist mit $e \geq h$. Für $h \geq 1$ ist also ein Versuch $\vec{y}$ nur dann erfolgreich, wenn mindestens ein $|y_{\sigma(i)}|$ **keine 2er–Potenz** ist. Die bisher effizientesten Versuche sind also nun nicht mehr erfolgreich. Es erscheint also plausibel, anzunehmen, daß durch die Modifikation des Preprocessings die minimale Workload eines Versuches erhöht wird.

Das Programm zur Berechnung der Effizienz eines Versuches folgendermaßen abgeändert:
Für eine Positionierung P werden anstatt bisher 2 (positiv und nicht–positiv) jetzt 4 Typen von Spalten einer Konfiguration $K \in P$ unterschieden:
1) Die Spalten mit einem α_ν^ν aber keiner -1
2) Die Spalten mit einer -1 aber keinem α_ν^ν
3) Die Spalten mit einem α_ν^ν und einer -1
4) Die restlichen Spalten
Diese Einteilung ist von der Wahl der Konfiguration unabhängig.

Sei P eine Positionierung. Für die Spalten vom Typ 1) definieren wir $H(\vartheta_P(j), j) := |A(P,j)|$, denn dieser Wert hängt nur von $\vartheta_P(j)$ und j ab. Das Programm errechnet nun für eine Spalte j vom Typ 1) bzw. Typ 3) $|A(P,j)|$ folgendermaßen:

1. Sei $z(j)$ die Zeile, in der in der Spalte j von K das α_ν^ν steht, d.h. $z(j) = \sigma^{-1}(2j-1)$. Berechne u und $v(1) < \ldots < v(u)$ mit

$$\sum_{i=1}^{u} 2^{v(i)-1} = \vartheta_P(j) - 2^{z(j)-1}$$

aus $\vartheta_P(j)$.

2. Durchlaufe alle Tupel $(e(1), \ldots, e(u), e)$ mit $e(i) \in \{0, \ldots, d_{v(i) \bmod 2} - 1\}$ und

$e \in \{h, \ldots, d_1 - 1\}$. Teste, ob

$$\sum_{i=1}^{u} 2^{e(i)} y_{\sigma(v(i))} + y_{2j-1}(2^e + 1) = \begin{cases} 0 & \text{falls } j \text{ vom Typ 1) ist} \\ y_{2(j-k)} & \text{falls } j \text{ vom Typ 3) ist} \end{cases}$$

gilt. Die Anzahl der Tupel, für die das gilt, ist $|A(P, j)|$.
Auch die Berechnung von $|\vec{y}|$ ändert sich. Statt (19) gilt jetzt nämlich

$$|\vec{y}| = \left[\binom{k-2}{l_1 - 3} d_1^{l_1 - 2} (d_1 - h) \right]^{m(\vec{y})} \left[\binom{k-1}{l_0 - 1} d_0^{l_0} \right]^{n - m(\vec{y})} \tag{22}$$

Bemerkung: Die Ergebnisse aus Abschnitt 6 sind auch für das modifizierte Preprocessing gültig. In den Beweisen wurden nicht die Werte der Einträge, sondern nur ihre Positionen betrachtet. Diese bleiben unverändert.

Wir werden in Abschnitt 9 empirisch überprüfen, ob die Modifikation des Preprocessings wirklich zu höheren Workloads führt. Dazu ist es jedoch notwendig — wie in Kapitel 6 für das alte Preprocessing — den Worst–Case enger einzugrenzen. Analog zu Abschnitt 6.3 führen die 5–Zeilen–Versuche mit $x = 3,\ t = 2$ und den Vorzeichen $+ - - + +$ zu den höchsten Effizienzen.

Das neue Preprocessing benötigt pro Signatur eine Multiplikation mehr, also höchstens $l_0 + l_1 + d_0 + d_1 - 3$ viele. Der Zugewinn an Sicherheit, also die Erhöhung der minimalen Workload, ist aber — wie wir sehen werden — ungleich höher.

7.2 Statistik zur Effizienz

Analog zu Abschnitt 6.4 versuchen wir nun für das modifizierte Preprocessing den Worst–Case enger einzugrenzen:

Beobachtung: Unabhängig von h gilt für die effektivsten Versuche $x = 3$, $t = 2$ und Vorzeichen $+ - - + +$

Test: $h = 1,\ k = 8,\ l_1 = 4,\ d_0 = d_1 = 5$ und $1 \leq |y_{\sigma(i)}| \leq 20$. Wir geben jeweils $-\log_2(\max(\mathcal{E}(\vec{y})))$ an. Wo kein erfogreicher Versuch gefunden wurde, schreiben wir "–"

Vorzeichen	x	t	$l_0 = 4$	Koeffizienten					$l_0 = 5$	Koeffizienten				
$+--++$	3	2	48.0	1	-3	-1	1	1	50.4	1	-3	-1	1	1
	3	3	56.3	1	-3	-1	1	1	58.2	1	-3	-1	1	1
	5	k-1	–						–					
$+-+--$	3	2	50.4	2	-3	1	-1	-1	55.7	4	-3	1	-1	-1
	3	3	52.7	4	-3	1	-1	-1	–					
	5	3	52.6	4	-3	1	-1	-1	57.9	4	-3	1	-1	-1
	5	4	54.2	4	-3	1	-1	-1	57.2	4	-3	1	-1	-1
	5	k-1	52.2	4	-3	1	-1	-1	–					

Test: $h = 3,\ k = 8,\ l_1 = 4,\ d_0 = 6,\ d_1 = 5$ und $1 \leq |y_{\sigma(i)}| \leq 20$. Wir geben jeweils $-log_2(\max(\mathcal{E}(\vec{y})))$ an.

Vorzeichen	x	t	$l_0 = 4$	Koeffizienten					$l_0 = 5$	Koeffizienten				
$+--++$	3	2	56.0	4	-9	-3	1	1	60.1	4	-9	-7	4	1
	3	3	–						–					
	5	k-1	–						–					
$+-+--$	3	2	58.0	16	-17	3	-5	-5	–					
	3	3	–						–					
	5	3	–						–					
	5	4	–						–					
	5	k-1	–						–					

Aufgrund dieser Beobachtung betrachten wir von nun an nur noch Versuche mit $x = 3$, $t = 2$ und Vorzeichen $+--++$. Für diesen Fall ist es möglich, das Programm wesentlich zu beschleunigen. Notwendig wird das vor allem für $h \geq 3$, da wir dann nicht mehr von kleinen $y_{\sigma(i)}$ für die effizientesten Versuche ausgehen können.

8 Versuche mit fünf Zeilen

8.1 Das beschleunigte Programm zur Berechnung der Effizienz

Wir beschränken uns auf die Versuche mit $x = 3$ und $t = 2$. In diesem Fall können wir das Programm wesentlich beschleunigen. Außerdem wird die Laufzeit für $k \geq l_0 + l_1 - 2$ von k unabhängig. Dann können wir z.B. den Fall $k = 12$ behandeln. Zunächst benötigen wir jedoch noch ein Lemma:

Lemma 8.1 *Sei ein 5–Zeilen–Versuch $\vec{y}$ mit $x = 3$ und $t = 2$ gegeben. Dann gilt für jede 5–Konfiguration, die von $\vec{y}$ annulliert wird:*

Von den Spalten $3 < i < k$ sind höchstens $\min(k - 4, l_0 + l_1 - 6)$ viele nicht–leer

Beweis: Es ist nur $\#\{\text{nichtleere Spalten } i \,|3 < i < k\} \leq l_0 + l_1 - 6$ zu zeigen.

Vorzeichen $+--++$: Die Anzahl der Wahleinträge in den Zeilen mit negativen Koeffizienten ist l_0+l_1-4. Wegen α_3^3 muß einer dieser Einträge in Spalte 3 stehen, wegen α_1^2 ein anderer in Spalte k.

Vorzeichen $+-+--$: Die Anzahl der Wahleinträge in den Zeilen mit positiven Koeffizienten ist höchstens $2l_1 - 6 \leq l_0 + l_1 - 6$.

In den Spalten $3 < i < k$ stehen keine Pflichteinträge. Deshalb ist die Anzahl der nichtleeren Spalten $3 < i < k$ höchstens so groß, wie die Anzahl der Wahleinträge der Zeilen mit positiven (bzw. negativen) Koeffizienten in diesen Spalten. Damit ist das Lemma bewiesen.

□

Sei $\vec{y}$ ein 5–Zeilen–Versuch. Wir definieren eine Äquivalenzklasse auf $M(\vec{y})$:

Definition 8.1 *Seien $P, Q \in M(\vec{y})$:*

$$P \sim Q :\Longleftrightarrow \underset{\substack{B\in P\\ C\in Q}}{\exists} \quad \underset{\pi\in S_{\{4,\ldots,k-1\}}}{\exists} \quad \left(\underset{3<i<k}{\forall} B_i^{\perp} = C_{\pi(i)}^{\perp} \;\wedge\; \underset{j\leq 3 \,\vee\, j\geq k}{\forall} B_j^{\perp} = C_j^{\perp} \right)$$

D.h. zwei Positionierungen sind dann zueinander äquivalent, wenn sie durch Permutation der Spalten $3 < i < k$ auseinander hervorgehen.

Satz 8.2 *Für $P \sim Q$ gilt:* $\quad |P \cap N(\vec{y})| = |Q \cap N(\vec{y})|$

Beweis: Die Permutation $\pi \in S_{\{4,\ldots,k-1\}}$, die die Äquivalenz von P und Q sichert, definiert eine Bijektion $\Pi : \ P \to Q$, für die trivialerweise gilt:

$$B \in P \cap N(\vec{y}) \quad \Leftrightarrow \quad \Pi(B) \in Q \cap N(\vec{y}) \quad \square$$

Definition 8.2 *Sei K eine Äquivalenzklasse und P ein Repräsentant. Dann sei für $2 \le j \le l_0 + l_1 - 6$*

$$e_j := \#\{3 < i_1 < \ldots < i_j < k \mid \ \vartheta_{P}(i_1) = \ldots = \vartheta_{P}(i_j) \neq 0 \ \wedge \underset{3<i<k}{\forall} \ i \notin \{i_1, \ldots, i_j\} \ \Rightarrow \ \vartheta_{P}(i) \neq \vartheta_{P}(i_1)\}$$

und $e_0 := \#\{3 < i < k \mid \vartheta_{P}(i) \neq 0\}$

Diese Werte sind von der Wahl des Repräsentanten P unabhängig.

Bemerkung: Man beachte, daß in den Spalten $3 < i < k$ nur Wahleinträge stehen. Deshalb ist die Mächtigkeit einer Äquivalenzklasse durch

$$\frac{(k-4)!}{(k-4-e_0)!} \prod_{i=2}^{l_0+l_1-6} (i!)^{-e_i} \tag{23}$$

gegeben

Das Programm berechnet $|P \cap N(\vec{y})|$ nun nicht mehr für jedes $P \in M(\vec{y})$, sondern für jede Äquivalenzklasse K nur einmal, und zwar für jenes P_{K}, für das $\vartheta_{P_K}(4) \ge \vartheta_{P_K}(5) \ge \ldots \ge \vartheta_{P_K}(k-1)$ gilt. Mit Satz 8.2 folgt dann

$$\sum_{Q \in K} |Q \cap N(\vec{y})| = |K||P_{K} \cap N(\vec{y})|$$

und damit

$$|N(\vec{y})| = \sum_{\text{Äq.Kl. } K} |K||P_{K} \cap N(\vec{y})|$$

Sei $w := \min(k-1, l_0+l_1-3)$. Für alle $w < j < k$ gilt wegen Lemma 8.1 $\vartheta_{P_K}(j) = 0$ und deshalb $|A(P_{K}, j)| = 1$. Daraus folgt

$$|P_{K} \cap N(\vec{y})| = \prod_{1 \le j \le w} |A(P, j)| \prod_{k \le j \le k+t} |A(P, j)|$$

Da die Spalten $w < j < k$ in $P_{_K}$ leer sind, müssen auch die x_i^j diese Werte nicht durchlaufen. Dadurch wird die Rechenzeit für $k \geq l_0 + l_1 - 2$ von k unabhängig.

Die Werte e_i für die Bestimmung von $|K|$ können bei der Berechnung der $|A(P,j)|$ gleich mitberechnet werden.

Bemerkung: Der Wert $A_1 = |A(P,1)|$ ist auch hier von der Positionierung unabhängig. Er kann also im Hauptprogramm einmal berechnet und am Ende auf *gesamt* aufmultipliziert werden.Wir setzen also

$$A_P := \prod_{2 \leq j \leq w} |A(P,j)| \prod_{k \leq j \leq k+t} |A(P,j)|$$

8.1.1 Die beschleunigte Routine Schleife

FOR $x_u^i = x_{u-1}^i$ TO $ende(i) - c_i + u$ DO
  IF $(x_u^i \leq w \vee x_u^i \geq k)$ THEN DO
    $\vartheta_{_P}(x_u^i) := \vartheta_{_P}(x_u^i) + 2^{i-1}$
    IF $u < \tilde{c}_i$ THEN DO
      $u := u + 1$
      Schleife
      $u := u - 1$
    ELSE IF $i < n$ THEN DO
      $u := 2$
      $i := i + 1$
      Schleife
      $i := i - 1$
      $u := \tilde{c}_i + 1$
    ELSE DO
      IF $\vartheta_{_P}(4) \leq \ldots \leq \vartheta_{_P}(k-1)$ THEN DO
        berechne A_P und $|K|$
        $gesamt := gesamt + |K| A_P$
    $\vartheta_{_P}(x_u^i) := \vartheta_{_P}(x_u^i) - 2^{i-1}$

Bemerkung: Die beschriebene Modifikation des Programms ist für beide Varianten des Preprocessings — ohne h und mit h — identisch, da die Berechnung der Werte $|A(P,j)|$ unverändert bleibt.

Das Programm wird z.B. für $k=8$, $l_0=l_1=4$, $d_0=d_1=5$ für die Variante ohne h um den Faktor 30 und für $k=12$, $l_0=5$, $l_1=4$, $d_0=d_1=5$, $h=3$ um den Faktor 4000 beschleunigt.

Da nach unseren Beobachtungen für die effektivsten 5–Zeilen–Versuche $x = 3$ und $t = 2$ gilt, führen wir von nun an alle Tests mit dem beschleunigten Programm durch. Das ermöglicht es uns, mehr Versuche zu testen, und wir können gesichertere Aussagen machen. Das ist wichtig, weil wir im Gegensatz zum alten Preprocessing (ohne h), beim neuen nicht davon ausgehen können, daß bei den effektivsten Versuchen die $|y_{\sigma(i)}|$ alle kleine 2er–Potenzen sind. Für $h > 3$ müssen wir sogar mit ziemlich großen Werten für die Koeffizienten $|y_{\sigma(i)}|$ rechnen. Außerdem können wir jetzt Tests mit $k > 8$ durchführen.

8.2 Vergleich des Preprocessings mit bzw. ohne h

Wir stellen in diesem Abschnitt das modifizierte Preprocessing mit dem neuen Parameter h dem alten gegenüber und vergleichen die Workloads. Dabei führen wir alle Tests nur für $n = 5$ durch, da die Rechenzeit des Programms mit $n \geq 6$ exponentiell anwächst. Die Tendenzen, die wir finden, sind so stark, daß wir auf weitere Tests mit $n \geq 6$ verzichten. Wir testen nur den Fall $t = 2$, $x = 3$ und Vorzeichen $+--++$. Dazu können wir das beschleunigte Programm verwenden. Wir finden:
Beobachtung: Bei dem modifizierten Preprocessing ist die maximale Effizienz von 5–Zeilen–Versuchen geringer als bei dem alten.

Beobachtung: Für das modifizierte Preprocessing gilt: je größer h ist, desto geringer ist die maximale Effizienz von 5–Zeilen–Versuchen.

Test: Wir ermitteln für $k = 12$, $l_0 = 4 \vee l_0 = 5$, $l_1 = 4$, $d_0 = d_1 = 5$
$-\log_2(\max(\mathcal{E}(\vec{y})))$ über alle 5–Zeilen–Versuche mit $|y_{\sigma(i)}| \leq 40$ für das alte Preprocessing und das neue mit $h = 0, 1, 2, 3$. Wir geben jeweils $-\log_2(\max(\mathcal{E}(\vec{y})))$ an.

l_0	ohne h	mit h			
		$h=0$	$h=1$	$h=2$	$h=3$
4	43.7	48.5	53.8	57.6	61.6
5	45.9	50.8	57.3	63.0	67.5

Für die effektivsten Versuche gilt jeweils $|y_{\sigma(i)}| \leq 16$

Dieses Ergebnis ist eindeutig:
Das modifizierte Preprocessing mit Parameter h ist sicherer als das alte ohne h. Wir legen uns daher von nun an auf das modifizierte Preprocessing mit Parameter h fest.

8.3 Optimale Parameter

Wir gehen von jetzt an von $h \geq 3$ aus. Für erfolgreiche Versuche gilt stets $|y_2| \geq 2^h + 1$. Für $h \geq 4$ sind deshalb sehr große Koeffizienten bei den effizientesten Versuchen zu erwarten. Um einigermaßen sicher zu sein, den effizientesten Versuch gefunden zu haben, müssten wir also die Effizienzen sehr vieler Versuche berechnen. Damit die Laufzeiten des Programms nicht zu groß werden, beschränken wir uns zunächst auf den Fall $h = 3$. Den Fall $h \geq 4$ werden wir in Abschnitt 12.1 diskutieren.

Wir hatten uns bereits auf $l_0 \geq l_1 \geq 4$ festgelegt. Für $l_1 = 5$ können wir wegen der langen Laufzeiten keine umfangreichen Tests durchführen und deshalb keine gesicherten Aussagen machen. Deshalb gehen wir von jetzt an stets von $l_1 = 4$ aus. Wegen $h = 3$ gilt $d_1 \geq 4$. Außerdem forden wir $d_0 \geq 2$ um eine ausreichende Randomisierung der Gleichungen zu gewährleisten. Wir versuchen nun, die optimalen Parameter zu bestimmen, die diesen Bedingungen genügen und für die das Preprocessing 16 Multiplikationen pro Signatur benötigt.

Test: Das Programm durchläuft für $k = 12$, $h = 3$ und die in der Tabelle angegebenen Parameter alle 5–Zeilen–Versuche $\vec{y}$ mit $t = 2$, $x = 3$, Vorzeichen $+--++$ und $|y_{\sigma(i)}| \leq 100$ und ermittelt $-\log_2(\max(\mathcal{E}(\vec{y})))$.

l_0	l_1	d_0	d_1	$-\log_2(\max(\mathcal{E}(\vec{y})))$	y_1	y_2	y_3	y_4	y_5
4	4	2	9	66.7	2	-9	-15	1	1
4	4	3	8	66.4	2	-9	-13	1	1
4	4	4	7	66.1	8	-9	-15	1	1
4	4	5	6	65.0	16	-9	-15	1	1
4	4	6	5	62.7	4	-9	-3	1	1
4	4	7	4	60.8	8	-9	-7	1	1
5	4	2	8	–					
5	4	3	7	67.6	4	-9	-13	1	1
5	4	4	6	67.7	8	-9	-13	1	1
5	4	5	5	66.6	1	-9	-41	32	4
5	4	6	4	64.0	1	-9	-25	16	2
6	4	2	7	–					
6	4	3	6	–					
6	4	4	5	–					
6	4	5	4	–					

Bisher haben wir in allen Tests nur nach der maximalen Effizienz eines Versuches gesucht, obwohl für die Sicherheit des Preprocessings die maximale Effizienz einer Attacke maßgebend ist. Diese ist gleich der Summe der Effizienzen aller Versuche, die sie erzeugen. Unsere nächste Beobachtung rechtfertigt a posteriori unser Vorgehen für den Fall $l_0 > l_1$.

Wird eine Attacke $\vec{c}$ von einer 5–Zeilen–Attacke $\vec{y}$ mit $t = 3$ und $x = 3$ mit dem Tupel $(\alpha_k^2, \ldots, \alpha_k^{k+2})$ erzeugt, so gilt

$$\begin{aligned} c_1 &= \alpha_k^2 y_1 \\ c_2 &= \alpha_k^3 y_3 - y_1 \\ c_3 &= \alpha_k^4 y_5 - y_3 \\ c_4 &= -y_5 \end{aligned} \tag{24}$$

Alle anderen c_i sind 0.

Beobachtung: Für $h \geq 3$ und $l_0 > l_1$ ist die maximale Effizienz von 5–Zeilen–Versuchen mit $t = 2$, $x = 3$ eine gute Näherung für die Effizinz der Attacken, die sie erzeugen.

Test: Wir listen für $k = 12$, $l_0 = 5$, $l_1 = 4$, $d_0 = 4$, $d_1 = 6$, $h = 3$ alle Versuche $\vec{y}$ mit $\mathcal{E}(\vec{y}) \geq 2^{-71}$ auf.

y_1	y_2	y_3	y_4	y_5	$-\log_2(\mathcal{E}(\vec{y}))$
8	-9	-13	1	1	67.7

Test: Wir listen für $k = 12$, $l_0 = 5$, $l_1 = 4$, $d_0 = 5$, $d_1 = 5$, $h = 3$ alle Versuche $\vec{y}$ mit $\mathcal{E}(\vec{y}) \geq 2^{-70}$ auf.

y_1	y_2	y_3	y_4	y_5	$-\log_2(\mathcal{E}(\vec{y}))$
1	-9	-41	32	2	67.7
1	-9	-41	32	4	66.6
1	-9	-41	32	8	67.2
1	-9	-41	32	16	68.2
4	-9	-7	1	1	67.8
8	-9	-7	1	1	67.5
16	-9	-7	1	1	67.8

Nach 24 können zwei Versuche $\vec{y}$ und $\vec{z}$ nur dann dieselbe Attacke erzeugen, wenn $y_5 = z_5$ gilt. Dies gilt nur für die letzten drei aufgeführten Versuche. Gälte für diese $c_2 = \alpha_k^3 y_3 - y_1 = \alpha_k^3 z_3 - z_1$, so wäre wegen $y_3 = z_3 = -7$ die Differenz $y_1 - z_1$ ein Vielfaches von 7. Es gibt also keine zwei effizienten Versuche, die dieselbe Attacke erzeugen.

Für $l_0 = l_1 = 4$ gilt diese Aussage nicht. Auch bei den 6–Zeilen–Versuchen werden wir für $l_0 = l_1 = 4$ in Abschnitt 9.1 Schwierigkeiten haben, die maximale Effizienz einer Attacke abzuschätzen. Wir werden uns dann auf $l_1 > l_0$ festlegen.

Bemerkung: Eine Attacke kann von Versuchen erzeugt werden, die eine verschiedene Anzahl von Zeilen verwenden. D.h. die Anzahl der Zeilen aus der sich eine Attacke ergibt ist nicht eindeutig festgelegt. So erzeugt der 6–Zeilen–Versuch $(y_1, y_2, y_3, 0, y_5, y_6, y_7, 0, \ldots)$ die gleichen Attacken wie der 7–Zeilen–Versuch $(y_1, y_2, y_3, y_4, y_5, y_6, y_7, 0, \ldots)$ mit beliebigem $y_4 \neq 0$, denn die geraden Komponenten eines Versuches gehen nicht in die Komponenten der Attacke mit ein. Beide Versuche könnten erfolgreich sein. Für Attacken, die von einem 5–Zeilen–Versuch mit $t = 2$ und $x = 3$ erzeugt wird, kann dieser Fall aber nicht eintreten, denn nur diese Versuche erzeugen Attacken $\vec{c}$ mit $c_i = 0$ für alle $i > 4$.

9 Versuche mit sechs Zeilen

9.1 Die Auswahl der Zeilen

Die Laufzeit des Programms steigt mit n exponentiell an. Um einigermaßen gesicherte Aussagen machen zu können, müssen wir das Programm für sehr viele Versuche laufen lassen. Praktisch sind solche ausführlichen Berechnungen jedoch schon für $n = 7$ nicht mehr durchführbar. Auch im Fall $n = 6$ können wir die Berechnungen nicht mehr mit der gleichen Ausführlichkeit durchführen wie im Fall $n = 5$. Glücklicherweise zeigt sich, daß im Fall $h = 3$ für die effizientesten 6–Zeilen–Versuche, die wir mit $|y_{\sigma(i)}| \leq 20$ finden, immer $|y_{\sigma(i)}| \leq 9$ gilt. Wir können dann also annehmen, den Worst–Case wirklich gefunden zu haben. Aus diesem Grunde führen wir alle Tests mit $h = 3$ durch.

Wir werden in diesem Abschnitt zeigen, daß von $n = 5$ auf $n = 6$ die **deutliche** Tendenz besteht, daß die maximale Effizienz von Versuchen mit zunehmendem n abnimmt. Eine solche Tendenz wäre ein starkes Argument dafür, den Worst–Case bei den Versuchen mit $n = 5$, $t = 2$, $x = 3$ anzunehmen.

Für einen erfolgreichen 6–Zeilen–Versuch gilt allgemein (o.B.d.A. sei $\sigma(1) = 1$):

$$\sigma(2) = 2, \quad \sigma(3) = x_1, \quad \sigma(4) = x_2, \quad \sigma(5) = 2t, \quad \sigma(6) = 2t + 1$$

mit $2 < x_1 < x_2 < 2t$. Ähnlich wie in Abschnitt 6 kann man auch im Fall $n = 6$ den Worst–Case enger eingrenzen. So gilt:

Satz 9.1 *Für jeden 6–Zeilen–Versuch mit $3 \leq t \leq k-1$ gibt es einen 6–Zeilen–Versuch mit $3 \leq t \leq 5$, der mindestens genauso effizient ist.*

Wir verzichten auf den langen, aber einfachen Beweis.

Test: Für $k = 6$, $h = 3$ und die Parameter l_0, l_1, d_0, d_1 wie in der Tabelle angegeben durchläuft das Programm alle $3 \leq t \leq 5$ und $2 < x_1 < x_2 < 2t$. Die $|y_{\sigma(i)}|$ laufen von 1 bis 10. Wir geben alle Tupel (t, x_1, x_2) an, für die Versuche mit Effizienz $> 2^{-54}$ bei $l_0 = 4$, $> 2^{-57.5}$ bei $l_0 = 5$ oder > 0 bei $l_0 = 6$ gefunden wurden. Für diese Tupel geben wir die größte gefundene Effizienz an.

l_0, l_1, d_0, d_1			4455	5455	6445
t	x_1	x_2	$-\log_2(\max(\mathcal{E}(\vec{y})))$	$-\log_2(\max(\mathcal{E}(\vec{y})))$	$-\log_2(\max(\mathcal{E}(\vec{y})))$
3	3	4	53.0	54.8	52.5
3	3	5	52.3	54.9	54.5
3	4	4	53.8	57.1	55.8
4	3	4	53.3	57.2	–
4	3	5	53.5	–	–
4	3	7	52.3	56.0	–
4	5	7	54.4	56.0	–
4	6	7	53.8	57.1	–
5	3	5	52.1	–	–
5	3	7	52.0	–	–
5	3	9	51.4	–	–
5	5	7	51.6	55.7	–
5	5	9	51.5	57.4	–
5	6	7	52.3	59.4	–
5	7	9	51.6	–	–
5	8	9	51.1	–	–

Im Fall $t = 3$ lassen sich die erfolgreichen Versuche auf wenige Fälle einschränken. Die effizientesten scheinen sich sogar auf 2 Fälle zu reduzieren. In diesen beiden Fällen läßt sich das Programm dann ähnlich wie im Fall $n = 5$ erheblich beschleunigen. Dadurch werden umfangreichere und damit auch aussagekräftigere Tests möglich. Im Fall $l_0 = l_1$ scheint für den Worst–Case jedoch nicht $t = 3$ zu gelten. Um ausreichend sichere Aussagen machen zu können legen wir uns deshalb im folgenden auf $l_0 > l_1$ fest.

Von nun an gehen wir von $t = 3$, $x_1 = 3$, $x_2 = 4$ aus. Es bleiben für die Wahl von x_1 und x_2 noch drei Möglichkeiten:
1. $x_1 = 3$, $x_2 = 4$
2. $x_1 = 3$, $x_2 = 5$
3. $x_1 = 4$, $x_2 = 5$

9.2 Die Wahl der Vorzeichen

Wir untersuchen nun, welche Möglichkeiten der Vorzeichen der $y_{\sigma(i)}$ für erfolgreiche Versuche möglich sind. Es gilt o.B.d.A. $y_{\sigma(1)} > 0$ und $y_{\sigma(2)} < 0$

9.2.1 $x_1 = 3$, $x_2 = 4$

Sei $\vec{y}$ ein 6–Zeilen–Versuch mit $x_1 = 3$, $x_2 = 4$ und $t = 3$ und B eine Konfiguration, die von ihr annulliert wird.

Skizze ($k = 12$):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
x											x	-	-	-	-	-	-	-	-	-	-	-	-
x												-1	-	-	-	-	-	-	-	-	-	-	-
-	x											x	-	-	-	-	-	-	-	-	-	-	-
-	x												-1	-	-	-	-	-	-	-	-	-	-
-	-	x												-1	-	-	-	-	-	-	-	-	-
-	-	-	x											x	-	-	-	-	-	-	-	-	-

Wegen Spalte $k+3$ haben y_{2t} und y_{2t+1} dasselbe Vorzeichen. Wegen Spalte $k+2$ besitzt auch y_{x_2} dieses Vorzeichen. Hätte auch y_{x_1} dieses Vorzeichen, so müßte wegen Spalte $k+1$ auch y_2 dieses Vorzeichen haben, d.h. $y_2, y_{x_1}, y_{x_2}, y_{2t}, y_{2t+1} < 0$. Allein in der Zeile 2 stehen aber mehr Einträge als in Zeile 1. Das ist ein Widerspruch zu der Annahme, daß B von $\vec{y}$ annulliert wird.

Es bleiben also 2 Möglichkeiten für die Vorzeichen :

a) $+--++$
b) $+-+--$

9.2.2 $x_1 = 3$, $x_2 = 5$

Sei $\vec{y}$ ein 6–Zeilen–Versuch mit $x_1 = 3$, $x_2 = 5$ und $t = 3$ und B eine Konfiguration, die von ihr annulliert wird.

Skizze ($k = 12$):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
x											x	-	-	-	-	-	-	-	-	-	-	-	-
x												-1	-	-	-	-	-	-	-	-	-	-	-
-	x											x	-	-	-	-	-	-	-	-	-	-	-
-	-	x											x	-	-	-	-	-	-	-	-	-	-
-	-	x												-1	-	-	-	-	-	-	-	-	-
-	-	-	x											x	-	-	-	-	-	-	-	-	-

Wieder besitzen y_{2t} und y_{2t+1} dasselbe Vorzeichen. Wegen Spalte $k+2$ gilt aber $\mathrm{sgn}(y_{x_2}) = -\mathrm{sgn}(y_{2t})$. Also bleiben 4 Möglichkeiten für die Vorzeichen:

c) $+-++--$
d) $+-+-++$
e) $+--+--$
f) $+---++$

Wir zeigen nun, daß im Fall $h = d_1 - 1 \geq 3$ ein erfolgreicher Versuch mit Vorzeichen e) oder f) sehr hohe Koeffizienten haben muß.

Wegen Spalte 1 gilt

$$|y_1|\gamma = |y_2|2^{i_1}$$

mit $\gamma = 2^d + 1$ und $d \geq h$.
Wegen Spalte 2 gilt

$$|y_{x_1}|\gamma + |y_2|2^{i_2} = |y_1|2^{i_3}$$

Aus diesen beiden Gleichungen ergibt sich:

$$|y_{x_1}|\gamma\, 2^{i_1} + |y_1|\gamma\, 2^{i_2} = |y_1|2^{i_1+i_3}$$

Daraus folgt $\gamma/|y_1|$ und damit $\gamma^2/|y_2|$. Wegen $\gamma \geq 9$ ergibt sich also $|y_2| \geq 81$.

Bemerkung: Man kann auch zeigen, daß für $h \geq 3,\ d_1 \leq 5$ und $d_0 \leq 6$ erfolgreiche Attacken mit Vorzeichen e) oder f) nur mit $y_1 \geq 8,\ y_2 \leq -9$ und $y_{x_1} \leq -7$ möglich sind.

9.2.3 $x_1 = 4$, $x_2 = 5$

Sei $\vec{y}$ ein 6–Zeilen–Versuch mit $x_1 = 4$, $x_2 = 5$ und $t = 3$ und B eine Konfiguration, die von ihr annulliert wird.

Skizze ($k = 12$):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
x											x	-	-	-	-	-	-	-	-	-	-	-	-
x												-1	-	-	-	-	-	-	-	-	-	-	-
-	x												-1	-	-	-	-	-	-	-	-	-	-
-	-	x											x	-	-	-	-	-	-	-	-	-	-
-	-	x												-1	-	-	-	-	-	-	-	-	-
-	-	-	x											x	-	-	-	-	-	-	-	-	-

Gilt $\text{sgn}(y_{x_2}) = \text{sgn}(y_{2t})$, so folgt wegen Spalte $k+2$ $\text{sgn}(y_{x_1}) = \text{sgn}(y_{x_2})$ und wegen Spalte $k+1$ $\text{sgn}(y_2. = \text{sgn}(y_{x_1})$, d.h. $y_2,\ y_{x_1},\ y_{x_2},\ y_{2t},\ y_{2t+1} < 0$. Es folgt wie bei 1. ein Widerspruch.

Also gilt: $\text{sgn}(y_{x_2}) = -\text{sgn}(y_{2t})$. Es bleiben 4 Möglichkeiten für die Vorzeichen:

c) $+-++--$
d) $+-+-++$
e) $+--+--$
f) $+---++$

Analog zu 2. sieht man, daß für $h = d_1 - 1 \geq 3$ erfolgreiche Versuche mit Vorzeichen e) oder f) hohe Koeffizienten haben müssen und für $h \geq 3,\ d_1 \leq 5$ und $d_0 \leq 5$ bei erfolgreichen Versuchen mit e) oder f) $y_1 \geq 8,\ y_2 \leq -9$ und $y_{x_1} \leq -7$ gilt.

9.3 Statistik zur Effizienz

Beobachtung: Die effizientesten 6–Zeilen–Versuche sind die mit $x_1 = 3,\ x_2 = 4$ und Vorzeichen a) oder mit $x_1 = 3,\ x_2 = 5$ und Vorzeichen d)

Test: $k = 7,\ l_0 = 5,\ l_1 = 4,\ d_0 = d_1 = 5,\ h = 3,\ t = 3$ und $|y_{\sigma(i)}| < 18$.

t	x_1	x_2	Vorzeichen	$-\log_2(\max(\mathcal{E}(\vec{y})))$	y_1	y_2	y_{x_1}	y_{x_2}	y_{2t}	y_{2t+1}
3	3	4	$+--++$	59.4	1	-9	-1	1	1	1
3	3	4	$+-+---$	68.7	8	-17	12	-5	-5	-5
3	3	5	$+-++--$	61.6	2	-9	1	2	-1	-1
3	3	5	$+-+-++$	58.8	1	-9	1	-1	1	1
3	3	5	$+--+--$	–						
3	3	5	$+---++$	63.4	8	-9	-7	-1	2	2
3	4	5	$+-++--$	61.7	2	-9	2	2	-1	-1
3	4	5	$+-+-++$	63.3	8	-9	1	-1	1	1
3	4	5	$+--+--$	68.4	8	-9	-7	9	-2	-2
3	4	5	$+---++$	65.4	8	-9	7	-1	1	1
4	3	4	$+--++$	60.4	1	-9	-1	1	1	1

Wir beschränken uns im folgenden auf 2 Typen von Versuchen:

1. $t = 3$, $x_1 = 3$, $x_2 = 4$ und Vorzeichen $+ - + - --$
2. $t = 3$, $x_1 = 3$, $x_2 = 5$ und Vorzeichen $+ - + - ++$

Für diese beiden Fälle ist es möglich, die Berechnung von $\mathcal{E}(\vec{y})$ — analog zu Abschnitt 8 — erheblich zu beschleunigen.

9.4 Die Berechnung der Effizienz von Attacken

Anders als bei den 5–Zeilen–Versuchen, gibt es im Fall $n = 6$ verschiedene effiziente Versuche, die dieselbe Attacke erzeugen.

Definition 9.1 *Wir bezeichnen eine Attacke $\vec{c}$ als n–Zeilen–Attacke, wenn n die minimale Anzahl von Zeilen ist, mittels der sich $\vec{c}$ erzeugen läßt, d.h. wenn*

$$n = \min(i \mid \underset{i\text{–Z.–V. } \vec{y}}{\exists} \vec{y} \text{ erzeugt } \vec{c})$$

gilt.

Für Versuche vom Typ 1. bzw. 2. ist eine erzeugte Attacke gegeben durch

	1.	2.
c_1	$y_1\alpha_k^2$	$y_1\alpha_k^2$
c_2	$y_3\alpha_k^3 - y_1$	$y_3\alpha_k^3 - y_1$
c_3	$-y_3$	$y_5\alpha_k^4 - y_3$
c_4	$y_7\alpha_k^5$	$y_7\alpha_k^5 - y_5$
c_5	y_7	y_7

Im Fall 1. ist $c_3 > 0$, im Fall 2. dagegen $c_3 < 0$. Ein Versuch mit 1. kann also nicht die gleiche Attacke erzeugen wie einer mit 2.. Um die effizienteste 6–Zeilen–Attacke zu ermitteln, brauchen wir also nur Versuche desselben Types aufzusummieren. Hierbei vernachlässigen wir alle Versuche, die nicht vom Typ 1. oder 2. sind. Wir rechtfertigen dies mit ihrer wesentlich geringeren Effizienz. Ebenso vernachlässigen wir alle Versuche mit mehr als 6 Zeilen. Diese können in manchen Fällen dieselbe Attacke erzeugen wie ein 6–Zeilen–Versuch. Wir werden jedoch in Kapitel 10 sehen, daß sie wesentlich ineffizienter sind als die effizientesten 6–Zeilen–Versuche. Ihr Beitrag zu einer 6–Zeilen–Attacke ist deshalb vernachlässigbar. Auf die Möglichkeit, daß 5–Zeilen–Versuche dieselbe Attacke

erzeugen wie ein 6–Zeilen–Versuch, werden wir am Ende dieses Kapitels zurückkommen. Wir schätzen also die maximale Effizienz einer 6–Zeilen–Attacke aus den Effizienzen der 6–Zeilen–Versuche vom Typ 1. und 2. mit Hilfe eines Computerprogramms ab. Das Programm durchläuft alle 6–Zeilen–Versuche $\vec{y}$ vom Typ 1. bzw. 2. mit $\|\vec{y}\|_\infty < K$ für ein positives K. Für jeden Versuch durchläuft es alle Tupel $(\alpha_k^2, \ldots, \alpha_k^5)$ mit $\alpha_k^\nu \in \{2^0, \ldots, 2^{d_1}\}$ und bildet die von $\vec{y}$ mit $(\alpha_k^2, \ldots, \alpha_k^5)$ erzeugte Attacke $\vec{c}$. Dann summiert es $\mathcal{E}(\vec{y})$ auf $\mathcal{E}(\vec{c})$ auf. Aus programmtechnischen Gründen müssen wir uns jedoch auf Attacken $\vec{c}$ mit $\|\vec{c}\|_\infty \leq C$ beschränken. Für Versuche vom Typ 1. setzen wir $C = 64$ und $C = 16$ für Versuche vom Typ 2..

Mit dem Programm berechnen wir also

$$D_1 := \max_{\substack{\vec{c} \in \mathbf{Z}_q^{k+2} \\ \|\vec{c}\|_\infty \leq C}} \sum_{\substack{\vec{y} \text{ vom Typ 1.} \\ \vec{y} \text{ erzeugt } \vec{c} \\ \|\vec{y}\|_\infty \leq K}} \mathcal{E}(\vec{y})$$

bzw.

$$D_2 := \max_{\substack{\vec{c} \in \mathbf{Z}_q^{k+2} \\ \|\vec{c}\|_\infty \leq C}} \sum_{\substack{\vec{y} \text{ vom Typ 2.} \\ \vec{y} \text{ erzeugt } \vec{c} \\ \|\vec{y}\|_\infty \leq K}} \mathcal{E}(\vec{y})$$

Die maximale Effizienz einer Attacke schätzen wir dann durch

$$D := \max(D_1, D_2)$$

ab.

Für $l_0 = 5$ und $l_1 = 4$ setzen wir $K = 16$ und für $l_0 = 6$ und $l_1 = 4$ setzen wir $K = 10$. Dies ist notwendig, um die Laufzeit des Programms zu beschränken. Daß wir für diese Werte dennoch eine gute Abschätzung für $\mathcal{E}(\vec{c})$ erhalten, zeigt uns die folgende Beobachtung:

Beobachtung: Es gibt keine effizienten 6–Zeilen–Versuche $\vec{y}$ mit $\|\vec{y}\|_\infty > 10$

Test: Das Programm durchläuft für $k = 12$, $l_0 = 5$, $l_1 = 4$, $d_0 = 4$, $d_1 = 6$ alle Versuche $\vec{y}$ mit $\|\vec{y}\|_\infty \leq 24$. Der effizienteste mit $\|\vec{y}\|_\infty > 10$ ist $\vec{y} = (4, -9, -9, 17, 1, 1)$ und hat die Effizienz $2^{-78.1}$. Der effizienteste überhaupt ist $\vec{y} = (1, -9, -1, 1, 1, 1)$ und hat die Effizienz $2^{-74.7}$.

Für die Parameter $l_0 = 6$ und $l_1 = 4$ läßt sich ein solcher Test wegen der langen Laufzeit nicht mehr durchführen.

9.5 Vergleich der Fälle $n = 5$ und $n = 6$

In dem folgenden Test ermitteln wir für $k = 12$, $h = 3$ und $l_0 = 6$, $l_1 = 4$ oder $l_0 = 5$, $l_1 = 4$ und alle d_0, d_1 mit $d_0 \geq 2$, $d_1 \geq 4$ und $l_0+l_1+d_0+d_1 = 19$ den Wert D. Dies geschieht in der im letzten Abschnitt beschriebenen Weise. Wir stellen dem stets die maximale Effizienz eines 5–Zeilen–Versuches gegenüber. Diese war nach unseren Beobachtungen eine gute Abschätzung für die maximale Effizienz einer 5–Zeilen–Attacke.

Test: Sei $k = 12$ und $h = 3$. Wir ermitteln für folgende Parameter den Wert D. Dabei durchläuft das Programm alle 6–Zeilen–Versuche mit $\|\vec{y}\|_\infty \leq 16$ für $l_0 = 5$, $l_1 = 4$ und $\|\vec{y}\|_\infty \leq 10$ für $l_0 = 6$, $l_1 = 4$.

				$n = 6$	$n = 5$
l_0	l_1	d_0	d_1	D	$-\log_2(\max(\mathcal{E}(\vec{y})))$
5	4	2	8	72.0	–
5	4	3	7	71.2	67.6
5	4	4	6	69.7	67.7
5	4	5	5	67.3	66.6
5	4	6	4	64.3	64.0
6	4	2	7	73.1	–
6	4	3	6	73.6	–
6	4	4	5	72.4	–
6	4	5	4	72.3	–

Bemerkung: Wir haben bisher die Möglichkeit vernachlässigt, daß 5–Zeilen–Versuche dieselbe Attacke erzeugen wie ein 6–Zeilen–Versuch. Dies ist jedoch für 5–Zeilen–Versuche mit $t = 2$ unmöglich, da für diese $c_i = 0$ für alle $i > 4$ gilt. Die 5–Zeilen–Versuche mit $t \geq 3$ sind aber wesentlich — d.h. mindestens um den Faktor 20 — ineffizienter als die mit $t = 2$. Es ist deshalb ausgeschlossen, daß eine von 5–Zeilen–Versuchen mit $t \geq 3$ und 6–Zeilen–Versuchen erzeugte Attacke effizienter ist als die von 5–Zeilen–Versuchen mit $t = 2$ erzeugten.

Diese Ergebnisse legen die Vermutung nahe, daß die effizienteren Attacken von Versuchen mit wenigen Zeilen erzeugt werden. Zumindest steigt die Effizienz nicht mit der Anzahl der verwendeten Zeilen an. Um diese Vermutung zu fundieren, müssen wir zusätzliche Tests mit $n = 7$ durchführen. Für $n = 8$ läßt die hohe Laufzeit des Programms keine aussagekräftigen Tests mehr zu.

10 Versuche mit sieben Zeilen

Die hohen Laufzeiten des Programms erlauben im Fall $n = 7$ keine so ausführlichen Tests wie im Fall $n = 5$ oder $n = 6$. Wir testen deshalb stets nur die Versuche mit $|y_{\sigma(1)}| = |y_{\sigma(3)}| = |y_{\sigma(4)}| = |y_{\sigma(5)}| = |y_{\sigma(6)}| = |y_{\sigma(7)}| = 1$. Wir rechtfertigen dies mit der Beobachtung, daß von $n = 5$ auf $n = 6$ die deutliche Tendenz zu kleineren Koeffizienten bei den effizientesten Versuchen bestand und diese schon im Fall $n = 6$ sehr klein waren. Außerdem führen wir alle Tests mit $k = 7$ durch. Es ist uns im Fall $n = 7$ nämlich nicht mehr möglich, das Programm zu beschleunigen.

Wir werden zunächst die Auswahl der Zeilen ermitteln, für die die höchsten Effizienzen von Versuchen erzielt werden. Für diese Auswahl ermitteln wir dann die effizientesten Versuche mit $|y_{\sigma(1)}| = |y_{\sigma(3)}| = |y_{\sigma(4)}| = |y_{\sigma(5)}| = |y_{\sigma(6)}| = |y_{\sigma(7)}| = 1$. Den effizientesten gefundenen nehmen wir dann als den effizientesten 7–Zeilen–Versuch überhaupt an. Anschließend vergleichen wir die maximalen Effizienzen von 5–, 6– und 7–Zeilen–Versuchen. Aus diesem Vergleich werden wir folgern, daß die effizientesten Attacken von Versuchen mittels 5 aufeinanderfolgenden Zeilen erzeugt werden. Das bedeutet, die maximale Effizienz von Attacken ist in etwa durch die in Abschnitt 8.3 gefundenen maximalen Effizienzen von 5–Zeilen–Versuchen gegeben.

10.1 Die Auswahl der Zeilen

Für einen erfolgreichen 7–Zeilen–Versuch gilt allgemein (o.B.d.A. sei $\sigma(1) = 1$):

$$\sigma(2) = 2,\ \ \sigma(3) = x_1,\ \ \sigma(4) = x_2,\ \ \sigma(5) = x_3,\ \ \sigma(6) = 2t,\ \ \sigma(7) = 2t + 1$$

mit $2 < x_1 < x_2 < x_3 < 2t$. Ähnlich wie für $n = 5$ und $n = 6$ kann man für $n = 7$ den Worst–Case enger eingrenzen. So gilt:

Satz 10.1 *Für jeden 7–Zeilen–Versuch mit $3 \leq t \leq k-1$ gibt es einen 7–Zeilen–Versuch mit $3 \leq t \leq 6$, der mindestens genauso effizient ist.*

Wir verzichten auf den langen, aber einfachen Beweis.

Test: Für $k = 7$, $l_0 = 5$, $l_1 = 4$, $d_0 = 5$, $d_1 = 5$, $h = 3$ testen wir alle Versuche mit $t \leq 6$, $2 < x_1 < x_2 < x_3 < 2t$ und $y_1 = 1,\ y_2 = -9$ und $|y_{\sigma(i)}| = 1$ für

$i = 3, \ldots, 7$. Wir geben alle Wertetupel (t, x_1, x_2, x_3) an, für die ein Versuch y mit $\mathcal{E}(\vec{y}) > 2^{-63.5}$ gefunden wurde.

t	x_1	x_2	x_3	$-\log_2(\max(\mathcal{E}(\vec{y})))$
3	3	4	5	61.8
4	3	5	7	62.3
4	3	6	7	63.2
4	5	6	7	62.8
5	3	5	7	63.2
5	3	5	9	62.4
5	3	7	9	62.4
6	3	5	7	62.4
6	3	5	9	62.3
6	3	5	11	61.8
6	3	7	9	62.2
6	3	7	11	61.8
6	3	9	11	61.8
6	3	10	11	62.7
6	5	7	9	62.3
6	5	7	11	62.2
6	5	9	11	62.4
6	5	10	11	63.3
6	7	9	11	62.4
6	7	10	11	63.2

Wie man sieht, sind die Versuche mittels 7 aufeinanderfolgenden Zeilen mit die effizientesten. Wir gehen daher von nun an von $\sigma(i) = i$ für alle i aus.

10.2 Das Verhalten der maximalen Effizienz von Versuchen bei wachsendem n

Wegen der hohen Laufzeiten des Programms können wir die folgenden Test nicht mit $k = 12$ durchführen.

Test: Wir ermitteln für $h = 3$ und die gleichen Parameter $l_0,\ l_1,\ d_0,\ d_1$ wie im Fall $n = 6$ die maximalen Effizienzen von 7–Zeilen–Versuchen mittels der ersten

7 Zeilen mit $y_1 = 1,\ y_2 = -9$ und $|y_{\sigma(i)}| = 1$ für $i = 3, \ldots, 7$. Diese stellen wir den entsprechenden Werten für $n = 5$ und $n = 6$ gegenüber. Wir geben jeweils $-\log_2(\max(\mathcal{E}(\vec{y})))$ an.

k	l_0	l_1	d_0	d_1	$n = 5$	$n = 6$	$n = 7$
11	5	4	2	8	–	71.5	74.8
11	5	4	3	7	66.1	72.5	76.5
11	5	4	4	6	66.2	71.1	74.4
11	5	4	5	5	65.1	68.5	72.4
11	5	4	6	4	62.4	65.2	72.8
10	6	4	2	7	–	68.6	74.2
10	6	4	3	6	–	70.3	75.3
10	6	4	4	5	–	71.1	73.7
10	6	4	5	4	–	69.3	72.2

Beim Vergleich von $n = 5$ und $n = 6$ hatte es sich gezeigt, daß der Abstand zwischen der maximalen Effizienz von 5– und 6–Zeilen–Versuchen so groß ist, daß auch die 5–Zeilen–Attacken deutlich effizienter sind als die 6–Zeilen–Attacken. Der Abstand zwischen der maximalen Effizienz von 6– und 7–Zeilen–Versuchen ist nun ähnlich groß. Die Tendenz, die wir von $n = 5$ auf $n = 6$ beobachtet haben, scheint sich also fortzusetzen:

> *Die maximale Effizienz von n–Zeilen–Attacken nimmt mit zunehmendem n ab.*

Daraus folgt, daß die Sicherheit des Preprocessings — d.h. die minimale Erfolgswahrscheinlichkeit eines Angriffes — durch die maximale Effizienz von 5–Zeilen–Attacken gegeben ist. Nach unseren Beobachtungen in Abschnitt 8.3 ist diese gleich der maximale Effizienz von 5–Zeilen–Versuchen. Gibt es keinen erfolgreichen 5–Zeilen–Versuch, so ist die Sicherheit durch die maximale Effizienz einer 6–Zeilen–Attacke gegeben.

Für $k = 12$ und $h = 3$ erhalten wir für die Sicherheit des Preprocessings also folgende Werte:

l_0	l_1	d_0	d_1	Sicherheit
5	4	2	8	72.0
5	4	3	7	67.6
5	4	4	6	67.7
5	4	5	5	66.6
5	4	6	4	64.0
6	4	2	7	73.1
6	4	3	6	73.6
6	4	4	5	72.4
6	4	5	4	72.3

Besonders geeignet erscheinen die Parameter $l_0 = 6$, $l_1 = 4$, $d_0 = 3$, $d_1 = 6$, da wir in diesem Fall sogar $h = 5$ setzen können. Auf die Erhöhung des Parameters h werden wir in Abschnitt 12.1 noch einmal zurückkommen.

11 Eine Gittertheoretische Attacke

11.1 Die Attacke

Eine ganz andere Art von Angriff ergibt sich aus einer gittertheoretischen Betrachtung des Preprocessings. Ein Versuch $\vec{y}$ mit $\vec{y}Z = 0$ ist eine simultane $\mathbf{Z}$–Relation zu den Spalten von Z. Sei $\langle\ ,\ \rangle$ das Standard–Skalarprodukt im $\mathbf{R}^n$.

Definition 11.1 *Zu $x = (x_1, \ldots, x_n) \in \mathbf{R}^n$ ist $m = (m_1, \ldots, m_n) \in \mathbf{Z}^n \backslash \{0\}$ eine* **Z–Relation***, wenn $\langle m, x\rangle = \sum_{i=1}^{n} x_i m_i = 0$ gilt.*

Seien $x^1, \ldots, x^q \in \mathbf{R}^n$. $m \in \mathbf{Z}^n \backslash \{0\}$ heißt **simultane Z–Relation** *zu $x^1, \ldots, x^q$, wenn es zu jedem x^i eine $\mathbf{Z}$–Relation ist.*

Sei $Z = (x^1, \ldots, x^{2k})^T \in \mathcal{R}$ gegeben. Dann bilden die simultanen $\mathbf{Z}$–Relationen zu $x^1, \ldots, x^{2k}$ einschließlich dem Nullvektor ein Gitter $L_{x^1,\ldots,x^{2k}}$. Es ist nun möglich die Länge λ_1 des kürzesten Gittervektoren abzuschätzen. Hat ein Angreifer eine von der Wahl von $Z \in \mathcal{R}$ unabhängige Abschätzung $\lambda_1 \leq K$, so gibt es also für jedes $Z \in \mathcal{R}$ mindestens ein $\vec{y}_0$ mit $\|\vec{y}_0\| \leq K$ und $\vec{y}_0 Z = 0$.

Sei (a_i, e_i) die Signatur der Runde i. Der Angreifer müßte nun alle $\vec{y}$ mit $\|\vec{y}\| \leq K$ und dann alle möglichen α_k^{i+1} für alle $i \equiv 1 \bmod 2$ mit $y_{2i-1} \neq 0$ durchlaufen. Mit der Gleichung

$$\sum_{i=1}^{k+1} y_{2i-1}(\alpha_k^{i+1} r_k^i - r_k^{i+1}) = 0 \tag{25}$$

und den Gleichungen $a_i = se_i$ für $i = 1, \ldots, k+1$ kann er dann einen Wert für s ermitteln und mit der Gleichung $v = \alpha^{-s}$ überprüfen, ob der so ermittelte Wert mit s übereinstimmt. Für mindestens ein $\vec{y}$ wird das der Fall sein.

Da die ungeraden Komponenten von $\vec{y}$ nicht in die Gleichung (25) eingehen, muß der Angreifer nur alle Tupel $y' = (y_1, y_3, \ldots, y_{2k-1}, y_{2k+1})$ mit $\|y'\| \leq K$ durchlaufen. Daraus ergibt sich eine Workload von

$$\begin{aligned} \#\{\vec{z} \in \mathbf{Z}^{k+1} \mid \|\vec{z}\| \leq K\}/2 &\approx \mathrm{Vol}(S_{k+1}(K))/2 \\ &= \frac{K^{k+1}\pi^{(k+1)/2}}{2 \cdot \Gamma((k+3)/2)} \end{aligned} \tag{26}$$

Dabei haben wir das Durchlaufen der α_k^i unberücksichtigt gelassen.

11.2 Eine obere Schranke K für λ_1

Die Abschätzung, die wir nun herleiten, ist an [4] angelehnt. Wir setzen Grundkenntnisse in Gittertheorie voraus, geben aber die Sätze, die wir verwenden, an.

Definition 11.2 *Sei L ein Gitter. Dann heißen $b_1, \ldots, b_i \in L$ primitives System, wenn*

1. $b_1, \ldots, b_i$ linear unabhängig sind und

2. $L(b_1, \ldots, b_i) = L \cap Span(b_1, \ldots, b_i)$ gilt.

Satz 11.1 *Sei L ein Gitter und $b_1, \ldots, b_i \in L$. Dann sind $b_1, \ldots, b_i$ genau dann zu einer Basis von L ergänzbar, wenn sie ein primitives System sind.*

Für viele Matrizen $Z \in \mathcal{R}$ sind die Spalten $x^1, \ldots, x^{2k}$ nicht linear unabhängig. In diesem Fall kann man eine wesentlich bessere Abschätzung als die folgende herleiten. Es ist jedoch nicht klar, wie man den Anteil dieser Matrizen in der Menge $\mathcal{R}$ bestimmen kann. Wir können daher für eine Attacke, die eine bessere Abschätzung für Matrizen Z mit $rg(Z) < 2k$ verwendet, keine Workload abschätzen. Deshalb beschränken wir unsere Abschätzung auf Matrizen Z mit $rg(Z) = 2k$, indem wir $2k - rg(Z)$ viele linear abhängigen Spalten durch linear unabhängige ersetzen. Da die α–Zeilen in jeder Matrix $Z \in \mathcal{R}$ linear unabhängig sind, ist der Rang mindestens $k+1$. Außerdem ist wegen der Stufenform jede der ersten $k+1$ Spalten von den vorhergehenden linear unabhängig. Deshalb bleiben die ersten $k+1$ Höhen $\hat{x}^i$ bei der Ersetzung erhalten.

Lemma 11.2 *Für alle $i = 0, \ldots, 2k-1$ gilt:*

$$Span(L_{x^1,\ldots,x^{i+1}}) \cap L_{x^1,\ldots,x^i} = L_{x^1,\ldots,x^{i+1}}$$

Unter $L_{x^1,\ldots,x^0}$ verstehen wir hierbei $\mathbf{Z}^{2k+1}$.

Beweis: Sei $z \in Span(L_{x^1,\ldots,x^{i+1}}) \cap L_{x^1,\ldots,x^i} \Rightarrow \langle z, x^{i+1} \rangle = 0$, wegen $Span(L_{x^1,\ldots,x^{i+1}}) = Span(x^1, \ldots, x^{i+1})^\perp$ aber auch $z \in L_{x^1,\ldots,x^i} \subseteq \mathbf{Z}^{2k+1}$ $\Rightarrow z \in L_{x^1,\ldots,x^{i+1}}$ (andere Richtung ist trivial). $\square$

Mit Lemma 11.1 erhalten wir also für alle $i = 0, \ldots, 2k-1$:

Jede Basis $c_1, \ldots, c_{2k-i}$ von $L_{x^1,\ldots,x^{i+1}}$ läßt sich zu einer Basis $c_1, \ldots, c_{2k+1-i}$ von $L_{x^1,\ldots,x^i}$ ergänzen.

Damit ergibt sich sofort:

Es gibt eine Basis $c_1, \ldots, c_{2k+1}$ von $\mathbf{Z}^{2k+1}$, so daß $c_1, \ldots, c_{2k+1-i}$ Basis von $L_{x^1,\ldots,x^i}$ für alle $i = 0, \ldots, 2k$ ist.

Seien $\hat{x}^1, \ldots, \hat{x}^{2k}$ und $\hat{c}_1, \ldots, \hat{c}_{2k+1}$ die Vektoren, die aus $x^1, \ldots, x^{2k}$ bzw. aus $c_1, \ldots, c_{2k+1}$ durch das Schmidt'sche Orthogonalisierungs–Verfahren hervorgehen. Dann gilt

$$\begin{aligned} 1 &= det\left(\mathbf{Z}^{2k+1}\right) \\ &= \prod_{i=1}^{2k+1} \|\hat{c}_i\| \\ &= det\left(L_{x^1,\ldots,x^{2k}}\right) \prod_{i=2}^{2k+1} \|\hat{c}_i\| \end{aligned}$$

Daraus folgt

$$det\left(L_{x^1,\ldots,x^{2k}}\right) = \prod_{i=2}^{2k+1} \|\hat{c}_i\|^{-1} \tag{27}$$

Für alle $i = 2, \ldots, 2k+1$ gilt:

$\hat{c}_i = \pi(c_i)$, wobei π die Projektion von $\mathbf{Z}^{2k+1}$ auf

$$\begin{aligned} Span\left(\hat{c}_1, \ldots, \hat{c}_{i-1}\right)^{\perp} &= Span\left(c_1, \ldots, c_{i-1}\right)^{\perp} \\ &= Span\left(L_{x^1,\ldots,x^{2k+2-i}}\right)^{\perp} \\ &= Span\left(x^1, \ldots, x^{2k+2-i}\right) \\ &= Span\left(\hat{x}^1, \ldots, \hat{x}^{2k+2-i}\right) \end{aligned}$$

ist. Damit erhalten wir für alle $i = 2, \ldots, 2k+1$:

$$\begin{aligned} \hat{c}_i &= \sum_{j=1}^{2k+2-i} \langle c_i, \hat{x}^j \rangle \frac{\hat{x}^j}{\|\hat{x}^j\|^2} \\ &= \sum_{j=1}^{2k+2-i} \langle c_i, x^j \rangle \frac{\hat{x}^j}{\|\hat{x}^j\|^2} \end{aligned} \tag{28}$$

wegen

$$\begin{aligned} x^j - \hat{x}^j &\in Span\left(x^1, \ldots, x^{j-1}\right) \\ &\subseteq Span\left(x^1, \ldots, x^{2k+2-i}\right) \\ &= Span\left(c_1, \ldots, c_i\right)^{\perp} \end{aligned}$$

Aus $c_i \in L_{x^1,\ldots,x^{2k+1-i}}$ folgt $\langle c_i, x^j \rangle = 0$ für $j \leq 2k+1-i$ und wir erhalten

$$\hat{c}_i = \langle c_i, x^{2k+2-i} \rangle \frac{\hat{x}^{2k+2-i}}{\|\hat{x}^{2k+2-i}\|^2}$$

Daraus ergibt sich für alle $i = 2, \ldots, 2k+1$ wegen $\langle c_i, x^{2k+2-i} \rangle \in \mathbf{Z} \backslash \{0\}$:

$$\|\hat{c}_i\| \geq \|\hat{x}^{2k+2-i}\|^{-1}$$

und mit (27)

$$\begin{aligned} \left| det\left(L_{x^1,\ldots,x^{2k}}\right)\right| &\leq \prod_{i=2}^{2k+1} \|\hat{x}^{2k+2-i}\| \\ &= \prod_{i=1}^{2k} \|\hat{x}^i\| \end{aligned} \tag{29}$$

Allgemein gilt für ein Gitter L vom Rang n: (Minkowski–Ungleichung)

$$\prod_{i=1}^{n} \lambda_i(L) \leq \gamma_n^{n/2} |det(L)|$$

Dabei bezeichnet γ_n die $n$te Hermite–Konstante. Diese Schranke ist scharf.

Damit erhalten wir für $L = L_{x^1,\ldots,x^{2k}}$ mit (29): ($\gamma_1 = 1$)

$$\lambda\left(L_{x^1,\ldots,x^{2k}}\right) \leq \prod_{i=1}^{2k} \|\hat{x}^i\|$$

Gilt $\prod_{i=1}^{2k} \|\hat{x}^i\| \leq K$ für eine Konstante K, so ergibt sich daraus eine Workload von

$$\frac{K^{k+1}\pi^{(k+1)/2}}{2 \cdot \Gamma((k+3)/2)} =: W(K)$$

11.3 Eine untere Schranke für K

Wir zeigen, daß es keine Abschätzung $\prod_{i=1}^{2k} \|\hat{x}^i\| \leq K$ mit $W(K) < 2^{500}$ gibt. Wegen der Stufenform einer Matrix $Z \in \mathcal{R}$ in den ersten k+1 Spalten gilt

$(\hat{x}^i)_{2i-1} = \alpha_{i+1}^{i+1}$ für alle $i = 1, \ldots, k+1$. Damit folgt:

$$\begin{aligned}
\prod_{i=1}^{2k} \|\hat{x}^i\| &\geq \prod_{i=1}^{k+1} \|\hat{x}^i\| \\
&\geq \prod_{\nu=2}^{k+2} \alpha_\nu^\nu \\
&\geq \left(2^h + 1\right)^{k+1} \\
&\overset{h\geq 3}{\geq} 2^{41}
\end{aligned}$$

Für $k = 12$ und $d_1 \geq 4$ ergibt sich $W(K) > 2^{500}$.

Selbst wenn man eine günstige Strategie zum Durchlaufen aller Tupel $y' = (y_1, y_3, \ldots, y_{2k-1}, y_{2k+1})$ mit $\|y'\| \leq K$ wählt (z.B. mit den y' mit minimaler Länge beginnt), erscheint ein Angriff auf der Basis dieser Überlegungen aussichtslos.

12 Optionen zur Erhöhung der Sicherheit

Wir diskutieren Varianten des Preprocessings, die auf den ersten Blick geeignet erscheinen, die Sicherheit des Preprocessings — d.h. die maximale Effizienz einer Attacke — zu erhöhen.

12.1 Erhöhung des Parameters h

Damit gilt für erfolgreiche Versuche stets $|y_{\sigma(2)}| \geq 17$. Es erscheint plausibel anzunehmen, daß — zumindest für kleine n — erfolgreiche n–Zeilen–Versuche allgemein größere Koeffizienten haben. Dies könnte zu größerer Sicherheit führen.

Zumindest gilt:
Für eine Attacke $\vec{y}$ ist $|N(\vec{y})|$ im Fall $h \geq 4$ höchstens so groß wie im Fall $h = 3$. Diese Aussage ist anschaulich klar.

Klar ist auch, daß $|\vec{y}|$ mit wachsendem h abnimmt. Dies entnimmt man Gleichung 22. Da $\mathcal{E}(\vec{y}) = |N(\vec{y})|/|\vec{y}|$ gilt, kann das eine größere Effizienz von Versuchen im Fall $h \geq 4$ zur Folge haben. Dieser Effekt ist jedoch für die effizientesten Versuche gering, da für diese $m(\vec{y})$ klein ist.

Bei einem Test fanden wir für $k = 12$, $l_0 = 5$, $l_1 = 4$, $d_0 = 4$, $d_1 = 6$ und $h = 4$ keinen erfolgreichen 5–Zeilen–Versuch $\vec{y}$ mit $\|\vec{y}\| \leq 200$. Erfolgreiche Attacken mit Koeffizienten > 200 wird es wohl nicht geben, so daß die Sicherheit des Preprocessings für diese Parameter durch die maximale Effizienz einer 6–Zeilen–Attacke gegeben ist. Dieser Wert ist wohl größer als der für $h = 3$, $2^{-67.7}$.

Das Problem liegt darin, die Sicherheit des Preoprocessings zu bestimmen. Da wir höhere Koeffizienten für die erfolgreichen Versuche erwarten, müssen wir auch die Effizienzen sehr vieler Versuche berechnen, bevor wir mit einiger Gewißheit annehmen können, die maximale Effizienz einer 6–Zeilen–Attacke hinreichend genau berechnet zu haben. Die langen Laufzeiten des Programms im Fall $n = 6$ elauben uns jedoch solche umfangreichen Rechnungen nicht mehr.

Ein weiteres Problem, das die Erhöhung der Koeffizienten mit sich bringt, ist, daß dieser Effekt sich nur bei Versuchen von wenigen Zeilen auswirkt. So gilt z.B. mit $\vec{y}_h = (1, -2^h - 1, 1, 1, 1, -1, -1)$ im Fall $k = 7,\ l_0 = l_1 = 4,\ d_0 = 4,\ d_1 = 7$ für

$h = 3$: $\mathcal{E}(\vec{y}_h) = 2^{64.2}$

$h = 4$: $\mathcal{E}(\vec{y}_h) = 2^{65.2}$
$h = 5$: $\mathcal{E}(\vec{y}_h) = 2^{64.5}$

d.h. auf 7–Zeilen–Versuche hat die Erhöhung von h fast keinen Einfluß. Sogar für $h = 6$ findet man noch $\mathcal{E}(8, -65, 1, 1, 1, -1, -1) = 2^{67.9}$. Dadurch scheint die Möglichkeit, durch die Erhöhung von h die Sicherheit des Preprocessings zu erhöhen, begrenzt.

12.2 Keine Speicherung des Wertes r_k

Die Bedingung $k \in \{a(i, \nu)\}$ sollte sichern, daß das Tupel $(r_0^\nu, \ldots, r_k^\nu)$ in jeder Runde ν von allen Anfangswerten $(r_0^0, \ldots, r_k^0)$ abhängt. Fordern wir $k \notin \{a(i, \nu)\}$, so brauchen wir den Wert r_k^ν nicht zu speichern und auch keinen Anfangswert r_k^0 zu laden. Auch in diesem Fall hängt also das Tupel $(r_0^\nu, \ldots, r_k^\nu)$ in jeder Runde ν von allen Anfangswerten $(r_0^0, \ldots, r_{k-1}^0)$ ab — mit dem Unterschied, daß wir nun einen Anfangswert weniger haben. Dafür benötigt das Preprocessing einen Speicherplatz weniger und wir können k um 1 erhöhen. Außerdem sparen wir eine Multiplikation — die mit x_k^ν. Wir können dafür z.B. l_0 um 1 erhöhen.

Der Zusammenhang zwischen $\mathcal{E}(\vec{c})$ und $\mathcal{E}(\vec{y})$ ändert sich jedoch: die Attacke, die ein Versuch erzeugt, ist durch die ungeraden Koeffizienten des Versuches bereits eindeutig bestimmt. Eine Attacke wird daher von weniger vielen Versuchen erzeugt und die Differenz zwischen $\mathcal{E}(\vec{c})$ und $\mathcal{E}(\vec{y})$ ist geringer.

Für die Werte $k = 13$, $l_0 = 7, l_1 = 3$ und $h = 3$ fanden wir folgende maximale Effizienzen von Attacken:

d_0	d_1	$-\log_2(\max(\mathcal{E}(\vec{y})))$
2	7	70.4
3	6	73.0
4	5	74.6
5	4	74.4

Für diese Parameter benötigt das Preprocessing — wie bei allen Parametern, die wir ab Abschnitt 8.3 betrachtet haben — 16 Multiplikationen und 13 gespeicherte Paare (r_i, x_i).

12.3 Flexiblere Wahl der Werte $a(i,\nu)$ und $b(i,\nu)$

Seit Kapitel 2 haben wir gefordert, die Werte $a(i,\nu)$ und $b(i,\nu)$ verschieden zu wählen. Dies geschah unter dem Eindruck der einfachen Attacke von de Rooij [6]. Solche einfachen Attacken können jedoch schon durch die Forderung abgewehrt werden, daß eine Mindestanzahl von Werten $a(i,\nu)$ bzw. $b(i,\nu)$ verschieden sind. Diese Mindestanzahlen sind so zu wählen, daß die Ungleichung

$$\#\{\alpha_\mu^\nu \neq 0 \,|\, 0 \leq \mu \leq k-1\} < \#\{\beta_\mu^\eta \neq 0 \,|\, 0 \leq \mu \leq k-1\}$$

für alle ν und η erhalten bleibt. Dies hatten wir durch die Bedingungen 3., 5. und $l_0 \geq l_1$ erreicht. Nur so können wir 4–Zeilen–Attacken ausschließen. Das entnimmt man dem Beweis von Satz 4.1. Um die Effizienzen von Attacken gegen diese Art von Preprocessing zu berechnen, benötigen wir aber ein Programm, welches völlig anders aufgebaut ist als das in Kapitel 5 vorgestellte. Uns liegen daher noch keine Erkenntnisse über die Sicherheit dieser Art von Preprocessing vor.

12.4 Miteinbeziehung des geheimen Schlüssels s

Bei den bisherigen Varianten des Preprocessings gehen die geraden Koeffizienten eines Versuches nicht in die Koeffizienten der Attacke mit ein. Zwei Versuche, die sich nur in den geraden Koeffizienten unterscheiden, erzeugen also dieselbe Attacke. Setzen wir in Schritt 2. des Preprocessings

$$r_{\nu \bmod k}^\nu := s + \sum_{i=1}^{l_0} 2^{g(i,\nu)} r_{b(i,\nu)}^{\nu-1} \bmod q \qquad x_{\nu \bmod k}^\nu = s \cdot \prod_{i=1}^{l_0} (x_{b(i,\nu)}^{\nu-1})^{2^{g(i,\nu)}} \bmod p$$

so muß ein Angreifer annehmen, daß sich aus den Gleichungen der Schritte 1. und 2. eine Gleichung

$$c_0 s + \sum c_i r_k^i \equiv 0$$

ergibt. Es ist also sinnvoll, eine Attacke nun als ein Tupel $(c_0, c_1, \ldots, c_{k+2})$ zu definieren. Der Koeffizient c_0 hängt hierbei von den geraden Koeffizienten eines Versuches ab, der diese Attacke erzeugt — er ergibt sich gerade aus ihrer Summe. Das bedeutet, daß zwei Versuche nur dann dieselbe Attacke erzeugen können, wenn die Summe ihrer geraden Koeffizienten gleich ist. Dadurch verringert sich die Anzahl der Versuche, die dieselbe Attacke erzeugen können, und damit auch die maximale Effizienz einer Attacke.

Der Nachteil dieser Modifikation liegt auf der Hand: sie benötigt eine zusätzliche Multiplikation. Da die maximale Effizienz einer Attacke mindestens so groß ist wie

die maximale Effizienz eines Versuches, ist der Zugewinn an Sicherheit begrenzt. Nach unseren Beobachtungen verringert sich die maximale Effizienz einer Attacke dadurch nicht mehr als um den Faktor 2. Die zusätzliche Multiplikation könnte z.B. durch die Erhöhung des Parameters l_0 sinnvoller genutzt werden.

12.5 Eine zusätzliche Bedingung

Es hat sich bei unseren Untersuchungen gezeigt, daß die effizientesten Versuche stets kleine Koeffizienten besitzen. Die Modifikation des Preprocessings mit Einführung des Parameters h hatte zur Folge, daß bei erfolgreichen Versuchen $\vec{y}$ immer y_2 ein ganzzahliges Vielfaches von $2^h + 1$ ist. Die anderen Koeffizienten $y_{\sigma(i)}$ sind jedoch bei den effizientesten 6–Zeilen–Versuchen immer noch klein — es gilt z.B. fast immer $|y_5| = |y_6| = 1$.

Wir könnten durch die zusätzliche Bedingung

$$a(i, \nu) = \nu - 1 \Rightarrow f(i, \nu) \geq \bar{h}$$

mit einem $\bar{h} \geq 1$ erzwingen, daß für erfolgreiche n–Zeilen–Versuche $\vec{y}$ wegen Spalte $k+\sigma(n-1)/2$ stets $y_{\sigma(n-1)} \geq \bar{h}$ gilt. Uns liegen im Moment noch keine Ergebnisse über die Sicherheit des Preprocessings mit dieser zusätzlichen Bedingung vor. Es ist jedoch zu beachten, daß für $\bar{h} \geq 2$ der Wert $|\vec{y}|$ deutlich abnimmt.

12.6 Die Permutation Werte r_k^ν

Eine weitere Möglichkeit, die Sicherheit des Preprocessings zu erhöhen, ist, die r_k's zu permutieren. Naheliegend ist folgendes Verfahren:

Es werden n Werte r_k^μ abgespeichert. In jeder Runde wird einer davon zufällig ausgewählt und für das Protokoll dieser Runde verwendet. Der in dieser Runde berechnete Wert r_k^ν wird an seiner Stelle abgespeichert.

Sei r_k^i der Wert r_k, der in Runde i berechnet wird und R_k^i der Wert r_k, der in Runde i für das Protokoll verwendet wird. Die Permutation entspricht einer Bijektion $\eta : \mathbf{Z}_{>-n} \to \mathbf{N}$ mit $\eta(i) \geq i + 1$ im folgenden Sinne: $\eta(i)$ ist die Runde, in der der Wert r_k^i für das Protokoll verwendet wird, d.h. $r_k^i = R_k^{\eta(i)}$. Hierbei werden in den Runden $-n + 1$ bis 0 nur die Werte $r_k^1, \ldots r_k^n$ berechnet, aber kein Protokoll durchgeführt. Wenn wir im folgenden von der Runde i sprechen, so gehen wir stets von $i > 0$ aus.

Dem Verfahren entnimmt man, daß für alle i und j gilt

$$\Pr_{\eta}[\eta(i) = j] = \left(\frac{n-1}{n}\right)^{j-i-1} n^{-1}$$

Diese Wahrscheinlichkeit wird für $j = i + 1$ maximal.
Äquivalent dazu gilt für alle $i_1 \neq \ldots \neq i_m$, daß

$$\Pr_{\eta}\left[\forall_{1 \leq l \leq m} \eta(i_l) = j_l\right]$$

maximal wird, wenn $j_l = i_l + 1$ für alle $l = 1, \ldots, m$ gilt. Wir verzichten auf den einfachen Beweis.

Im Fall $j_l = i_l + 1$ für $l = 1, \ldots, m$ wird diese Wahrscheinlichkeit für aufeinanderfolgende i_l maximal, d.h. wenn $i_l = l + C$ gilt für alle $l = 1, \ldots, m$. Diese maximale Wahrscheinlichkeit beträgt n^{-m}. Wir verzichten auch hier auf den einfachen Beweis.

Ein Angreifer muß nun wie folgt vorgehen:
Wähle eine effiziente Attacke $\vec{c}$. Sei $m := \#\{i \,|c_i \neq 0\}$ und o.B.d.A. $c_1, \ldots, c_m \neq 0$. Wähle $j_1, \ldots, j_m$ und nehme an, daß $\sum_{i=1}^{m} c_i R_k^{j_i} = 0$ gilt. Ist diese Gleichung erfüllt, so läßt sich mit den Gleichungen $y^{j_i} = R_k^{j_i} + se^{j_i}$ für $i = 1, \ldots, m$ der geheime Schlüssel s ermitteln.

Gilt

$$\sum_{i=1}^{m} c_i R_k^{j_i} \equiv \sum_{i=1}^{m} c_i r_k^i$$

so ist die Wahrscheinlichkeit, daß die Gleichung $\sum_{i=1}^{m} c_i R_k^{j_i} = 0$ gilt, durch $\mathcal{E}(\vec{c})$ gegeben.

Gilt dagegen

$$\sum_{i=1}^{m} c_i R_k^{j_i} \not\equiv \sum_{i=1}^{m} c_i r_k^i$$

so können wir über die Erfolgswahrscheinlichkeit der Annahme $\sum_{i=1}^{m} c_i R_k^{j_i} = 0$ allgemein keine Aussage machen. Um einen effizienten Angriff zu gewährleisten, ist es für einen Angreifer also wichtig, die Werte $j_1, \ldots, j_m$ so zu wählen, daß

$$P := \Pr_{\eta}\left[\sum_{i=1}^{m} c_i R_k^{j_i} \equiv \sum_{i=1}^{m} c_i r_k^i\right]$$

möglichst groß wird. Im folgenden untersuchen wir für die effizientesten Attacken, für welche Wahl der j_i P maximal wird.

Ein Angreifer kann versuchen, die Werte $\eta(i)$ für $i = 1, \ldots, m$ zu erraten. Ist eine Attacke jedoch symmetrisch, d.h. gilt $c_i = c_j \neq 0$ für $i \neq j$ für mindestens ein Paar (i, j), so führt auch eine Verwechslung von $\eta(i)$ und $\eta(j)$ zur Identität $\sum_{i=1}^{m} c_i R_k^{j_i} \equiv \sum_{i=1}^{m} c_i r_k^i$.

Für einige Parameter werden die effizientesten Attacken von 5–Zeilen–Versuchen mit $t = 3$ und $x = 3$ erzeugt. Eine solche Attacke ist folgendermaßen gegeben:

$$c_1 = \alpha_k^1 y_1 > 0 \qquad c_2 = \alpha_k^2 y_3 - y_1 < 0$$

$$c_3 = \alpha_k^3 y_5 - y_3 > 0 \qquad c_4 = -y_5 < 0$$

Die anderen Koeffizienten c_i sind 0. Die effizienten 5–Zeilen–Versuche hatten stets das Vorzeichen $(+--++)$. Daraus folgt c_1, $c_3 > 0$ und c_2, $c_4 < 0$. Außerdem ist bei den effizienten 5–Zeilen–Versuchen (im Fall h=3) der Koeffizient y_3 stets groß und teilerfremd zu den anderen Koeffizienten. Deshalb ist es ausgeschlossen, daß eine effiziente 5–Zeilen–Attacke symmetrisch ist. Ein Angreifer muß in diesem Fall also alle Werte $\eta(i)$ erraten. Die Erfolgswahrscheinlichkeit hierfür ist n^{-4}.

Für andere Parameter werden die effizientesten Attacken von 6–Zeilen–Versuchen erzeugt. Für solche Attacken sind höchstens 5 Koeffizienten c_i nicht 0 und o.B.d.A. sind das die ersten 5. Die effizientesten teilen sich in 4 Typen auf:

$$\begin{aligned} &a) \quad c_1 = c_3 = c_4 = c_5 \\ &b) \quad\quad c_3 = c_4 = c_5 \\ &c) \quad\quad\quad c_4 = c_5 \\ &d) \quad\quad c_1 = c_4 = c_5 \end{aligned}$$

Bemerkung: Eine Attacke kann von verschiedenen Typen sein, d.h. diese Zerlegung in verschiedene Typen ist nicht disjunkt.

Im folgenden gehen wir von $n = 4$ aus.

Für eine Attacke vom Typ a) gilt

$$P = \sum_{\sigma \in S_{\{1,3,4,5\}}} \Pr_{\eta} \left[\eta(2) = j_2 \wedge \bigwedge_{i \in \{1,3,4,5\}} \eta(i) = j_{\sigma(i)} \right]$$

Zwar wird

$$\Pr_{\eta} \left[\bigwedge_{1 \leq i \leq 5} \eta(l) = j_l \right]$$

maximal mit $j_l = l+1$ für $l = 1, 2, 3, 4, 5$, alle anderen Summanden verschwinden aber. Für alle $\sigma \in S_{1,3,4,5} - \{Id\}$ gilt dann nämlich $j_{\sigma(i)} < i + 1$ für mindestens ein $i \in \{1, 3, 4, 5\}$ und wegen $\eta(i) \geq i + 1$ ist $\eta(i) = j_{\sigma(i)}$ unmöglich.

P wird maximal für z.B. $(j_1, j_2, j_3, j_4, j_5) = (5, 3, 6, 7, 8)$. Wir verzichten auf den einfachen Beweis. Wir erhalten also

$$P \leq 24\frac{(n-1)(n-2)^2}{n^8} \approx 2^{-8.3}$$

Für Attacken vom Typ b) erhalten wir den maximalen Wert für P bei $(j_1, j_2, j_3, j_4, j_5) = (2, 3, 6, 7, 8)$. Es gilt dann also

$$P \leq 6\frac{(n-1)(n-2)}{n^7} \approx 2^{-8.8}$$

Für Attacken vom Typ c) erhalten wir

$$P \leq 2\frac{n-1}{n^6} \approx 2^{-9.4}$$

Für Attacken vom Typ d) erhalten wir

$$P \leq 6\frac{(n-1)(n-2)}{n^7} \approx 2^{-8.8}$$

Wir betrachten nun die maximalen Effizienzen der Attacken vom Typ a), b) und c) für $k = 8$ und $h = 3$. Wir geben jeweils $-\log_2(\max(\mathcal{E}(\vec{c})))$ an.

l_0	l_1	d_0	d_1	a)	b)	c)	d)
5	4	2	8	63.2	62.3	62.3	64.4
6	4	2	7	61.2	60.3	60.3	64.4
6	4	3	6	62.5	62.0	62.0	64.4
6	4	4	5	62.6	61.9	61.8	63.9
6	4	5	4	–	–	60.3	61.9

Die Sicherheit des Preprocessings erhalten wir, indem wir diese Effizienzen mit $2^{-8.3}$, $2^{-8.8}$ bzw. $2^{-9.4}$ multiplizieren und von diesen 3 Werten das Maximum bilden.

Wir vergleichen nun die Sicherheit des Preprocessings für $k = 12$, $n = 0$ und $k = 8$, $n = 4$. In beiden Fällen benötigt das Preprocessing 13 zu speichernde Paare (r_i, x_i). Wir setzen jeweils $h = 3$.

l_0	l_1	d_0	d_1	$k = 12$, $n = 0$	$k = 8$, $n = 4$
5	4	2	8	72.0	71.1
5	4	3	7	67.6	68.0
5	4	4	6	67.7	68.0
5	4	5	5	66.6	67.0
5	4	6	4	64.0	64.3
6	4	2	7	73.1	69.1
6	4	3	6	73.6	70.8
6	4	4	5	72.4	70.7
6	4	5	4	72.3	69.7

Bei $k = 6$ und $n = 6$ sind die Sicherheiten noch niedriger.

Interessant sind für uns vor allem die Parameter, für die es keine 5–Zeilen–Versuche gibt. Für diese bringt die Permutation der Werte r_k^ν bei gleichbleibender Zahl der zu speichernden Werte keine Erhöhung der Sicherheit.

13 Zusammenfassung

Ziel dieser Arbeit war es, ein sicheres und trotzdem effizientes Preprocessing zu finden. Nach den zurückliegenden Untersuchungen können wir annehmen, dies erreicht zu haben. Wir haben gezeigt, daß eine minimale Workload von Attacken von 2^{72} mit nur 16 Multiplikationen pro Runde und 13 gespeicherten Paaren (r_i, x_i) erreicht werden kann. Mit der in Abschnitt 12.3 erklärten Variation — der Wert r_k^ν geht nicht in die Gleichungen mit ein — erreichen wir sogar eine Sicherheit von 2^{74}. In diesem Fall können wir die Anzahl der gespeicherten Paare auf 12 verringern. Auch von der in Abschnit 12.5 besprochenen Variation erwarten wir eine Erhöhung der Sicherheit. Ergebnisse dazu werden bald vorliegen.

Folgender Preprocessing–Algorithmus erscheint z.B. nach unserem derzeitigen Wissensstand geeignet:

Setze $k = 12$, $l_0 = 7$, $l_1 = 3$, $d_0 = 4$, $d_1 = 5$, $h = 4$, $\bar{h} = 1$.

Initiation: lade k Paare $(r_0^0, x_0^0) \ldots, (r_{k-1}^0, x_{k-1}^0)$ mit $x_i^0 = \alpha^{r_i^0} \bmod p$.
$\nu := 1$. ν ist die Rundennummer

1. Wähle $l_1 - 2$ verschiedene Zufallszahlen
$a(3,\nu), \ldots, a(l_1,\nu) \in \{\nu + 1 \bmod k, \ldots, \nu - 2 \bmod k\}$
$a(1,\nu) := \nu \bmod k$, $a(2,\nu) := \nu - 1 \bmod k$

Wähle $l_1 - 2$ verschiedene Zufallszahlen $f(3,\nu), \ldots, f(l_1,\nu) \in \{0, \ldots, d_1 - 1\}$,
$f(1,\nu)$ zufällig aus $\{h, \ldots, d_1 - 1\}$ und $f(2,\nu)$ zufällig aus $\{\bar{h}, \ldots, d_1 - 1\}$

$$r_k^\nu := r_{\nu \bmod k}^\nu + \sum_{i=1}^{l_1} 2^{f(i,\nu)} r_{a(i,\nu)}^{\nu-1} \bmod q \qquad x_k = x_{\nu \bmod k}^\nu \cdot \prod_{i=1}^{l_1} (x_{a(i,\nu)}^{\nu-1})^{2^{f(i,\nu)}} \bmod p$$

2. wähle $l_0 - 1$ verschiedene Zufallszahlen
$b(2,\nu), \ldots, b(l_0,\nu) \in \{\nu + 1 \bmod k, \ldots, \nu - 1 \bmod k\}$
$b(1,\nu) := \nu \bmod k$

Wähle l_0 verschiedeneZufallszahlen $g(1,\nu), \ldots, g(l_0,\nu) \in \{0, \ldots, d_0 - 1\}$

$$r_{\nu \bmod k}^\nu := \sum_{i=1}^{l_0} 2^{g(i,\nu)} r_{b(i,\nu)}^{\nu-1} \bmod q \qquad x_{\nu \bmod k}^\nu = \prod_{i=1}^{l_0} (x_{b(i,\nu)}^{\nu-1})^{2^{g(i,\nu)}} \bmod p$$

3. verwende (r_k^ν, x_k^ν) für die ν–te Signatur (e_ν, y_ν) gemäß
$y_\nu = r_k^\nu + s e_\nu \bmod q$

4. $\nu := \nu + 1$
GOTO 1. für die nächste Signatur

Die Zufallszahlen $a(3,\nu),\ldots,a(l,\nu)$, $b(2,\nu),\ldots,b(l,\nu)$, $f(1,\nu),\ldots,f(l,\nu)$ und $g(1,\nu),\ldots,g(l,\nu)$ werden unabhängig gewählt.

Dies ist selbstverständlich nur ein Beispiel.

Unsere Untersuchungen sind noch nicht abgeschlossen. Wir glauben aber nicht, daß feste Werte $a(i,\nu)$ und $b(i,\nu)$ ein effizientes Preprocessing definieren. Wir haben einige Variationen mit solchen weniger randomisierten Gleichungen studiert und immer effiziente Attacken gefunden.

A Es gibt keine erfolgreichen 5–Zeilen–Versuche

A.1 Die 5×5–Untermatrix

Im folgenden sei $k \geq 8$, $l_0 = 4$, $l_1 \geq 6$, $d_0 \leq 6$, $d_1 = 5$ und $h = 4$. Wir zeigen, daß es unter diesen Vorraussetzungen keine erfolgreichen 5–Zeilen–Versuche gibt.

O.B.d.A. sei im folgenden stets $\sigma(1) = 1$ und $\sigma(2) = 2$. Wir wissen, daß die effizientesten 5–Zeilen–Versuche eine der folgenden Zuordnungen besitzen:
1. $\sigma(3) = 3,\ \sigma(4) = 4,\ \sigma(5) = 5$
2. $\sigma(3) = 3,\ \sigma(4) = 6,\ \sigma(5) = 7$
3. $\sigma(3) = 5,\ \sigma(4) = 6,\ \sigma(5) = 7$
4. $\sigma(3) = 5,\ \sigma(4) = 8,\ \sigma(5) = 9$
Wir gehen deshalb von nun an stets von diesen 4 Fällen aus.

Sei eine 5–Zeilen–Konfiguration gegeben. Wir betrachten für jeden der 4 Fälle eine andere 5×5–Untermatrix A, die aus folgenden Spalten besteht:
1. $1, 2, 3, k+1, k+2$
2. $1, 2, 3, k+1, k+3$
3. $1, 3, 4, k+1, k+3$
4. $1, 3, 4, k+1, k+4$

Diese Untermatrix hat folgende Gestalt:

1.

17	∗	∗		
x	∗	∗	-1	
	17	∗	x	
	x	∗	∗	-1
		17	∗	x

2.

17	∗	∗		
x	∗	∗	-1	
	17	∗	x	
		x	∗	-1
			∗	x

3.

17	∗	∗		
x	∗	∗	-1	
	17	∗	∗	
	x	∗	∗	-1
		17	∗	x

4.

17	∗	∗		
x	∗	∗	-1	
	17	∗	∗	
		x	∗	-1
			∗	x

Hierbei steht “x” für einen Pflichteintrag und “∗” für einen Wahleintrag. Wie

man sieht unterscheiden sich die Fälle 1. und 3. bezüglich dieser Untermatrix A nur darin, daß $A_{3,4}$ im Fall 3. auch 0 sein kann.

Lemma A.1 *In jeden der 4 Fälle hat für jede mögliche Belegung der Einträge die Untermatrix A Rang* ≥ 4.

Beweis: In den Fällen 2. und 4. ist das unmittelbar klar. In den Fällen 1. und 3. betrachte man die Untermatrix D von A die aus den Zeilen 1,3,5,4 und den Spalten 1,2,3,5 besteht. D_i sei die i–te Zeile von D. Formt man D gemäß

$$D_4 \longrightarrow D_4 - \frac{D_{4,2}}{17} D_2 - \frac{17 \cdot D_{4,3} - D_{4,2} D_{2,3}}{17^2} D_3$$

in eine obere Dreiecksmatrix um, so sieht man, daß

$$det(D) = 17^3 \cdot \left(-1 - \frac{17 \cdot D_{4,3} - D_{4,2} D_{2,3}}{17^2} D_{3,4} \right)$$

gilt. Dieser Term ist wegen $D_{2,3}, D_{3,4}, D_{4,2}, D_{4,3} \in \{0, 1, 2, 4, 8, 16, 32\}$ nicht Null.□

Ein Computerprogramm durchläuft nun für einen der 4 Fälle alle Möglichkeiten dieser Untermatrizen. Dabei setzen wir o.B.d.A. $\bar{h} = 0$ und $d_0 = 6$. Für jede dieser Möglichkeiten berechnet es die Determinante. Ist diese gleich 0, so ermittelt es einen Versuch $\vec{y}$, der diese Untermatrix annulliert. Da der Rang der Matrix stets ≥ 4 ist, ist dieser Versuch bis auf Multiplikation mit einem Faktor eindeutig bestimmt. Deshalb wird von dem Programm jeder erfolgreiche Versuch mit Zuordnung σ (bis auf Multiplikation mit einem Faktor) mindestens einmal durchlaufen.

A.2 Der Test

Für jeden berechneten Versuch stellt ein kurzer Test fest, ob der Versuch erfolgreich sein kann. Hierbei wird jedoch nur ein notwendige und keine hinreichende Bedingung dafür geprüft. Dieser Test ist nicht in allen 4 Fällen gleich.

Der Test wählt für jede Konfiguration B mit Zuordnung σ und 5×5–Untermatrix A eine Spalte j aus, die nicht in A liegt, und testet ob $\sum B_{i,j} y_{\sigma(i)} = 0$ gilt. Ist dies für keine Konfiguration B der Fall, so ist $\vec{y}$ nicht erfolgreich. Unser Ziel ist es, j so zu wählen, daß es für keinen der berechneten Versuche $\vec{y}$ die Gleichung $\sum B_{i,j} y_{\sigma(i)} = 0$ gilt. Daraus folgt dann, daß es keinen erfolgreichen Versuch mit

Zuordnung σ gibt. Gilt das für jeden der 4 Fälle, so gibt es überhaupt keinen erfolgreichen 5–Zeilen–Versuch.

Dabei kommt es auf den expliziten Wert j nicht an: dieser kann von der jeweiligen Konfiguration abhängen. Wichtig ist nur, daß das Programm alle Möglichkeiten für die Spalte j aus einer Konfiguration B mit der 5×5–Untermatrix A durchläuft. Die Forderung, daß A die 5×5–Untermatrix von B ist, erzeugt zusätzliche Bedingungen für die Spalte j: gilt z.B. $A_{1,2} \neq 0$, so folgt daraus wegen $l_1 = 4$ schon $B_{1,j} = 0$.

A.2.1 Die Fälle 1. und 3.

Für den Test wählen wir j so, daß für eine gegebene Konfiguration B mit Zuordnung σ für mindestens ein $1 \leq i \leq 5$ $B_{i,j} = 0$ gilt. Die Existenz eines solchen j garantiert uns folgendes Lemma:

Lemma A.2 *Im den Fällen 1. und 3. gibt es in jeder Konfiguration B mit Zuordnung σ eine Spalte j mit $A_{5,j} = 0$ und $A_{2,j} \neq 0$*

Beweis: Dies folgt sofort aus der Gestalt der Konfiguration B und $l_1 = 4$ und $l_0 \geq 6$ □

A.2.2 Die Fälle 2. und 4.

In diesen Fällen wählen wir für j den festen Wert $t + 1$. Das ist die Spalte mit α_t^t. Wir können noch weitere Aussagen über diese Spalte machen:

Lemma A.3 *Unter der Vorraussetzung $l_0 \geq 6$ und $l_1 = 4$ gibt es in den Fällen 2. und 4. keinen erfolgreichen Versuch mit den Vorzeichen $+ - + - -$*

Beweis: Man betrachte die Spalten $\{j \,|\, 1 < j < \frac{x+1}{2} \vee 3 \leq j \leq k-1\}$. In den Zeilen 1 und 3 stehen dort zusammen höchstens 2 Einträge, allein in der Zeile 2 aber mindestens 3 Einträge. Die Einträge in diesen Spalten können sich also nicht wegkürzen. □

Korrolar A.4 *In den Fällen 2. und 4. gilt für die* 5×5*–Untermatrix A:*

$$A_{1,2} \neq 0 \qquad A_{1,3} = 0 \qquad A_{3,4} \neq 0$$

Korrolar A.5 *In den Fällen 2. und 4. gilt für jede Konfiguration B mit Zuordnung* σ*:*

$$B_{1,t+1} = 0$$

A.3 Ergebnisse

In keinem der 4 Fälle wurden vom Computerprogramm erfolgreiche Versuche gefunden. Da das Programm alle erfolgreichen 5-Zeilen–Versuche aufzählt, erhalten wir damit:

Satz A.6 *Es gibt unter den obigen Vorraussetzungen keine erfolgreichen* 5*–Zeilen–Versuche*

Diese Aussage gilt natürlich auch für $h \geq 4$ und $\bar{h} \geq 0$.

Literatur

[1] T. Beth: Effizient Zero–Knowledge Identifikation Scheme for Smart Cards. Advances in Cryptology—Eurocrypt'88, Lecture Notes in Computer Science, Vol.330 (1988), Springer Verlag, Berlin, pp 77–86

[2] T. ElGamal: A Public Key Cryptosystem and a Signature Scheme Based On Discrete Logarithms. IEEE Trans. Inform. Theory, 31 (1985), pp 469–472

[3] C. G. Günther: An Identity–Based Key–Exchange Protokoll. Advances in Cryptology—Eurocrypt'89, Lecture Notes in Computer Science, Vol.434 (1990), Springer Verlag, Berlin, pp 29–37

[4] J. Håstad, B. Just, J. C. Langrias, C. P. Schnorr: Polynomial Time Algorithms For Finding Integer Relations Among Real Numbers. SIAM J. Comput., Vol. 18, No 5, pp. 859–881.

[5] R. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Comm. ACM, 21 (1978), pp 120–126

[6] P. de Rooij: On the security of the Schnorr scheme using preprocessings. Advances in Cryptology — Proceedings of Eurocrypt'91, Lecture Notes in Computer Science, vol 547 (1991), Springer Verlag, Berlin, pp. 71–80.

[7] P. de Rooij: On Schnorr's Preprocessing for Digital Signatur Schemes. Eurocrypt'93, Lecture Notes in Computer Science, Vol 765 (1993), Springer Verlag, Berlin, pp 435–449.

[8] C. P. Schnorr: Efficient Identification and Signatures for Smart Cards. Advances in Cryptology—Crypto'89, Lecture Notes in Computer Science, Vol.435(1990), Springer Verlag, Berlin, pp. 239–252.

[9] C. P. Schnorr: Efficient Signature Generation by Smart Cards. Journal of Cryptology vol. 4 (1991), no. 3, pp. 161–174.