# Eine Sicherheitsanalyse des Public-Key Kryptosystems $MST_3$

## Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Mathematiker (DII)
an der Universität Duisburg-Essen, Campus Essen
im Fachbereich Mathematik

vorgelegt von:

**Paul Wolf**
Matr. 2225558

Datum der Abgabe: _____

# Inhaltsverzeichnis

# 1. Einleitung

Mit der zunehmenden weltweiten Verbreitung neuer Kommunikationsmethoden, wie Internet und Handy, werden immer neuere und effizientere Methoden zum Absichern der Nachrichten gesucht. Die moderne Kryptographie bietet Techniken an, um Vertraulichkeit der übertragenen Informationen, Authentizität, Nicht-Abstreitbarkeit des Senders und die Integrität der gesendeten Botschaft zu gewährleisten. Bedingt durch die stets anwachsenden Leistungsfähigkeiten der heutigen Rechner und den theoretischen Möglichkeiten von zukünftigen Quantencomputern gelten viele asymmetrische kryptographische Verfahren bereits als unsicher, weswegen es entscheidend ist, neue Verfahren anzubieten und gründlich zu untersuchen. Im Folgenden wollen wir ein System vorstellen und untersuchen, welches uns Sicherheit auch gegenüber leistungsstarken Computern verspricht.

Das Public Key Kryptosystem $MST_3$ basiert auf der Theorie für nicht-abelschen endlichen Gruppen und wurde in [1] von Lempken, Magliveras, van Trung und Wei vorgestellt. Wir wollen in dieser Arbeit zunächst die Grundlagen zusammenfassen, so dass ein tiefergehendes Verständnis des neuen Kryptosystems möglich ist. Die $MST_3$-Autoren haben weiterhin eigene Kryptoanalysen betrieben und Angriffe betrachtet, welche letztlich die Sicherheit des Kryptosystems aufzeigen, in dem nun gewisse Schwächen und Eigenarten bekannt sind und vermieden werden können. Im weiteren Verlauf der Arbeit wollen wir eine neuere Kryptoanalyse aus [3] von Blackburn, Cid und Mullan begutachten und werden feststellen, dass die dort erzielten Ergebnisse, welche $MST_3$ als unsicher bezeichnen, auf keiner soliden Argumentationsbasis stehen und teilweise sogar stark anzuzweifeln sind. Dies soll nicht nur theoretisch, sondern auch durch durch gezielte Computeranalyse geschehen, welche dann auch letztlich die Arbeit abschließen wird. Weiterhin konnte eine Beweisidee aus [3] verwendet und abgewandelt werden um einen Faktorisierungsalgorithmus für bestimmte Signaturen zu entwickeln. Dieser wird am Anfang des fünften Kapitels vorgestellt und stellt eine zentrale Erkenntnis dieser Arbeit dar.

Die Arbeit ist wie folgt gegliedert:
Kapitel 2 führt die grundlegensten Begriffe ein und weist auf Tatsachen hin, die im späteren Verlauf häufig verwendet werden.
In Kapitel 3 betrachten wir sogenannte Cover und logarithmische Signaturen, wodurch wir dann in Kapitel 4 in der Lage sind das Kryptosystem $MST_3$ zu analysieren.
Kapitel 5 behandelt die bereits erwähnte Kryptoanalyse [3], zeigt die Mängel auf und belegt schließlich die Sicherheit von $MST_3$. Der erwähnte Faktorisierungsalgorithmus ist ebenfalls hier zu finden.

# 2. Grundlagen

## 2.1. Definitionen und Beispiele

Um einen gemeinsamen Bezugspunkt zu haben listen wir im Folgenden die grundlegendsten Definitionen der Kryptographie auf, wie man sie in nahezu jeder Einführung finden kann (z.B. [4]). Wir wollen in dieser Arbeit allerdings nicht näher auf die Grundlagen der Zahlentheorie eingehen, sondern verweisen auch hierfür auf [4].

**Definition 2.1.** Als *Kryptosystem* bezeichnet man ein 5-Tupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$. Hierbei gilt:

1. Der Klartextraum $\mathcal{P}$ ist eine endliche Menge, deren Elemente Klartexte heißen.
2. Der Chiffretextraum $\mathcal{C}$ ist eine endliche Menge, deren Elemente Chiffretexte heißen.
3. Der Schlüsselraum $\mathcal{K}$ ist eine endliche Menge, deren Elemente Schlüsseln heißen.
4. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{P} \to \mathcal{C}$, deren Elemente Verschlüsselungsfunktionen genannt werden.
5. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \to \mathcal{P}$, deren Elemente Entschlüsselungsfunktionen genannt werden.
6. Für jedes $e \in \mathcal{K}$ gibt es ein $d \in \mathcal{K}$, dass für alle $x \in \mathcal{P}$ die Gleichung $D_d(E_e(x)) = x$ erfüllt.

Klassische Beispiele verwenden als Klar- bzw. Chiffretexträume das Alphabet oder entsprechend $\mathbb{Z}_{26}$, wogegen es heute üblich ist für die Verwendung mit Computern Nachrichten als Bitstrings zu behandeln, also $\mathcal{P} = \mathcal{C} = \{0,1\}^l$ mit $l \in \mathbb{N}$.

**Definition 2.2.** Wir nennen ein Kryptosystem *symmetrisch* oder bezeichnen es als *Private-Key-Verfahren*, wenn der Verschlüsselungsschlüssel $e$ mit dem entsprechenden Entschlüsselungsschlüssel $d$ übereinstimmt oder leicht aus $e$ zu berechnen ist. Bei einem sogenannten *asymmetrischen* Kryptosystem, besser bekannt als *Public-Key-Verfahren*, sind $d$ und $e$ verschieden und nicht leicht voneinander ableitbar.

Beim Private-Key-Verfahren müssen Sender und Empfänger - wir nennen sie üblicherweise Alice und Bob - bereits vorab einen geheimen Schlüssel ausgemacht haben oder mittels eines sicheren Kanals austauschen. Wird ein Public-Key-Verfahren verwendet, so ist dies nicht nötig. Möchte Alice chiffrierte Nachrichten empfangen, so veröffentlicht sie den Verschlüsselungsschlüssel und behält den Entschlüsselungsschlüssel geheim. Somit kann jeder verschlüsselte Nachrichten senden, aber nur Alice kann sie dechiffrieren. Betrachten wir nun zwei ausgewählte Beispiele.

## 2.1.1. Beispiele

### 2.1.1.1. Permutations-Chiffre

Wähle $m, l \in \mathbb{N}$, $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_m)^l$ und $\mathcal{K} = Sym(l) = Sym(\{1, 2, ..., l\})$. Für ein $\pi \in \mathcal{K}$ definieren wir:

$$\begin{aligned} e_\pi((x_1, ..., x_l)) &= (x_{\pi(1)}, ..., x_{\pi(l)}) \\ d_\pi((y_1, ..., y_l)) &= (y_{\pi^{-1}(1)}, ..., y_{\pi^{-1}(l)}) \end{aligned}$$

wobei $\pi^{-1}$ die inverse Permutation von $\pi$ ist.
Nach unserer Definition ist $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \{e_\pi : \pi \in K\}, \{d_\pi : \pi \in K\}$ ein symmetrisches Kryptosystem.
Konkret wird beispielsweise mit $m = 26$, $l = 6$, $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 6 & 1 & 3 & 5 \end{pmatrix}$ der Klartext KRYPTOSYSTEM zu RPOKYTYTMSSE chiffriert. Wir sehen hier, dass dieses System polyalphabetisch ist, denn T wird sowohl zu Y, als auch zu S chiffriert.

### 2.1.1.2. El-Gamal

Die Sicherheit des El-Gamal Systems beruht auf dem diskreten Logarithmusproblem (DLP) in $\mathbb{Z}_p^*$, d.h. es soll schwer sein zu gewählter Primitivwurzel $g \in \mathbb{Z}_p^*$ und einem Element $z \in \mathbb{Z}_p^*$ ein $n \in \{0, 1...p-1\}$ zu finden, so dass $g^n = z$.

*Schlüsselgenerierung:* Alice wählt

- eine große Primzahl $p$ derart, dass das diskrete Logarithmusproblem in $\mathbb{Z}_p^*$ schwer zu lösen ist
- eine Primitivwurzel $g \in \mathbb{Z}_p^*$
- zufällig und gleichverteilt einen Exponenten $a \in \{1, ..., p-2\}$ und bestimmt zudem $A := g^a \, mod \, p$.

Der öffentliche Schlüssel des Public-Key-Systems ist $(p, g, A)$, während der geheime Schlüssel lediglich aus $a$ besteht. Also ist insbesondere $\mathcal{P} = \mathbb{Z}_p^*$ und $\mathcal{C} = \mathbb{Z}_p^* \otimes \mathbb{Z}_p^*$.

*Verschlüsselung:* Will Bob eine Nachricht $m \in \mathbb{Z}_p^*$ an Alice senden, so wählt er eine Zufallszahl $b \in \{1, ..., p-2\}$ und berechnet $B := g^b \, mod \, p$ und $C := A^b m \, mod \, p$.
Er sendet $(B, C)$ an Alice.

*Entschlüsselung:* Alice berechnet $X := B^{(p-1)-a} C \, mod \, p$ und kann den Klartext lesen, denn da $g^{p-1} = 1 \, mod \, p$ gilt:

$$B^{(p-1)-a} C = g^{b(p-1)-ba} g^{ab} m = (g^{(p-1)})^b m = m$$

Die Anzahl der verschiedenen kryptographischen Verfahren ist besonders im Laufe des

letzten Jahrhunderts enorm gestiegen, daher wäre es nicht zielführend an dieser Stelle weitere Systeme aufzulisten. Lediglich das RSA-Verfahren sei hier noch am Rande erwähnt, denn hier wird das Faktorisierungsproblem genutzt, was in seinem Schwierigkeitsgrad bisher dem diskreten Logarithmusproblem ähnelt. Wählt man zwei große Primzahlen $p$, $q$ und berechnet $n = pq$, so gilt es im Allgemeinen als unmöglich in vertretbarer Zeit $n$ zu faktorisieren, wenn $p$ und $q$ geheim sind. Details findet man z.B. in [4].

### 2.1.2. Kryptoanalyse

Die Kryptoanalyse ist die Lehre von den Angriffen auf Kryptosysteme.
Üblicherweise nimmt man an, dass dem Angreifer (Oskar) das verwendete Kryptosystem bekannt ist und dass sowohl der verwendete Schlüssel, als auch der Klartext geheim sind. Wir unterscheiden fünf Typen von Angriffen:

- *Ciphertext-Only-Attacke:* Der Angreifer kennt nur den Chiffretext und sucht den zugehörigen Klartext oder den verwendeten Schlüssel.
- *Known-Plaintext-Attacke:* Der Angreifer kennt (mindestens) einen Klartext und den zugehörigen Chiffretext. Er sucht den verwendeten Schlüssel oder versucht andere Chiffretexte zu entschlüsseln.
- *Chosen-Plaintext-Attacke:* Der Angreifer kann zu selbst gewählten Klartexten die Chiffretexte erzeugen, ohne aber den Schlüssel zu kennen. Er sucht selbigen oder versucht andere Chiffretexte zu entschlüsseln.
- *Chosen-Ciphertext-Attacke:* Der Angreifer kann ohne den Schlüssel zu kennen selbst gewählte Chiffretexte entschlüsseln. Er sucht den Schlüssel.
- *Chosen-Text-Attacke:* Chosen-Plaintext und Chosen-Ciphertext zusammen.

Die Angriffsarten sind nach Gefahrenstufe sortiert, wobei die Ciphertext-Only-Attacke die ungefährlichste, aber auch die üblichste, ist.
Man unterscheidet weiterhin zwischen *passiven* und *aktiven* Attacken. Im ersten Fall spricht man auch von Spionage, im zweiten von Sabotage. Eine spezielle Angriffsart ist die sogenannte *Brute-Force-Attacke* (Exhaustive Search), hier werden alle möglichen Schlüssel ausprobiert. Ein sehr großer Schlüsselraum verhindert dies sofort.
Zuletzt sollte noch der Begriff des *Orakels* geklärt werden: Unter bestimmten Bedingungen kann es vorkommen, dass der Angreifer eine Art „Blackbox" zu Verfügung hat (z.B. weil sie öffentlich ist), welche es ihm gestattet eine Eingabe zu machen um sofort eine Auswertung zu erhalten. Die Arbeitsschritte dieser Blackbox sind dem Angreifer unbekannt und manchmal kann man auch nur eine bestimmte Anzahl von Abfragen starten. Daher wird diese Blackbox auch Orakel genannt.

### 2.1.3. Geburtstagsparadoxon

Das sogenannte Geburtstagsparadoxon (siehe z.B. [4]) behandelt folgende Frage: Wieviele Leute müssen in einem Raum sein, damit die Chancen gut stehen ($> \frac{1}{2}$), dass wenigstens

zwei von ihnen am gleichen Tag Geburtstag haben?
Die Antwort wurde in der Wahrscheinlichkeitstheorie schnell beantwortet und lautet $1,17 \cdot \sqrt{365} \approx 22,3$, das heißt, es genügen 23 Personen. Dieses Ergebnis überrascht viele, da es nicht intuitiv erscheint, weswegen man von einem Paradoxon spricht. Interessant für uns ist jedoch, dass man die Idee verallgemeinern kann, denn im Grunde wird in diesem Versuch nach Kollisionen von Geburtstagen gesucht. Will man also eine große Liste von Elementen auf doppelte Einträge, also Kollisionen, prüfen, so genügt es durchschnittlich $1,17 \cdot \sqrt{m}$ Elemente zu kontrollieren, wenn $m$ die Gesamtzahl der Elemente in der Liste ist. Diese nützliche Eigenschaft werden wir später noch gebrauchen können.

## 2.1.4. Sonstiges

Wie bereits angedeutet müssen wir davon ausgehen, dass Grundlagen der endlichen Gruppentheorie bekannt sind, jedoch sollen an dieser Stelle ein paar für uns im weiteren Verlauf relevante Themen angesprochen werden. Man kann sie wohl in fast jedem Grundlagenbuch für Algebra nachlesen, beispielsweise in [7]. Anzumerken ist noch, dass $e$ (bzw. 1 oder 0) im Allgemeinen das neutrale Element der jeweiligen Gruppe meint. Ausnahmen werden explizit erwähnt.

### 2.1.4.1. Konjugation

Sei $G$ eine (im folgenden stets endliche) Gruppe, so wird für ein $g \in G$ der innere Automorphismus von $G$ folgendermaßen definiert:

$$int_g : G \to G \text{ mit } h \mapsto ghg^{-1}$$

Die Gruppe der inneren Automorphismen von $G$ wird mit $\mathrm{Inn}(G)$ bezeichnet und ist bekanntermaßen ein Normalteiler der Automorphismengruppe $\mathrm{Aut}(G)$. Somit können wir das Zentrum von $G$ auch als Kern von $T : G \to \mathrm{Inn}(G)$ mit $g \mapsto int_g$ ansehen, da

$$\forall g, h \in G : T(g)(h) = int_g(h) = ghg^{-1} \stackrel{!}{=} h \Leftrightarrow g \in Z(G) = \{g \in G : hg = gh \, \forall h \in G\}$$

Mit dem Homomorphiesatz folgt also, dass $G/Z(G)$ ismorph zu $\mathrm{Inn}(G)$ ist.

### 2.1.4.2. Gruppenringe

Sei $R$ ein kommutativer Ring mit Eins und $G$ ein Monoid (bzw. bei uns eine Gruppe), so ist

$$R[G] := \{f : G \to R : f(g) = 0 \text{ für alle bis auf endlich viele } g \in G\}$$

mit der Addition

$$(f_1 + f_2)(g) := f_1(g) + f_2(g)$$

und der Multiplikation

$$(f_1 f_2)(g) := \sum_{hi=g} f_1(h) f_2(i)$$

ein Ring. Man kann dies als eine Verallgemeinerung des Polynomrings ansehen. Im Übrigen ist $1 \cdot e$ das Einselement von $R[G]$, wobei 1 das neutrale Element von $R$ und $e$ das Neutralelement von $G$ ist.
Wir bemerken abschließend, dass $R[G]$ abelsch ist genau dann, wenn $G$ abelsch ist.

## 2.2. Die Suzuki-2-Gruppen

Im weiteren Verlauf dieser Arbeit wollen wir konkrete Beispiele und sinnvolle Implementierungsgrundlagen für ein neues Kryptosystem betrachten, allerdings benötigen wir dafür eine spezielle Gruppenart. Diese wollen wir nun schrittweise einführen und ihre - für uns - wichtigsten Eigenschaften kennenlernen. Wir stützen uns dabei insbesondere auf die Arbeit [6] von Higman, sowie auf Hinweise aus [1, 5]. Da wir in dieser Arbeit diese Gruppen hauptsächlich für Beispiele benötigen, wollen wir die meisten Eigenschaften ohne Beweise auflisten und verweisen für letztere wieder auf [6]. Zunächst aber wollen wir allgemein p-Gruppen definieren.

**Definition 2.3.** Sei $p$ eine Primzahl. Eine endliche Gruppe $G$ der Ordnung $p^n$ ($n \in \mathbb{N}$) heißt *p-Gruppe*. Die kleinst-mögliche Ordnung von Elementen aus $G$ ist der *Exponent* von $G$, d.h. das kleinste $n \in \mathbb{N}$ mit $x^n = e \, \forall x \in G$. Eine abelsche $p$-Gruppe mit Exponent $p$ nennt man *elementar-abelsch*.

**Beispiel.** Eine sehr einfache 2-Gruppe ist die symmetrische Gruppe $S_2$ mit $2! = 2^1$ Elementen. Da $S_2 = \{e, (12)\}$ und $(12) \cdot (12) = e$ ist der Exponent ebenfalls 2 und die Gruppe ist daher elementar-abelsch. Ein Beispiel für eine nicht-abelsche 2-Gruppe ist die Diedergruppe $D_8$ mit $2^4 = 16$ Elementen.
Wenden wir uns nun besonderen Untergruppen zu, die wir interessanterweise später (unter bestimmten Bedingungen) alle zu einer zusammenfassen können.

**Definition 2.4.** Sei $G$ eine endliche Gruppe. Die *Kommutator-Untergruppe* von $G$ ist definiert durch $G' :=< \{x^{-1}y^{-1}xy : x, y \in G\} >$. Die sogenannte *Frattini-Untergruppe* von $G$ ist durch den Schnitt aller maximalen Untergruppen von $G$ gegeben. Wir bezeichnen sie üblicherweise mit $\Phi(G)$.

Die Größe der Kommutator-Untergruppe gibt an, wie weit die Gruppe von der Kommutativität entfernt ist, denn offensichtlich ist die Gruppe genau dann abelsch, wenn $G' = \{e\}$. Weiterhin ist allgemein bekannt, dass $\Phi(G)$ ein Normalteiler von $G$ ist, daher können wir die Faktor-Gruppe bilden.

**Proposition 2.1.** *Wenn G eine p-Gruppe ist, so ist die Faktorgruppe $G/\Phi(G)$ elementar abelsch.*

Der Beweis erfordert die Einführung weiterer Begriffe und würde daher hier unnötig lang werden. Hier soll ein Verweis auf [8] genügen.

**Proposition 2.2.** *Jede endliche p-Gruppe ($\neq \{e\}$) besitzt ein nicht-triviales Zentrum.*

Diese Behauptung wird über die bekannte Klassengleichung und die Tatsache, dass $p$ prim ist, bewiesen. Details finden sich z.B. in [9]. Diese Proposition wird später noch wichtig werden, da wir nur mit Gruppen arbeiten wollen, die ein nicht-triviales Zentrum besitzen, was also die Suzuki-2-Gruppen erfüllen.
Wir können nun zur zentralen Definition dieses Abschnitts vorrücken.

**Definition 2.5.** Eine *Suzuki-2-Gruppe* ist definiert als eine nicht-abelsche 2-Gruppe mit mehr als einer *Involution* (d.h. die Elementordnung ist 2) und einer zyklischen Gruppe von Automorphismen, die die Involutionen permutieren.

Wir listen nun einige Eigenschaften auf, die im späteren Verlauf noch benutzt werden.

**Proposition 2.3.** *Ist $G$ eine Suzuki-2-Gruppe, so gilt:*

- $Z(G) = \Phi(G) = G' = \Omega_1(G) :=< g \in G : g^2 = e >$
- $|Z(G)| = q = 2^m$ *mit* $1 < m \in \mathbb{N}$
- $|G| = q^2$ *oder* $|G| = q^3$.
- $Z(G)$ *und* $G/\Phi(G)$ *sind elementar abelsch.*

Wir folgern sofort, dass der Exponent von $G$ gleich 4 sein muss, denn ist $g \in G\backslash Z(G)$, so ist $ord(g^2) = 2$ und daher ist $ord(g) = 4$.

Im Folgenden betrachten wir nur den Fall $|G| = q^2$ mit $q = 2^m$ und $m > 2$, so dass der endliche Körper $\mathbb{F}_q$ einen nicht-trivialen Automorphismus von ungerader Ordnung besitzt, wobei die Ordnung angibt, wie oft die Abbildung mit sich selbst verkettet werden muss, bis die Identitätsabbildung erscheint. Unter diesen Bedingungen existiert eine Suzuki-2-Gruppe, welche nach Higman mit $A(m, \Theta)$ bezeichnet wird. Glücklicherweise gibt es eine einfache Möglichkeit mit diesen Gruppen zu arbeiten.

**Definition 2.6.** Sei $G := \{S(a, b) : a, b \in \mathbb{F}_q\}$ mit

$$S(a,b) := \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & a^\Theta & 1 \end{pmatrix}$$

wobei $a^\Theta$ gleichbedeutend mit $\Theta(a)$ ist. Man kann zeigen, dass $G \backsimeq A(m, \Theta)$ ist. Üblicherweise verwendet man den Frobenius-Automorphismus (insbesondere das Quadrieren) als $\Theta$.

Da $a, b \in \mathbb{F}_q$ sind, folgt direkt, dass $|G| = q^2$. Mittels der vorherigen Propositionen können wir zudem folgern, dass $Z(G) = \Phi(G) = G' = \Omega_1(G) = \{S(0, b) : b \in \mathbb{F}_q\}$ und $|Z(G)| = q$. Dies ist klar, denn

$$\begin{aligned} S(a,b)S(c,d) &= \begin{pmatrix} 1 & 0 & 0 \\ a+c & 1 & 0 \\ b+ca^\Theta+d & a^\Theta+c^\Theta & 1 \end{pmatrix} \\ S(c,d)S(a,b) &= \begin{pmatrix} 1 & 0 & 0 \\ a+c & 1 & 0 \\ b+c^\Theta a+d & a^\Theta+c^\Theta & 1 \end{pmatrix} \end{aligned}$$

Ein Vergleich der Einträge zeigt, dass $a$ (oder $c$) gleich Null sein muss. Aus dieser Rechnung und mit dem Gaußverfahren zum Invertieren von Matrizen folgern wir weiterhin die nützlichen Regeln:

$$\begin{aligned} S(a,b)S(c,d) &= S(a+c, b+d+a^{\Theta}c) \\ S(a,b)^{-1} &= S(a, a^{\Theta}a+b) \end{aligned}$$

Da $\det(S(a,b)) = 1$ können wir die Suzuki-2-Gruppen als Untergruppen von $GL_3(\mathbb{F}_q)$ auffassen. Weiterhin ergibt sich durch Betrachtung der Matrix $S(a,b)$:

**Proposition 2.4.** *Mit der eingeführten Notation gilt:* $A(m,\Theta) \backsimeq A(m,\Psi) \Leftrightarrow \Psi = \Phi^{\pm 1}$.

Abschließend zeigen wir noch zwei Eigenschaften, die sich später als nützlich erweisen werden.

**Proposition 2.5.** $\forall x, y \in G$ *mit* $ord(x) = ord(y) = 4$, *die verschiedenen Nebenklassen zu* $Z(G)$ *angehören, gilt:* $xy \neq yx$.

*Beweis.* Zunächst ist klar, dass wenn $x, y$ aus der gleichen Nebenklasse wären, so ist

$$[x][y] = [x][x] = \{xuxv \,:\, u,v \in Z(G)\} = \{x^2uv \,:\, u,v \in Z(G)\} \overset{x^2 \in Z(G)}{=} Z(G)$$

Daher gilt

$$[x] \neq [y] \Rightarrow [x][y] = [xy] \neq Z(G) \Rightarrow xy \neq yx$$

□

**Proposition 2.6.** *Sind* $a, c \in \mathbb{F}_q \backslash \{0\}$ *und* $a \neq c$, *so gehören* $S(a,b)$ *und* $S(c,d)$ *verschiedenen Nebenklassen unter* $Z(G)$ *an.*

*Beweis.* Da $S(a,b)S(0,d) = S(a,b+d)$ entscheidet nur das $a$ über die Nebenklasse. □

Die hier eingeführte Matrizen-Schreibweise für Suzuki-2-Gruppen erlaubt eine effiziente Implementierung in den Computer. Üblicherweise speichert man dann ein Element $S(a,b)$ als Tripel $(a, b, a^{\Theta})$ ab um so Rechenleistung zu sparen.

# 3. Cover und logarithmische Signaturen

Die Begriffe des Covers (oder auch „Überdeckung") und der logarithmischen Signatur wurden bereits in einigen Arbeiten (z.B. [2, 1, 5]) erläutert und lassen uns das Faktorisierungsproblem auch auf abstraktere Elemente als den ganzen Zahlen anwenden. Diese speziellen Mengen werden uns die gesamte Arbeit über begleiten und sollen daher im Folgenden ausführlich eingeführt werden. Konkrete Beispiele werden stets das Verständnis erleichtern.

## 3.1. Definitionen und Transformationen

Wir wollen uns zunächst den grundlegensten Definitionen widmen und dann die daraus folgenden Eigenschaften ableiten.

### 3.1.1. Grundlegende Definitionen

Sei $G$ eine endliche Gruppe. Wir definieren die *Breite* von G als $\omega := \lceil log_2|G| \rceil$, um so die benötigte Speicherkapazität abschätzen zu können.
Im Folgenden meinen wir mit $G^{[\mathbb{Z}]}$ die *Menge aller endlichen Folgen* von Elementen in $G$ und betrachten die Elemente von $G^{[\mathbb{Z}]}$ als einzeilige Matrizen mit Einträgen in $G$.
Seien also $X = [x_1, ..., x_r]$, $Y = [y_1, ..., y_s] \in G^{[\mathbb{Z}]}$, so definieren wir die Multiplikation folgendermaßen:

$$X \cdot Y = XY := [x_1y_1, x_1y_2, ..., x_ry_s]$$

Damit hat $XY$ insgesamt $r \cdot s$ Einträge. Weiterhin setzen wir

$$\overline{X} := \sum_{i=1}^{r} x_i \in \mathbb{Z}[G]$$

wobei $\mathbb{Z}[G]$ der Gruppenring ist (siehe 2.1.4.2).

**Definition 3.1.** Sei $\alpha = [A_1, ..., A_s]$ eine Folge von $A_i \in G^{[\mathbb{Z}]}$ und $\sum_{i=1}^{s} A_i$ ist beschränkt durch ein Polynom in $log|G|$ (d.h. für die Anwendung mit dem Computer geeignet). Sei weiterhin $S$ eine Teilmenge von $G$ und

$$\overline{A_1} \cdot ... \cdot \overline{A_s} = \sum_{g \in G} a_g g \text{ für } a_g \in \mathbb{Z}$$

so sagen wir:

1. $\alpha$ ist ein *Cover* von $G$ (oder $S$), wenn $a_g > 0 \, \forall g \in G$ (bzw. $\forall g \in S$)

2. $\alpha$ ist eine *logarithmische Signatur* von $G$ (oder $S$), wenn $a_g = 1 \forall g \in G$ (bzw. $\forall g \in S$)

3. Ist $\alpha$ ein Cover, so nennen wir es ein *gleichmäßiges Cover*, wenn $\frac{max\{a_g : g \in G\}}{min\{a_g : g \in G\}} = 1$.

Wir sehen sofort, dass natürlich jede logarithmische Signatur auch ein gleichmäßiges Cover ist. Weiterhin ergibt sich aus der Definition sofort, dass unter einer logarithmischen Signatur $\alpha$ jedes $y \in G$ eindeutig als Produkt dargestellt werden kann: $y = q_1 \cdot ... \cdot q_s$ mit $q_i \in A_i$ für $i = 1, ..., s$. In sehr großen Gruppen ist es nahezu unmöglich eben diese Faktorisierung zu finden, weswegen der Vergleich mit dem klassischen Faktorisierungsproblem nahe liegt.

**Definition 3.2.** Sei $\alpha = [A_1, ..., A_s] =: (\alpha_{ij})$ ein Cover von G mit $r_i := |A_i|$ für $i = 1, .., s$. Wir nennen die $A_i$ die *Blöcke* von $\alpha$ und den Vektor $(r_1, ..., r_s)$ den *Typ* von $\alpha$. Die *Länge* von $\alpha$ ist $l = \sum_{i=1}^{s} r_i \in \mathbb{Z}$. Wir wollen $\alpha$ *nicht-trivial* nennen, wenn $s \geq 2$ und $r_i \geq 2 \forall i = 1, ..., s$, ansonsten ist es trivial. Mit $C(G)$ und $\Lambda(G)$ bezeichnen wir die Mengen der Cover bzw. der logarithmischen Siganturen von $G$. Weiterhin wird ein gleichmäßiges Cover vom Typ $(r, ..., r)$ auch $[s, r] - Sieb$ genannt.

Betrachten wir nun ein Beispiel zur Verdeutlichung.

**Beispiel.** Sei $G$ die alternierende Gruppe $\mathfrak{A}_4$, d.h. $G$ hat $|G| = 12$ Elemente. Betrachten wir $\alpha = [A_1, A_2]$ mit

$$A_1 = \{e, (132), (123)\},\ A_2 = \{e, (243), (13)(24), (124)\}$$

so liegt eine nicht-triviale logarithmische Sigantur der Länge $l = 7$ vom Typ $(3, 4)$ vor, denn

$$\begin{aligned} \overline{A_1} \cdot \overline{A_2} &= (e + (132) + (123))(e + (243) + (13)(24) + (124)) \\ &= e + (243) + (13)(24) + (124) + (132) + (12)(34) \\ &\quad +(234) + (134) + (123) + (143) + (142) + (14)(23) \\ &= \sum_{g \in \mathfrak{A}_4} a_g g \text{ mit } a_g = 1 \forall g \in \mathfrak{A}_4 \end{aligned}$$

Manchmal kann es sinnvoll sein eine Signatur als $(s, max\{r_i, i = 1, ..., s\})-$Matrix darzustellen. Dies erfolgt sehr natürlich und würde in diesem Beispiel folgendermaßen aussehen:

$$\alpha = \begin{pmatrix} e & (132) & (123) & e \\ e & (243) & (13)(24) & (124) \end{pmatrix}$$

wobei wir eventuelle Leerstellen mit „Nullen" auffüllen.
Wir können in diesem Beispiel jedes Element eindeutig als Produkt von Elementen aus $A_1$ und $A_2$ schreiben. Beispielsweise ist $(234) = (132) \cdot (13)(24) = \alpha_{12} \cdot \alpha_{23}$.
Gerade diese Faktorisierung ist für uns interessant, daher wollen wir nun einen weiteren wichtigen Begriff einführen.

### 3.1.2. Faktorisierbarkeit

**Definition 3.3.** Sei $\Gamma = \{(G_l, \alpha_l)\}_{l\in\mathbb{N}}$ eine Familie von Paaren mit Indexparameter $l$, wobei $G_l$ Gruppen sind und $\alpha_l$ spezielle Cover von $G_l$ mit polynomialer Länge in $l$ darstellen. Die Familie $\Gamma$ (und insbesondere auch ein einzelnes Paar) heißt *zahm* (bzw. „tame"), wenn es einen probalistischen Algorithmus in polynomieller Zeit gibt, welcher für alle $g \in G_l$ das Paar $(\alpha_l, g)$ als Input akzeptiert und mit sehr hoher Wahrscheinlichkeit eine Faktorisierung von $g$ unter $\alpha$ zurück gibt. Ist diese Wahrscheinlichkeit gering, so heißt $\Gamma$ *wild*.

Man kann statt „zahm" auch „faktorisierbar" sagen, was den Kern der Sache ganz gut trifft. Auch wenn die Definition Familien von Paaren betrachtet, so ist es oft üblich lediglich die Cover als zahm oder wild zu bezeichnen, da man die zugehörigen Gruppen als bekannt voraussetzt. Im Allgemeinen betrachten wir solche Familien als einelementige Mengen. Im letzten Beispiel liegt natürlich eine zahme logarithmische Sigantur vor, da wir sogar per Hand alle Möglichkeiten in kürzester Zeit ausprobieren könnten, um die passende Faktorisierung zu finden. Bei größeren Gruppen ist es jedoch sehr schwer eine Aussage über das Faktorisierungsverhalten zu machen, sofern keine nützlichen Eigenschaften vorliegen. Man beachte: Selbst wenn man keinen solchen Algorithmus (mit hoher Output-Wahrscheinlichkeit) finden konnte, so hat man damit noch nicht bewiesen, dass das Cover wild ist. Es gibt aber Fälle, in denen man sich recht sicher sein kann.

**Beispiel.** Sei $q$ eine Primzahlpotenz, d.h. $q = p^n$ für eine Primzahl $p$ und ein $n \in \mathbb{N}$, für die das diskrete Logarithmusproblem in $(\mathbb{F}_q, \cdot)$ schwer zu lösen ist. Sei $2^{l-1} \leq q - 1 \leq 2^l$ und $G_l = \mathbb{F}_q^*$ (Einheitengruppe). Weiterhin sei $f$ ein Erzeuger von $G_l$, also $ord(f) = |G_l|$, und $\alpha_l = [A_1, ..., A_l]$ mit $A_i = [e, f^{2^{i-1}}]$, so ist $\alpha_l$ ein Cover von $G_l$ und die Faktorisierung beläuft sich auf das Lösen des DLP in $G_l$. Denn:

$$\begin{aligned}\overline{A_1} \cdot ... \cdot \overline{A_l} &= (1+f) \cdot ... \cdot (1+f^{2^{l-1}}) \\ &= 1 + f + f^2 + ... + f^{2^{l-1}} + ... + f^{2^l-1} \\ &\stackrel{!}{=} \sum_{g\in G} a_g g\end{aligned}$$

und da $f$ ein Erzeuger ist muss $a_g > 0 \, \forall g \in G$ gelten. Das DLP wurde bereits in 2.1.1.2 beschrieben und daher sehen wir, dass die Wildheit so lange gegeben ist, wie das DLP als schwer gilt. Insbesondere könnten also zukünftige Computer hier viele Veränderungen mit sich bringen, was aber - wie wir später sehen werden - das neue Kryptosystem nur wenig beeinträchtigen wird.

### 3.1.3. Äquivalenz von Covern

Äquivalente Cover und Signaturen zeichnen sich dadurch aus, dass sie sich in der Faktorisierung von Elementen sehr ähnlich sind. Kennen wir die Faktorisierung eines bestimmten Elements unter der einen Signatur, so kennen wir sie automatisch auch unter allen dazu

äquivalenten Signaturen. Insbesondere kann man sofort sehen, wenn eine Signatur zahm ist, so sind es auch alle dazu Äquivalenten.

**Definition 3.4.** Sei $\alpha = [A_1, ..., A_s]$ ein Cover vom Typ $(r_1, ..., r_s)$ zu einer Gruppe $G$. Wir setzen $m := \prod_{i=1}^{s} r_i$ und $m_i := \prod_{j=1}^{i-1} r_j \, \forall i = 2, ..., s$, wobei $m_1 := 1$. Die kanonische Bijektion $\iota$ sei zudem gegeben durch:

$$\begin{aligned} \iota : \mathbb{Z}_{r_1} \oplus ... \oplus \mathbb{Z}_{r_s} &\to \mathbb{Z}_m \\ (j_1, ..., j_s) &\mapsto \sum_{i=1}^{s} j_i m_i \end{aligned}$$

Dies induziert eine surjektive Abbildung (da $\alpha$ ein Cover ist!) mittels $\alpha$:

$$\breve{\alpha} : \mathbb{Z}_m \to G \text{ mit } \breve{\alpha}(x) := \alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s}$$

wobei $(j_1, ..., j_s) = \iota^{-1}(x)$. Wir wollen also zwei Cover $\alpha$, $\beta$ *äquivalent* nennen, wenn $\breve{\alpha} = \breve{\beta}$.

Wir bemerken sofort, dass wenn $\iota$ und $\iota^{-1}$ effizient berechenbar sind, so ist es $\breve{\alpha}(x)$ ebenfalls. Andererseits ist klar, dass wenn Cover $\alpha$ und ein $y \in G$ gegeben sind, so braucht man für die Bestimmung von $x = \breve{\alpha}^{-1}(x)$ die mögliche Faktorisierung von $y$. Nach Definition 3.3 ist dies genau dann möglich, wenn $\alpha$ zahm ist.

**Beispiel.** Sei $G = S_4$ die symmetrische Gruppe mit $4! = 24$ Elementen und eine Signatur $\alpha$ vom Typ $(3, 4, 2)$ gegeben durch

$$\alpha = [\{e, (123), (132)\}, \{e, (243), (13)(24), (124)\}, \{e, (1234)\}]$$

Betrachten wir also $\breve{\alpha} : \mathbb{Z}_{24} \to S_4$ und speziell $x = 18$. So ist

$$\begin{aligned} 18 &= \iota(j_1, j_2, j_3) \\ &= m_1 j_1 + m_2 j_2 + m_3 j_3 \\ &= j_1 + 3j_2 + 12j_3 \end{aligned}$$

In einfachen Fällen wie hier kann man leicht das Ergebnis raten, aber wir wollen an dieser Stelle auch ein allgemeineres Verfahren an diesem Beispiel vorstellen:

$$\begin{aligned} 18 : 12 &= 1 \; \textit{Rest} \, 6 \\ 6 : 3 &= 2 \; \textit{Rest} \, 0 \\ 0 : 1 &= 0 \; \textit{Rest} \, 0 \end{aligned}$$

Einfach gesagt wollen wir wissen, wie oft $x$ von $m_s$ geteilt wird, um dann den Rest durch die anderen $m_i$ darzustellen. Hier erhalten wir also $18 = 1 \cdot 0 + 3 \cdot 2 + 12 \cdot 1$ und damit ist $\iota^{-1}(18) = (0, 2, 1)$.
Jetzt sehen wir, dass $\breve{\alpha}(18) = \alpha_{10} \cdot \alpha_{22} \cdot \alpha_{31} = e \cdot (13)(24) \cdot (1234) = (1432)$ und kennen insbesondere die Faktorisierung von $(1432)$.
Wir wollen die Idee auf die ganze Gruppe anwenden. Dazu ist es sinnvoll eine Tabelle anzulegen, wobei die erste Spalte aus einer logarithmischen Signatur $\delta$ von $\mathbb{Z}_{|G|} = \mathbb{Z}_{24}$ und die zweite Spalte aus $\alpha$ besteht:

| $\delta_{\mathbb{Z}_{24}}$ | $\alpha$ |
|---|---|
| 0 | e |
| 1 | (123) |
| 2 | (132) |
| 0 | e |
| 3 | (243) |
| 6 | (13)(24) |
| 9 | (124) |
| 0 | e |
| 12 | (1234) |

Tabelle 3.1.: Hilfstabelle für $\breve{\alpha}$

Wollen wir nun ein $x \in \mathbb{Z}_{24}$ unter $\breve{\alpha}$ abbilden, so suchen wir seine Faktorisierung unter der Signatur zu $\mathbb{Z}_{24}$ in der linken Spalte und können rechts die entsprechenden Elemente der Gruppe ablesen. Analog geht man rückwerts vor, wenn man $\breve{\alpha}^{-1}(g)$ für ein $g \in G$ sucht. Hierfür benötigt man natürlich erst einmal dessen Faktorisierung unter $\alpha$.
Wir wollen den Begriff der Äquivalenz weiter präzisieren, allerdings benötigen wir dazu bestimmte Transformationen, die wir jetzt vorstellen wollen.

### 3.1.4. Sandwich-Transformation

**Definition 3.5.** Sei $\alpha = [A_1, ..., A_s]$ ein Cover einer Gruppe $G$. Seien $g_0, g_1, ..., g_s \in G$ und $\beta = [B_1, ..., B_s]$ durch $B_i := g_{i-1}^{-1} A_i g_i$ definiert. Wir nennen $\beta$ eine *zwei-seitige Transformation* von $\alpha$ bzgl. $g_0, g_1, ..., g_s$. Im Spezialfall $g_0 = g_s = e$ heißt $\beta$ *Sandwich* von $\alpha$.

Folgende nützliche Eigenschaft ist schnell gezeigt:

**Proposition 3.1.** *Ist $\beta$ eine zwei-seitige Transformation eines Covers $\alpha$ einer Gruppe $G$, so ist auch $\beta$ selbst ein Cover zu der gleichen Gruppe.*

*Beweis.* Laut Definition ist $\beta = [g_0^{-1} A_1 g_1, ..., g_{s-1}^{-1} A_s g_s]$. Offensichtlich haben $\alpha$ und $\beta$ den gleichen Typ und die gleiche Länge. Weiterhin sehen wir mit $g_i^{-1} \cdot g_i = e$, dass

$$\begin{aligned}
\overline{B_1} \cdot ... \cdot \overline{B_s} &= g_0^{-1} \overline{A_1} g_1 \cdot ... \cdot g_{s-1}^{-1} \overline{A_s} g_s \\
&= g_0^{-1} \overline{A_1} \cdot ... \cdot \overline{A_s} g_s \\
&\overset{\alpha \in C(G)}{=} g_0^{-1} \cdot (\sum_{g \in G} a_g g) \cdot g_s \\
&\overset{a_g \in \mathbb{Z}}{=} \sum_{g \in G} a_g (g_0^{-1} \cdot g \cdot g_s) \\
&= \sum_{g' \in G} a_g g'
\end{aligned}$$

wobei $g' := g_0^{-1} \cdot g \cdot g_s$. Da $\alpha$ ein Cover ist, gilt $a_g > 0$, was unsere Behauptung beweist. □

Wir werden in späteren Betrachtungen manchmal annehmen wollen, dass ohne Beschränkung der Allgemeingültigkeit Signaturen, deren Blöcke (bis auf der letzte) stets mit dem neutralen Element beginnen, die Gesamtheit aller Signaturen vertreten können.

**Proposition 3.2.** *Man kann zu jeder logarithmischen Signatur eine äquivalente Signatur durch Sandwich-Transformation finden, so dass das erste Element eines jeden Blocks, außer beim letzten Block, das neutrale Element der zugrunde liegenden Gruppe ist.*

*Beweis.* Sei $\alpha = [A_1, ..., A_s]$ eine logarithmische Signatur zu einer Gruppe $G$. Wir führen eine Sandwich-Transformation mit

$$g_0 = e,\ g_1 = \alpha_{11}^{-1},\ g_2 = (\alpha_{11}\alpha_{21})^{-1}, ..., g_{s-1} = (\alpha_{11}\alpha_{21} \cdot ... \cdot \alpha_{(s-1)1})^{-1},\ g_s = e$$

durch und erhalten

$$\begin{aligned} \beta_{11} &= e \cdot \alpha_{11} \cdot \alpha_{11}^{-1} = e \\ \beta_{21} &= \alpha_{11} \cdot \alpha_{21} \cdot \alpha_{21}^{-1} \cdot \alpha_{11}^{-1} = e \\ &... \\ \beta_{(s-1)1} &= \alpha_{11}\alpha_{21} \cdot ... \cdot \alpha_{(s-2)1} \cdot \alpha_{(s-1)1} \cdot (\alpha_{11}\alpha_{21} \cdot ... \cdot \alpha_{(s-1)1})^{-1} = e \end{aligned}$$

Da laut Proposition 3.1 unser $\beta$ wieder eine Signatur sein muss, ist die Behauptung gezeigt. □

Diesen Vorgang wollen wir *Normalisierung* nennen.
Wie bereits angekündigt sollen uns diese Erkenntnisse ermöglichen den Begriff der Äquivalenz von Signaturen zu präzisieren:

**Theorem 3.1.** *Seien $\alpha, \beta$ zwei logarithmische Signaturen von $G$ mit Typ $(r_1, ...r_s)$, so gilt, dass $\alpha$ und $\beta$ äquivalent sind genau dann, wenn $\alpha$ ein Sandwich von $\beta$ (oder anders herum) ist.*

*Beweis.* Seien also $\alpha = [A_1, ..., A_s]$ und $\beta = [B_1, ..., B_s]$ zwei Signaturen des gleichen Typs.
Rückrichtung („⇐"): Es seine $A_i = g_{i-1}^{-1} B_i g_i \, \forall i = 1...s$ mit $g_i \in G \forall i = 1...s-1$ und $g_0 = g_s = e$. Wir prüfen einfach die Definition 3.4 für $x \in \mathbb{Z}_{|G|}$ nach:

$$\begin{aligned} \breve{\alpha}(x) &= \alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s} \\ &= g_0^{-1}\beta_{1j_1}g_1 \cdot ... \cdot g_{s-1}^{-1}\beta_{sj_s}g_s \\ &= \beta_{1j_1} \cdot ... \cdot \beta_{sj_s} \\ &= \breve{\beta}(x) \end{aligned}$$

Daher sind die Signaturen äquivalent.
Hinrichtung („⇒"): Da nun $\breve{\alpha}(x) = \breve{\beta}(x)$ für alle $x \in \mathbb{Z}_{|G|}$ gilt, wissen wir, dass

$$\alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s} = \beta_{1j_1} \cdot ... \cdot \beta_{sj_s}$$

und daher sehen wir mit $j_1 = 1$, dass

$$\alpha_{11} = \beta_{11} \cdot \beta_{1j_2} \cdot ... \cdot \beta_{sj_s} \cdot (\alpha_{1j_s} \cdot ... \cdot \alpha_{sj_s})^{-1}$$

wobei wir nun $g_0 = e$ und $g_1 = \beta_{1j_2} \cdot ... \cdot \beta_{2j_s} \cdot (\alpha_{1j_s} \cdot ... \cdot \alpha_{sj_s})^{-1}$ setzen können. Analog ergeben sich alle Werte für $\alpha_{1j}$ $(j = 2...r_1)$. Wir betrachten noch den Fall $j_2 = 1$ und können dann induktiv fortfahren.

$$\alpha_{21} = \alpha_{11}^{-1}\beta_{11} \cdot \beta_{21} \cdot \beta_{3j_3} \cdot ... \cdot \beta_{sj_s} \cdot (\alpha_{1j_s} \cdot ... \cdot \alpha_{sj_s})^{-1}$$

Da $g_1^{-1} = \alpha_{2j_2} \cdot ... \cdot \alpha_{sj_s} \cdot \beta_{sj_s}^{-1} \cdot ... \cdot \beta_{2j_2}^{-1} = \alpha_{11}^{-1} \cdot \beta_{11}$ handelt es sich hier also um eine Sandwich-Transformation. Induktiv setzt sich dies fort und beweist somit auch die Hinrichtung. □

## 3.2. Konstruktion von Covern und Signaturen

Für unser Kryptosystem wird es wichtig sein, dass wir Cover und Signaturen mit möglichst geringem Aufwand konstruieren können. Im Folgenden wollen wir einige Herangehensweisen genauer analysieren, wobei wir die Konstruktion von neuen Covern durch (Sandwich-) Transformation bzw. Normalisierung von alten Covern bereits zuvor kennengelernt haben und sie daher hier nicht wieder erläutern. Die Konstruktionen sind oft nicht abhängig von der endlichen Gruppe $G$, allerdings werden wir die bereits bekannten Suzuki-2-Gruppen und die symmetrischen Gruppen bevorzugen.

### 3.2.1. Transversale Signaturen

Beginnen wir mit dem für uns wichtigsten Verfahren, welches insbesondere in [2] vorgestellt wurde.

**Theorem 3.2.** *Wenn $\gamma : 1 = G_0 < G_1 < ... < G_{s-1} < G_s = G$ eine Kette von Untergruppen von $G$ ist und jedes $A_i$ für $i = 1...s$ aus einer komplette Menge von Rechtsnebenklassenrepräsentanten von $G_{i-1}$ in $G_i$ besteht, so ist $\alpha = [A_1, ..., A_s]$ eine logarithmische Signatur von $G$.*

Man nennt $\alpha$ in diesem Fall *exakt-transversal* mit Bezug zu $\gamma$.

*Beweis.* Wir beweisen die Behauptung durch vollständige Induktion über $s \in \mathbb{N}$.

*Induktions-Anfang* $(s = 1)$: Die triviale Kette $\gamma : 1 = G_0 < G$ liegt vor. Da der Index offensichtlich $[G : G_0] = \text{ord}(G)$ ist, muss jedes Element der Gruppe seine eigene Nebenklasse sein. Somit ist $A_1 = G$ und dies ist natürlich eine logarithmische Signatur.

*Induktions-Schritt* $(s - 1 \to s)$: Wir wissen durch die Induktionsvoraussetzung, dass

$$\overline{A_1} \cdot ... \cdot \overline{A_{s-1}} = \sum_{g \in G_{s-1}} g$$

Daraus können wir folgern

$$\overline{A_1} \cdot ... \cdot \overline{A_{s-1}} \cdot \overline{A_s} = \left( \sum_{g \in G_{s-1}} g \right) \cdot \overline{A_s}$$

wobei $\overline{A_s}$ insgesamt $t := \frac{ord(G_s)}{ord(G_{s-1})}$ Elemente hat. Diese haben die Gestalt

$$[g_j] = \{hg_j \ : \ h \in G_{s-1}\}, \ j = 1...t$$

Da Nebenklassen bekanntermaßen disjunkt sind, muss $hg_i \neq hg_j \, \forall i \neq j$ gelten. Hieraus folgt, dass $G_s$ eindeutig und komplett getroffen wird, denn $ord(G_{s-1}) \cdot t = ord(G_s)$. □

Wir sind nun in der Lage mit wenigen Mitteln effektiv eine logarithmische Signatur zu erzeugen. Betrachten wir jetzt ein Beispiel.

**Beispiel.** Sei die Untergruppenkette der symetrischen Gruppe $S_4$ mit 24 Elementen gegeben durch

$$\gamma : \ 1 = G_0 < G_1 < G_2 < G_3 = S_4$$

wobei

$$\begin{aligned} G_1 &= \{e, (14)(23), (13)(24), (12)(34)\} \\ G_2 &= \{e, (14)(23), (13)(24), (12)(34), (234), (132), (124), (143), (123), (243), (134), (142)\} \end{aligned}$$

so ergeben sich folgende Nebenklassen:

$$\begin{aligned} G_3/G_2 &= \{[e] = G_2, [(1234)] = G_3 \backslash G_2\} \\ G_2/G_1 &= \{[e] = G_1, [(234)] = \{(234), (132), (123), (143)\}, \\ &\quad [(123)] = \{(123), (243), (134), (142)\}\} \\ G_1/G_0 &= \{[e] = \{e\}, [(14)(23)] = \{(14)(23)\}, \\ &\quad [(13)(24)] = \{(13)(24)\}, [(12)(34)] = \{(12)(34)\}\} \end{aligned}$$

Daher ist unsere logarithmische Signatur

$$\alpha = [A_1, A_2, A_3] = [\{e, (14)(23), (13)(24), (12)(34)\}, \{e, (234), (123)\}, \{e, (1234)\}]$$

vom Typ $(4, 3, 2)$. Wir sehen insbesondere, dass der Typ durch die Größe der Untergruppen gesteuert werden kann.
Man beachte, dass im Allgemeinen viele mögliche Repräsentanten vorhanden sind, da jedes Element einer Nebenklasse diese vertreten kann. Somit können viele verschiedene Signaturen erzeugt werden, die üblicherweise nicht äquivalent sind. Der angenehme Nebeneffekt von exakt-transversalen Signaturen ist, dass man eine Liste (s.o.) dazu erstellen kann und sie zahm sind. Der in [14] vorgestellte Algorithmus „Factorization with respect to transversal logarithmic signature" gibt sogar eine genaue Vorschrift dazu an.
Wir haben bereits die Sandwich-Transformation kennengelernt, daher liegt folgende Definition nahe.

**Definition 3.6.** Eine logarithmische Signatur heißt genau dann *transversal*, wenn sie ein Sandwich von einer exakt-transversalen Signatur (zur gleichen Gruppe) ist.

Kurz gesagt: Eine transversale Signatur ist stets äquivalent zu einer exakt-transversalen Signatur. Der Vollständigkeit halber benennen wir noch das Gegenteil.

**Definition 3.7.** Eine logarithmische Signatur $\alpha$ zu einer Gruppe $G$ heißt *total nicht-transversal*, wenn jeder Block von $\alpha$ keine Nebenklasse einer nicht-trivialen Untergruppe von $G$ ist.

Ein Beispiel für eine transversale Signatur kennen wir bereits, daher wollen wir noch ein nicht-transversales Beispiel und eine leicht abgewandelte Version des „Algorithm to determine if the logarithmic signature is transversal" aus [14] betrachten.

**Algorithm 3.1** Algorithmus zum Prüfen ob eine Signatur transversal ist

**Input:** Normalisierte logarithmische Signatur $\alpha = [A_1, ..., A_s]$ zu Gruppe $G$ mit Typ $(r_1, ..., r_s)$.
**Output:** **true** wenn $\alpha$ transversal ist, ansonsten **false**
1: **if** $r_1 \cdot ... \cdot r_s \neq |G|$ **then**
2: **return false**
3: **end if**
4: $N \leftarrow 1$
5: **for** $i \leftarrow 1$ **to** $s$ **do**
6: $N \leftarrow N \cdot r_i$
7: **if** $| < A_1, ..., A_i > | \neq N$ **then**
8: **return false**
9: **end if**
10: **end for**
11: **return true**

Wir prüfen also zunächst, ob das Produkt der einzelnen Blocklängen überhaupt die Gruppengröße ergibt. Anschließend werden die Größen der aufgespannten Unterräume (per $A_1 \cup ... \cup A_i$) betrachtet. Die Signatur ist nur dann transversal, wenn die Größe der Unterräume entsprechend mit den Blockgrößen anwächst.

**Beispiel.** In der nachstenden Liste findet man eine logarithmische Signatur zur Gruppe $\mathfrak{A}_5$. Mit dem oben angegebenen Algorithmus wird im Fall $i \leftarrow 2$ ein **false** ausgegeben, was uns zeigt, dass $\beta$ tatsächlich nicht-transversal ist.

### 3.2.2. Zufällig erzeugte Cover

In der Arbeit [10] von Svaba und van Trung werden zufällige Cover eingehend untersucht. Wir wollen an dieser Stelle ihre Ideen betrachten und zusammenfassen.

| $\beta$ |
|---|
| e |
| (234) |
| (243) |
| (12)(34) |
| (12345) |
| (124) |
| (12543) |
| (132) |
| (134) |
| (13245) |
| (13)(25) |
| (142) |
| (145) |
| (14)(23) |
| (14253) |
| e |
| (24)(35) |
| (254) |
| (25)(34) |

Tabelle 3.2.: Eine nicht-transversale Signatur

Im Folgenden sei $G$ eine endliche (sehr große) Gruppe und $\alpha = [A_1, ..., A_s]$ ein Cover einer Teilmenge $S \subseteq G$ vom Typ $(r_1, ..., r_s)$ und der Länge $l = r_1 + ... + r_s$. Wir bezeichnen mit $\lambda_g$ die Anzahl aller möglichen Darstellungen von $g \in G$ unter $\alpha$. Auf Grund der oft gewaltigen Größe von $G$ kann man zwar Wahrscheinlichkeiten betrachten, aber nicht konkret prüfen, ob es sich bei einer zufälligen Menge von Teilmengen um ein Cover von $G$ handelt. Es gibt allerdings einen Zusammenhang zum bekannten Problem der Zuordnung („occupancy problem"), bei dem es darum geht die Wahrscheinlichkeit zu bestimmen, wieviele Bälle maximal pro Korb aufzufinden sind, wenn man $M$ Bälle zufällig in $m$ Körbe wirft. Da dieses Problem bekannt und gelöst ist (siehe z.B. [11]), würde ein direkter Zusammenhang auch unser Problem lösen.

Es sei $|G| = n$ und $\alpha$ eine geordnete Menge zufälliger Teilmengen $A_i \subseteq G$ für $i = 1...s$. Das heißt, es liegt eine Gleichverteilung vor und jedes Element aus $G$ wurde mit einer Wahrscheinlichkeit von $\frac{1}{n}$ ausgewählt. Wir setzen

$$N := r_1 \cdot ... \cdot r_s \; mit \; r_i = |A_i| \; für \; i = 1...s$$

und interpretieren die Elemente der Teilmengen als Menge von zufällig gewählten Elementen von $G$ (mit Vertauschung). Im Allgemeinen wird $\lambda_g > 1$ sein für die $g \in G$, die von $\alpha$ erzeugt werden können.

Angelehnt an das Zuordnungsproblem betrachten wir nun $n$ „Körbe" und verteilen die erzeugten Elemente $g_i = \alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s}$ („Bälle") auf Selbige. Im $i$-ten Korb landen

also alle Darstellungen von $g_i$, mit $i = 1...N$ (auch doppelt gezählte). Daher haben wir jetzt $N$ Bälle zufällig in $n$ Körbe verteilt, wobei jede mögliche Verteilung damit die Wahrscheinlichkeit $n^{-N}$ hat.

Wir betrachten das Gegenereignis zu unserem eigentlich Gewünschtem: Sei $E_{j_1,...,j_k}$ das Ereignis, dass die Elemente $g_{j_1}, ..., g_{j_k} \in G$ *nicht* von $\alpha$ erzeugt werden, bzw. dass die Körbe $j_1, ..., j_k$ leer sind. In diesem Fall befinden sich die von $\alpha$ erzeugten $N$ Elemente in den übrigen $n - k$ Körben und dies kann wiederum auf $(n - k)^N$ Wegen geschehen. Somit ist

$$P(E_{j_1,...,j_k}) = n^{-N}(n - k)^N = (1 - \frac{k}{n})^N$$

Mit der Bezeichnung $T_k := \binom{n}{k}(1 - \frac{k}{n})^N$ und der Methode der Inklusion-Exklusion (siehe z.b. [12]) erhalten wir die Wahrscheinlichkeit, dass wenigstens einer der Körbe leer ist:

$$\sum_{i=1}^{n}(-1)^{i-1}T_i$$

Wir betrachten weiterhin mit $p_k(N, n)$ die Wahrscheinlichkeit, dass genau $k$ Körbe leer bleiben, dann ist die Wahrscheinlichkeit, dass kein Korb leer ist:

$$p_0(N, n) = \sum_{j=0}^{n}(-1)^j \binom{n}{j}(1 - \frac{j}{n})^N$$

Beispielsweise ist also

$$\begin{aligned} p_0(24, 24) &\approx 3,6^{-10} \\ p_0(50, 24) &\approx 0,02 \\ p_0(100, 24) &\approx 0,7 \end{aligned}$$

und insgesamt muss in diesem Fall $N \geq 85$ sein, damit $p_0 > 0,5$ ist. Konkret heißt dies, dass wir zu einer Gruppe mit 24 Elementen wenigstens $N \geq 85$ wählen müssen, damit wir von einer großen Wahrscheinlichkeit sprechen können, dass jedes Element getroffen wird (Cover).

Als nächstes schauen wir uns wieder $p_k(N, n)$ an und stellen fest, da $k$ Körbe auf $\binom{n}{k}$ Wege gewählt werden und da jeder der verbleibenden $n - k$ Körbe belegt ist, muss die Anzahl der Möglichkeiten dieser Verteilung gleich $(n - k)^N p_0(N, n - k)$ sein. Wie weiter oben bereits erwähnt, müssen wir nun nur noch durch $n^N$ teilen, um die gesuchte Formel

zu erhalten:

$$\begin{aligned} p_k(N,n) &= \binom{n}{k} n^{-N}(n-k)^N p_0(N,n-k) \\ &= \binom{n}{k} n^{-N}(n-k)^N \sum_{j=0}^{n-k}(-1)^j \binom{n-k}{j}(1-\frac{j}{n-k})^N \\ &= \binom{n}{k} \sum_{j=0}^{n-k}(-1)^j \binom{n-k}{j}(\frac{n-k}{n})^N(1-\frac{j}{n-k})^N \\ &= \binom{n}{k} \sum_{j=0}^{n-k}(-1)^j \binom{n-k}{j}(1-\frac{k+j}{n})^N \end{aligned}$$

In [10]wurde weiterhin erwähnt, dass durch Grenzwertbetrachtungen ($N, n \to \infty$) gezeigt werden kann, dass für große $n$ gilt:

$$p_k(N,n) \approx \exp(-n \cdot \exp(-\frac{N}{n})) \cdot \frac{(n \cdot \exp(-\frac{N}{n}))^k}{k!}$$

und damit insbesondere

$$p_0(N,n) \approx \exp(-n \cdot \exp(-\frac{N}{n})).$$

Vergleicht man diese Werte zum Beispiel mit unserer Rechnung von oben, so ergeben sich nur geringe Abweichungen selbst bei kleinen $n$. Fassen wir die Ergebnisse nun zusammen:

**Theorem 3.3.** *Sei $G$ eine endliche Gruppe mit $|G| = n$. Für jedes $v \in ]0,1[$ gibt es ein $N_0 \in \mathbb{N}$, so dass jede Menge $\alpha = [A_1, ..., A_s]$ zu zufälligen $A_i$ mit $i = 1...s$ von $G$ mit $N := \prod_{i=1}^{s} |A_i| \geq N_0$ ein Cover für $G$ mit Wahrscheinlichkeit $p_0(N,n) \geq v$ ist. Für große $n$ gilt zudem: $p_0(N,n) \approx \exp(-n \cdot \exp(-\frac{N}{n}))$.*

Das Fazit der Arbeit von Svaba und van Trung - welches sie durch Experimente belegen konnten - ist, dass ein großes $N$ für eine hohe Wahrscheinlichkeit sorgt, dass ein Cover vorliegt, und dass ein großes $l = \sum_{i=1}^{s} r_i$ zu einer hohen Wahrscheinlichkeit für Gleichmäßigkeit führt. Wollen wir also eine logarithmische Signatur zufällig finden, so sollten wir $N$ und $l$ möglichst groß wählen.

### 3.2.3. Kanonische Signaturen

Die im Folgenden recht spezielle Art von Signaturen wurden insbesondere in [5] vorgestellt und sollen nun erläutert werden, obwohl und auch gerade weil sie nicht genügend Sicherheit für unser Kryptosystem bieten können.

**Definition 3.8.** Sei $V$ ein $m$-dimensionaler Vektorraum über dem Körper $\mathbb{F}_2$. Weiterhin sei $P = C_1 \cup ... \cup C_s$ mit $|C_i| = k_i \, \forall i = 1...s$ und $\sum_{i=1}^{s} k_i = m$ eine zufällige Partitionierung von der Menge $\{1, ..., m\}$. Eine logarithmische Signatur $\beta = [B_1, ..., B_s]$ von $V$ heißt

*kanonisch*, wenn für alle $i = 1...s$ gilt, dass Block $B_i$ aus allen möglichen $2^{k_i}$ Vektoren besteht, wobei an den durch die $C_i$ gegebenen Stellen eine 1 (bzw. 0) steht und ansonsten nur Nullen. Weiter heißt $\beta$ in *Standardform*, wenn die $C_i$ für $i = 1...s$ aufsteigende Listen sind (d.h. $C_1 = \{1, ..., k_1\}, C_2 = \{k_1 + 1, ..., k_1 + k_2\}, ..., C_s = \{k_1 + ... + k_{s-1} + 1, ..., m\}$) und die Blöcke sind sortiert (d.h. $\beta_{ij_1} < \beta_{ij_2} \, \forall i, j_1 < j_2$).

Eine Partitionierung ist eine vollständige Aufteilung einer Menge $M$ in Teilmengen $C_i$, so dass $M = C_1 \cup ... \cup C_n$ und $C_i \cap C_j = \emptyset \, \forall i \neq j$. Zur Verdeutlichung betrachten wir nun ein einfaches Beispiel.

**Beispiel.** Sei $\dim(V) = 3$ und $P = \{1, 3\} \cup \{2\}$, das heißt also $|C_1| = 2$, $|C_2| = 1$. Offensichtlich ist $P$ eine Partitionierung der Menge $\{1, 2, 3\}$. Nach unserer Definition erhalten wir also zu $C_1$ die Vektoren

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

und zu $C_2$ lediglich

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

was uns also die kanonisch logarithmische Signatur

$$\beta = [\{\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\}, \{\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}\}]$$

bringt.
Offensichtlich erhält man aus jeder der Partitionen $C_i$ exakt $2^{|C_i|}$ Vektoren, was uns - zusammen mit den Erkenntnissen aus dem Beispiel - zu folgendem Algorithmus bringt.

**Algorithmus zur Erzeugung von kanonisch logarithmischen Signaturen.**
*Schritt 1:* Erzeuge eine zufällige Partionierung $P$ von der Menge $\{1, ..., m\}$, wobei hier $P = C_1 \cup ... \cup C_s$ und $|C_i| = k_i \, \forall i = 1...s$.

*Schritt 2:* Erzeuge für alle $i = 1...s$ die Blöcke $B_i$ von $\beta = [B_1, ..., B_s]$ wie folgt. Nehme alle möglichen $2^{k_i}$ Vektoren $u_j$ der Dimension $k_i$ (über $\mathbb{F}_2$) und identifiziere die Elemente („Zahlen") der Reihe nach von $C_i$ mit den Einträgen von $u_j$. Nun erschaffe für jedes $u_j$ die $m$-dimensionalen Vektoren, in dem an entsprechender Stelle eine 1 (bzw. 0) steht. Fülle den Rest mit Nullen auf.

Man kann nun zu unserem Beispiel auch eine Tabelle anlegen, die die Erzeugung verdeutlicht und durch den Algorithmus verallgemeinert werden kann. Man beachte, dass lediglich die Diagonale wichtig ist und der Rest mit Nullen aufgefüllt wurde.

| $\beta$ | |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 0 |
| 0 0 | 0 |
| 0 0 | 1 |

Tabelle 3.3.: Beispiel für eine kan. log. Sig.

Im Abschnitt über die Faktorisierbarkeit haben wir uns bereits mit zahmen und wilden Covern beschäftigt, daher ist es angebracht nun die kanonisch logarithmischen Signaturen auf ihre Faktorisierbarkeit hin zu prüfen. Dies kann kurz und knapp zusammengefasst werden:

**Proposition 3.3.** *Kanonisch logarithmische Signaturen sind zahm.*

*Beweis.* Laut unserer Definition agieren die Elemente von Block $B_i$ (stets $i = 1...s$) nur auf den von $C_i$ gegebenen Stellen und jedes $B_i$ enthält eine komplette Menge von $2^{k_i}$ Vektoren der Dimension $k_i$ auf den von $C_i$ angezeigten Positionen. Um nun ein $y \in V$ in die Form $y = \beta_{1j_1} \cdot ... \cdot \beta_{sj_s}$ zu faktorisieren, splitten wir die Bits von $y$ in Vektoren $\beta_{ij_i}$ bezüglich $P$ folgendermaßen auf:
Wir kopieren die Bits von $y$ auf die von $C_i$ vorbestimmten Positionen um $\beta_{ij_i}$ zu bestimmen und setzen den Rest von $\beta_{ij_i}$ gleich Null. Die Position von solch einem erzeugten Vektor $\beta_{ij_i}$ bzgl. $B_i$ definiert den Index $j_i$. Bei logarithmischen Signaturen in Standardform ist dies offensichtlich besonders einfach. □

Anschaulich ist dies sofort klar, denn die Form der Vektoren ist angenehm einfach. So würde in unserem Beispiel der Vektor $(1, 0, 1)$ einfach die Faktorisierung $(1, 0, 0)+(0, 0, 1)$ haben und wir haben hierbei einfach nur die Diagonale der Tabelle geprüft.

In der Definition der kan. log. Sig. wurde auch die Standardform angesprochen. Natürlich können wir jede kan. log. Sig in eben solche überführen, indem wir die Elemente zwischen den $C_i$ permutieren und entsprechend die Vektoren der Blöcke sortieren. Aus der Linearen Algebra wissen wir, dass solche Transformationen bestimmte Permutationsmatrizen induzieren. Diese Überlegung führt uns zu folgender Proposition:

**Proposition 3.4.** *Eine kanonisch logarithische Signatur $\beta := (\beta_{ij})$ kann man als eine lineare Transformation der kanonischen standardisierten Signatur $\epsilon := (\epsilon_{ij})$ vom selben Typ zum gleichen $V$ schreiben. Anders gesagt: Es gibt eine Matrix $M \in GL(m, 2)$, so dass $\beta = (\beta_{ij}) = (M \cdot \epsilon_{ij})$.*

**Beispiel.** Wir benutzen unsere kan. log. Sig. $\beta$ von oben und suchen die Matrix $M$ zur Standard-kan. log. Sig $\epsilon$ mit

$$\epsilon = [\{\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}\}, \{\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\}]$$

wobei wir die Einträge $a, b, c, d, e, f, g, h, i \in \mathbb{F}_2$ derart suchen, dass

$$\begin{pmatrix}a & b & c\\ d & e & f\\ g & h & i\end{pmatrix}\begin{pmatrix}\epsilon_{ij}\end{pmatrix} = \begin{pmatrix}\beta_{ij}\end{pmatrix}$$

und erhalten nach kurzer Rechnung

$$M = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix}.$$

Bezüglich der Faktorisierbarkeit brauchen wir uns keine Sorgen zu machen, denn die Transformationen kann man, da M regulär, wieder rückgängig machen. Also bleibt eine zahme Signatur auch nach der Transformation noch zahm. Dies ermöglicht uns zahme logarithmische Signaturen zufällig zu erzeugen, da wir nach der Generierung einer kanonisch logarithmischen Signatur einfach eine zufällige Matrix $M \in GL(m, 2)$ darauf anwenden können. Gerade diese Zufälligkeit der Matrix ist interessant für kryptographische Zwecke.

Wir wenden uns einer Eigenschaft kanonischer Signaturen zu, die auf den ersten Blick lediglich weitere solcher Signaturen erzeugt, jedoch können wir das Prinzip auf logarithmische Signaturen erweitern und somit verallgemeinern. Dies wird dann der letzte Satz in diesem Abschnitt erledigen.

**Proposition 3.5.** *Sei $\beta = [B_1, ..., B_s]$ eine kanonische logarithmische Signatur für $V$, so gilt, dass $\tilde{\beta} := (\tilde{\beta_{ij}}) := (\beta_{ij} d_i)$ mit $d_i \in B_i$ für $i = 1...s$ ebenfalls eine kanonische logarithmische Signatur für $V$ ist.*

*Beweis.* Wir wissen bereits, dass die Blöcke von $\beta$ auf disjunkten Mengen von Bits von $C_i$ agieren und jeder Block $B_i$ enthält die komplette Menge an $2^{k_i}$ Vektoren mit Bits an den von $C_i$ vorgegebenen Positionen. Wenn wir also alle Elemente von $B_i$ mit einem festen $d_i \in B_i$ verknüpfen, so vertauschen wir nur Nullen und Einsen an den Stellen, die durch $C_i$ gegeben waren und bleiben daher in $B_i$ (wenn auch anders geordnet) und $\beta$ bleibt kanonisch für $V$. □

Auch wenn der Einfachheit halber stets die Multiplikation als Operation angezeigt wird, so ist bei kanonischen Signaturen natürlich stets die Addition der Vektoren gemeint. Betrachten wir ein kurzes Beispiel.

**Beispiel.** Wir betrachten wieder

$$\beta = [\{\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}\}, \{\begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}\}]$$

wählen

$$d_1 = \begin{pmatrix}0\\1\\0\end{pmatrix}, d_2 = \begin{pmatrix}0\\0\\1\end{pmatrix}$$

und erhalten somit

$$\tilde{\beta} = [\{\begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}\}, \{\begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}\}]$$

was laut unserer Proposition wieder eine kanonische Signatur ist. Tatsächlich bleiben die Blöcke (in neuer Ordnung) erhalten. Die zugrunde liegenden Permutationen bilden also stets eine Untergruppe.

**Theorem 3.4.** *Sei $G$ eine endliche Gruppe und $\beta = [B_1, ..., B_s]$ eine zahme logarithmische Signatur für $G$. Weiterhin sei $\tilde{\beta} := (\tilde{\beta_{ij}}) := (\beta_{ij} d_i)$ mit $d_i \in G$.*
*Es gilt: $\tilde{\beta}$ ist zahm, wenn ein der folgenden Bedingungen erfüllt ist:*
*1. $d_i \in Z(G)$ für $i = 1...s$*
*2. $d_i \in G_{i-1}$ für $i = 1...s$, wenn $\beta$ exakt-transversal von $G$ zu einer Kette von Untergruppen $\gamma$ und $B_i$ eine komplette Menge von Nebenklassenrepräsentanten von $G_{i-1}$ in $G_i$ ist.*

*Beweis.* Wir betrachten die beiden Fälle getrennt.
*Fall 1*: Da $d_i \in Z(G)$ kommutieren $b_{ij}$ und $d_i$, daher können wir eine zu $\tilde{\beta}$ äquivalente logarithmische Signatur $\hat{\beta} := (\hat{\beta_{ij}})$ angeben, wobei

$$\hat{\beta_{ij}} := \begin{cases} \beta_{ij} & \forall i = 1...s-1 \\ \beta_{sj} d & \text{mit } d = d_1 \cdot ... \cdot d_s \end{cases}$$

Denn es gilt:

$$\begin{aligned} \tilde{\beta}(x) &= \tilde{\beta}_{1j_1} \cdot ... \cdot \tilde{\beta}_{sj_s} \\ &= \beta_{1j_1} d_1 \cdot ... \cdot \beta_{sj_s} d_s \\ &\overset{d_i \in Z(G)}{=} \beta_{1j_1} \cdot ... \cdot \beta_{sj_s} \cdot d_1 \cdot ... \cdot d_s \\ &= \hat{\beta}_{1j_1} \cdot ... \cdot \hat{\beta}_{s-1 j_{s-1}} \cdot d \\ &= \hat{\beta}(x) \end{aligned}$$

Sind wir also in der Lage in $\beta$ zu faktorisieren (da zahm), so ist es uns möglich in den ersten $s-1$ Blöcken von $\tilde{\beta}$ zu faktosieren. Das letzte gesuchte $j_s$ finden wir einfach durch

vollstände Suche. Somit ist $\tilde{\beta}$ zahm, denn eine äquivalente Signatur, nämlich $\hat{\beta}$, stellte sich als zahm heraus.
*Fall 2:* Wir setzen nun voraus, dass $g = \beta_{1j_1} \cdot ... \cdot \beta_{sj_s}$ bezüglich $\beta$ faktorisieren können und versuchen $\tilde{g} = \tilde{\beta}_{1j_1} \cdot ... \cdot \tilde{\beta}_{sj_s}$ bezüglich $\tilde{\beta}$ zu faktorisieren. Mit $\tilde{B}_i = B_i d_i$ und $d_i \in G_{i-1}$ folgt, dass $\beta_{ij}$ und $\tilde{\beta}_{ij}$ in derselben Nebenklasse von $G_{i-1}$ in $G_i$ sind. Wir versuchen also nun zunächst $\tilde{g} = \tilde{\beta}_{1j_1} \cdot ... \cdot \tilde{\beta}_{sj_s}$ bezüglich des Blocks $\tilde{B}_s$ zu faktorisieren.
Da $\tilde{\beta}_{1j_1} \cdot ... \cdot \tilde{\beta}_{s-1j_{s-1}} \in G_{s-1}$ folgt (analog wie zuvor), dass $\tilde{g}$ und $\tilde{\beta}_{sj_s}$ in der gleichen Nebenklasse von $G_{s-1}$ in $G_s$ sind. Das bedeutet, wir können die Nebenklassen von $\tilde{g}$ eindeutig als $\tilde{\beta}_{sj_s} G_{s-1}$ identifizieren. Somit kennen wir den ersten Faktor von $\tilde{g}$, nämlich $\tilde{\beta}_{sj_s}$. Wir fahren dann analog mit der Faktorisierung von $\tilde{g}(\tilde{\beta}_{sj_s})^{-1}$ bezüglich $\tilde{B}_{s-1}$ fort und identifizieren $\tilde{\beta}_{s-1j_{s-1}}$ und so weiter. □

# 4. Das Kryptosystem $MST_3$

## 4.1. Vorgänger

Da es in dieser Arbeit zentral um das Kryptosystem $MST_3$ geht, wollen wir nicht viel Zeit mit den Vorgängern verlieren. Wir beschränken uns auf die Betrachtung der eigentlichen Verfahren, ohne tiefer auf Angriffe und dergleichen einzugehen. Hierfür verweisen wir auf [2]. Beide Kryptosysteme, also $MST_1$ und $MST_2$, beruhen ebenfalls auf der Theorie der Cover und logarithmischen Signaturen zu endlichen Gruppen.

### 4.1.1. $MST_1$

**Setup:** Alice wählt ein Paar von logarithmischen Signaturen $(\alpha, \beta)$ zu der gleichen endlichen Gruppe $G$, wobei $\alpha$ total-nicht-transversal und $\beta$ transversal sind. Sie kennt weiterhin die Faktorisation

$$\sigma = \breve{\alpha}\breve{\beta}^{-1} = \breve{\Theta}_1 \cdot ... \cdot \breve{\Theta}_k$$

wobei $\Theta_i$ , $i = 1...k$ sind exakt-transversal und $k \geq 2$.
Alice veröffentlicht $((\alpha, \beta), G)$ als ihren öffentlichen Schlüssel, aber behält $\Theta_1, ..., \Theta_k$ geheim.
Da $\Theta_i$ transversal für alle $i = 1...k$ kann sie $\breve{\Theta}^{-1}$ und damit auch $\sigma^{-1}$ effektiv berechnen.
Der Nachrichtenraum (Klartext und Chiffretext) ist $\mathbb{Z}_{|G|}$.

**Chiffrierung:** Bob verschlüsselt $m \in \mathbb{Z}_{|G|}$ zu

$$c = \sigma(m) = (\breve{\alpha}\breve{\beta}^{-1})(m)$$

und sendet $c$ an Alice.

**Dechiffrierung:** Alice entschlüsselt

$$\breve{\Theta}_k^{-1}(\breve{\Theta}_{k-1}^{-1}(...(\breve{\Theta}_1^{-1}(c))...)) = m$$

und kennt damit den gesuchten Klartext.

### 4.1.2. $MST_2$

**Setup:** Alice wählt zwei große Gruppen $G$ und $H$, sowie weiterhin einen Epimorphismus $f : G \to H$ und erzeugt ein zufälliges $[s, r]$-Sieb $\alpha = (\alpha_{ij})$ für $G$.
Alice berechnet

$$\beta = f(\alpha) = (\beta_{ij}) = (f(\alpha_{ij}))$$

und veröffentlicht $(\alpha, \beta)$, aber behält die Abbildung $f$ geheim.

**Chiffrierung:** Wenn Bob eine Nachricht $x \in H$ senden will, so

- wählt er eine zufällige Zahl $R \in \mathbb{Z}_{r^s}$
- berechnet $y_1 = \breve{\alpha}(R)$, $y_2 = \breve{\beta}(R)$, $y_3 = xy_2$

und sendet $y = (y_1, y_3)$ an Alice.

**Dechiffrierung:** Alice berechnet

$$y_2 = \breve{\beta}(R) = f(\breve{\alpha}(R)) = f(y_1)$$

und da ihr $y_3 = xy_2$ bekannt ist, kennt sie damit den gesuchten Klartext $x = y_3 y_2^{-1}$.

## 4.2. $MST_3$

Im zweiten und dritten Kapitel haben wir die notwendigen Definitionen und Eigenschaften kennen gelernt, die wir im Folgenden nun nutzen wollen, um das neue Kryptosystem $MST_3$, bekannt aus [1], zu beschreiben. Im weiteren Verlauf wollen wir ein eigenes Beispiel betrachten, dass zwar von Hand nachgerechnet werden kann, aber zugleich auch die Arbeitsschritte des Systems deutlich macht. Die Autoren von $MST_3$ haben zudem in ihren Arbeiten (insb. [1, 5]) einige Angriffe vorgestellt und deren Probleme erläutert. Hiermit wollen wir dann das Kapitel abschließen.

### 4.2.1. Das originale System

Sei $G$ eine endliche nicht-abelsche Gruppe mit nicht-trivialem Zentrum $Z(G)$, die nicht über $Z(G)$ zerfällt. Das heißt, es gibt keine Untergruppe $H < G$ mit $H \cap Z(G) = \{e\}$, so dass $G = H \cdot Z(G) = \{hz : h \in H, z \in Z(G)\}$ - oder kurz gesagt: $G$ soll nicht über $Z(G)$ faktorisierbar sein. Natürlich soll weiterhin $Z(G)$ genügend groß sein, damit Computer-Angriffe undurchführbar sind.
Die wichtige Hypothese, welche besagt, dass das Faktorisierungproblem über einem zufälligem Cover für eine große Teilmenge $S \subset G$ im Allgemeinen unlösbar ist, sollen wir bei folgenden Erläuterungen im Hinterkopf behalten:

**Setup:** Alice wählt eine große Gruppe $G$ wie wir sie gerade beschrieben haben und

1. nimmt eine zahme logarithmische Signatur $\beta = [B_1, ..., B_s] =: (\beta_{ij})$ vom Typ $(r_1, ..., r_s)$ zu $Z(G)$
2. wählt ein zufälliges Cover $\alpha = [A_1, ..., A_s] =: (\alpha_{ij})$ vom selben Typ wie $\beta$ zu einer Teilmenge $J \subset G$, so dass $A_1, ..., A_s \subseteq G \backslash Z(G)$
3. wählt $t_0, ..., t_s \in G \backslash Z(G)$

4. berechnet $\tau := [T_1, ..., T_s]$, wobei $T_i := t_{i-1}^{-1} A_i t_i \, \forall i = 1, ..., s$

5. und bestimmt $\gamma := (\gamma_{ij}) = (\beta_{ij}\tau_{ij})$.

Alice veröffentlicht den Schlüssel $(\alpha, \gamma)$ und behält $(\beta, (t_0, ..., t_s))$ geheim.

**Chiffrierung:** Wenn Bob eine Nachricht $x \in \mathbb{Z}_{|Z(G)|}$ senden will, dann berechnet er

$$y_1 := \breve{\alpha}(x),\ y_2 := \breve{\gamma}(x)$$

und sendet $y = (y_1, y_2)$ an Alice.

**Dechiffrierung:** Alice kennt also $y_1, y_2, \alpha, \beta, \gamma$. Da

$$\begin{aligned} y_2 &= \breve{\gamma}(x) \\ &= \beta_{1j_1}\tau_{1j_1} \cdot ... \cdot \beta_{sj_s}\tau_{sj_s} \\ &= \beta_{1j_1} t_0^{-1} \alpha_{1j_1} t_1 \cdot ... \cdot \beta_{sj_s} t_{s-1}^{-1} \alpha_{sj_s} t_s \\ &\overset{\beta_{ij} \in Z(G)}{=} \beta_{1j_1} \cdot ... \cdot \beta_{sj_s} t_0^{-1} \alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s} t_s \\ &= \breve{\beta}(x) t_0^{-1} \breve{\alpha}(x) t_s \end{aligned}$$

gilt können wir folgende Gleichung aufstellen:

$$\breve{\beta}(x) = y_2 t_s^{-1} y_1^{-1} t_0.$$

Wir haben vorausgesetzt, dass $\beta$ zahm sein soll, daher lässt sich $\breve{\beta}^{-1}$ effizient berechnen und Alice erhält somit die Nachricht $x$.

### 4.2.2. Ein Beispiel

Im einem vorherigen Kapitel haben wir bereits die Suzuki-2-Gruppen betrachtet und wollen nun mit deren Hilfe ein konkretes Anwendungsbeispiel für $MST_3$ aufzeigen. Natürlich wird $Z(G)$ hier viel zu klein sein, als dass man von Sicherheit sprechen darf, aber andererseits wäre ein Beispiel mit sinnvoller Größe sehr unhandlich und vorallem wenig anschaulich.
Sei also $G = \{S(a, b) : a, b \in \mathbb{F}_{2^3}\}$, wobei

$$\mathbb{F}_{2^3} = \mathbb{F}_8 = \mathbb{F}_2[x] \diagup (x^3 + x + 1) = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$$

Wir wählen als zugehörigen Automorphismus $\Theta$ das Quadrieren in $\mathbb{F}_{2^3}$. Es ist somit klar, dass $|G| = q^2 = 64$ und ein Element aus $G$ hat die Gestalt

$$\begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & a^2 & 1 \end{pmatrix} \text{ mit } a, b \in \mathbb{F}_{2^3}.$$

Wie wir wissen gilt

$$Z(G) = \Phi(G) = G' = \Omega_1(G) = \{S(0,b) \,:\, b \in \mathbb{F}_8\} = \{ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ b & 0 & 1 \end{pmatrix} \,:\, b \in \mathbb{F}_8\}$$

und daher ist $|Z(G)| = q = 8$. Wir erinnern uns, dass $S(a,b)S(c,d) = S(a+c, b+d+a^2c)$ und $S(a,b)^{-1} = S(a, a^3+b)$. Wie üblich bezeichnen wir das neutrale Element $S(0,0)$ auch abkürzend einfach mit $e$.
Beginnen wir mit dem Setup.

**Bsp.-Setup:** Alice benötigt eine zahme logarithmische Signatur $\beta$ zu $Z := Z(G)$ und wählt als einfachste Variante eine transversal logarithmische Signatur zu folgender Kette von Untergruppen:

$$Z_0 = \{e\} < Z_1 = \{z \in Z \,:\, b \in \{0,1\}\} < Z_2 = \{z \in Z \,:\, b \in \{0,1,x,x+1\}\} < Z_3 = Z.$$

Wir bestimmen entsprechend die Nebenklassen:

$$\begin{aligned} Z_3 / Z_2 &= \{[e], [S(0,x^2)]\} \\ Z_2 / Z_1 &= \{[e], [S(0,x)]\} \\ Z_1 / Z_0 &= \{[e], [S(0,1)]\} \end{aligned}$$

und erhalten eine zahme logarithmische Signatur vom Typ $(2,2,2)$ zu $Z(G)$:

$$\beta = [\{e, S(0,x^2)\}, \{e, S(0,x)\}, \{e, S(0,1)\}]$$

Alice benötigt jedoch noch ein Cover $\alpha = [A_1, A_2, A_3]$ vom gleichen Typ zu einer Teilmenge $J \subset G$ mit $A_1, A_2, A_3 \subseteq G \backslash Z(G)$. Wir gehen rückwärts vor und stellen einfach ein $\alpha$ auf und prüfen dann, zu welcher Teilmenge es ein Cover ist. Sei also

$$\begin{aligned} A_1 &:= \{S(1,0), S(x,1)\} \\ A_2 &:= \{S(x,0), S(x,x)\} \\ A_3 &:= \{S(1,x), S(1,1)\} \end{aligned}$$

und damit ergibt sich

$$\begin{aligned} \bar{A_1} \cdot \bar{A_2} \cdot \bar{A_3} &= (S(1,0) + S(x,1))(S(x,0) + S(x,x))(S(1,x) + S(1,1)) \\ &= S(x, x^2+1) + S(x, x^2+x) + S(x, x^2+x+1) + S(x, x^2) \\ &\quad + S(1,0) + S(1, x+1) + S(1,x) + S(1,1) \end{aligned}$$

Also ist $\alpha$ ein Cover (sogar eine logarithmische Signatur) zu

$$J = \{S(1,0), S(1,1), S(1,x), S(1,x+1), S(x,x^2), S(x,x^2+1), S(x,x^2+x), S(x,x^2+x+1)\}$$

Jetzt wählt Alice

$$t_0 = S(1,1),\ t_1 = S(1,0),\ t_2 = S(x,1),\ t_3 = S(1,x)$$

und wir bemerken vorausschauend, dass

$$t_0^{-1} = S(1,0),\ t_1^{-1} = S(1,1),\ t_2^{-1} = S(x,x),\ t_3^{-1} = S(1,x+1)$$

Fahren wir fort und berechnen $\tau = [T_1, T_2, T_3]$ wie im Setup oben beschrieben:

$$\begin{aligned}
\tau_{11} &= S(1,0)S(1,0)S(1,0) = S(1,1)\\
\tau_{12} &= S(1,0)S(x,1)S(1,0) = S(x,x^2+x)\\
\tau_{21} &= S(1,1)S(x,0)S(x,1) = S(1,x+1)\\
\tau_{22} &= S(1,1)S(x,x)S(x,1) = S(1,1)\\
\tau_{31} &= S(x,x)S(1,x)S(1,x) = S(x,x+1)\\
\tau_{32} &= S(x,x)S(1,1)S(1,x) = S(x,0)
\end{aligned}$$

Also ist $\tau = [\{S(1,1), S(x,x^2+x)\}, \{S(1,x+1), S(1,1)\}, \{S(x,x+1), S(x,0)\}]$.
Jetzt fehlt uns nur noch $\gamma = (\beta_{ij}\tau_{ij})$, $i,j \in \{1,2,3\}$, welches sich aus den bisher errechneten Werten sofort bestimmen lässt:

$$\gamma = [\{S(1,1), S(x,x)\}, \{S(1,x+1), S(1,x+1)\}, \{S(x,x+1), S(x,1)\}]$$

Alice veröffentlicht nun $(\alpha, \gamma)$ und behält wie vorgegeben $(\beta, (t_0, t_1, t_2, t_3))$ geheim.

**Bsp.-Chiffrierung:** Jetzt will Bob die Nachricht $x = 5 \in \mathbb{Z}_8$ senden und berechnet dazu $y_1 = \breve{\alpha}(5)$ und $y_2 = \breve{\gamma}(5)$ wie folgt:
Wir beachten, dass $\breve{\alpha} : \mathbb{Z}_8 \to Z(G)$ mit $\breve{\alpha}(x) = \alpha_{1j_1} \cdot \alpha_{2j_2} \cdot \alpha_{3j_3}$ und $\iota^{-1}(x) = (j_1, j_2, j_3)$ und erstellen die Tabelle, welche wir bereits in ähnlicher Art im letzten Kapitel angetroffen haben:

| $\delta_{\mathbb{Z}_8}$ | $\alpha$ | $\gamma$ |
|---|---|---|
| 0 | $S(1,0)$ | $S(1,1)$ |
| 1 | $S(x,1)$ | $S(x,x)$ |
| 0 | $S(x,0)$ | $S(1,x+1)$ |
| 2 | $S(x,x)$ | $S(1,x+1)$ |
| 0 | $S(1,x)$ | $S(x,x+1)$ |
| 4 | $S(1,1)$ | $S(x,1)$ |

Tabelle 4.1.: Hilfstabelle

Wir sehen, dass $5 = 1 + 0 + 4$ und lesen ab:

$$\breve{\alpha}(5) = \alpha_{12} \cdot \alpha_{21} \cdot \alpha_{32} = S(x,1)S(x,0)S(1,1) = S(1,x+1)$$

Analog ergibt sich $\breve{\gamma}(5) = \gamma_{12} \cdot \gamma_{21} \cdot \gamma_{32} = S(1,x^2+1)$.
Bob sendet nun $y = (S(1,x+1), S(1,x^2+1))$ an Alice.

**Bsp.-Dechiffrierung:** Alice rechnet:

$$\begin{aligned}\breve{\beta}(x) &\stackrel{!}{=} S(1,x^2+1)S(1,x+1)S(1,x+1)^{-1}S(1,1)\\ &= S(0,x^2+1)\end{aligned}$$

und da sie die zahme logarithmische Signatur $\beta$ kennt, kann sie $\breve{\beta}^{-1}(S(0,x^2+1))$ bestimmen:

$$S(0,x^2+1) = S(0,x^2)\cdot e\cdot S(0,1) = \beta_{12}\cdot\beta_{21}\cdot\beta_{32}$$

woraus Alice folgern kann, dass $x = 1+0+4 = 5$, was der von Bob gesendeten Nachricht entspricht.

### 4.2.3. Die verbesserte Version

Im Laufe der Zeit konnten ein paar Schwachstellen am ersten (originalen) Kryptosystem entdeckt werden, so dass die Autoren sich entschlossen in [16] eine verbesserte Version vorzustellen. Wir wollen an dieser Stelle kurz das neue System betrachten. Im fünften Kapitel werden wir darauf wieder eingehen und anmerken, wenn eine alte Schwachstelle durch die Veränderungen behoben wurde.
Im Folgenden sei $G$ eine Suzuki-2-Gruppe über $\mathbb{F}_{2^m}$. Wir betrachten das Zentrum $Z(G)$ als einen Vektorraum der Dimension $m$ über $\mathbb{F}_2$. Somit ist $\mathrm{Aut}(Z(G)) = \mathrm{GL}(m,\mathbb{F}_2)$. Weiterhin benutzen wir die Bezeichnungen $g_{.a} := x$ und $g_{.b} := y$ falls $g = S(x,y) \in G$. Aus der Algebra kennen wir zudem folgende Eigenschaft:

**Proposition 4.1.** *Ist $f$ ein Homomorphismus von $G$ nach $Z(G)$ und $N := ker(f)$, so ist $N$ ein Normalteiler von $G$ und $G/N$ ist isomorph zu $f(G) \subseteq Z(G)$. Daher ist $G/N$ abelsch. Weiterhin ist hier $G' = Z(G)$, also ist $N \geq Z(G)$ und somit ist $f(z) = 1$ für alle $z \in Z(G)$.*

Für die Realisierung wird folgende Klasse von Homomorphismen genutzt: Sei $g = S(g_{.a}, g_{.b}) \in G$ und $\sigma \in \mathrm{Aut}(Z(G)) = \mathrm{GL}(m,\mathbb{F}_2)$, so definieren wir

$$f : G \to Z(G) \text{ mit } f(g) := S(0, g_{.a}^{\sigma})$$

dann ist $f$ laut den Rechenregeln in Suzuki-2-Gruppen ein Homomorphismus von $G$ nach $Z(G)$.

**Setup:** Alice wählt eine große Gruppe $G$ wie wir sie gerade beschrieben haben und erzeugt

1. eine zahme logarithmische Signatur $\beta = [B_1,...,B_s] =: (\beta_{ij})$ vom Typ $(r_1,...,r_s)$ zu $Z(G)$
2. ein zufälliges Cover $\alpha = [A_1,...,A_s] =: (\alpha_{ij})$ vom selben Typ wie $\beta$ zu einer Teilmenge $J \subset G$, so dass $A_1,...,A_s \subseteq G\backslash Z(G)$, wobei in den einzelnen Blöcken jeweils gelten soll, dass paarweise verschiedene Elemente in verschiedenen Nebenklassen unter $Z(G)$ sein sollen (also $\alpha_{(ij_1).a} \neq \alpha_{(ij_2).a}$ für $j_1 \neq j_2$). Zudem wird $\sum_{j=1}^{r_i} \alpha_{(ij).a} = 0$ gefordert.

3. Alice wählt $t_0, ..., t_s \in G \backslash Z(G)$
4. und einen Homomorphismus $f : G \to Z(G)$
5. und berechnet $\gamma := (\gamma_{ij}) = (t_{i-1}^{-1} \cdot \alpha_{ij} \cdot f(\alpha_{ij}) \cdot \beta_{ij} \cdot t_i)$.

Alive veröffentlicht den Schlüssel $(\alpha, \gamma)$ und behält $(\beta, (t_0, ..., t_s), f)$ geheim

**Chiffrierung:** Wenn Bob eine Nachricht $x \in Z(G)$ senden will, so wählt er zufällig $R \in \mathbb{Z}_{|Z(G)|}$ und berechnet

$$\begin{aligned} y_1 &= \breve{\alpha}(R) \cdot x \\ y_2 &= \breve{\gamma}(R) \cdot x = t_0^{-1} \cdot \breve{\alpha}(R) \cdot f(\breve{\alpha}(R)) \cdot \breve{\beta}(R) \cdot t_s \cdot x \end{aligned}$$

und sendet $(y_1, y_2)$ an Alice.

**Dechiffrierung:** Alice hat von Bob $(y_1, y_2)$ erhalten. Sie nutzt die Tatsache, dass $x \in Z(G)$ und daher $f(y_1) = f(\breve{\alpha}(R))$ und berechnet

$$\begin{aligned} \breve{\beta}(R) &= f(\breve{\alpha}(R))^{-1} \cdot y_1^{-1} \cdot t_0 \cdot y_2 \cdot t_s^{-1} \\ &= f(y_1)^{-1} \cdot y_1^{-1} \cdot t_0 \cdot y_2 \cdot t_s^{-1} \end{aligned}$$

Da $\beta$ zahm ist, kann sie effizient $R$ aus $\breve{\beta}(R)$ bestimmen. Mittels der Berechnung von $\breve{\alpha}(R)$ kann sie schließlich die gesuchte Nachricht $x$ aus $y_1$ erhalten.

### 4.2.4. Angriffe

Im weiteren Verlauf dieses Kapitels geht es stets um die erste Version von $MST_3$. Wir wollen uns nun verschiedenen Angriffen auf unser Kryptosystem widmen. Diese sind unter anderem bekannt aus [1, 5].

#### 4.2.4.1. Ciphertext-Only-Attacke

Wir entsinnen uns, dass der Angreifer bei einer Ciphertext-Only-Attacke nur die öffentlichen Informationen besitzt, das heißt in unserem Fall, dass ihm $(\alpha, \gamma)$ bekannt sind, aber $(\beta, t_0...t_s)$ gesucht sind. Nach unseren Erkenntnissen über Signaturen, würde es dem Angreifer genügen eine zu $\beta$ äquivalente Signatur zu finden, denn diese würde offensichtlich das System brechen. Dies ist bekanntermaßen gleichbedeutend damit, dass ein $\tilde{\beta}$ gefunden wird, welches eine Sandwichtransformation von $\beta$ ist. Insbesondere wissen wir nun (Prop. 3.2), dass wir ohne Beschränkung der Allgemeinheit davon ausgehen dürfen, dass $\beta$ normalisiert ist, also dass jeder (außer der letzte) Block mit dem neutralen Element der zugrunde liegenden Suzuki-2-Gruppe $G$ beginnt: $\beta_{i1} = e \, \forall i = 1...s$.
Zunächst sehen wir aus dem Setup heraus, dass für alle $i = 1...s$, $1 \leq j \leq r_i$ gilt:

$$\gamma_{ij} = \beta_{ij} t_{i-1}^{-1} \alpha_{ij} t_i \tag{4.2.1}$$

wobei uns $\gamma_{ij}$ und $\alpha_{ij}$ ja bekannt sind. Da $\beta$ normalisiert ist, folgt für $j = 1$

$$\begin{aligned} \gamma_{11} &= t_0^{-1}\alpha_{11}t_1 \\ \gamma_{21} &= t_1^{-1}\alpha_{21}t_2 \\ &... \\ \gamma_{(s-1)1} &= t_{s-2}^{-1}\alpha_{(s-1)1}t_{s-1} \end{aligned}$$

was uns folgern lässt, dass sich die $(t_0, ..., t_{s-1})$ berechnen lassen, wenn man eines der $t_i$ einmal fest gewählt hat. Betrachten wir wieder die Formel 4.2.1 und sehen

$$\beta_{ij} = \gamma_{ij}t_i^{-1}\alpha_{ij}^{-1}t_{i-1}$$

daher können wir - stets abhängig vom geratenen $t_0$ - ebenfalls die Blöcke $B_1, ..., B_{s-1}$ berechnen. Um sowohl den letzten Block, als auch $t_s$ zu erhalten, sind wir jedoch gezwungen $\beta_{s1}$ zu wählen.
Betrachten wir unsere Erfolgswahrscheinlichkeit: Laut Setup muss $t_0 \in G\backslash Z(G)$ sein und da $G$ eine Suzuki-2-Gruppe ist, hat der Angreifer $q^2 - q$ Möglichkeiten $t_0$ zu raten. Da $\beta$ eine Signatur für das Zentrum ist, gibt es zudem jedesmal noch $q$ Möglichkeiten $\beta_{s1}$ zu wählen. Dies ergibt zunächst eine Gesamtzahl von $(q^2 - q)q = q^2(q - 1)$ Möglichkeiten, welche wir jedoch noch weiter reduzieren können. Dazu muss man einsehen, dass es genügt, wenn man ein $t$ aus der Nebenklasse vom richtigen $t_0$ erraten hat, d.h. $t = t_0Z(G)$. Dies ergibt sich mit $z \in Z(G)$ ebenfalls aus 4.2.1:

$$\gamma_{11} = (t_0z)^{-1}\alpha_{11}t_1 = z^{-1}t_0^{-1}\alpha_{11}t_1 \stackrel{!}{=} t_0^{-1}\alpha_{11}t_1$$

was äquivalent dazu ist, dass $t_1 = t_1z$. Analog folgt also, dass $(t_0, ..., t_s)$ durch $(t_0z, ..., t_sz)$ ersetzt wird. Davon bleibt laut Setup $\alpha$ unberührt. Dies gilt auch für $\beta$ und $\gamma$, denn mit 4.2.1 sehen wir, dass

$$\begin{aligned} \gamma'_{ij} &= \beta_{ij}(t_{i-1}z)^{-1}\alpha_{ij}(t_iz) \\ &= \beta_{ij}z^{-1}t_{i-1}^{-1}\alpha_{ij}t_iz \\ &\stackrel{z\in Z(G)}{=} \beta_{ij}t_{i-1}^{-1}\alpha_{ij}t_i \\ &= \gamma_{ij} \end{aligned}$$

bzw. analog zu $\beta_{ij}$. Insbesondere ist klar, dass

$$\breve{\beta}'(x) = y_2(t_sz)^{-1}y_1^{-1}(t_0z) \stackrel{z\in Z(G)}{=} y_2t_s^{-1}y_1^{-1}t_0 = \breve{\beta}(x)$$

Insgesamt können wir also sehen, dass wir tatsächlich nur die Nebenklasse von $t_0$ beim Raten treffen müssen.
Laut dem Satz von Lagrange teilt jede Untergruppenordnung die Ordnung der Gruppe und da weiterhin Nebenklassen stets gleich groß sind, gibt es insgesamt $\frac{q^2}{q} = q$ Nebenklassen. Schlußendlich ergibt sich also eine Gesamtzahl von $\frac{q^2(q-1)}{q} = q(q-1)$ Möglichkeiten -

oder anders gesagt: Ein einzelner Angriff hat eine Erfolgswahrscheinlichkeit von $(q^2-q)^{-1}$. Wird also $q$ entsprechend groß gewählt (z.B. $q = 2^{81}$), so kann diese Art von Attacke als ungefährlich eingestuft werden. Anzumerken ist noch, dass ein falsches $t_0$ und/oder ein falsches $\beta_{s1}$ erst einmal als solches entlarvt werden muss. Eine Möglichkeit besteht darin selbst eine (oder besser mehrere) Nachrichten zu verschlüsseln und sie anschließend mit dem Angriff zu entschlüsseln. Gelingt dies nicht, so muss der nächste Versuch gestartet werden.

**Beispiel.** Wir wollen im Folgenden immer wieder auf unser $MST_3$ von oben zurückgreifen. Nach unseren letzten Überlegungen hat ein einzelner Angriff eine Erfolgswahrscheinlichkeit von $(8^2 - 8)^{-1} = \frac{1}{56} \approx 0,018$, wir wollen aber dennoch der Einfachheit halber annehmen, dass wir sowohl die Nebenklasse, als auch $\beta_{s1}$ richtig erraten haben. Wir wählen also (mit $z = S(0,1)$)

$$\begin{aligned} t_0 &= S(1,0) \\ \beta_{31} &= e \end{aligned}$$

und berechnen (aus 4.2.1)

$$\begin{aligned} t_1 &= S(1,0)^{-1}S(1,0)S(1,1) = S(1,1) \\ t_2 &= S(x,0)^{-1}S(1,1)S(1,x+1) = S(x,0) \\ t_3 &= S(1,x)^{-1}S(x,0)S(x,x+1) = S(1,x+1) \end{aligned}$$

wodurch wir wiederum mittels 4.2.1

$$\beta = [\{e, S(0,x^2)\}, \{e, S(0,x)\}, \{e, S(0,1)\}]$$

erhalten und einen äquivalenten privaten Schlüssel besitzen. Wir müssten, wie bereits erwähnt, nun noch den Schlüssel einige Male testen, um (mit hoher Wahrscheinlichkeit) sagen zu können, dass wir das System gebrochen haben. Man bemerke, dass wir sogar in diesem kleinen Beispiel erahnen können, welche Schwierigkeiten eine Ciphertext-Only-Attacke dem Angreifer bereiten würde. Wurde falsch geraten, so können wir in großen Gruppen nicht einmal sicher sein, ob das gefundene $\beta$ eine Signatur ist. Doch dazu später mehr.

#### 4.2.4.2. Chosen-Plaintext-Attacke

Neben den Mitteln in der Ciphertext-Only-Attacke stehen dem Angreifer nun auch Chiffretexte zu selbst gewählten Klartexten zur Verfügung, allerdings wieder ohne den Schlüssel zu kennen. Somit sind $\beta$ und $(t_0, t_s)$ gesucht, welche mit der Gleichung

$$\begin{aligned} y_2 &= \breve{\beta}(x)t_0^{-1}y_1t_s \\ \Leftrightarrow \quad \breve{\beta}(x) &= y_2t_s^{-1}y_1^{-1}t_0 \end{aligned} \tag{4.2.2}$$

berechnet werden können. Es wird versucht genug Werte $\breve{\beta}(x_i)$ zu bestimmen, um somit $\beta$ rekonstruieren zu können. Wir können, wie bereits mehrfach erwähnt, o.B.d.A. annehmen, dass die Blöcke einer zu $\beta$ äquivalente Signatur stets (bis auf den letzten Block) mit der Identität beginnen. Daraus lässt sich folgende nützliche Eigenschaft sofort ableiten:

**Proposition 4.2.** *Wenn $G$ eine Gruppe der Ordnung $N$ ist und die dazu logarithmische Signatur $\beta$ vom Typ $(r_1, ..., r_s)$ ist, so kann man eine zu $\beta$ äquivalente Signatur rekonstruiren, in dem man insgesamt $1 - s + \sum_{i=1}^{s} r_i$ passend gewählte Werte von $\breve{\beta}(x_i)$ verwendet.*

Wir bemerken an dieser Stelle, dass ein $\breve{\beta}$-Orakel das System leicht brechen könnte, denn eine äquivalente Signatur $\beta'$ wäre schnell bestimmt. Da o.B.d.A. für $i = 1...s$ gilt $\beta'_{i1} = id$ und mit $x \cong (1, ..., 1)$ erhalten wir:

$$\breve{\beta}'(x) = \beta'_{11} \cdot ... \cdot \beta'_{s1} = 1 \cdot ... \cdot 1 \cdot \beta'_{s1} \stackrel{!}{=} \breve{\beta}(x)$$

und wenn ein Orakel vorliegt, so wäre nun $\beta'_{s1}$ bekannt. Analog kann man durch geschickte Wahl von $x$ alle $\beta'_{ij}$ erhalten. In unserer folgenden Betrachtung liegt aber natürlich kein solches Orakel vor.

Sei nun $\{x_1, ..., x_n\}$ eine Menge von Klartexten (mit $x_i \in \mathbb{Z}_{|Z(G)|}$ für alle $i = 1...n$), die vom Angreifer gewählt wurden um $\beta$ zu bestimmen. Wir erinnern uns, dass

$$\breve{\beta}(x_i) = y_{i,2} t_s^{-1} y_{i,1}^{-1} t_0 \text{ für } i = 1...n \text{ und } y_{i,1} := \breve{\alpha}(x_i) \text{ und } y_{i,2} := \breve{\gamma}(x_i)$$

durch unser Kryptosystem gegeben ist. Öffentlich bekannt sind an dieser Stelle lediglich $y_{i,1}$ und $y_{i,2}$. Zunächst formen wir den Ausdruck ein wenig um:

$$\begin{aligned} \breve{\beta}(x_i) &= y_{i,2}(t_s^{-1} y_{i,1}^{-1} t_s) t_s^{-1} t_0 \\ &= y_{i,2} y_{i,1}^{-1} y_{i,1} (t_s^{-1} y_{i,1}^{-1} t_s) t_s^{-1} t_0 \end{aligned}$$

Wenn wir nun beachten, dass $\beta$ eine Signatur des Zentrums ist, so folgt

$$y_{i,2} y_{i,1}^{-1} (y_{i,1} t_s^{-1} y_{i,1}^{-1} t_s) t_s^{-1} t_0 \in Z(G)$$

Mittels der Klammern wurde hier bereits angedeutet, dass $y_{i,1} t_s^{-1} y_{i,1}^{-1} t_s \in G'$, also ein Element der Kommutator-Untergruppe von $G$ ist. In Suzuki-2-Gruppen gilt aber bekanntlich, dass $G' = Z(G)$ und daher gilt für $i = 1...n$:

$$t_0^{-1} t_s \in y_{i,2} y_{i,1}^{-1} Z(G) \Rightarrow t_s \in t_0 y_{i,2} y_{i,1}^{-1} Z(G)$$

Insbesondere sind also die Möglichkeiten für $t_s$ nach der Wahl von $t_0$ deutlich eingeschränkt.

Wir beachten nun, dass Nebenklassen entweder gleich oder disjunkt sind und setzen daher als *Annahme*, dass für $i \neq j$

$$y_{i,2} y_{i,1}^{-1} Z(G) \neq y_{j,2} y_{j,1}^{-1} Z(G)$$

gilt. Wir können daher folgern, dass

$$t_0 y_{i,2} y_{i,1}^{-1} Z(G) \cap t_0 y_{j,2} y_{j,1}^{-1} Z(G) = \emptyset$$

aber, da $t_s \in t_0 y_{i,2} y_{i,1}^{-1} Z(G)$ für alle $i = 1...n$ erhalten wir einen Widerspruch zur Annahme und wissen nun, dass insbesondere für $j = 1$ und $i = 1...n$

$$y_{i,2} y_{i,1}^{-1} \in y_{1,2} y_{1,1}^{-1} Z(G)$$

gilt.

Wie beim ersten Angriff, so müssen wir auch hier ein $t_0 \in G \backslash Z(G)$ aus $q^2 - q$ Möglichkeiten wählen. Ist es gesetzt, so muss nach unseren Betrachtungen nun $t_s \in t_0 y_{1,2} y_{1,1}^{-1} Z(G)$ sein, d.h. es gibt $q = |Z(G)|$ Möglichkeiten für $t_s$. Die Anzahl möglicher Paare $(t_0, t_s)$ beschränkt sich daher auf $q^3 - q^2$. Offensichtlich kann man sofort folgern, dass für $z \in Z(G)$ auch $(t_0 z, t_s z)$ die Eigenschaft $t_s \in t_0 y_{1,2} y_{1,1}^{-1} Z(G)$ besitzt, wenn dies bereits für $(t_0, t_s)$ gilt. Das bedeutet, dass es für jedes Lösungspaar $q$ nahestehende Lösungen gibt.
Nehmen wir nun an, dass $(\tau_0, \tau_s)$ und $(t_0, t_s)$ die Gleichung

$$y_{i,2} t_s^{-1} y_{i,1}^{-1} t_0 = z = \breve{\beta}(x_i) = y_{i,2} \tau_s^{-1} y_{i,1}^{-1} \tau_0$$

erfüllen. Durch Anwendung von $y_{i,2}^{-1}$ auf beide Seiten von links, erhalten wir für alle $i, j \in \{1, ..., n\}$ die zwei Gleichungen

$$\begin{aligned} \tau_0^{-1} y_{i,1} \tau_s &= t_0^{-1} y_{i,1} t_s \\ \tau_s^{-1} y_{j,1}^{-1} \tau_0 &= t_s^{-1} y_{j,1}^{-1} t_0 \end{aligned}$$

welche uns direkt

$$\tau_0^{-1} y_{i,1} \tau_s \tau_s^{-1} y_{j,1}^{-1} \tau_0 = t_0^{-1} y_{i,1} t_s t_s^{-1} y_{j,1}^{-1} t_0$$

und damit

$$\tau_0^{-1} y_{i,1} y_{j,1}^{-1} \tau_0 = t_0^{-1} y_{i,1} y_{j,1}^{-1} t_0 \tag{4.2.3}$$

bringen. Also wissen wir, dass wenn es genug Paare $(i, j)$ gibt, so dass die unterschiedlichen Elemente $y_{i,1} y_{j,1}^{-1}$ die Gruppe $G$ erzeugen, dann induzieren $\tau_0$ und $t_0$ nach unseren Betrachtungen der Grundlagen (siehe Kapitel 2) den gleichen inneren Automorphismus, d.h. $\tau_0 \equiv t_0 \bmod Z(G)$. Damit ist folglich $\tau_0 = t_0 z$ und $\tau_s = t_s z$ für ein $z \in Z(G)$. Schlußendlich können wir sagen, dass die Anzahl möglicher Paare $(t_0, t_s)$, welche $\breve{\beta}(x_i)$ bestimmen ist gleich $\frac{q^3 - q^2}{q} = q^2 - q$. Unser Fazit ist, dass der Angreifer also mindestens $q^2 - q$ Lösungsvektoren $(\breve{\beta}(x_1), ..., \breve{\beta}(x_n))$ konstruiren muss und nur ein Tupel ist dann tatsächlich korrekt. Damit entspricht die Erfolgswahrscheinlichkeit dem Wert $(q^2 - q)^{-1}$, aber wird natürlich wesentlich kleiner, wenn dem Angreifer nicht genug Gleichungen der Art 4.2.3 zur Verfügung stehen.

**Beispiel.** In unserem Beispiel war stets $s = 3$ und $r_1 = r_2 = r_3 = 2$, damit folgt aus der letzten Proposition, dass wir $1 - 3 + 2 + 2 + 2 = 4$ Auswertungen von $\breve{\beta}$ benötigen. Wir nehmen an, dass wir viel Glück hatten und ein korrektes Tupel $(\breve{\beta}(x_1), ..., \breve{\beta}(x_4))$ erraten haben:

$$(\breve{\beta}(5), \breve{\beta}(4), \breve{\beta}(3), \breve{\beta}(2)) = (S(0, x^2 + 1), S(0, 1), S(0, x^2 + x), S(0, x))$$

Da $(\alpha, \gamma)$ sowieso öffentlich sind, werden folgende Werte schnell bestimmt:

$$\begin{aligned}(\breve{\alpha}(5), \breve{\alpha}(4), \breve{\alpha}(3), \breve{\alpha}(2)) &= (S(1, x+1), S(x, x^2+x), S(1, x), S(x, x^2+x+1)) \\ (\breve{\gamma}(5), \breve{\gamma}(4), \breve{\gamma}(3), \breve{\gamma}(2)) &= (S(1, x^2+1), S(x, x), S(1, x^2+x+1), S(x, 0))\end{aligned}$$

Wir stellen nach 4.2.2 das Gleichungssystem

$$\begin{aligned}S(0, x^2+1) &= S(1, x^2+1)t_3^{-1}S(1, x+1)^{-1}t_0 \\ S(0, 1) &= S(x, x)t_3^{-1}S(x, x^2+x)^{-1}t_0 \\ S(0, x^2+x) &= S(1, x^2+x+1)t_3^{-1}S(1, x)^{-1}t_0 \\ S(0, x) &= S(x, 0)t_3^{-1}S(x, x^2+x+1)^{-1}t_0\end{aligned}$$

auf und erhalten (wie oben beschrieben) ein korrektes Paar $(t_0, t_3) = (S(1,1), S(1,x))$. Damit liegt uns eine Art Orakel für $\breve{\beta}$ vor und wir können vorgehen wie oben beschrieben um das System zu knacken. Auch dieser Angriff hängt wieder von zu vielen Parametern ab, als dass er bei großen Gruppen realisierbar wäre.

#### 4.2.4.3. Angriff auf kanonische Signaturen

Um diesen Angriff, welcher aus [5] bekannt ist, erklären zu können, müssen wir zunächst eine neue Notation einführen.

**Definition 4.1.** Sei $G$ eine Suzuki-2-Gruppe in Matrixdarstellung über dem Körper $\mathbb{F}_{2^m}$ mit dem Automorphismus $\theta$. Wenn $g = S(x, y) \in G$, so bezeichnen wir $x$ mit $g_{.a}$ und $y$ mit $g_{.b}$, also $g = S(g_{.a}, g_{.b})$. Ist zudem $M \in \mathrm{GL}(m, 2)$, so definiere $gM := S(g_{.a}, g_{.b}M)$.

Es ist klar, dass die Operation unter der regulären Matrix den $.a$-Part unberührt lässt. Schnell sehen wir nun zudem folgende zwei Eigenschaften ein:

**Lemma 4.1.** *Mit obiger Notation gilt* $g^{-1} = S(g_{.a}, g_{.b})^{-1} = S(g_{.a}, g_{.b} + g_{.a}^{\theta} g_{.a})$. *Weiterhin haben Elemente der gleichen Nebenklasse unter* $Z(G)$ *den selben* $.a-$*Part, d.h.* $\forall t_j \in t_i Z(G) : t_{(i).a} = t_{(j).a}$.

*Beweis.* Da

$$\begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & a^{\theta} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -a & 1 & 0 \\ a^{\theta}a - b & -a^{\theta} & 1 \end{pmatrix} \overset{\mathbb{F}_{2^m}}{=} \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b + a^{\theta}a & a^{\theta} & 1 \end{pmatrix}$$

haben wir den ersten Teil bereits gezeigt. Und mit $z \in Z(G)$ beweist

$$\begin{aligned}t_j &= S(t_{j.a}, t_{j.b}) \\ &= t_i z \\ &= S(t_{i.a}, t_{i.b})S(0, z_{.b}) \\ &= S(t_{i.a}, t_{i.b} + z_{.b})\end{aligned}$$

dass $t_{j.a} = t_{i.a}$ gilt. Dies war zu zeigen. □

Wir wollen nun zeigen, wie der Angreifer ein $\tilde{t_0} \in t_0 Z(G)$ wählen kann um unser Kryptosystem zu brechen. Hierbei will er eine spezielle Eigenschaft der kanonischen Signaturen ausnutzen, welche einen Schlüsselgenerator erzeugt. Betrachten wir also zunächst den ersten Block von $\gamma$:

$$\gamma_{1j} = \beta_{1j} t_0^{-1} \alpha_{ij} t_1$$

wobei mit obiger Notation und den bekannten Rechenregeln zu sehen ist, dass

$$\begin{aligned}
\gamma_{(1,1).a} &= t_{0.a}^{-1} + \alpha_{(1,1).a} + t_{1.a} \\
\gamma_{(1,j).b} &= \beta_{(1,j).b} + t_{0.b}^{-1} + \alpha_{(1,j).b} + t_{1.b} + (t_{0.a})^{\Theta} \alpha_{(1,j).a} \\
&\quad + (t_{0.a})^{\Theta} t_{1.a} + (\alpha_{(1,j).a})^{\Theta} t_{1.a}
\end{aligned}$$

Weiterhin gilt zu einer Indexmenge $J$ mit gerader Größe (also $\exists m \in \mathbb{N} : |J| = 2m$):

$$\begin{aligned}
\sum_{j \in J} \gamma_{(1,j).b} &= \sum_{j \in J} \beta_{(1,j).b} + \sum_{j \in J} \alpha_{(1,j).b} + (t_{0.a})^{\Theta} \sum_{j \in J} \alpha_{(1,j).a} + t_{1.a} \sum_{j \in J} (\alpha_{(1,j).a})^{\Theta} \\
&= \sum_{j \in J} \beta_{(1,j).b} + \sum_{j \in J} \alpha_{(1,j).b} + (t_{0.a})^{\Theta} \sum_{j \in J} \alpha_{(1,j).a} \\
&\quad + (\gamma_{(1,1).a} + t_{0.a} + \alpha_{(1,1).a}) \sum_{j \in J} (\alpha_{(1,j).a})^{\Theta}
\end{aligned}$$

denn, da wir im $\mathbb{F}_{2^m}$ rechnen, sind die Rechnungen XOR (somit sind Einheiten mit ihrem Inversen identisch) und da $|J|$ gerade ist, gilt

$$\sum_{j \in J} t_{0.b}^{-1} + t_{1.b} + t_{0.a}^{\Theta} t_{1.a} = |J|(t_{0.b}^{-1} + t_{1.b} + t_{0.a}^{\Theta} t_{1.a}) = 0.$$

Wir führen zunächst ein paar Abkürzungen ein, um die folgende Formel übersichtlicher zu gestalten:

$$\begin{aligned}
A &= \sum_{j \in J} \alpha_{(1,j).a} \\
B &= \sum_{j \in J} (\alpha_{(1,j).a})^{\Theta} \\
C &= \sum_{j \in J} \beta_{(1,j).b} + \sum_{j \in J} \gamma_{(1,j).b} + \sum_{j \in J} \alpha_{(1,j).b} \\
&\quad + (\gamma_{(1,1).a} + \alpha_{(1,1).a}) \sum_{j \in J} (\alpha_{(1,j).a})^{\Theta}
\end{aligned}$$

Damit erhalten wir also insgesamt die Formel

$$A(t_{0.a})^{\Theta} + B(t_{0.a}) + C = 0 \tag{4.2.4}$$

Sinnvollerweise sollte $J$ nun so gewählt werden, dass die Unbekannte, also die Summe der Elemente des ersten Blocks von $\beta$, Null ergibt. Dazu können wir die spezielle Struktur

kanonischer logarithmischer Signaturen nutzen. Zunächst ist klar, dass für $\beta = (\beta_{ij}) = (\epsilon_{ij}M)$ für eine kanonische Signatur $\epsilon$ und $M \in \mathrm{GL}(m, 2)$ gilt

$$\sum_{j\in J} \beta_{(1,j).b} = \sum_{j\in J} \epsilon_{(1,j).b}M = (\sum_{j\in J} \epsilon_{(1,j).b})M.$$

Der Block der kanonisch logarithmischen Signatur $\epsilon$ enthält eine komplette Menge von $2^{k_i}$ Vektoren, daher können wir immer ein $J \subseteq \{1..r_1\}$ so finden, dass $\sum_{j\in J} \epsilon_{(1,j).b} = 0$ ist. Nun ist das Problem das System zu brechen auf das Problem eine Lösung von 4.2.4 über $\mathbb{F}_q$ zu finden reduziert. Laut [5] ist dies in polynomialer Zeit möglich. Insgesamt sehen wir also, dass kanonische Signaturen ohne Modifikation nicht sicher genug sind, um sie für eine Implementierung unseres Kryptosystems zu nutzen.

### 4.2.5. Bemerkungen

Wie bereits erwähnt ist es in der Praxis sinnvoll die Elemente der Suzuki-2-Gruppe als Tripel $(a, b, a^\Theta)$ abzuspeichern, um somit die Zeitkomplexität zu senken und insgesamt für eine Multiplikation von Gruppenelementen lediglich vier Additionen in $\mathbb{F}_q$ und eine einzige Multiplikation zu benötigen, denn

$$(a, b, a^\Theta)(c, d, c^\Theta) = (a + c, b + d + a^\Theta c, a^\Theta + c^\Theta)$$

Der somit verkleinerte Speicherplatzbedarf und die höchst effizienten Operationsmöglichkeiten in der Suzuki-2-Gruppe machen diese sehr attraktiv für die Implementierung unseres Kryptosystems.

Ein wichtiger Wert, bekannt aus [1], der im Zusammenhang mit der Sicherheit des Systems steht, soll noch angemerkt werden. Die durchschnittliche Anzahl an Urbildern eines $y \in J$ (siehe Setup) kann mit dem Wert $\frac{|Z(G)|}{|J|}$ beschrieben werden. Will man also nicht davon ausgehen, dass zufällig erzeugte Cover zu großen endlichen Gruppen Einweg-Funktionen induzieren, so muss man diesen Wert möglichst groß wählen, um weiterhin eine gute Sicherheit erwarten zu können.

# 5. Eine Kryptoanalyse von $MST_3$

In diesem Kapitel soll es hauptsächlich um die Arbeit [3] von Blackburn, Cid und Mullan gehen. Wir wollen zunächst den Inhalt der Arbeit betrachten und analysieren, um schließlich Mängel und Fehler einsehen zu können. Unsere Argumente werden wir teils theoretisch, aber auch teils praktisch durch Computertests stützen, um letztlich die Behauptung, dass $MST_3$ angeblich unsicher sei, entschieden zurückweisen zu können. Fehlende oder fehlerhafte Beweise versuchen wir stets zu ergänzen und zu korrigieren. Entstammt ein Beweis aus [3], so wird darauf hingewiesen.

## 5.1. Grundlagen

Die Arbeit [3] beginnt damit die Grundlagen zu Covern und Signaturen kurz zu erläutern. Es fällt auf, dass bei der Beschreibung des $MST_3$-Setup die Eigenschaft, dass $A_i \subseteq G \backslash Z(G) \, \forall i = 1...s$ ignoriert wird. Dies wird damit begründet, dass dies weder für die Ver- noch für die Entschlüsselung gebraucht wird und sowieso fast immer erfüllt sei, falls $\alpha$ zufällig gewählt wurde. Weiterhin ignorieren die Autoren die Forderung, dass keine zwei Elemente von $A_i$ in der selben Nebenklasse von $Z(G)$ liegen sollen - mit ähnlicher Begründung.
Kanonische und transversale Signaturen werden ebenfalls erläutert, allerdings tritt an dieser Stelle zum ersten Mal ein neuer Begriff auf, welchen wir nun genauer untersuchen wollen.

### 5.1.1. ATLS

**Definition 5.1.** Sei $\alpha = [A_1, ..., A_s]$ eine zahme transversale logarithmische Signatur. Wir bezeichnen $\beta$ als eine *ATLS (Amalgated Transversal Log. Sig.)*, falls es durch wenigstens eine der folgenden Operationen durch $\alpha$ entstanden ist:

- Permutation(en) innerhalb der Blöcke $A_i$ für $i = 1...s$
- Permutation(en) der Blöcke miteinander
- Ersetzung von $A_i$ durch $A_i z$, für ein $z \in Z(G)$ und $i \in \{1, ..., s\}$
- Fusion von zwei Blöcken $A_i$ und $A_j$ zu $A_i A_j := \{gh : g \in A_i, h \in A_j\}$

Laut eigenen Aussagen der Autoren ist dies die allgemeinste Methode, um eine logarithmische Signatur zu einer elementar-abelschen 2-Gruppe zu konstruieren.

**Proposition 5.1.** *Wurde die ATLS $\beta$ aus $\alpha$ wie oben beschrieben konstruiert, so ist $\beta$ eine logarithmische Signatur.*

*Beweis.* Wir prüfen die Operationen einzeln.
Dass die Permutationen (innerhalb der Blöcke, oder die Blöcke selbst) nicht die Signatur an sich zerstören ist klar, denn in Gruppenringen gilt das Distributivgesetz.

Ersetzen wir o.B.d.A. nun $A_1$ durch $A_1 z$ mit $z \in Z(G)$, so hat die Bijektion $\breve{\alpha}(x)$ die Gestalt $\breve{\alpha}(x) \cdot z$, was wiederum eine Bijektion ist. Wir erhalten also wieder eine Signatur.

Führen wir o.B.d.A. eine Fusion (der Übersichtlichkeit halber) mit den ersten beiden Blöcken durch, so haben wir $\alpha' = [A_1 A_2, A_3, ..., A_s]$. Es gilt

$$\begin{aligned}
\overline{A_1 A_2} \cdot \overline{A_3} \cdot ... \cdot \overline{A_s} &= \sum_{k=1}^{r_1} \sum_{j=1}^{r_2} \alpha_{1k}\alpha_{2j} \cdot \prod_{i=3}^{s} \sum_{j=1}^{r_i} \alpha_{ij} \\
&= (\alpha_{11}\alpha_{21} + ... + \alpha_{1r_1}\alpha_{2r_2})(\alpha_{31} + ... + \alpha_{3r_3}) \\
&\quad \cdot ... \cdot (\alpha_{s1} + ... + \alpha_{sr_s}) \\
&= (\alpha_{11} + ... + \alpha_{1r_1})(\alpha_{21} + ... + \alpha_{2r_2}) \cdot ... \cdot (\alpha_{s1} + ... + \alpha_{sr_s}) \\
&= \overline{A_1} \cdot \overline{A_2} \cdot ... \cdot \overline{A_s}
\end{aligned}$$

was uns zeigt, dass wir wieder eine Signatur vorliegen haben. □

Anhand eines kurzen Beispiels wollen wir die praktische Umsetzung der Fusions- und Permutationsoperation sehen. Die dritte Operation aus der Definition wird später noch Anwendung finden.

**Beispiel.** Mittels des angehängten Programms (per GAP: [13]) wird eine logarithmische Signatur $\beta$ des Typs $[4, 4, 4, 4]$ über eine reguläre Matrix erzeugt (siehe nachfolgende Tabelle). Nehme dazu vier mal zwei Zeilen und erzeuge die Blöcke $B_i$ durch Linearkombination dieser je zwei Zeilenvektoren. Als nächstes fusionieren wir die Blöcke 1 und 3, sowie die Blöcke 2 und 4 und permutieren die Elemente innerhalb der Blöcke. Das Ergebnis bezeichnen wir mit $\beta'$. Man bemerke, dass die relativ unhandlich gewordene Menge $\beta'$ laut unserer Proposition immer noch eine logarithmische Signatur ist!

Für eine der zentralen Aussagen der Arbeit benötigen wir zunächst noch eine weitere Eigenschaft der ATLS, der wir uns nun zuwenden wollen.

**Proposition 5.2.** *Sei $\beta = [B_1, ..., B_s]$ ein ATLS zu einer elementar-abelschen 2-Gruppe $Z$ mit $1 \in B_i \forall i = 1...s$, so gibt es eine Teilmenge $B_i$ und ein nicht-triviales Element $\beta_{ij} \in B_i$, so dass $B_i \cdot \beta_{ij} = B_i$ gilt. Diese Eigenschaft wird auch als Periodizität bezeichnet.*

| $\beta$ | | | $\beta'$ | |
|---|---|---|---|---|
| | | | $\pi_1(B_1 \cdot B_3)$ | $\pi_2(B_2 \cdot B_4)$ |
| $B_1$ | 00000000 | | 01000100 | 10001011 |
| | 00110010 | | 11011001 | 11000111 |
| | 11011001 | | 10011101 | 10000110 |
| | 11101011 | | 01110110 | 11100111 |
| $B_2$ | 00000000 | | 10010011 | 00001101 |
| | 10000110 | | 01111000 | 01001100 |
| | 01000001 | | 00111100 | 00101101 |
| | 11000111 | | 11101011 | 01101100 |
| $B_3$ | 00000000 | | 01001010 | 00100000 |
| | 00001110 | | 10101111 | 10101011 |
| | 01001010 | | 11010111 | 11001010 |
| | 01000100 | | 00110010 | 01000001 |
| $B_4$ | 00000000 | | 10100001 | 10100110 |
| | 01100001 | | 00000000 | 00000000 |
| | 10001011 | | 11100101 | 01100001 |
| | 11101010 | | 00001110 | 11101010 |

Tabelle 5.1.: Fusion und Permutation

Wir bemerken, dass der Fall $B_i = Z_1 z$ in dieser Proposition durch die Voraussetzung, dass $1 \in B_i$ für alle $i = 1...s$ gelten soll, ausgeschlossen wird, da er sonst zum Widerspruch führen würde. Blackburn et al. haben eine kurze Beweisskizze angegeben. Wir versuchen sie weiter auszuführen.

*Beweis.* Der Beweis verläuft in zwei Schritten. Zunächst zeigen wir, dass aus der Tatsache, dass die Untergruppe $Z_1$ in eine der Teilmengen $B_i$ fusioniert wurde, folgt, dass sowohl $B_i Z_1 = B_i$ als auch $Z_1 \subseteq B_i$ gilt. Damit können wir also ein $\beta_{ij} \in Z_1 \backslash \{1\}$ wählen, welches die gewünschte Eigenschaft erfüllt.
Als erstes brauchen wir für transversale Signaturen eine Kette von Untergruppen, die hier durch

$$1 = Z_0 < Z_1 < ... < Z_k = G$$

gegeben sei. Wir haben also durch Anwendung der vier oben genannten Operationen $B_i$ aus $Z_1$ erhalten und wollen nun prüfen, ob $B_i Z_1 = B_i$ gilt. Prüfen wir also die Operationen nacheinander:

- Beide Permutationsmöglichkeiten ändern nur die Reihenfolge der Elemente von $Z_1$, daher ist $B_i Z_1 = B_i$
- Ist $B_i = Z_1 B_j$ für $j \neq i$, so ist $B_i Z_1 = Z_1 Z_1 B_j = Z_1 B_j = B_i$

Schließlich prüfen wir analog ob $Z_1 \subseteq B_i$ ist:

- Permutationen ändern, wie bereits erwähnt, nur die Reihenfolge, daher ist $Z_1 \subseteq B_i$ klar
- Ist $B_i = Z_1 B_j = \{gh : g \in Z_1, h \in B_j\}$ für $j \neq i$, so enthält dieses Produkt insbesondere (mit $h = 1$) ganz $Z_1$

Damit ist die Behauptung bewiesen. □
Man kann (beispielsweise mittels GAP) in dem Beispiel von oben schnell sehen, dass $\beta_{2,16} = 11101010$ die beschriebene Eigenschaft aufweist.

### 5.1.2. Der zentrale Satz

Kommen wir nun zu dem eigentlich zentralen Satz der Arbeit von Blackburn et al. (original als Lemma bezeichnet).

**Theorem 5.1.** *Sei $\beta = [B_1, .., B_s]$ eine ATLS zu einer elementar-abelschen 2-Gruppe $Z$, so ist $\beta$ zahm.*

Bevor wir uns dem von den Autoren gegebenen Beweis zuwenden, müssen wir den Inhalt des Satzes genau erfassen. Wir haben also eine transversale Signatur durch Permutationen und Fusionen wie oben zu einer ATLS verwandelt und behaupten nun, dass die Faktorisierbarkeit der zugrundeliegenden Signatur auf die ATLS vererbt wird. Wenn wir die Faktorisierbarkeit prüfen wollen, so sind wir gezwungen zu prüfen, ob jedes Gruppenelement auf genau eine Art in polynomieller Zeit dargestellt wird. Dies ist gerade bei großen Gruppen ein nahezu unmöglich zu bewältigender Aufwand, auch wenn wir uns mit einer großen Wahrscheinlichkeit begnügen wollen und die Geburtstagsattacke verwenden (die im Übrigen nicht in der Arbeit [3] erwähnt wird). Das hier gestellte Problem ist also durchaus als schwierig zu betrachten.

*Beweis.* Die Behauptung soll per Induktion über die Gruppengröße $|Z| = n$ bewiesen werden. Der Fall $n = 1$ ist trivial. Betrachten wir also den Induktionsschritt $(n-1 \to n)$. Wie bereits häufig bemerkt ist es o.B.d.A. möglich anzunehmen, dass $1 \in B_i \, \forall i = 1...s$. Es sei $\epsilon$ eine transversale logarithmische Signatur und $\epsilon = \epsilon_0, \epsilon_1, ..., \epsilon_l = \beta$ eine Sequenz von Signaturen, die bei der Erzeugung von $\beta$ generiert wurden. Somit ist jedes $\epsilon_i$ durch eine der vier ATLS-Operationen aus $\epsilon_{i-1}$ entstanden. Es gibt nach der vorherigen Proposition ein $z = \beta_{ij} \in Z$ (eigentlich aus $Z \backslash \{1\}$), so dass $B_i z = B_i$. Solch ein Element kann man effizient finden in dem man höchstens $\sum_{i=1}^{s} |B_i|$ Elemente prüft.
Sei nun $N = \{1, z\}$ die von $z$ erzeugte Untergruppe. Da also $|Z/N| = \frac{1}{2}|Z|$ definieren wir eine Signatur $\bar{\beta}$ zu $Z/N$ durch $\bar{B}_k := B_k N/N$, wobei wir die überflüssigen Elemente aus $B_i$ streichen, so dass $\bar{B}_i$ nur halb so groß ist wie $B_i$ (also jeweils ein Bit weniger, was $n-1$ entspricht, da $Z$ eine 2-Gruppe ist). Jetzt ist $\bar{\beta}$ eine ATLS, was sich dadurch zeigt, dass die Sequenz $\bar{\epsilon} = \bar{\epsilon}_0, \bar{\epsilon}_1, ..., \bar{\epsilon}_l = \bar{\beta}$ von logarithmischen Signaturen von $Z/N$ durch die selben ATLS-Operationen wie zuvor definiert ist. Da $\bar{\beta}$ eine ATLS ist muss es laut Induktionsvoraussetzung zahm sein und daher können wir $\breve{\bar{\beta}}^{-1}(xN)$ für alle $x \in Z$

effizient berechnen. Wir können $\breve{\beta}$ effizient invertieren, da es nur zwei Möglichkeiten für $\breve{\beta}^{-1}(x)$ gibt, wenn $\breve{\beta}^{-1}(xN)$ bekannt ist. Daher ist $\beta$ zahm. □

Ein Problem des Beweises haben wir bereits kurz angeschnitten. Es steckt in den letzten Behauptungen, denn die jeweils zwei Möglichkeiten für $\breve{\beta}^{-1}(x)$ sind schwer zu bestimmen und potenzieren sich auf und gerade bei großen Gruppen ist dieses Produkt nicht zu unterschätzen. Weiterhin ist die Induktion nicht vollständig, denn sofort sehen wir an einem kleinen Beispiel (siehe Tabelle), dass die Modulo-Rechnung nur auf einen Block anwendbar ist - nämlich bei demjenigen, bei dem das Element $z$ gefunden wurde. Wie aber sollen wir faktorisieren, wenn dieser Weg uns letztlich nur einen Block vereinfacht, aber alle anderen weiterhin unhandlich bleiben?

| $\beta$ |
|---|
| 0 0 0 0 |
| 0 1 0 0 |
| 1 0 0 0 |
| 1 1 0 0 |
| 0 0 0 0 |
| 1 1 0 1 |
| 1 0 1 0 |
| 1 0 1 1 |

Tabelle 5.2.: ATLS - Der untere Block besitzt kein $z$ mit $Bz = B$

Die Argumentation ist also nicht als Beweis anzusehen, sondern nur als unvollständige Ideenskizze. Dennoch ist es nach eingehender Analyse der Beweisidee möglich gewesen einen Algorithmus aufzustellen, der die gesuchte Faktorisierung für ATLS übernimmt.

## 5.2. Ein Faktorisierungsalgorithmus

In diesem Abschnitt erklären wir zunächst den neuen Faktorisierungsalgorithmus und werden ihn anschließend diskutieren und auf weitere Anwendungsmöglichkeiten hin untersuchen.

### 5.2.1. Der Algorithmus

Um den Algorithmus leichter erklären zu können, definieren wir zunächst die Spaltennormalisierung.

**Definition 5.2.** Es seien $v$ und $z$ Elemente einer elementar-abelschen 2-Gruppe $G$. Wir definieren die *Spaltennormalisierung von $v$ nach $z$* folgendermaßen: Sei $k$ eine beliebige aber fortan feste Stelle von $z$ die gleich 1 ist. Wir prüfen nun den $k$-ten Eintrag von $v$ und behalten $v$ bei, wenn hier eine 0 zu finden ist, oder aber ersetzen sonst $v$ durch $v \oplus z$.

**Algorithm 5.1** Faktorisierungsalgorithmus für ATLS

**Input:** ATLS $\beta = [B_1, ..., B_s]$ zu Gruppe $G$ mit Typ $(r_1, ..., r_s)$, wobei $|G| = 2^m$, sowie Chiffretext $y \in G$

**Output:** Faktorisierung von $y$ unter $\beta$ in der Gestalt $y = \beta_{1j_1} \cdot ... \cdot \beta_{sj_s}$

1: $y^0 := y$; $list := []$;
2: **for** $t$ **in** $[0, ..., m-1]$ **do**
3: Suche ein $z_t$ mit $B_i^t z_t = B_i^t$ für ein $i \in \{1..s\}$
4: $N_t := \{e, z_t\}$, $B_i^{t+1} := B_i^t N/N$;
5: $y^{t+1} := y^t N_t / N_t$;
6: Add($list$,$i$);
7: **end for**;
8: Lies triviale Faktorisierung von $y^m$ ab, also $y^m = \beta_{1j_1}^m \cdot ... \cdot \beta_{sj_s}^m$
9: **for** $k$ **in** $[m-1, ..., 0]$ **do**
10: $\lambda := \beta_{list[k],j_h}^{k+1}$
11: $\mu := \lambda \cdot z_k$;
12: **if** $\lambda$ **in** $B_h^k$ **then**
13: $y^k := \beta_{1j_1}^{k+1} \cdot ... \cdot \beta_{list[k]-1,j_{list[k]-1}}^{k+1} \cdot \lambda \cdot \beta_{list[k]+1,j_{list[k]+1}}^{k+1} ... \cdot \beta_{sj_s}^k$;
14: **else**
15: $y^k := \beta_{1j_1}^{k+1} \cdot ... \cdot \beta_{list[k]-1,j_{list[k]-1}}^{k+1} \cdot \mu \cdot \beta_{list[k]+1,j_{list[k]+1}}^{k+1} ... \cdot \beta_{sj_s}^k$;
16: **end if**;
17: **end for**;
18: **return** Faktorisierung von $y^0$;

Offensichtlich gibt es nach diesem Verfahren an der $k$-ten Stelle von $v$ in jedem Fall eine 0. Bei häufiger Anwendung spricht man auch von *n-facher Spaltennormalisierung*.

Im Folgenden erklären wir einen einzelnen Durchlauf des Algorithmus. Das Verfahren kann entsprechend rekursiv fortgesetzt werden.

**Spaltennormalisierung:**
Sei $\beta = [B_1, ..., B_s]$ eine ATLS zu einer elementar-abelschen 2-Gruppe $G$ mit $1 \in B_i$ für alle $i = 1..s$. Weiterhin sei $y \in G$ der zu faktorisierende Chiffretext. Wir wissen es gibt einen Block $B$ und ein nicht-triviales Element $z \in B$, so dass $B \cdot z = B$ gilt und setzen $N = \{\bar{0}, z\}$. Somit können wir $B_i^* := B_i N/N$ für alle $i = 1..s$ berechnen, was also letztlich eine Spaltennormalisierung von $\beta$ nach $z$ ist. Analog wird $y$ nach $z$ normalisiert. Insgesamt haben wir also eine Nullspalte, die in der normalisierten Signatur $\beta^*$ und im normalisierten Chiffretext $y^*$ gleichermaßen zu finden ist.

**Faktorisierung:**
Wir nehmen an, dass nach der (normalerweise mehrfachen) Spaltennormalisierung eine Faktorisierung trivial geworden ist, beispielsweise weil nur noch eine Spalte nicht normalisiert ist. Nun wollen wir die Spaltennormalisierung rückgängig machen. Sei also die triviale Faktorisierung gegeben durch $y^* = \beta_{1j_1}^* \cdot ... \cdot \beta_{sj_s}^*$. Da wir wissen, wie $y$ aussieht,

können wir überlegen, welche Möglichkeiten es gibt, um die Normalisierung rückgängig zu machen. Ist die in der Definition angesprochene $k$-te Stelle von $y$ beispielsweise eine 0, so gibt es 2 Möglichkeiten diese darzustellen. Beispielsweise ist $0 = 0+0 \, mod \, 2 = 1+1 \, mod \, 2$, also gibt es hier zwei Kombinationsmöglichkeiten (die man durch Umstellung der Gleichung auf eine Möglichkeit reduzieren kann). Analog im Falle, dass die besagte Stelle eine 1 ist. Wir ersetzen also den Faktor, der aus dem durch die Modulo-Rechnung halbierten Block stammt, jeweils an der $k$-ten Stelle um die entsprechende Zahl und erhalten somit 2 Faktorisierungsmöglichkeiten für $y$. Da $\beta$ eine ATLS ist, kann es jedoch nur eine Faktorisierung geben und diese erhalten wir einfach durch simples Nachprüfen, ob die ersetzten Elemente wirklich in den entsprechenden Blöcken vorhanden sind. Alle anderen Faktoren können einfach abgelesen werden, da ihr jeweiliger Block in der Modulo-Rechnung unverändert blieb und somit ihre Position die gleiche blieb. Am Ende bleibt nur eine Kombination übrig, welche dann auch die gesuchte Faktorisierung sein muss.

**Beispiel:**
Es sei $y := 1110$ und $\beta$ sei gegeben durch folgende Tabelle.

| $\beta$ |
|---|
| 0 0 0 0 |
| 0 1 0 0 |
| 1 0 0 0 |
| 1 1 0 0 |
| 0 0 0 0 |
| 1 1 0 1 |
| 1 0 1 0 |
| 1 0 1 1 |

Tabelle 5.3.: Signatur $\beta$

Wir sehen, dass $z := 0\mathbf{1}00$ aus dem ersten Block die gewünschte Eigenschaft erfüllt und führen eine Spaltennormalisierung von $y$ und $\beta$ nach $z$ durch. Wir erhalten

$$y^* = 1\mathbf{1}10 \oplus 0\mathbf{1}00 = 1\mathbf{0}10$$

da $y$ an der zweiten Stelle eine 1 hat. Weiterhin bestimmen wir $\beta^*$ (siehe Tabelle) und sehen, dass eine Faktorisierung nun deutlich leichter ist, da wir doppelte Einträge ignorieren können und nur noch drei, anstatt vier, Spalten beachten müssen.

Wir lesen also ab, dass

$$y^* = \beta^*_{10} \cdot \beta^*_{22}$$

und suchen dazu nun die möglichen „Urbilder". Da $z$ aus dem ersten Block stammt, wissen wir direkt, dass die Position des gesuchten Faktors im zweiten Block die gleiche wie zuvor ist (also $\beta_{22}$). Wir sehen, dass $y$ an der zweiten Stelle eine 1 besitzt, also bleiben von den

| $\beta$ | $\longrightarrow$ | $\beta^*$ |
|---|---|---|
| 0 **0** 0 0 | | 0 **0** 0 0 |
| 0 **1** 0 0 | $\oplus z$ | 0 **0** 0 0 |
| 1 **0** 0 0 | | 1 **0** 0 0 |
| 1 **1** 0 0 | $\oplus z$ | 1 **0** 0 0 |
| 0 **0** 0 0 | | 0 **0** 0 0 |
| 1 **1** 0 1 | $\oplus z$ | 1 **0** 0 1 |
| 1 **0** 1 0 | | 1 **0** 1 0 |
| 1 **0** 1 1 | | 1 **0** 1 1 |

Tabelle 5.4.: $\beta$ wird zu $\beta^*$

zwei Möglichkeiten

$$\begin{aligned} y &= 0\mathbf{0}00 \oplus 1\mathbf{0}10 \\ y &= 0\mathbf{1}00 \oplus 1\mathbf{0}10 \end{aligned}$$

nur die zweite übrig. Insofern ist die gesuchte Faktorisierung durch

$$y = \beta_{11} \cdot \beta_{22}$$

gegeben und wir sind fertig. Statt der Urbildbestimmung kann man natürlich auch einfach $\beta_{1j_1} = y \cdot \beta_{22}^{-1}$ eindeutig berechnen und das Element im ersten Block suchen.
Im Anhang B findet sich ein ausführlicheres Beispiel zu dem vorgestellten Algorithmus.

**Lemma 5.1.** *Der genannte Faktorisierungsalgorithmus gibt als Output die Faktorisierung von $y \in G$ unter $\beta$ in der Gestalt $y = \beta_{1j_1} \cdot ... \cdot \beta_{sj_s}$ zurück und hat eine Komplexität von $\mathcal{O}(\sum_{i=1}^{k} r_i^{-1} t_i^2 \log_2^2(t_i))$, wobei die ATLS $\beta$ vom Typ $(t_1, ..., t_k)$ und die zugrunde liegende transversale Signatur $\alpha$ vom Typ $(r_1, ..., r_s)$ gegeben sind. Die ATLS darf nur durch eine Standard-Fusion entstanden sein, d.h. $B_i = A_i A_{i+\frac{s}{2}}$.*

*Beweis.* Wir müssen nur noch die Komplexität nachrechnen.
Eine ATLS wird laut Definition über eine Kette von Untergruppen zu $G$ erzeugt. Sei diese also hier gegeben durch

$$\gamma : 1 = G_0 < G_1 < G_2 < ... < G_s = G$$

so ist $\alpha = [A_1, ..., A_s]$ mit $A_1 = G_1/G_0 = G_1$, d.h. $A_1$ ist eine Untergruppe. Wird nun eine Fusion von $A_1$ mit $A_j$ durchgeführt, so sind mindestens die Elemente aus $A_1$ periodisch, denn $A_1 A_1 A_j = A_1 A_j$, da $A_1$ abgeschlossen ist. Die Wahrscheinlichkeit in der ATLS im ersten Block ein periodisches Element zu finden ist also größer als $\frac{r_1}{r_1 r_j} = \frac{1}{r_j}$, wenn $\alpha$ vom Typ $(r_1, ..., r_s)$ ist. Man beachte, dass $\alpha$ nicht die ATLS, sondern nur die zugrunde liegende transversale Signatur ist! Wir wollen im Folgenden die resultierende ATLS mit $\beta = [A_1, ..., A_k]$ bezeichnen. Ihr Typ sei $(t_1, ..., t_s)$.

Wird also $\alpha$ mehrfach fusioniert um $\beta$ zu erhalten, so ist die Wahrscheinlichkeit entsprechend $r_1(\prod_{i\in I} r_i)^{-1}$ mit $I$ als die Indexmenge der fusionierten Blöcke.
Führen wir nun eine maximale Spaltennormalisierung durch und erhalten $\beta'$, so enthält $B_1'$ nur noch das neutrale Element und die Untergruppenkette $\gamma'$ hat die Gestalt:

$$\gamma' : 1 = G_1/G_1 < G_2/G_1 < ... < G_s/G_1$$

das heißt insbesondere, dass nun $A_2' = G_2/G_1$ wieder eine Untergruppe ist und wir fahren fort wie oben beschrieben.
Um also im Faktorisierungsalgorithmus das periodische Element zu finden müssen wir nur den jeweils ersten nicht-trivialen Block untersuchen. Dabei müssen wir durchschnittlich $r_i^{-1}t_i$-viele Elemente auf Periodizität prüfen, also mit dem Block mulitplizieren und prüfen, ob das Resultat wieder im Block enthalten ist, was jeweils einen Aufwand von $\mathcal{O}(r_i^{-1}t_i^2\log_2(t_i))$ bedeutet. Um einen Block auf die Größe 1 zu halbieren bedarf es offensichtlich $\log_2(t_i)$-viele Durchläufe, was zur Folge hat, dass ein einzelner Block den Aufwand $\mathcal{O}(r_i^{-1}t_i^2\log_2^2(t_i))$ mit sich bringt. Da wir $k$-viele Blöcke haben folgt also schließlich die Behauptung. □

Will man anders fusionieren, als im Lemma angegeben, so muss man vor der Faktorisierung eine Blockpermutation durchführen, welche die vorgegebene Ordnung herstellt. Man beachte zudem, dass die Formel nur ein Minimum darstellt, denn wenn die Signatur mehrfach fusioniert wird, so wird die Komplexität natürlich größer, allerdings auch unübersichtlicher.

Wir bemerken, dass die Komplexität i.A. *nicht* polynomial ist, daher kann man nicht davon reden, dass $\beta$ zahm ist. Man kann ihn anwenden, wenn sich die ATLS gut abspeichern lässt, daher ist der Algorithmus für die Praxis interessant, jedoch in der Theorie eben dennoch bei großen Blöcken sehr komplex. Insofern ist das Verfahren zwar eine interessante Erkenntnis, jedoch kein Beweis für das Theorem von Blackburn et al, welches also weiterhin kritisch zu hinterfragen ist. Im Übrigen ist der Algorithmus nicht auf Betrachtungen über dem Grundkörper $\mathbb{F}_2$ beschränkt, sondern kann auf $p$-Gruppen erweitert werden, wenn sich weiterhin ein $z \in G$ mit $B_i z = B_i$ finden lässt.

Einen Sonderfall stellen exakt-transversale logarithmische Signaturen dar, denn hier ist jeder Block eine Untergruppe und damit automatisch auch periodisch. Die Suche nach dem periodischen Element besteht also darin ein beliebiges nicht-neutrales Element zu wählen. Damit ist der Algorithmus in diesem Fall tatsächlich zahm.

Bevor wir uns den weiteren Anwendungsmöglichkeiten widmen soll noch angemerkt werden, dass man problemlos und schnell Signaturen entwerfen kann, die keine ATLS, aber dennoch periodisch sind. In der Tabelle 5.5 sieht man ein gutes Beispiel, wieso hier der Faktorisierungsalgorithmus nicht funktionieren kann: Eine Spaltennormalisierung hätte keinen Effekt auf die nicht-periodischen Blöcke und so bricht der Algorithmus in Ermangelung eines periodischen Elements nach wenigen Schritten ab.

| $\beta$ | |
|---|---|
| 000 | 000000 |
| 001 | 000000 |
| 010 | 000000 |
| 100 | 000000 |
| 011 | 000000 |
| 110 | 000000 |
| 101 | 000000 |
| 111 | 000000 |
| 000 | 000000 |
| 000 | 100000 |
| 000 | 010000 |
| 000 | 110000 |
| 000 | 000001 |
| 000 | 001001 |
| 000 | 000101 |
| 000 | 001101 |
| 000 | 000000 |
| 000 | 101000 |
| 000 | 010100 |
| 000 | 111100 |
| 000 | 000010 |
| 000 | 100110 |
| 000 | 111010 |
| 000 | 011110 |

Tabelle 5.5.: Im ersten Block periodische Nicht-ATLS

### 5.2.2. Weitere Anwendungsmöglichkeiten

Eine weitere Anwendung könnte der Algorithmus bei der Suche in großen Datenmengen finden. Wählt man die komplette Gruppe als eine logarithmische Signatur vom Typ ($|G|$), so ist die Periodizität trivialerweise immer gegeben. Wir können also ein beliebiges nichtneutrales Element wählen und können sofort eine Spaltennormalisierung durchführen, welche den Block (also die gesamte Signatur) entsprechend verkleinert. Bei einer entsprechenden 2-Gruppe hätten wir jetzt nur noch halb so viele Elemente mit den üblichen Methoden (z.B. Hashtabellen) zu durchsuchen und müssten am Ende beim Rückschritt nur 2 mögliche Urbilder prüfen. Natürlich lohnt dies nur, wenn man mehrmals in der Gruppe suchen muss, da sonst ein normaler Suchlauf schneller ist.

Kommt man auf die Ideen, den Algorithmus rückwärts auszuführen, so fällt auf, dass man die ursprünglich periodischen Elemente $z \in G$ nun so wählen muss, dass sie eine

Art Basis ergeben, bei der man lediglich angeben muss, welches Element welchen Block erweitern soll (quasi eine „Re-Spaltennormalisierung") um eine eindeutige logarithmische Signatur daraus zu erhalten. Betrachten wir zur Verdeutlichung ein einfaches Beispiel.

| $\beta_0$ | $\longrightarrow$ | $\beta_1$ | $\longrightarrow$ | $\beta_2$ |
|---|---|---|---|---|
| 0 0 0 0 | | 0 0 0 0 | | 0 0 0 0 |
| | $\oplus z_{11}$ | 0 1 0 1 | | 0 1 0 1 |
| | | | $\oplus z_{12}$ | 1 0 0 1 |
| | | | $\oplus z_{12}$ | 1 1 0 1 |
| 0 0 0 0 | | 0 0 0 0 | | 0 0 0 0 |
| | | | | |
| | | | | |
| | | | | |

Tabelle 5.6.: Erzeugung von $\beta$ über eine Basis (Teil I)

Wir wollen eine log. Sig. zu einer Gruppe mit $2^4$ Elementen erzeugen und wählen uns als Basis

$$\begin{aligned} Z &= \{z_{11}, z_{12}, z_{21}, z_{22}\} \\ &= \{0101, 1000, 0010, 0001\} \end{aligned}$$

und zeigen in den Indizes an, auf welchen Block das jeweilige Element angewendet wird. In der Tabelle 5.6 sieht man, dass wir von einem trivialen Cover ausgehen und die Blockelemente immer einmal abschreiben und zusätzlich noch mit $z_{11}$ bzw. $z_{12}$ addieren. Somit ist der erste Block nach zwei Schritten fertig. Analog fahren wir mit dem zweiten Block fort.

| $\beta_2$ | $\longrightarrow$ | $\beta_3$ | $\longrightarrow$ | $\beta$ |
|---|---|---|---|---|
| 0 0 0 0 | | 0 0 0 0 | | 0 0 0 0 |
| 0 1 0 1 | | 0 1 0 1 | | 0 1 0 1 |
| 1 0 0 1 | | 1 0 0 1 | | 1 0 0 1 |
| 1 1 0 1 | | 1 1 0 1 | | 1 1 0 1 |
| 0 0 0 0 | | 0 0 0 0 | | 0 0 0 0 |
| | $\oplus z_{21}$ | 0 0 1 0 | | 0 0 1 0 |
| | | | $\oplus z_{22}$ | 0 0 0 1 |
| | | | $\oplus z_{22}$ | 0 0 1 1 |

Tabelle 5.7.: Erzeugung von $\beta$ über eine Basis (Teil II)

Man kann schnell nachrechnen, dass $\beta$ tatsächlich eine logarithmische Signatur ist. Offensichtlich ist die Reihenfolge der Elemente in der Basis entscheidend und ebenso auch die lineare Unabhängigkeit. Eine andere Reihenfolge bringt uns eine andere Signatur (evtl. nur permutiert) und linear abhängige $z_{ij}$ erzeugen lediglich ein Cover, da durch die gegenseitige Darstellbarkeit doppelte Elemente auftreten.

## 5.3. Der Angriff

Blackburn et al. bemerken, dass obgleich der private Schlüssel aus der zahmen logarithmischen Signatur $\beta$ und den zufälligen $t_i$, $i = 0...s$ besteht, so werden $t_1, ..., t_{s-1}$ eigentlich nicht gebraucht, denn nur $\beta, t_0, t_s$ sind in der Entschlüsselung entscheidend. Diese Überlegung wird in allen folgenden Betrachtungen miteinbezogen.

### 5.3.1. Eine Vereinfachung von $MST_3$

Blackburn et al. versuchen im Laufe der Arbeit [3] das Kryptanalysisproblem von $MST_3$ zu vereinfachen, in dem sie zeigen wollen, dass es ausreicht eine deutlich kleinere Menge von Schlüsseln zu betrachten, als in der originalen Definition angegeben sind, wobei die grundlegende Gruppenauswahl nun nicht mehr auf Suzuki-2-Gruppen beschränkt ist.
Sei $(\alpha, \gamma)$ der öffentliche und $(\beta, (t_0, ..., t_s))$ der private Schlüssel von $MST_3$. Wir erinnern uns, dass

$$\gamma_{ij} = \beta_{ij} t_{i-1}^{-1} \alpha_{ij} t_i \tag{5.3.1}$$

gilt und definieren uns für $i = 1..s$:

$$\begin{aligned} p_i &:= \prod_{k=1}^{i} \alpha_{k1} \\ q_i &:= \prod_{k=1}^{i} \gamma_{k1} \\ z_i &:= \prod_{k=1}^{i} \beta_{k1} \end{aligned}$$

und $p_0 = q_0 = z_0 = 1$. Da $\beta_{ij} \in Z(G)$ (und natürlich auch $z_i \in Z(G)$) können wir mit 5.3.1 folgern:

$$\begin{aligned} q_i &= \prod_{k=1}^{i} \gamma_{k1} \\ &= \prod_{k=1}^{i} \beta_{k1} t_{k-1}^{-1} \alpha_{k1} t_k \\ &= \prod_{k=1}^{i} t_{k-1}^{-1} \beta_{k1} \alpha_{k1} t_k \\ &\overset{t_k t_k^{-1}=1}{=} t_0^{-1} (\prod_{k=1}^{i} \beta_{k1} \alpha_{k1}) t_i \\ &= t_0^{-1} z_i p_i t_i \end{aligned}$$

Insgesamt haben wir also (stets für $i = 1..s$)

$$q_i = z_i t_0^{-1} p_i t_i \tag{5.3.2}$$

und definieren weiterhin in üblicher Notation

$$\begin{aligned}\alpha^* &= [p_0 A_1 p_1^{-1}, ..., p_{s-1} A_s p_s^{-1}] \\ \gamma^* &= [p_0 H_1 p_1^{-1}, ..., p_{s-1} H_s p_s^{-1}] \\ \beta^* &= [p_0 B_1 p_1^{-1}, ..., p_{s-1} B_s p_s^{-1}]\end{aligned}$$

und können nun folgendes Lemma beweisen.

**Lemma 5.2.** *Mit der geraden eingeführten Notation gilt für $i = 1..s$: $\alpha^*_{i1} = \beta^*_{i1} = \gamma^*_{i1} = 1$. Weiterhin gilt*

$$\begin{aligned}\breve{\alpha}^*(x) &= \breve{\alpha}(x) p_s^{-1} \\ \breve{\gamma}^*(x) &= \breve{\gamma}(x) q_s^{-1} \\ \breve{\beta}^*(x) &= \breve{\beta}(x) z_s^{-1}\end{aligned}$$

*und insbesondere ist $\beta^*$ eine logarithmische Signatur für $Z(G)$ und $\alpha^*$ ist ein Cover für eine Teilmenge $J^*$ von $G$.*

Hierzu wird in [3] nur gesagt, dass der Beweis einfach sei. Wir wollen ihn aber an dieser Stelle konkret angeben.

*Beweis.* Wir rechnen die Behauptung für $\alpha^*_{i1}$ nach und erkennen die Analogie zu $\beta^*_{i1}$ und $\gamma^*_{i1}$(für $i = 1..s$).

$$\begin{aligned}\alpha^*_{i1} &= p_{i-1} \alpha_{i1} p_i^{-1} \\ &= \prod_{k=1}^{i-1} \alpha_{k1} \cdot \alpha_{i1} \cdot (\prod_{k=1}^{i} \alpha_{k1})^{-1} \\ &= (\prod_{k=1}^{i} \alpha_{k1})(\prod_{k=1}^{i} \alpha_{k1})^{-1} \\ &= 1\end{aligned}$$

Weiterhin sehen wir

$$\begin{aligned}\breve{\alpha}^*(x) &= \alpha^*_{1j_1} \cdot ... \cdot \alpha^*_{sj_s} \\ &= p_0 \alpha_{1j_1} p_1^{-1} \cdot ... \cdot p_{s-1} \alpha_{sj_s} p_s^{-1} \\ &\overset{p_0=1}{=} \alpha_{1j_1} \cdot ... \cdot \alpha_{sj_s} p_s^{-1} \\ &= \breve{\alpha}(x) p_s^{-1}\end{aligned}$$

und erkennen, dass es analog für $\breve{\beta}^*(x)$ und $\breve{\gamma}^*(x)$ verläuft. Die letzte Bemerkung im Lemma haben wir bereits im Abschnitt zu den ATLS bewiesen. □

Wir benötigen für die Hauptaussage dieses Abschnittes noch ein weiteres Lemma.

**Lemma 5.3.** *Sei $(\alpha, \gamma)$ ein öffentlicher und $(\beta, (t_0, ..., t_s))$ ein entsprechend privater Schlüssel zu $MST_3$. Mit obiger Notation und $t_0^* = ... = t_s^* = t_0$ gilt, dass $(\alpha^*, \gamma^*)$ ein öffentlicher und $(\beta^*, (t_0^*, ..., t_s^*))$ ein zugehöriger privater Schlüssel zu $MST_3$ ist.*

Dieses Lemma wurde in der Arbeit von Blackburn et al. ausführlich bewiesen. Wir wollen der Vollständigkeit halber die Erklärungen verfolgen.

*Beweis.* Wir nehmen an, dass wir $\alpha^*, \beta^*$ und $t_0^*, ..., t_s^*$ genutzt haben um einem öffentlichen Schlüssel $(\alpha^*, \delta)$ mit $\delta = [D_1, ..., D_s]$ und

$$\delta_{ij} = \beta_{ij}^* (t_{i-1}^*)^{-1} \alpha_{ij}^* t_i^*$$

zu erzeugen. Es genügt zu zeigen, dass $\delta = \gamma^*$ gilt. Zunächst sehen wir, dass

$$\begin{aligned} \delta_{ij} &= \beta_{ij}^* t_0^{-1} \alpha_{ij}^* t_0 \\ &= z_{i-1} \beta_{ij} z_i^{-1} t_0^{-1} p_{i-1} \alpha_{ij} p_i^{-1} t_0 \\ &\overset{\beta_{ij} \in Z(G)}{=} \beta_{ij} z_{i-1} z_i^{-1} t_0^{-1} p_{i-1} \alpha_{ij} p_i^{-1} t_0 \end{aligned}$$

und da wir bereits aus dem Lemma zuvor wissen, dass

$$1 = \beta_{i1}^* = z_{i-1} \beta_{i1} z_i^{-1} \tag{5.3.3}$$

gilt nun insgesamt

$$\delta_{ij} = \beta_{ij} \beta_{i1}^{-1} t_0^{-1} p_{i-1} \alpha_{ij} p_i^{-1} t_0$$

Weiterhin folgern wir aus 5.3.2 und 5.3.3, dass

$$\begin{aligned} t_0^{-1} p_{i-1} &= z_{i-1}^{-1} q_{i-1} t_{i-1}^{-1} \\ p_i^{-1} t_0 &= t_i q_i^{-1} z_i \\ \beta_{i1}^{-1} z_{i-1}^{-1} &= z_i^{-1} \end{aligned}$$

gelten. Da $z_i \in Z(G)$ folgt nun

$$\begin{aligned} \delta_{ij} &= \beta_{ij} \beta_{i1}^{-1} z_{i-1}^{-1} q_{i-1} t_{i-1}^{-1} \alpha_{ij} t_i q_i^{-1} z_i \\ &= \beta_{ij} q_{i-1} t_{i-1}^{-1} \alpha_{ij} t_i q_i^{-1} \\ &= q_{i-1} \beta_{ij} t_{i-1}^{-1} \alpha_{ij} t_i q_i^{-1} \\ &= q_{i-1} \gamma_{ij} q_i \\ &= \gamma_{ij}^* \end{aligned}$$

Damit ist die Behauptung bewiesen. □

**Definition 5.3.** Wir definieren das *eingeschränkte OneWayEncryption-Problem* für $MST_3$ folgendermaßen: Gegeben ist ein öffentlichter Schlüssel $(\alpha, \gamma)$ und ein Chiffretext $(y_1, y_2)$. Es muss zudem, wie oben, für alle $i = 1..s$ nun $\alpha_{i1} = \beta_{i1} = \gamma_{i1} = 1$ und $t_0 = t_1 = ... = t_s$ gelten. Gesucht ist der Klartext $p$ zu $(y_1, y_2)$.

Das folgende Theorem besagt, dass wir uns stets auf den eingeschränkten Fall zurück ziehen können und beweist damit die Vereinfachung. Wir werden an dieser Stelle den Beweis von Blackburn et al. aus [3] weiter ausformulieren, aber die Grundstruktur beibehalten.

**Theorem 5.2.** *Es gibt eine polynomielle Reduktion vom OWE-Problem von $MST_3$ (für beliebige Schlüssel) zum eingeschränkten OWE-Problem.*

*Beweis.* Sei $O(\alpha, \gamma, y_1, y_2)$ ein Orakel für das eingeschränkte OWE-Problem für $MST_3$. Wir zeigen nun, dass man dieses Orakel auch für das allgemeine OWE-Problem nutzen kann.
Sei $(\alpha, \gamma)$ ein (uneingeschränkter) öffentlicher Schlüssel mit zugehörigem privaten Schlüssel $(\beta, t_0, ..., t_s)$. Sei $(y_1, y_2)$ ein Chiffretext bezüglich der Nachricht $p$. Es sind also $(\alpha, \gamma)$ und $(y_1, y_2)$ gegeben. Wir definieren $(\alpha^*, \gamma^*)$ wie bereits zuvor und bemerken, dass dies effektiv aus $(\alpha, \gamma)$ gewonnen werden kann (siehe oben).
Durch die vorherigen Lemmata ist bekannt, dass $(\alpha^*, \gamma^*)$ ein öffentlicher Schlüssel bezüglich des privaten Schlüssels $(\beta^*, (t_0, ..., t_0))$ ist und die Einschränkungen erfüllt. Daher definieren wir nun

$$\begin{aligned} y_1^* &= y_1 p_s^{-1} \\ y_2^* &= y_2 q_s^{-1} \end{aligned}$$

und bemerken auch hier, dass $p_s$ und $q_s$ nur aus den öffentlichen Informationen stammen und somit sind auch $y_1^*$ und $y_2^*$ effektiv berechenbar.
Wir befragen das Orakel $O$ nach $(\alpha^*, \gamma^*, y_1^*, y_2^*)$ und erhalten eine Nachricht $p$, so dass $(\alpha^*(p), \gamma^*(p)) = (y_1^*, y_2^*)$. Damit ist $p$ die gesuchte Nachricht, denn

$$\begin{aligned} \breve{\alpha}(p) = \breve{\alpha}^*(p) p_s = y_1^* p_s = y_1 p_s^{-1} p_s = y_1 \\ \breve{\gamma}(p) = \breve{\gamma}^*(p) q_s = y_2^* q_s = y_2 q_s^{-1} q_s = y_2 \end{aligned}$$

und wir haben den Satz bewiesen. □

### 5.3.2. Ein neuer Angriff

Im Folgenden sei $G/Z(G)$ abelsch (z.b. Suzuki-2-Gruppen) und $t^*$ eine Vermutung für den gesuchten Wert $t$ (bzw. der Nebenklasse).
Für die weitere Betrachtungen definieren wir, entsprechend zu $MST_3$, für alle $i, j$

$$\zeta_{ij} := \gamma_{ij} (t^*)^{-1} \alpha_{ij}^{-1} t^*$$

und sehen, dass nur der öffentliche Schlüssel benötigt wird, um $\zeta_{ij}$ zu bestimmen. Entsprechend erhalten wir also mit $\zeta = [D_1, ..., D_s]$ ein Cover für eine Teilmenge $J \subset G$ und setzen $\omega(x) = \breve{\gamma}(x)(t^*)^{-1} \breve{\alpha}(x) t^*$. Analog zu allen vorherigen Betrachtungen ergibt sich also wenn $t^*$ korrekt ist $\breve{\zeta} = \omega$ und es ist klar, dass $\zeta = \beta$ und $\breve{\zeta} = \breve{\beta}$. Insgesamt können wir also folgendes Lemma aufstellen:

**Lemma 5.4.** *Wenn die genannten Bedingungen erfüllt sind, so ist* $(\zeta, (t^*, ..., t^*))$ *ein äquivalenter privater Schlüssel für* $MST_3$.

*Beweis.* In [3] wird argumentiert, dass aus den genannten Bedingungen zunächst folgt, dass $\zeta$ eine zahme logarithmische Signatur für $Z(G)$ ist und daher ist $(\zeta, (t^*, ..., t^*))$ ein möglicher privater Schlüssel. Sei also $(y_1, y_2)$ der Chiffretext zum Klartext $p$ unter dem zu $(\beta, (t, ..., t))$ gehörigen öffentlichen Schlüssel. Also ist

$$\begin{aligned} y_1 &= \breve{\alpha}(p) \\ y_2 &= \breve{\gamma}(p) \end{aligned}$$

und wir benutzen $(\zeta, (t^*, ..., t^*))$ zum entschlüsseln:

$$\begin{aligned} \breve{\zeta}^{-1}(y_2(t^*)^{-1}y_1^{-1}t^*) &\overset{\breve{\zeta}=\omega}{=} \breve{\zeta}^{-1}(\breve{\gamma}(p)(t^*)^{-1}(\breve{\alpha}(p))^{-1}t^*) \\ &= \breve{\zeta}^{-1}(\breve{\zeta}(p)) \\ &= p \end{aligned}$$

Dies war zu zeigen. □

Die Autoren fahren damit fort zwei weitere Lemma zu beweisen, die sie für einen abschließenden Satz benötigen. Wir verfolgen nun zunächst ihre Erklärungen.

**Lemma 5.5.** *Ist* $G/Z(G)$ *abelsch, so ist* $\zeta$ *für jede beliebige Wahl von* $t^*$ *ein Cover zu einer Teilmenge von* $Z(G)$.

*Beweis.* Für alle $i, j$ gilt:

$$\begin{aligned} \zeta_{ij}Z(G) &= \gamma_{ij}(t^*)^{-1}\alpha_{ij}^{-1}t^*Z(G) \\ &\overset{G/Z(G)\,\text{abelsch}}{=} \gamma_{ij}\alpha_{ij}^{-1}(t^*)^{-1}t^*Z(G) \\ &= \gamma_{ij}\alpha_{ij}^{-1}Z(G) \\ &= \gamma_{ij}t^{-1}\alpha_{ij}^{-1}tZ(G) \\ &= \beta_{ij}Z(G) \\ &\overset{\beta_{ij}\in Z(G)}{=} Z(G) \end{aligned}$$

Daher folgt $\zeta_{ij} \in Z(G)$. □

**Lemma 5.6.** *Wenn* $G/Z(G)$ *abelsch ist, so gilt für jede beliebige Wahl von* $t^*$ *dass* $\omega = \breve{\zeta}$.

*Beweis.* Um Verwirrungen zu vermeiden sehen wir im Folgenden die Starträume von $\breve{\zeta}$ und $\omega$ als $\mathbb{Z}_{r_1} \times ... \times \mathbb{Z}_{r_s}$ anstatt $\mathbb{Z}_q$ an.
Zuerst bemerken wir, dass für alle $i = 1..s$ gilt

$$\begin{aligned} \zeta_{i1} &= \gamma_{i1}(t^*)^{-1}\alpha_{i1}^{-1}t^* \\ &= (t^*)^{-1}t^* \\ &= 1 \end{aligned}$$

Insbesondere ist

$$\breve{\zeta}(x_1,...,x_k,1,...,1) = \prod_{i=1}^{s} \zeta_{ix_i} = \prod_{i=1}^{k} \zeta_{ix_i}$$

und da $\alpha_{i1} = \gamma_{i1} = 1 \, \forall i = 1..s$ gilt analog

$$\begin{aligned} \breve{\alpha}(x_1,...,x_k,1,...,1) &= \prod_{i=1}^{k} \alpha_{ix_i} \\ \breve{\gamma}(x_1,...,x_k,1,...,1) &= \prod_{i=1}^{k} \gamma_{ix_i} \end{aligned}$$

Wir beweisen das Lemma per Induktion über $k$.
*Induktionsvoraussetzung:* Sei $P(k)$ die folgende Aussage:

$$\omega(x_1,...,x_s) = \breve{\zeta}(x_1,...,x_s) \text{ wenn } x_{k+1} = ... = x_s = 1$$

*Induktionsanfang:* Der Fall $k = 0$ ist klar, denn da stets $\zeta_{i1} = 1$ folgt $\breve{\zeta}(1,...,1) = 1$ und weiterhin $\breve{\alpha}(1,...,1) = \breve{\gamma}(1,...,1)$. Daher ist $\omega(1,...,1) = 1$ und $P(0)$ ist bewiesen.

*Induktionsschritt:* Angenommen $P(k-1)$ gilt. Da $G/Z(G)$ abelsch ist folgt, wie wir bereits gesehen haben, $\zeta_{ij} \in Z(G) \, \forall i,j$.
Seien $x_1,...,x_k$ fest, so definiere

$$\begin{aligned} x &:= (x_1,...,x_k,1,...,1) \\ x^* &:= (x_1,...,x_{k-1},1,...,1) \end{aligned}$$

dann gilt

$$\begin{aligned} \breve{\zeta}(x) &= \prod_{i=1}^{k} \zeta_{ix_i} \\ &= \breve{\zeta}(x^*)\zeta_{kx_k} \\ &\overset{\text{IV}}{=} \omega(x^*)\zeta_{kx_k} \\ &= \breve{\gamma}(x^*)(t^*)^{-1}\breve{\alpha}(x^*)^{-1}t^*\zeta_{kx_k} \\ &\overset{\zeta_{kx_k} \in Z(G)}{=} \breve{\gamma}(x^*)\zeta_{kx_k}(t^*)^{-1}\breve{\alpha}(x^*)^{-1}t^* \\ &= \breve{\gamma}(x^*)\gamma_{kx_k}(t^*)^{-1}\alpha_{kx_k}^{-1}t^*(t^*)^{-1}\breve{\alpha}(x^*)^{-1}t^* \\ &= (\prod_{i=1}^{k-1}\gamma_{ix_i})\gamma_{kx_k}(t^*)^{-1}\alpha_{kx_k}^{-1}(\prod_{i=1}^{k-1}\alpha_{ix_i})^{-1}t^* \\ &= \breve{\gamma}(x)(t^*)^{-1}\breve{\alpha}(x)^{-1}t^* \\ &= \omega(x) \end{aligned}$$

Damit haben wir aus $P(k-1)$ die Aussage $P(k)$ gefolgert und den Beweis abgeschlossen. □

Laut Blackburn et al. ist das folgende Theorem eine Konsequenz der letzten drei Lemmata. Wir betrachten zunächst die Aussage und begutachten dann den vorgeschlagenen Beweis.

**Theorem 5.3.** *Sei $G$ derart, dass $G/Z(G)$ abelsch ist, so gilt:*
*$(\zeta, (t^*, ..., t^*))$ ist genau dann ein äquivalenter privater Schlüssel für $MST_3$, wenn $\breve{\zeta} : \mathbb{Z}_{|Z(G)|} \to Z(G)$ eine Bijektion ist, deren Inverse effizient berechenbar ist.*

*Beweis.* Es ist klar, dass ein privater Schlüssel $(\zeta, (t^*, ..., t^*))$ zu $MST_3$ die Eigenschaft haben muss, dass $\breve{\zeta} : \mathbb{Z}_{|Z(G)|} \to Z(G)$ bijektiv und effizient invertierbar ist. Die andere Richtung beweist man folgendermaßen:
Wir bemerken, dass $\zeta$ laut Lemma 5.5. ein Cover zu einer Teilmenge von $Z(G)$ ist. Weiterhin behaupten wir, dass $\zeta$ eine Bijektion ist, deren Inverse effizient berechenbar ist, also dass $\zeta$ eine zahme logarithmische Signatur für $Z(G)$ ist. Mit Lemma 5.6. gilt $\omega = \breve{\zeta}$ und mit Lemma 5.4. sehen wir, dass $(\zeta, (t^*, ..., t^*))$ ein äquivalenter privater Schlüssel ist. □

Blackburn et al. beschreiben weiterhin einen angeblich generellen Ansatz um den privaten Schlüssel von $MST_3$ zu finden. Man solle die Tatsache nutzen, dass $\breve{\zeta}$ eine Bijektion sein muss um einige Eigenschaften von $t^*$ zu berechnen. Wenn dies funktioniert, so schränken diese Eigenschaften die Anzahl möglicher $t^*$ stark ein, woraufhin man nur noch die restlichen Möglichkeiten durchprobieren müsse. Wenn es dennoch viele mögliche $t^*$ gibt, so wähle man eines zufällig aus und hoffe, dass $\breve{\zeta}^{-1}$ effizient berechenbar ist.

Betrachten wir nun diese Ausführungen kritisch. Zunächst ist zu beachten, dass selbst wenn $\breve{\zeta}$ eine Bijektion ist, so muss es nicht die Gesuchte sein, wenn das $t^*$ bereits falsch gewählt war. Weiterhin ist noch viel entscheidender, dass es bei Blackburn et al. noch keinen effizienten Algorithmus gibt, der prüft, ob das gefundene $\zeta$ zahm ist, bzw. ob $\breve{\zeta}^{-1}$ effizient berechenbar ist. Selbst ein Kollisionstest mit Geburtstagsattacke ist bei großen Gruppen nur schwer oder gar nicht durchführbar. Wir werden im späteren Verlauf noch sehen, dass dieser Ansatz keine Hilfe bei der Kryptoanalyse ist, da man von vornherein bereits das korrekte $t^*$ gewählt haben muss und in diesem Fall könnte man auch schnellere Attacken verwenden.

### 5.3.3. Praktische Anwendung

Blackburn et al. verwendeten laut eigenen Angaben die mathematische Software SAGE [15] und die gerade vorgestellten Überlegungen bei ihren Experimenten. Sie setzten $m = 81$ und betrachteten $G$ als eine Suzuki-2-Gruppe über $\mathbb{F}_{2^m}$ und verwendeten den Quadrierungs-Automorphismus. Die logarithmische Signatur $\beta$ wurde über die genannte ATLS-Methode konstruiert und sollte stets $\beta_{i1} = 1$ für alle $i$ erfüllen. Die vorgestellten Überlegungen wurden laut den Beschreibungen in [3] durchgeführt, wobei $t^*$ immer zufällig gewählt wurde.

In den Experimenten wurden die Typen $(2, 2, ..., 2)$ und $(8, 64, 64, ..., 64)$, sowie zuletzt

$(2, 2, ..., 2, 16, 16, ..., 16)$ verwendet. Die Ergebnisse wurden teils in Tabellen veröffentlicht, wobei der erste Fall 10.000 mal auf einem Standard-PC zu jeweils zufälligen Initialisierungen von $MST_3$ geprüft wurde und lediglich wenige Minuten benötigte. Dabei wurden angeblich durchschnittlich lediglich ca. $3,47$ mal ein $t^*$ gewählt, bevor das System geknackt war. Ähnlich verhielt es sich, laut den Aussagen der Autoren, auch mit den anderen Fällen, wobei diese deutlich mehr Zeit benötigten und daher weniger oft geprüft wurden. Speziell im zweiten Fall wurde eine Angriffsmethode vorgestellt, der wir uns nun widmen wollen, da sie durchaus gefährlich erscheint.

Zunächst sei wie üblich die zahme logarithmische Signatur $\beta = [B_1, ..., B_{14}]$ zu $Z(G)$ (hier zum Typ $(8, 64, 64, ..., 64)$), $\alpha$ ein Cover zu einer Teilmenge $J \subseteq G \backslash Z(G)$ und $t \in G \backslash Z(G)$. Uns ist bereits bekannt, dass

$$\gamma_{ij} = \beta_{ij} t^{-1} \alpha_{ij} t \tag{5.3.4}$$

wobei wir uns in einer Suzuki-2-Gruppe (mit Automorphismus $\theta$) aufhalten und daher

$$\begin{aligned} t &= S(x, 0) \\ \beta_{ij} &= S(0, d_{ij}) \\ \alpha_{ij} &= S(e_{ij}, f_{ij}) \end{aligned}$$

setzen dürfen. Aus 5.3.4 folgern wir sofort

$$\begin{aligned} \gamma_{ij} &= S(0, d_{ij}) S(x, x^\theta x) S(e_{ij}, f_{ij}) S(x, 0) \\ &= S(e_{ij}, d_{ij} + f_{ij} + e_{ij} x^\theta + e_{ij}^\theta x) \\ &=: S(e_{ij}, g_{ij}) \end{aligned}$$

und sehen durch Umstellen der Formel, dass

$$d_{ij} = g_{ij} + f_{ij} + e_{ij}^\theta x + e_{ij} x^\theta \tag{5.3.5}$$

gilt. Ist $\beta$ periodisch, so gibt es für ein $i \in \{1..14\}$ ein $b \neq 0$ mit $B_i \cdot b = B_i$ und daher gibt es mindestens $|B_i| - 2$ Paare von Indices $(k, l)$ mit $d_{ij} + d_{ik} = d_{il}$ also

$$d_{ij} + d_{ik} + d_{il} = 0$$

für $j, k, l$ paarweise verschieden. Wir können also folgende drei Gleichungen (aus 5.3.5) nutzen

$$\begin{aligned} g_{ij} &= d_{ij} + f_{ij} + e_{ij} x^\theta + e_{ij}^\theta x \\ g_{ik} &= d_{ik} + f_{ik} + e_{ik} x^\theta + e_{ik}^\theta x \\ g_{il} &= d_{il} + f_{il} + e_{il} x^\theta + e_{il}^\theta x \end{aligned}$$

und erhalten durch Aufsummieren insgesamt

$$\underbrace{g_{ij} + g_{ik} + g_{il}}_{=:g} = \underbrace{f_{ij} + f_{ik} + f_{il}}_{=:f} + \underbrace{(e_{ij} + e_{ik} + e_{il})}_{=:e} x^\theta + \underbrace{(e_{ij}^\theta + e_{ik}^\theta + e_{il}^\theta)}_{=:e^\theta} x$$

also kurz gesagt

$$g + f = ex^{\theta} + e^{\theta}x \tag{5.3.6}$$

Da $\theta$ ein Automorphismus ist, folgt sofort, dass

$$f : \mathbb{F}_q \to \mathbb{F}_q \text{ mit } x \mapsto ex^{\theta} + e^{\theta}x$$

$\mathbb{F}_2$-linear ist. Ist $e \neq 0$ so muss also der Kern die Dimension 2 haben und insgesamt sehen wir, dass 5.3.6 höchstens zwei Lösungen für $x$ besitzen kann.

Auf Grund der Blocklängen kann es hier für $(i, j, k)$ nur weniger als $2^{18}$ Möglichkeiten geben, und da, wie gerade angesprochen, nur stets maximal zwei Lösungen für $x$ gefunden werden können, erhalten wir also höchstens $2^{19}$ Möglichkeiten die Formel 5.3.5 zu benutzen um ein mögliches $\beta'$ zu erzeugen, was laut Blackburn et al. auf Bijektivität (und natürlich korrekte Dechiffrierung) geprüft werden muss.

Das Fazit der Arbeit von Blackburn et al. kann man damit zusammenfassen, dass $MST_3$ als unsicher bewertet wurde. Wir widersprechen dieser Behauptung und zeigen im folgenden Abschnitt die Mängel des Angriffs auf, wobei wir natürlich nicht den Typ $(2, 2, ..., 2)$ betrachten werden, da hier alle nicht-neutralen Elemente eine Basis bilden und eine Faktorisierung daher unter den gegebenen Umständen trivial ist und keiner weiteren Analyse bedarf.

Wir wollen an dieser Stelle noch anmerken, dass die neue Version von $MST_3$ gegen diesen Angriff geschützt ist. Wir erinnern uns, dass wir hier einen Autormorphismus $\sigma \in \mathrm{Aut}(Z(G))$ und einen Homomorphismus

$$f : G \to Z(G) \text{ mit } g \mapsto S(0, g^{\sigma}_{.a})$$

vorliegen haben, so dass wir (im Kontext wie oben)

$$\begin{aligned} \gamma_{ij} &= \beta_{ij} t^{-1} \alpha_{ij} f(\alpha_{ij}) t \\ &= S(0, d_{ij}) S(x, x^{\theta}x) S(e_{ij}, f_{ij}) S(0, e^{\sigma}_{ij}) S(x, 0) \\ &= S(e_{ij}, d_{ij} + f_{ij} + e^{\sigma}_{ij} + x^{\theta}e_{ij} + e^{\theta}_{ij}x) \\ &= S(e_{ij}, g_{ij}) \end{aligned}$$

erhalten. Mit den gleichen Schritten wie oben (unter der Voraussetzung, dass $\beta$ periodisch ist), erhalten wir schließlich mit selbsterklärenden Kürzel:

$$g + f + e^{\sigma} = x^{\theta}e + e^{\theta}x$$

Die rechte Seite ist zwar weiterhin $\mathbb{F}_2$-linear, allerdings ist $\sigma$ immer noch eine geheime Unbekannte in der Gleichung. Ohne weiteres Wissen über $\sigma$ ist die Gleichung daher nicht nach $x$ auflösbar und der Angriff kann nicht durchgeführt werden.

## 5.4. Testergebnisse und Folgerungen

In der Arbeit von Blackburn et al. finden wir keine genaueren Hinweise zur Programmierung der Computertests, wie beispielsweise ob die Geburtstagsattacke verwendet wurde, und sind erstaunt, dass mit einem einfachen PC eine Gruppe der Größe $2^{81}$ in so kurzen Zeiten analysiert werden konnte (und der Arbeitsspeicher ausreichte). Die Ergebnisse erschienen bereits bei erster Betrachtung unglaubwürdig, weswegen eine eigene Programmierung der Tests unumgänglich war. Wir wollen in diesem Abschnitt diese Programmierung und die dahinter stehende Theorie erläutern, die Resultate der Versuche präsentieren und schließlich die Folgerungen aufzeigen.

### 5.4.1. Theorie und Programmierung

Im Folgenden verwenden wir eine Überlegung aus der Arbeit von Svaba [14]. Blackburn betrachtet stets den Fall, dass $t_0 = t_1 = ... = t_s = t$, daher verwenden auch wir diese Voraussetzung.
Wir brauchen eine zahme log. Sig. $\beta$ zu $Z(G)$ vom gleichen Typ wie ein Cover $\alpha$ zu $J \subseteq G \backslash Z(G)$ und erhalten

$$\gamma_{ij} = \beta_{ij} t^{-1} \alpha_{ij} t$$

Bei einem Angriff wird ein $t'$ geraten, wobei das System gebrochen ist, wenn es zufällig aus der gleichen Nebenklasse ist wie $t$. Sei also $u \notin tZ(G)$. Wir nehmen an $(\epsilon, (u, ..., u))$ ist ein äquivalenter Schlüssel. So folgt (da $\beta_{ij} \in Z(G)$)

$$\gamma_{ij} = t^{-1} \alpha_{ij} t \beta_{ij} \stackrel{!}{=} u^{-1} \alpha_{ij} u \epsilon_{ij}$$

und da es ein $w \in G$ geben muss, so dass $u = w^{-1} t$ folgt

$$t^{-1} \alpha_{ij} t \beta_{ij} \quad = \quad t^{-1} w \alpha_{ij} w^{-1} t \epsilon_{ij}$$

also insbesondere

$$\epsilon_{ij} = t^{-1} w \alpha_{ij}^{-1} w^{-1} \alpha_{ij} t \beta_{ij}$$

Da es sich bei $w\alpha_{ij}^{-1} w^{-1} \alpha_{ij}$ um einen Kommutator handelt und diese in Suzuki-2-Gruppen auch aus dem Zentrum sind, kürzen sich $t$ und $t^{-1}$ weg und wir sehen, da die Elementordnung im Zentrum gleich zwei ist,

$$\epsilon_{ij} = (\alpha_{ij}^{-1} w^{-1} \alpha_{ij} w) \beta_{ij} \in Z(G)$$

Unter Zuhilfenahme der Rechnenregeln erhalten wir schließlich

$$\epsilon_{ij} = S(0, \beta_{ij.b} + w_{.a} \alpha_{ij.a}^{\Theta} + \alpha_{ij.a} w_{.a}^{\Theta})$$

Wäre $u \in tZ(G)$, so wäre auch $w \in Z(G)$ und damit $w_{.a} = 0$, was zur Folge hätte, dass $\epsilon = \beta$. Ist aber $u \notin tZ(G)$, so muss $\epsilon$ im Allgemeinen keine logarithmische Signatur sein. Wäre $\epsilon$ eine Signatur, so wäre das Kryptosystem gebrochen. Wir haben also experimentell zu prüfen, ob die Wahl zufälliger $w_{.a}$ zu einer logarithmischen Signatur führen können. Dazu suchen wir mittels Geburtstagsattacke nach Kollisionen im Bildbereich von $\check{\epsilon}$.
Die Realisierung erfolgte über GAP [13], der Quellcode ist mit Kommentaren versehen und im Anhang C unter „signaturprüfung.g" zu finden.

### 5.4.2. Die Resultate

Der erste Test erfolgte zunächst mit $m = 24$ und dem Typ $(16, 16, 16, 16, 16, 16)$ unter dem Quadrieren-Automorphismus. Nachdem $\epsilon$ wie gefordert implementiert war, haben wir nach besagten Kollisionen gesucht. Bereits hier zeigte sich, dass die Ergebnisse von Blackburn anzuzweifeln sind, denn selbst bei nur 1000 zufälligen Wahlen wurde immer eine Kollision gefunden, wodurch jeweils eine Signatur ausgeschlossen wurde. Die Anzahl erzeugter Elemente, bevor die erste Kollision gefunden wurde, entspricht im Durchschnitt in etwa dem durch das Geburtstagsparadoxon erwarteten Wert von $1,17 \cdot \sqrt{2^{24}} \approx 4792$. Als nächstes betrachteten wir kleinere Gruppen, so dass wir in der Lage waren sämtliche Elemente der Gruppe anzeigen zu lassen. Wir wollten hier alle möglichen $w_{.a}$ durchgehen und sehen, wie oft nun tatsächlich eine Signatur erscheint (also keine Kollision) und wie weit die anderen Cover von der Bijektivität entfernt sind. Diesen Wert messen wir mit dem Bruch $\frac{|G|}{|J|}$ wobei also die Gruppengröße durch die Anzahl der erzeugten Elemente geteilt wird. Ist diese Zahl nahe bei 1, so handelt es sich (nahezu) um eine Signatur. Ist dieser Wert dagegen nahe bei $\frac{e}{e-1} \approx 1,58$, so wissen wir nach [10], dass es sich um ein zufälliges Cover handeln muss.

In der folgenden Tabelle findet man einige Resultate der Signaturtests, welche deutlich zeigen, dass stets zufällige Cover, aber nie Signaturen (außer $\beta$ selbst) gefunden werden. Dies widerlegt die Behauptungen und Ergebnisse von Blackburn et al. und zeigt, dass die Angriffe bereits bei solch kleinen Gruppen deutliche Schwierigkeiten bereiten. Insbesondere eine Gruppe der Größe $2^{81}$ kann also nicht auf die in [3] vorgestellte Weise kontrolliert werden. Es ist auch fraglich, wie Cover unter solch einer großen Gruppe auf Bijektivität geprüft werden sollen - insbesondere ohne Geburtstagsattacke - dies übersteigt den üblichen Arbeitsspeicher bei weitem. Man kann insgesamt sagen, dass man die eine richtige Nebenklasse bezüglich der gesuchten $t$ suchen muss und nicht irgendwelche anderen das System brechen können.

| **Typ, m** | **Signaturen** | $\mathbb{E}(X)$ **zu** $\frac{\lvert G\rvert}{\lvert J\rvert}$ | **Minimum** |
|---|---|---|---|
| [4,4,4,4], m=8 | Betha | 1,595 | 1,23 |
| | | | |
| [8,8,8], m=9 | Betha | 1,584 | 1,38 |
| | | | |
| [4,4,4,4,4], m=10 | Betha | 1,611 | 1,25 |
| | | | |
| [4,4,4,4,4,4], m=12 | Betha | 1,595 | 1,34 |
| | | | |
| [8,8,8,4,4], m=13 | Betha | 1,585 | 1,45 |

Table 5.8.: Erster Test

Eine ausführliche Tabelle zu den Resultaten findet sich im Anhang A.

Unser zweiter Test bestand darin in 100 (und mehr) Durchläufen, in denen jeweils eine Signatur $\beta$ und ein zufälliges $w$ gewählt wurden, unser $\epsilon$ auf Kollisionen zu prüfen und die Anzahl der bis zur ersten Kollision erzeugten Elemente zu zählen. In folgender Tabelle kann man einige der Resultate sehen und erkennen, dass mindestens eine Geburtstagsattacke nötig ist, um die Bijektivität zu prüfen. Die Testergebnisse von Blackburn sind also auch von diesem Standpunkt aus anzuzweifeln. Man bemerke, dass ein Test auf Bijektivität die komplette Abbildung untersuchen muss (was bei großen Gruppen unmöglich sein kann) und insgesamt sehen wir hier, dass es nicht einmal eine hohe Wahrscheinlichkeit für Bijektivität gibt, wenn man weniger als $1,17 \cdot 2^{\frac{m}{2}}$ Elemente prüft.

| **Typ, m, Tests** | $\mathbb{E}(X)$ | $\lvert\mathbb{E}(X) - (1,17 \cdot 2^{\frac{m}{2}})\rvert$ |
|---|---|---|
| **[4,4,4,4], m=8** | | *18,72=100%* |
| 100 | 20,25 | 1,53 (8%) |
| 200 | 20,26 | 1,54 (8%) |
| **[8,8,8], m=9** | | *26,47=100%* |
| 100 | 29,31 | 2,83 (10%) |
| 200 | 31,27 | 4,81 (18%) |
| **[4,4,4,4,4], m=10** | | *37,44=100%* |
| 100 | 41,33 | 3,89 (10%) |
| 200 | 41,02 | 3,58 (9%) |
| **[4,4,4,4,4,4], m=12** | | *74,88=100%* |
| 100 | 79,21 | 4,33 (5%) |
| 200 | 77,97 | 3,11 (4%) |
| **[8,8,8,8,8], m=15** | | *211,79=100%* |
| 100 | 223,71 | 11,93 (5%) |
| 200 | 212,48 | 0,72 (0%) |

Tabelle 5.9.: Zweiter Test

Auch hierzu findet man weitere Ergebnisse im Anhang A.
Man beachte den Erwartungswert, der sich immer mehr dem theoretischen Optimalwert, gegeben durch die Geburtstagsattacke, nähert. Die Differenz zum Optimalwert wurde auch prozentual angegeben, so dass die Annäherung bei größeren Gruppen deutlich zu sehen ist.

### 5.4.3. Schlussfolgerungen

Zunächst müssen wir feststellen, dass der letzt genannte Angriff von Blackburn et al. natürlich nur dann möglich ist, wenn $\beta$ periodisch ist. Wir haben bereits weiter oben ein simples Beispiel für eine nicht-periodische Signatur gegeben, die gegen eben diesen Angriff schon gewappnet ist. Außerdem sehen wir, dass die Autoren offensichtlich noch keinen Faktorisierungalgorithmus entworfen hatten, da sie immer wieder anmerken, dass die Bijektivität geprüft werden muss. Dies ist allerdings ein aufwendiger und oft unmöglicher

Arbeitsschritt, der schon bei den von Blackburn et al. verwendeten Gruppen nicht in den angegebenen Zeiten möglich ist.

Wir haben durch unsere Tests herausgefunden, dass es nur ein korrektes $t_0$ (bzgl. Nebenklassen) gibt und daher empfehlen wir den Bijektivitätstest durch den hier vorgestellten und in der Praxis sehr günstigen Faktorisierungsalgorithmus zu ersetzen. Dazu werden lediglich einige Nachricht-Chiffretext-Paare $(x, y)$ benötigt, die man aus den öffentlichen Informationen erzeugen kann. Statt also nun ein gefundenes $\beta'$ auf Bijektivität zu prüfen, testen wir ob die jeweilige Faktorisierung korrekt verläuft, so dass schließlich nur die richtige Signatur übrig bleibt.

## 5.5. Abschließende Worte

Wir haben uns in dieser Arbeit den Argumenten von Blackburn et al. gestellt und gezeigt, dass die Behauptung, dass $MST_3$ unsicher ist, anzuzweifeln ist. Die Schlußfolgerung der kritisierten Arbeit weisen wir entschieden ab und sehen $MST_3$ insgesamt als eine solide Basis für moderne Public-Key-Kryptosysteme an, die auch im Zeitalter von Quantencomputern noch bestehen können. Im Laufe dieser Diplomarbeit konnten wir zudem einen Faktorisierungsalgorithmus für ATLS präsentieren, welcher auch für zukünftige Arbeiten interessant sein könnte, sofern man beispielsweise andere Grundkörper oder Nicht-ATL-Signaturen untersuchen möchte und den Algorithmus entsprechend anpassen kann.
Die Forschung zu Covern und Signaturen ist noch lange nicht abgeschlossen und auch $MST_3$ konnte bereits weiter verbessert werden und in anderen Gebieten Anwendung finden, wie man unter Anderem in der Arbeit von Svaba [14] erfahren kann.

# A. Testergebnisse

Zu Tabelle A.1:
Es wurden in den 100 (200, 500) Durchläufen jeweils eine zufällige Signatur $\beta$ und ein zufälliges $w$ (siehe Programm) gewählt. Epsilon wurde dann auf Kollisionen geprüft. Die Anzahl der Elemente, die bis zur ersten Kollision erzeugt wurden, wurde jeweils gespeichert. Man beachte insbesondere den Erwartungswert, der sich immer mehr dem theoretisch optimalen Wert aus der Geburtstagsattacke nähert. Gleiche Tests mit jeweils festem Betha ergaben übrigens sehr ähnliche Resultate.

Zu Tabelle A.2:
Wir prüften zu festem $\beta$ aber zufälligen (oder allen möglichen) $w$ Epsilon auf Bijektivität. Dabei bestimmten wir, wieviele Elemente gecovert wurden (also $|J|$) und berechneten den Quotienten $\frac{|G|}{|J|}$. Man bemerke, dass $1,58$ auf zufällige Cover deutet, und beachte sowohl die niedrigen Varianzen, als auch die angegebenen Minima, welche deutlich zeigen, dass keines der Cover auch nur annäherend eine Signatur sein könnte.

| **Typ, m, Tests** | $\mathbb{E}(X)$ | $\lvert\mathbb{E}(X) - (1,17 \cdot 2^{\frac{m}{2}})\rvert$ | $\mathbb{V}(X)$ | $\sqrt{\mathbb{V}(X)}$ |
|---|---|---|---|---|
| | | | | |
| **[4,4,4], m=6** | | *9,36=100%* | | |
| 100 | 10,84 | 1,48 (15%) | 35,59 | 5,96 |
| 200 | 10,36 | 1,01 (10%) | 32,52 | 5,71 |
| 500 | 11,24 | 1,88 (20%) | 28,25 | 5,31 |
| | | | | |
| **[4,4,4,4], m=8** | | *18,72=100%* | | |
| 100 | 20,25 | 1,53 (8%) | 139,62 | 11,81 |
| 200 | 20,26 | 1,54 (8%) | 102,92 | 10,14 |
| 500 | 21,26 | 2,54 (13%) | 133,02 | 11,53 |
| | | | | |
| **[8,8,8], m=9** | | *26,47=100%* | | |
| 100 | 29,31 | 2,83 (10%) | 232,15 | 15,23 |
| 200 | 31,27 | 4,81 (18%) | 221,02 | 14,86 |
| 500 | 29,38 | 2,91 (10%) | 204,66 | 14,31 |
| | | | | |
| **[4,4,4,4,4], m=10** | | *37,44=100%* | | |
| 100 | 41,33 | 3,89 (10%) | 513,88 | 22,66 |
| 200 | 41,02 | 3,58 (9%) | 449,11 | 21,19 |
| 500 | 41,11 | 3,66 (9%) | 446,47 | 21,12 |
| | | | | |
| **[4,4,4,4,4,4], m=12** | | *74,88=100%* | | |
| 100 | 79,21 | 4,33 (5%) | 1705,67 | 41,29 |
| 200 | 77,97 | 3,11 (4%) | 1968,54 | 44,36 |
| 500 | 80,69 | 5,82 (7%) | 1960,99 | 44,28 |
| | | | | |
| **[8,8,8,4,4], m=13** | | *105,89=100%* | | |
| 100 | 108,34 | 2,45 (2%) | 3536,54 | 59,46 |
| 200 | 107,16 | 1,28 (1%) | 3285,27 | 57,31 |
| 500 | 112,28 | 6,41 (6%) | 3429,31 | 58,56 |
| | | | | |
| **[8,8,8,8,8], m=15** | | *211,79=100%* | | |
| 100 | 223,71 | 11,93 (5%) | 10620,41 | 103,05 |
| 200 | 212,48 | 0,72 (0%) | 13021,41 | 114,11 |
| 500 | 226,53 | 14,76 (6%) | 14335,71 | 119,73 |

Tabelle A.1.: Elementanzahl bis zur Kollision

| **Typ, m** | **Zufällige $w$** | **Signaturen** | $\mathbb{E}(X)$ | $\mathbb{V}(X)$ | **Minimum** |
|---|---|---|---|---|---|
| | | | | | |
| [4,4,4,4,4,4], m=12 | 100 | 0 | 1,587 | 0,011 | 1,36 |
| | | | | | |
| [4,4,4,4,4,4,8], m=15 | 100 | 0 | 1,589 | 0,004 | 1,48 |
| | | | | | |
| | | | | | |
| [4,4,4], m=6 | alle | Betha | 1,642 | 0,117 | 1,31 |
| | | | | | |
| [4,4,4,4], m=8 | alle | Betha | 1,595 | 0,043 | 1,23 |
| | | | | | |
| [8,8,8], m=9 | alle | Betha | 1,584 | 0,011 | 1,38 |
| | | | | | |
| [4,4,4,4,4], m=10 | alle | Betha | 1,611 | 0,041 | 1,25 |
| | | | | | |
| [8,8,8,4,4], m=13 | alle | Betha | 1,585 | 0,004 | 1,45 |
| | | | | | |
| | | | | | |
| [4,4,4,4,4,4], m=12 | | | | | |
| I | alle | Betha | 1,595 | 0,019 | 1,34 |
| II | alle | Betha | 1,592 | 0,02 | 1,34 |
| III | alle | Betha | 1,596 | 0,022 | 1,33 |
| IV | alle | Betha | 1,593 | 0,018 | 1,34 |
| V | alle | Betha | 1,596 | 0,025 | 1,34 |
| VI | alle | Betha | 1,592 | 0,019 | 1,34 |
| VII | alle | Betha | 1,596 | 0,021 | 1,35 |
| VIII | alle | Betha | 1,595 | 0,021 | 1,32 |
| IX | alle | Betha | 1,6 | 0,022 | 1,34 |
| X | alle | Betha | 1,594 | 0,021 | 1,34 |

Table A.2.: Signaturtests

# B. Faktorisierungsbeispiel

Mittels GAP [13] (siehe Anhang C, „betha.g") erzeugen wir eine Signatur $\beta$ mit Typ $(4,4,4,4)$ zu $|G| = 2^8$. Wir geben sie direkt fusioniert, permutiert und normalisiert an. Der gewählte Chiffretext $y$ und das periodische Element $z$ sind ebenfalls in der Tabelle B.1 zu finden. Wir führen nun nach und nach die Spaltennormalisierung durch (B.2-B.7), bis wir zu einer trivialen ATLS kommen und die Faktorisierung ablesen können. Wir können am Ende nach und nach zurück gehen (siehe Tabelle B.8). Zunächst sehen wir die Faktorisierung von $y^6$, da die zugehörige ATLS trivial geworden ist. Wir gehen nun folgendermaßen vor: Wurde ein Block nicht halbiert, so ändert sich die Position des zugehörigen Faktors natürlich auch nicht, daher merken wir sie uns. Den jeweils anderen Faktor erhalten wir, in dem wir einfach die möglichen Urbilder (also der Faktor selbst oder mit dem jeweiligen Periodenelement multipliziert) betrachten. Hierbei ergibt sich stets genau eine Lösung, da wir in ATLS arbeiten. Schließlich erhalten wir die gesuchte Faktorisierung.

| | | $\beta$ | | |
|---|---|---|---|---|
| $B_1$ | 00000000 | | $B_2$ | 00000000 |
| | 01101000 | | | 00110001 |
| | 00001111 | | | 10000000 |
| | 01110101 | | | 10100001 |
| | 01101001 | | | 10110011 |
| | 10011001 | | | 00000011 |
| | 00011100 | | | 10101001 |
| | 11111110 | | | 00011000 |
| | 10010110 | | | 10011011 |
| | 10001010 | | | 00110010 |
| | 00010011 | | | 10100010 |
| | 01110100 | | | 10000011 |
| | 11111111 | | | 00010010 |
| | 11100011 | | | 00111010 |
| | 10000101 | | | 10010001 |
| | 11100010 | | | 00100000 |
| $y$ | 10101100 | | | |

Tabelle B.1.: ATLS $\beta$

| | | $\beta^1$ | | |
|---|---|---|---|---|
| $B_1^1$ | 000**0**0000 | | $B_2^1$ | 000**0**0000 |
| | 011**0**1000 | | | 001**0**1101 |
| | 000**0**1111 | | | 100**0**0000 |
| | 011**0**1001 | | | 101**0**0001 |
| | 100**0**0101 | | | 101**0**1111 |
| | 111**0**0010 | | | 000**0**0011 |
| | 100**0**1010 | | | 101**0**1001 |
| | 111**0**0011 | | | 000**0**0100 |
| | | | | 100**0**0111 |
| | | | | 001**0**1110 |
| | | | | 101**0**0010 |
| | | | | 100**0**0011 |
| | | | | 000**0**1110 |
| | | | | 001**0**0110 |
| | | | | 100**0**1101 |
| | | | | 001**0**0000 |
| $y^1$ | | 101**0**1100 | | |
| $z$ aus $B_1$ | | 000**1**1100 | | |

Tabelle B.2.: Erste Normalisierung beendet

| | | $\beta^2$ | | |
|---|---|---|---|---|
| $B_1^2$ | **0**000**0**000 | | $B_2^2$ | **0**000**0**000 |
| | **0**11**0**1000 | | | **0**01**0**1101 |
| | **0**000**1**111 | | | **0**000**1**010 |
| | **0**11**0**1001 | | | **0**01**0**1011 |
| | | | | **0**01**0**0101 |
| | | | | **0**000**0**011 |
| | | | | **0**01**0**0011 |
| | | | | **0**000**0**100 |
| | | | | **0**000**1**101 |
| | | | | **0**01**0**1110 |
| | | | | **0**01**0**1000 |
| | | | | **0**000**1**001 |
| | | | | **0**000**1**110 |
| | | | | **0**01**0**0110 |
| | | | | **0**000**1**111 |
| | | | | **0**01**0**0000 |
| $y^2$ | | **0**01**0**0110 | | |
| $z^1$ aus $B_1^1$ | | **1**0001010 | | |

Tabelle B.3.: Zweite Normalisierung beendet

| | | $\beta^3$ | | |
|---|---|---|---|---|
| $B_1^3$ | **0**000000**0** | | $B_2^3$ | **0**000000**0** |
| | **0**11**0**100**0** | | | **0**01**0**11**0**1 |
| | **0**000**1**10**0** | | | **0**01**0**0**1**0**1** |
| | **0**11**0**10**0**1 | | | **0**0000**1**0**0** |
| | | | | **0**000**1**11**0**1 |
| | | | | **0**01**0**10**0**0 |
| | | | | **0**000**1**00**1** |
| | | | | **0**01**0**000**0** |
| $y^3$ | | **0**01**0**0**1**0**1** | | |
| $z^2$ aus $B_2^2$ | | 0000000**1**1 | | |

Tabelle B.4.: Dritte Normalisierung beendet

| | | $\beta^4$ | | |
|---|---|---|---|---|
| $B_1^4$ | 0**0000**00 | | $B_2^4$ | **0**0**00**0**0** |
| | **0**1**00**1**0**1 | | | **0**0**1**0**0**1**0**1 |
| | **0**0**0000**1 | | | **0**0**000**1**0**0 |
| | **0**1**00**1**0**0 | | | **0**0**1**0**0**0**0** |
| $y^4$ | | | **0**0**1**0**0**1**0**1 | |
| $z^3$ aus $B_2^3$ | | | 0000**1**101 | |

Tabelle B.5.: Vierte Normalisierung beendet

| | | $\beta^5$ | | |
|---|---|---|---|---|
| $B_1^5$ | 0**0000**00 | | $B_2^5$ | **0**0**00**0**0** |
| | **0**1**00**1**00** | | | **0**0**1**0**0**1**00** |
| | | | | **0**0**000**1**00** |
| | | | | **0**0**1**0**0**0**0** |
| $y^5$ | | | **0**0**1**0**0**1**00** | |
| $z^4$ aus $B_1^4$ | | | 0000000**1** | |

Tabelle B.6.: Fünfte Normalisierung beendet

| | | $\beta^6$ | | |
|---|---|---|---|---|
| $B_1^6$ | 0**0000000** | | $B_2^6$ | **0**0**00000** |
| | **0**1**00000** | | | **0**0**1**0**0000** |
| $y^6$ | | | **0**0**1**0**0000** | |
| $z^5$ aus $B_2^5$ | | | 0000**1**00 | |

Tabelle B.7.: Sechste Normalisierung beendet

| $y$ | Faktorisierung | Merke Position |
|---|---|---|
| $y^6$ | $\beta^6_{1,0} \cdot \beta^6_{2,1}$ | $(1,0)$ |
| $y^5$ | $\beta^5_{1,0} \cdot \beta^5_{2,1}$ | $(2,1)$ |
| $y^4$ | $\beta^4_{1,0} \cdot \beta^4_{2,1}$ | $(1,0)$ |
| $y^3$ | $\beta^3_{1,0} \cdot \beta^3_{2,2}$ | $(1,0)$ |
| $y^2$ | $\beta^2_{1,0} \cdot \beta^2_{2,13}$ | $(2,13)$ |
| $y^1$ | $\beta^1_{1,6} \cdot \beta^1_{2,13}$ | $(2,13)$ |
| $\mathbf{y = \beta_{1,8} \cdot \beta_{2,13}}$ | | |

Tabelle B.8.: Faktorensuche

# C. Quellcode

Alle Codes beziehen sich auf GAP [13].

**signaturprüfung.g**

```
F:=GF(2); x:=Indeterminate(F,"x"); poly:=1; m:=12;    #Globale Variabeln
alp:=[]; type:=[4,4,4,4,4,4]; typelength:=6;
bet:=[]; ordnung:=2^m; prim:=NextPrimeInt(2^(m-1));
eingabe:=Random([0..(2^m)-1]); gruppe:=[]; pliste:=[];
Print("\n— Programm bereit. —\n");

start:=function(typ, q)    # — START — Grundeinstellungen der Gruppe
local polynom;
m:=q; F:=GF(2^m); x:=Indeterminate(F,"x");
polynom:=DefiningPolynomial(F);
poly:=Value(polynom,x); ordnung:=2^m;
type:=typ; typelength:=Length(type);
alp:=alpha(); bet:=betha();
end;

input:=function(b)    # — INPUT — Werfe Bitstring der Länge m rein
local polynom, i;      #und erhalte entsprechendes Polynom in GF(2^m)
polynom:=0;
for i in [1..m] do
polynom:=polynom+(b[i]*x^(m-i));
od;
return polynom;
end;

output:=function(p)    # — OUTPUT — Werfe Polynom unter GF(2^m) rein
local bit, help, degree, i, v, h;  #und erhalte passenden Bitstring der Länge m
bit:=[]; v:=[]; h:=[];
if p=1 #Fehlervermeidung wenn p=1 oder 0
then p:=x-x+1;
elif p=0
then p:=x-x+0;
fi;
degree:=DegreeIndeterminate(p,x);
help:=x^(m+1)+p;
```

```
bit:=PolynomialCoefficientsOfPolynomial(help,x);
for i in [1..m] do    #Wandle Z(2)^0 in 1 und 0*Z(2) in 0 um
if (bit[i]=bit[m+2]) then
bit[i]:=1;
elif (bit[i]=bit[m+1]) then
bit[i]:=0;
else
Print("\nERROR in Output!\n");
fi;
od;
Unbind(bit[m+2]); Unbind(bit[m+1]);
v:=bit;     #Sortierung
for i in [1..m] do
h[i]:=v[m+1-i];
od;
return h;
end;

quadrieren:=function(v)    # — QUADRIEREN — Quadrierung in GF(2^m)
local polynom, quad, out;  #Bits rein, Bits raus
out:=[]; polynom:=input(v);
quad:=(polynom*polynom) mod poly;
out:=output(quad);
return out;
end;

multiplizieren:=function(v,w) # — MULTIPLIZIEREN — in GF(2^m)
local polynom1, polynom2, mult, out;
out:=[]; polynom1:=input(v); polynom2:=input(w);
mult:=(polynom1*polynom2) mod poly;
out:=output(mult);
return out;
end;

alpha:=function()   # — ALPHA — Das Cover Alpha wird mittels zufälliger
local hilfsmat, alpha, mat, i, j, k, a;          #Matrizen in GF(2) generiert
mat:=[]; hilfsmat:=[]; alpha:=[];
for i in [1..typelength] do
alpha[i]:=[]; mat[i]:=[];
for j in [1..type[i]] do
alpha[i][j]:=[]; mat[i][j]:=[];
od;
od;
a:=1;
```

```
for i in [1..typelength] do #Erzeuge typelength-viele reguläre mxm-Matrizen über GF(2)
while a=1 do
hilfsmat:=RandomInvertibleMat(m);
hilfsmat:=(hilfsmat) mod 2;
if (Determinant(hilfsmat)<>0)
then a:=0;
fi;
od;
a:=1;
for j in [1..type[i]] do
mat[i][j]:=hilfsmat[j];
od;
hilfsmat:=[];
od;
for i in [1..typelength] do    #Jeweils die ersten "type[i]" Zeilen jeder Matrix
for j in [1..type[i]] do   #geben einen Block A_i für alpha (mit Vektoren der Dim. m)
for k in [1..m] do
alpha[i][j][k]:=mat[i][j][k];
od; od; od;
return alpha;
end;

betha:=function()    #— BETHA —
# Die log. Sig Betha zu Z(G) wird über reguläre mxm Matrix erzeugt
#Nehme dazu "typelength" mal (log_2 |B_i|) Zeilen und erzeuge die Blöcke B_i durch
#Linearkombination dieser je (log_2 |B_i|) Zeilenvektoren
local i, j, k, a, b,loga, mat, mathelp, betha, V, bethahelp, t;
b:=1; mat:=[]; bethahelp:=[]; betha:=[];
loga:=[]; mathelp:=[]; a:=0;
for i in [1..typelength] do
betha[i]:=[];
for j in [1..type[i]] do
betha[i][j]:=[];
od;
od;
while b=1 do
mat:=RandomInvertibleMat(m); #Erzeuge reguläre mxm-Matrix über GF(2)
mat:=mat mod 2;
if (Determinant(mat)<>0)
then b:=0;
fi;
od;
for i in [1..typelength] do    #Loga ist gefüllt mit log_2 |B_i|
loga[i]:=LogInt(type[i],2);
```

```
od;
for i in [1..typelength] do #Linearkombinationen erzeugen und Matrix füllen
for j in [1..loga[i]] do
mathelp[j]:=mat[j+a];
od;
a:=a+loga[i];
V:=VectorSpace(GF(2), (mathelp)*Z(2));
bethahelp[i]:=List(V);
od;
for i in [1..typelength] do
for j in [1..type[i]] do
for k in [1..m] do
betha[i][j][k]:=bethahelp[i][j][k];
od; od; od;
t:=betha[1][1][1]+betha[1][1][1];
for i in [1..typelength] do    #Wandle Z(2)^0 in 1 und 0*Z(2) in 0 um
for j in [1..type[i]] do
for k in [1..m] do
if (betha[i][j][k]=t)
then betha[i][j][k]:=0;
elif (betha[i][j][k]<>t)
then betha[i][j][k]:=1;
else Print("\nERROR in Betha!\n");
fi;
od; od; od;
return betha;
end;

epsilon:=function(b,w,a)     # — EPSILON —
local eps, i, j, wquad;
eps:=[]; wquad:=quadrieren(w);
for i in [1..typelength] do;
eps[i]:=[];
for j in [1..type[i]] do
eps[i][j]:=[];
eps[i][j]:=(b[i][j]+multiplizieren(wquad,a[i][j])+multiplizieren(w,quadrieren(a[i][j])))mod 2;
od;
od;
return eps;
end;

nextpoint:=function(p)     # — NEXTPOINT — x-Erzeugung für e(x)
local i,j;     #p meint hier x, Erzeugung in Reihenfolge
if p=[] then
```

```
for i in [1..typelength] do
p[i]:=1;
od;
return p;
fi;
i:=1;
while p[i]>=type[i] do
i:=i+1;
od;
if i>typelength then
return [];
fi;
for j in [1..(i-1)] do
p[j]:=1;
od;
p[i]:=p[i]+1;
return p;
end;

int2point:=function(x)    # — INT2POINT — Zufällige x-Erzeugung
local i, result;
result:=[];
for i in [1..(typelength-1)] do
result[i]:=(x mod type[i])+1;
x:=Int(x/type[i]);
od;
result[typelength]:=(x mod type[typelength])+1;
return result;
end;

breve:=function(e,p)   # — BREVE — entspricht der Breve-Funktion in MST3
local i, result;
result:=e[1][(p[1])];
for i in [2..typelength] do
result:=(result+e[i][(p[i])]) mod 2;
od;
return result;
end;

go:=function() #    — GO — Allgemeiner Startbefehl
local e,w,x,y,i,j,p,z,list, h,bin,s, counter;
p:=[]; list:=[]; counter:=1; h:=Int(m/2); s:=1;
for i in [0..h] do    #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i);
```

```
od;
for i in [1..s] do    #Erzeuge passende Tabelle
list[i]:=[];
od;
x:=Random([1..typelengthh]); y:=Random([1..type[x]]);
w:=alp[x][y];      #Erzeuge zufälliges w und davon abhängiges epsilon
e:=epsilon(bet,w,alp);
for i in [1..ordnung] do
eingabe:=(eingabe+prim) mod (ordnung);
p:=int2point(eingabe);
z:=breve(e,p);
bin:=1;     #Hashtabelle
for j in [1..h] do    #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1)));
od;
if z in list[bin] then
Print("\n Kollision!!!\n");
return counter;
fi;
Append(list[bin],[z]);
counter:=counter+1;
od;
return 0;
end;

testgo:=function()
# — TESTGO — wie go, aber ignoriert Kollisionen um die
#Größe der gecoverten Teilmenge zu bestimmen
local e,w,x,y,i,j,p,z,list,h,bin,s,counter,summe;
p:=[]; list:=[]; summe:=0; counter:=0; h:=Int(m/2); s:=1;
for i in [0..h] do   #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i);
od;
for i in [1..s] do #Erzeuge passende Tabelle
list[i]:=[];
od;
x:=Random([1..typelengthh]); y:=Random([1..type[x]]);
w:=alp[x][y]; #Erzeuge zufälliges w und davon abhängiges epsilon
e:=epsilon(bet,w,alp);
for i in [1..ordnung] do
eingabe:=(eingabe+prim) mod (ordnung);
p:=int2point(eingabe);
z:=breve(e,p);
bin:=1; #Hashtabelle
```

```
for j in [1..h] do #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1)));
od;
if z in list[bin] then
counter:=counter+1;
else
Append(list[bin],[z]);
fi;
od;
return (ordnung-counter); #Gebe zurück wieviele Elemente gecovert werden (|J|)
end;

test:=function(typ, n, testzahl) # — TEST — ein zusammengefasster Startbefehl
# Bestimme Erwartungswert und Varianz von |G|/|J| mittels testgo
# zu "testzahl"-vielen Versuchen
local i, sammler, erwartungswert, varianz, tester;
sammler:=[]; erwartungswert:=0; varianz:=0; tester:=1;
while tester<>0 do
start(typ,n); tester:=hilfsfunktion();
od;
for i in [1..testzahl] do
sammler[i]:=testgo();
sammler[i]:=Float(ordnung/sammler[i]);
od;
erwartungswert:=Float(Sum(sammler)/testzahl);
for i in [1..testzahl] do
varianz:=varianz+((sammler[i]-erwartungswert)^2);
od;
varianz:=Float(varianz/testzahl);
Print("\nErwartungswert: ",erwartungswert,"\nVarianz: ",varianz,"\n");
return sammler;
end;

kompletteGruppe:=function(typ, n) # — KOMPLETTEGRUPPE —
#verläuft wie test, durchläuft aber alle möglichen w
local i, list, sammler, erwartungswert, varianz, counter, speicher, tester;
sammler:=[]; erwartungswert:=0; varianz:=0; counter:=0; speicher:=[]; tester:=1;
while tester<>0 do
start(typ,n); tester:=hilfsfunktion();
od;
list:=werzeugung();
list:=Set(list); Print("\nZur Ueberpruefung: Listelaenge ist ",Length(list));
if Length(list)<>ordnung then return "Error in kompletteGruppe!!!"; fi;
for i in [1..(ordnung)] do
```

```
sammler[i]:=wtester(list[i]);
sammler[i]:=Float(ordnung/sammler[i]);
if i=(2^(m-1)) then Print("\n50 Prozent erledigt!\n"); fi;
if i=(2^(m-2)) then Print("\n25 Prozent erledigt!\n"); fi;
if i=(2^(m-2)*3) then Print("\n75 Prozent erledigt!\n"); fi;
od;
erwartungswert:=Float(Sum(sammler)/(ordnung));
for i in [1..(ordnung)] do
varianz:=varianz+((sammler[i]-erwartungswert)^2);
od;
varianz:=Float(varianz/(ordnung));
Print("\nErwartungswert: ",erwartungswert,"\nVarianz: ",varianz,"\n");
return sammler;
end;

werzeugung:=function() #speziell für kompletteGruppe
#erzeuge mittels betha die ganze #Gruppe um w daraus wählen zu können
#Verwende dafür im Grunde die go-Funktion
local e,x,y,i,j,p,z,list;
p:=[]; list:=[];
for i in [1..(2^m)] do
eingabe:=(eingabe+prim) mod (ordnung);
p:=int2point(eingabe);
z:=breve(bet,p);
if z in list then return 0;
else Append(list,[z]); fi;
od;
return list;
end;

hilfsfunktion:=function() #speziell für kompletteGruppe um den Start zu testen
local e,w,x,y,i,j,p,z,list, h,bin,s, counter; #ähnlich der go-Funktion aber mit w=0
p:=[]; list:=[]; counter:=1; h:=Int(m/2); s:=1;
for i in [0..h] do #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i);
od;
for i in [1..s] do #Erzeuge passende Tabelle
list[i]:=[];
od;
x:=Random([1..typelength]); y:=Random([1..type[x]]);
w:=alp[x][y]*0; #Erzeuge zufälliges w und davon abhängiges epsilon
e:=epsilon(bet,w,alp);
for i in [1..ordnung] do
eingabe:=(eingabe+prim) mod (ordnung);
```

```
p:=int2point(eingabe);
z:=breve(e,p);
bin:=1; #Hashtabelle
for j in [1..h] do #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1)));
od;
if z in list[bin] then
return counter;
fi;
Append(list[bin],[z]);
counter:=counter+1;
od;
return 0;
end;

wtester:=function(vektor) # speziell für kompletteGruppe
#wie go, aber ignoriert Kollisionen um die Größe der gecoverten
# Teilmenge zu bestimmen
local e,w,x,y,i,j,p,z,list, h,bin,s, counter, summe;
p:=[]; list:=[]; summe:=0; counter:=0; h:=Int(m/2); s:=1; w:=vektor;
for i in [0..h] do #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i);
od;
for i in [1..s] do #Erzeuge passende Tabelle
list[i]:=[];
od;
e:=epsilon(bet,w,alp);
for i in [1..(ordnung)] do
eingabe:=(eingabe+prim) mod (ordnung);
p:=int2point(eingabe);
z:=breve(e,p);
bin:=1; #Hashtabelle
for j in [1..h] do #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1)));
od;
if z in list[bin] then
counter:=counter+1;
else
Append(list[bin],[z]);
fi;
od;
return (ordnung-counter); #Gebe zurück wieviele Elemente gecovert werden (|J|)
end;
```

```
kollisionstest:=function(typ, n, testzahl) # — KOLLISIONSTEST —
#macht bestimmte Anzahl von Tests und zählt wann immer das erste Mal Kollision kam
local i, sammler, erwartungswert, varianz, tester, gebattack, p, l,t;
sammler:=[]; erwartungswert:=0; varianz:=0; tester:=1; gebattack:=0; p:=[]; pliste:=[];
while tester<>0 do
start(typ,n); tester:=hilfsfunktion();
od;
gruppe:=werzeugung();
if Length(gruppe)<ordnung then Print("\nFehler bei W-Erzeugung!\n"); fi;
l:=[];t:=[];
for i in [1..(ordnung)] do
p:=nextpoint(p); Append(l,p); Append(t,[l]); l:=[];
od;
pliste:=t; #gobale pliste enthält alle möglichen p für breve
for i in [1..testzahl] do
sammler[i]:=goforkollision();
od;
erwartungswert:=Float(Sum(sammler)/testzahl);
for i in [1..testzahl] do
varianz:=varianz+((sammler[i]-erwartungswert)^2);
od;
varianz:=Float(varianz/testzahl);
gebattack:=(erwartungswert-(Float(117/100)*(Float(14142/10000)^n)));
Print("\nErwartungswert: ",erwartungswert,"\nVarianz: ",varianz,"\nGebAttacke:",gebattack,"\n");
end;

goforkollision:=function() # speziell für kollisionstest
#wie go aber stets mit neuem betha und w
local e,w,x,i,j,p,z,list, h, bin, s, counter, y, errorliste;
p:=[]; list:=[]; y:=Random([1..Length(pliste)]); errorliste:=[];
counter:=1; h:=Int(m/2); s:=1;
for i in [0..h] do #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i);
od;
for i in [1..s] do #Erzeuge passende Tabelle
list[i]:=[];
od;
x:=Random([2..ordnung]);
w:=gruppe[x]; #Erzeuge zufälliges w und davon abhängiges epsilon
e:=epsilon(bet,w,alp);
y:=Random([1..(ordnung)]);
for i in [1..ordnung] do
while y in errorliste do #zufällige p für breve erzeugen
y:=Random([1..(ordnung)]);
```

```
od;
p:=pliste[y];
Add(errorliste,y);
z:=breve(e,p);
bin:=1; #Hashtabelle
for j in [1..h] do #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1)));
od;
if z in list[bin] then
return counter;
fi;
Append(list[bin],[z]);
counter:=counter+1;
od;
return 0;
end;
```

**betha.g**

```
F:=GF(2); x:=Indeterminate(F,"x"); poly:=1; m:=12;
typelength:=6; bet:=[]; ordnung:=2^m; prim:=NextPrimeInt(2^(m-1));
eingabe:=Random([0..(2^m)-1]); gruppe:=[]; pliste:=[];
Print("\n— Programm bereit. —\n");

start:=function(typ, q) # — START — Grundeinstellungen der Gruppe
local polynom; m:=q; F:=GF(2^m); x:=Indeterminate(F,"x");
polynom:=DefiningPolynomial(F); poly:=Value(polynom,x); ordnung:=2^m;
type:=typ; typelength:=Length(type); bet:=betha();
end;

input:=function(b) # — INPUT — Werfe Bitstring der Länge m rein
local polynom, i; #und erhalte entsprechendes Polynom in GF(2^m)
polynom:=0;
for i in [1..m] do polynom:=polynom+(b[i]*x^(m-i)); od;
return polynom; end;

output:=function(p) # — OUTPUT — Werfe Polynom unter GF(2^m) rein
local bit, help, degree, i, v, h; #und erhalte passenden Bitstring der Länge m
bit:=[]; v:=[]; h:=[];
if p=1 #Fehlervermeidung wenn p=1 oder 0
then p:=x-x+1; elif p=0 then p:=x-x+0; fi;
degree:=DegreeIndeterminate(p,x);
help:=x^(m+1)+p; bit:=PolynomialCoefficientsOfPolynomial(help,x);
for i in [1..m] do #Wandle Z(2)^0 in 1 und 0*Z(2) in 0 um
if (bit[i]=bit[m+2]) then bit[i]:=1;
```

```
elif (bit[i]=bit[m+1]) then bit[i]:=0;
else Print("\nERROR in Output!\n");
fi; od; Unbind(bit[m+2]); Unbind(bit[m+1]);
v:=bit; #Sortierung
for i in [1..m] do h[i]:=v[m+1-i]; od;
return h;
end;

quadrieren:=function(v) # — QUADRIEREN — Quadrierung in GF(2^m)
local polynom, quad, out; #Bits rein, Bits raus
out:=[]; polynom:=input(v);
quad:=(polynom*polynom) mod poly; out:=output(quad);
return out;
end;

multiplizieren:=function(v,w) # — MULTIPLIZIEREN — in GF(2^m), Bits rein/raus
local polynom1, polynom2, mult, out; out:=[];
polynom1:=input(v); polynom2:=input(w);
mult:=(polynom1*polynom2) mod poly; out:=output(mult);
return out;
end;

betha:=function() # — BETHA —
#Die log. Sig betha zu Z(G) wird über reguläre mxm Matrix erzeugt
#Nehme dazu "typelength" mal (log2vonB_i) Zeilen und erzeuge die Blöcke B_i durch
#Linearkombination dieser je (log2vonB_i) Zeilenvektoren
local i, j, k, l, a, b,loga, mat, mathelp, betha, V, bethahelp, t, sumlog;
b:=1; mat:=[]; bethahelp:=[]; betha:=[]; sumlog:=0; loga:=[]; mathelp:=[]; a:=0;
for i in [1..typelength] do betha[i]:=[]; for j in [1..type[i]] do betha[i][j]:=[]; od; od;
while b=1 do mat:=RandomInvertibleMat(m); #Erzeuge reguläre mxm-Matrix in GF(2)
mat:=mat mod 2; if (Determinant(mat)<>0) then b:=0; fi; od;
for i in [1..typelength] do #Loga ist gefüllt mit log2vonB_i
loga[i]:=LogInt(type[i],2); od;
for i in [1..typelength] do #Linearkombinationen erzeugen und Matrix füllen
for j in [1..loga[i]] do mathelp[j]:=mat[j+a]; od; a:=a+loga[i];
V:=VectorSpace(GF(2), (mathelp)*Z(2));
bethahelp[i]:=List(V); od;
for i in [1..typelength] do
for j in [1..type[i]] do
for k in [1..m] do
betha[i][j][k]:=bethahelp[i][j][k];
od; od; od;
t:=betha[1][1][1]+betha[1][1][1];
for i in [1..typelength] do # Wandle Z(2)^0 in 1 und 0*Z(2) in 0 um
```

```
for j in [1..type[i]] do
for k in [1..m] do
if (betha[i][j][k]=t) then betha[i][j][k]:=0;
elif (betha[i][j][k]<>t) then betha[i][j][k]:=1;
else Print("\nERROR in Betha!\n");
fi; od; od; od;
for i in [1..typelength] do
for j in [1..type[i]] do
for l in [1..sumlog] do #Randomisierung
if Random([0,1])=0 then
betha[i][j]:=(betha[i][j]+mat[l]) mod 2;
fi;od; od; sumlog:=sumlog+loga[i]; od;
return betha;
end;

int2point:=function(x) # — INT2POINT — Zufällige x-Erzeugung
local i, result; result:=[];
for i in [1..(typelength-1)] do
result[i]:=(x mod type[i])+1; x:=Int(x/type[i]); od;
result[typelength]:=(x mod type[typelength])+1;
return result;
end;

breve:=function(e,p) # — BREVE — entspricht der Breve-Funktion in MST3
local i, result;
result:=e[1][(p[1])];
for i in [2..typelength] do result:=(result+e[i][(p[i])]) mod 2; od;
return result;
end;

BNmodN:=function(block, b) #speziell für bethabeweis, berechne BN/N
local i, j, e, result, sortiert, ende;
result:=[]; e:=0; j:=1; sortiert:=[]; ende:=[];
while e=0 do #bestimme wo b die erste 1 hat
if b[j]=0 then j:=j+1; else e:=j; fi; od;
for i in [1..Length(block)] do #wenn e-te Stelle gleich 0, so übernehme
if block[i][e]=0 then #wenn e-te Stelle gleich 1, so rechne +b mod 2
result[i]:=block[i]; else result[i]:=(block[i]+b) mod 2; fi; od;
for i in [1..Length(block)] do #Streiche Null-Spalte
sortiert[i]:=[]; for j in [1..(e-1)] do sortiert[i][j]:=result[i][j]; od;
for j in [e..(Length(b)-1)] do sortiert[i][j]:=result[i][j+1]; od; od;
for i in [1..Length(block)] do #Streiche doppelte Zeilen
if sortiert[i] in ende then j:=1; else Add(ende,sortiert[i]); fi; od;
return ende;
```

```
end;

blocktest:=function(block) #Teste ob B*b=B (für bethabeweis)
local i,j, k, B, b, h; B:=[]; b:=[]; k:=1; i:=1; h:=0;
while i<=Length(block) do
k:=1; b:=block[i];
if b<>block[1]*0 then B:=[]; h:=0;
for j in [1..Length(block)] do B[j]:=(block[j]+b) mod 2; od;
while k<=Length(block) do if B[k] in block then k:=k+1;
else i:=i+1; k:=Length(block)+1; h:=1; fi; od; if h=0 then i:=i+1; fi;
if k>=Length(block) then return b; #Wenn b gefunden dann return b
fi;
else i:=i+1; fi;
od; return 0; #Wenn kein b gefunden dann return 0
end;

permutation:=function(block) #Permutation in einem Block (für bethabeweis)
local i,result,z, hilf, x; result:=[]; z:=[]; hilf:=[1..Length(block)];
for i in [1..Length(block)] do #erzeuge zufällige Permutation
x:=Random([1..Length(hilf)]); z[i]:=hilf[x]; Unbind(hilf[x]);
hilf:=Set(hilf); od;
for i in [1..Length(block)] do #permutiere mit z
result[i]:=block[z[i]]; od;
Print("\nVerwendete Permutation: ",z);
return result;
end;

fusion:=function(blocka, blockb) #Fusion von zwei Blöcken (für bethabeweis)
local i,j, la, lb, result;
la:=Length(blocka); lb:=Length(blockb); result:=[];
for i in [1..la] do for j in [1..lb] do
Add(result,(blocka[i]+blockb[j]) mod 2); od; od;
return result;
end;

darstellung:=function(signatur,name) #Zur Darstellung von Sig. (für bethabeweis)
local i,j;
Print("\n",name);
for i in [1..Length(signatur)] do Print("\n");
for j in [1..Length(signatur[i])] do Print(signatur[i][j],"\n");
od; od; Print("\n");
end;

hilfsfunktion:=function() #speziell um den Start zu testen
```

```
local e,i,j,p,z,list, h,bin,s, counter; #ähnlich der go-Funktion aber mit w=0
p:=[]; list:=[]; counter:=1; h:=Int(m/2); s:=1;
for i in [0..h] do #Höchst mögliche Binarzahl der Länge h bestimmt
s:=s+(2^i); od;
for i in [1..s] do #Erzeuge passende Tabelle
list[i]:=[]; od;
e:=bet;
for i in [1..ordnung] do
eingabe:=(eingabe+prim) mod (ordnung); p:=int2point(eingabe);
z:=breve(e,p); bin:=1; #Hashtabelle
for j in [1..h] do #Weise z eine Zahl zu (Binär zur ersten Hälfte von z)
bin:=bin+(z[j]*(2^(j-1))); od;
if z in list[bin] then return counter; fi;
Append(list[bin],[z]); counter:=counter+1; od;
return 0;
end;

bethabeweis:=function(typ, n) # — BETHABEWEIS — Startfunktion
local i,j,k, tester, betfus, half, betfusper, b, hilf, zahl, BNN, BNNew, counter;
tester:=1; betfus:=[]; betfusper:=[]; b:=[]; hilf:=[]; zahl:=0;
BNN:=[]; BNNew:=[]; counter:=[]; k:=2;
while tester<>0 do
start(typ,n); tester:=hilfsfunktion();
od;
if ((typelength mod 2)=1) then # Hier sind nur gerade Typlängen erlaubt
return "! ! ! Fehler: TypeLength ungerade ! ! !";
fi;
darstellung(bet,"—> BETHA:");
if (typelength mod 2 = 0) then # Fusionierung von möglichst weit entfernten Blöcken
half:=(typelength/2);
for i in [1..half] do
betfus[i]:=fusion(bet[i],bet[i+half]);
od; fi;
#darstellung(betfus,"—> BETHA FUSIONIERT:");
for i in [1..Length(betfus)] do # zufällige Permutationen jeweils innerhalb der Blöcke
betfusper[i]:=permutation(betfus[i]);
od;
darstellung(betfusper,"—> BETHA Fusioniert Und Permutiert:");
i:=1;
while (zahl=0 and i<=Length(betfus)) do # suche b mit Bb=B
hilf:=blocktest(betfusper[i]);
if hilf<>0 then b:=hilf;
zahl:=i; else i:=i+1;
fi; od;
```

```
if b=[] then return "Fehler bei B*b=B, kein b gefunden!!!"; fi;
Print("\nDer gesuchte Vektor bij ist ",b,". Er gehört zum Block B ",zahl,"\n");
for i in [1..Length(betfus)] do # bestimme BN/N und lösche Nullspalte
BNN[i]:=BNmodN(betfusper[i],b);
counter[i]:=(Length(BNN[i])-1); #zähle die neuen Blocklängen(-1)
od;
darstellung(BNN,"—> BN mod N:");
# An dieser Stelle hat BNN nur noch Länge m-1.
# Fahre analog mit Bb=B und BN/N fort.
while counter<>counter*0 do #durchlaufe so lange bis Blocklängen alle gleich 1
i:=1; b:=[]; hilf:=[]; zahl:=0; BNNew:=[];
while (zahl=0 and i<=Length(BNN)) do # suche b mit Bb=B
hilf:=blocktest(BNN[i]);
if hilf<>0 then b:=hilf; zahl:=i;
else i:=i+1; fi; od; if b=[] then
return "Fehler bei B*b=B, kein b gefunden!!!";
fi;
Print("\nDer gesuchte Vektor bij ist ",b,". Er gehört zum Block B ",zahl,"\n");
for i in [1..Length(BNN)] do
# bestimme BN/N und lösche Nullspalte
BNNew[i]:=BNmodN(BNN[i],b);
counter[i]:=(Length(BNNew[i])-1); #zähle die neuen Blocklängen(-1)
od;
Print("\nDurchlauf Nummer ",k,":");
darstellung(BNNew,"—> BN mod N:");
BNN:=BNNew; k:=k+1;
od;
end;
```

# Literaturverzeichnis


[1] W. Lempken, S.S. Magliveras, T. van Trung, W. Wei. A public key cryptosystem based on non-abelian finite groups. *J. Cryptology*, **22**(1), 62-74 (2009)

[2] S.S. Magliveras, D.R. Stinson, T van Trung. New approaches to designing public key cryptosystems using one-way functions and trap-doors in finite groups. *J. Cryptology*, **15**, 285-297 (2002)

[3] S.R. Blackburn, C. Cid, C. Mullan. Cryptanalysis of the $MST_3$ Public Key Cryptosystem. *J. Math. Crypt.*, **3**, 321-338 (2009)

[4] J. Buchmann, *Einführung in die Kryptographie*, 4. erweiterte Auflage, Springer-Verlag Berlin Heidelberg, 2008

[5] S.S. Magliveras, P. Svaba, T. van Trung, P. Zajac. On the security of a realization of cryptosystem $MST_3$. *Tatra Mt. Math.*, **41**, 1-13 (2008)

[6] G. Higman. Suzuki 2-groups. *Ill. J. Math.*, **7**, 79-96 (1963)

[7] S. Bosch, *Algebra*, 6. Auflage, Springer-Verlag Berlin Heidelberg, 2006

[8] H. Kurzweil und B. Stellmacher, *Theorie der endlichen Gruppen*: eine Einführung, Springer-Verlag Berlin Heidelberg, 1998

[9] F. Lorenz, *Einführung in die Algebra I*, 3. Auflage, Akademischer Verlag Spektrum Heidelberg Berlin, 1996

[10] P. Svaba, T. van Trung. On generation of random covers for finite groups. *Tatra Mt. Math.*, **37**, 105-112 (2007)

[11] William Feller, *An Introduction to Probability Theory and Its Appliations.* 1. Auflage, Verlag John Wiley & Sons, 1957

[12] Stasys Jukna, *Extremal Cobinatorics: With Applications in Computer Science*, 1. Auflage, Springer-Verlag Berlin, 2001

[13] GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra. Version 4.4.12. *http://www.gap-system.org*

[14] P. Svaba, *Covers and Logarithmic Signatures of Finite Groups in Cryptography*, Ph.D. thesis, Universität Duisburg-Essen Deutschland (2010)

[15] SAGE Mathematical Software, Version 3.4.1, *http://www.sagemath.org*

[16] P. Svaba, T. van Trung, Public key cryptosystem $MST_3$: cryptanalysis and realization, *J. Math. Cryptol.*, **4**, 271-315 (2010)

## Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die Arbeit selbstständig erstellt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

_____

Ort, Datum, Unterschrift (Paul Wolf)