UNIVERSITÉ DE LIMOGES
ÉCOLE DOCTORALE Science – Technologie – Santé
FACULTÉ des SCIENCES et TECHNIQUES

Année 2005 **Thèse N°60-2005**

Thèse

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE LIMOGES

Discipline : Mathématiques et leurs applications
(Arrêté du 25 avril 2002)

présentée et soutenue publiquement
par

Samuel MAFFRE

le 21 novembre 2005

Conjugaison et cyclage dans les groupes de Garside, applications cryptographiques

Thèse dirigée par Thierry BERGER

Jury

Président :	
Jean-Pierre BOREL	Professeur à l'Université de Limoges
Rapporteurs :	
Marc GIRAULT	Expert Emérite, France Télécom R&D, Caen
Juan GONZALEZ-MENESES	Professeur à l'Université de Séville
Examinateurs :	
Thierry BERGER	Professeur à l'Université de Limoges
Philippe GABORIT	Maître de Conférences à l'Université de Limoges
Alexander ZIMMERMANN	Professeur à l'Université de Picardie
Invités :	
François ARNAULT	Maître de Conférences à l'Université de Limoges
Matthieu PICANTIN	Maître de Conférences à l'Université de Paris 7

Faculté des Sciences de Limoges
Département de Mathématiques
123, avenue Albert Thomas
87060 LIMOGES Cedex

Laboratoire d'Arithmétique, de Calcul formel et d'Optimisation
UMR CNRS 6090
123, avenue Albert Thomas
87060 LIMOGES Cedex

Remerciements

Je tiens à remercier Thierry Berger d'avoir accepté d'être mon directeur de thèse ; sa disponibilité et son soutien ont joué une grande part dans la réalisation de ce projet de recherche.

Merci à messieurs François Arnault, Jean-Pierre Borel, Philippe Gaborit, Marc Girault, Juan Gonzalez-Meneses, Matthieu Picantin et Alexander Zimmermann pour l'intérêt qu'ils portent à mon travail et aussi pour avoir accepté de participer à mon jury de thèse. Tout particulièrement, je remercie Jean-Pierre Borel de présider ce jury, ainsi que Marc Girault et Juan Gonzalez-Meneses d'avoir bien voulu être les rapporteurs.

Je tiens à remercier Jean-François Misarsky pour le stage de DEA qu'il m'a proposé au sein du département R&D de France Télécom à Caen. C'est lors de ce stage que j'ai découvert les groupes de tresses et que j'ai pris goût à l'étude de cette structure.

Je tiens à remercier Tor Helleseth de m'avoir accueilli au sein du projet FASTSEC du groupe Selmer, à Bergen (Norvège). C'est lors de ce séjour que mon travail sur le problème des cyclages a pris forme.

Je remercie François Arnault pour les discussions scientifiques que nous avons pu avoir, et qui m'ont permis en particulier de mieux comprendre l'aspect combinatoire des groupes de tresses. Je remercie également Philippe Gaborit pour son regard critique et ses idées, qui ont contibués à la mise en valeur de mon travail.

Je tiens à exprimer toute ma gratitude à Hervé Sibert et Matthieu Picantin, qui furent à plusieurs reprises d'une aide précieuse en ce qui concerne la cryptographie basée sur les tresses et les groupes de Garside. Ils ont toujours répondu efficacement à mes interrogations et sont des références pour moi.

Je remercie Martine Guerletin, Sylvie Laval, Nadine Tchefranoff, Patricia Vareille et Yolande Vieceli pour leur gentillesse, leur disponibilité et leur efficacité ; ce sont des paramètres importants dans le bon déroulement de tout travail.

Je remercie aussi les membres du Laboratoire de m'avoir accueilli parmi eux, et particulièrement Anne Bellido, Alexandre Cabot, Guilhem Castagnos, Laurent Dubreuil, Pierre Dusart, Nicolas Le Roux, Mikaël Lescop, Henri Massias, Chazad Movahhedi, Abdelkader Necer, Adrien Poteaux, Pascale Sénéchaud, Marc Rybowicz et Jacques-Arthur Weil pour leurs échanges, qu'ils soient de nature scientifique, pédagogique ou associative.

Je remercie mon épouse Magali et ma soeur Stéphanie pour leur soutien moral et technique dans la réalisation de ce manuscrit.

De façon générale, je remercie amis et famille pour leurs encouragements, en particulier Andrée et Jean, Béatrice et Alexandre, Brice, Colette et Georges, Christophe, Jacqueline et Franck, Jeanine et Guy, Laure et Jean-David, Nelly et Gilles, Olivia et Daniel.

Table des matières

Remerciements **5**

Introduction **11**

Notations **15**

1 Groupes de tresses et groupes de Garside **17**
1.1 Les tresses géométriques 17
1.2 Les groupes de Garside 19
1.2.1 Définitions 19
1.2.2 Présentations d'Artin et de Birman, Ko et Lee 22
1.2.3 La forme Δ-normale 24

2 Les tresses et la cryptographie **29**
2.1 Représentation 30
2.2 Problèmes algorithmiques 31
2.2.1 Problème des mots 31
2.2.2 Problèmes de conjugaison 32
2.3 Exemples de protocoles 33
2.3.1 Echange de clés et cryptosystème à clé publique 33
2.3.2 Authentification 34
2.4 Attaques existantes 35
2.4.1 Classe de conjugaison 35
2.4.2 Longueur et complexité 36
2.4.3 Approche linéaire 36
2.5 Problématique de la sécurité 37
2.5.1 Présentation 37
2.5.2 Générateur aléatoire 38

3 Propriétés de divisibilité **41**
3.1 Notations 41
3.2 Propriétés 42

4 Algorithme de réduction de la conjugaison **47**
4.1 Présentation 47
4.2 Algorithmes Multiple et Diviseur 49

4.3 Problème de conjugaison - Conj . . . 50
4.4 Complexité . . . 51
4.5 Preuve . . . 52

5 Application de l'algorithme aux groupes de tresses 55
5.1 Modes parallèle / séquentiel . . . 56
5.2 Evaluation des paramètres d'entrée . . . 57
5.2.1 Infimum du secret . . . 57
5.2.2 Longueur canonique du secret . . . 58
5.3 Problème de conjugaison multiple simultanée - ConjMS . . . 62
5.4 Problème de conjugaison de type Ko-Lee - ConjKL . . . 64

6 Analyse de l'efficacité 67
6.1 Evaluation du secret restant . . . 68
6.2 Dépendance aux paramètres d'entrée . . . 68
6.2.1 Dépendance aux générateurs aléatoires . . . 68
6.2.2 Dépendance aux formats des entrées . . . 69
6.3 Proposition d'un générateur aléatoire . . . 70
6.3.1 Explication des résultats . . . 70
6.3.2 Le GA-TronC . . . 71
6.4 Utilisation des modes parallèle / séquentiel . . . 73
6.5 Ordre lexicographique . . . 74
6.6 Conclusion . . . 77

7 Fonction *cyclage* 79
7.1 Présentation . . . 79
7.1.1 Partition de l'ensemble des antécédents . . . 80
7.1.2 Compléments sur la forme Δ-normale . . . 81
7.2 Inversion de la fonction . . . 83
7.2.1 Les critères . . . 83
7.2.2 Existence d'antécédent . . . 85
7.3 Inversion locale . . . 87
7.3.1 Eléments de longueur canonique nulle . . . 87
7.3.2 *Type 1* . . . 89
7.3.3 *Type 2* . . . 90
7.3.4 *Type 3* . . . 92
7.3.5 *Type 4* . . . 93

Conclusion 97

8 Annexe 99
8.1 Statistiques du biais et estimations . . . 99
8.2 Dépendance aux générateurs . . . 101
8.3 Dépendance aux formats des entrées . . . 102
8.4 Utilisation des modes . . . 104
8.5 L'ordre lexicographique . . . 107

Liste des figures **110**

Liste des tableaux **111**

Index **114**

Bibliographie **117**

Introduction

Les groupes de tresses ont été introduits comme support pour la cryptographie en 1999-2000 [1, 28]. Depuis, de nombreuses constructions et attaques ont vu le jour. Notre travail s'inscrit dans la thématique de la cryptographie basée sur les tresses. Nous proposons un algorithme de réduction du problème de conjugaison et résolvons en pratique le problème des cyclages. Les deux problèmes sont présentés dans [28].

La première partie de cette étude concerne l'élaboration d'un algorithme qui attaque une instance de conjugaison ; son action induit une réduction de la taille du secret. Il repose sur la différence de complexité entre les conjugués et s'adapte aux variantes du problème de conjugaison. Son utilisation fait appel à deux techniques particulières :

- L'algorithme prend en entrée une instance de conjugaison $(x, x' = axa^{-1})$ et ressort un diviseur et un multiple du secret a : (d, m). Ceci nous permet immédiatement de déterminer deux nouvelles instances : $(x, d^{-1}x'd)$ et $(x, m^{-1}x'm)$ dont les nouveaux secrets $d^{-1}a$ et $m^{-1}a$ sont de taille plus petite. Ces nouvelles instances peuvent être ensuite considérées comme des instances à part entière ; ainsi, elles peuvent être attaquées à nouveau.
- L'algorithme fonctionne sur une généralisation structurelle des groupes de tresses que sont les groupes de Garside. Ce n'est pas le premier algorithme étendu aux structures de Garside. Ici, nous utilisons le fait que les groupes de tresses possèdent deux structures différentes de groupe de Garside pour exploiter une même instance sous ces deux représentations. Ce mode d'utilisation de notre algorithme permet d'améliorer la réduction.

L'efficacité de l'algorithme dépend essentiellement de la méthode utilisée pour générer les tresses. Ainsi, cette attaque permet de réaffirmer que la génération aléatoire de tresses est un point crucial qui doit être résolu avant d'envisager l'utilisation de la conjugaison comme primitive cryptographique sur les tresses. Il y a très peu d'études sur les générateurs aléatoires alors qu'ils constituent la principale vulnérabilité des cryptosystèmes existants. Actuellement, à part certaines propositions de Sibert [37], il n'existe pas de générateur aléatoire de qualité sur les groupes de tresses. Or c'est exactement là que se situe le problème. La sécurité d'un cryptosystème ne réside pas dans le fait que toutes les instances soient de qualité, mais dans le fait que l'on soit capable de déterminer un ensemble de bonnes instances qui soit exploitable par un générateur aléatoire.

L'objectif de cette partie est de montrer que certains générateurs produisent des clés faibles et identifions des paramètres de notre attaque qui favorisent la réduction du secret. Ainsi, d'une part, nous complétons une liste déjà existante de critères sur les "bonnes" instances qui rendent un problème de conjugaison difficile, et d'autre part, nous montrons que la recherche de nouveaux générateurs aléatoires de tresses est prometteuse en proposant un générateur, simple de conception, permettant de se mettre hors de portée de notre attaque, mais aussi de celle de Hofheinz et Steinwandt [22].

La deuxième partie de cette étude montre que l'inversion de la fonction *cyclage* dans les groupes de Garside admet une solution de complexité polynomiale et donc que le problème des cyclages est résolu en pratique dans les groupes de tresses.

En particulier, nous définissons une partition de l'ensemble des antécédents d'un élément par la fonction *cyclage* et donnons un algorithme permettant d'inverser localement la fonction sur chaque ensemble de la partition.

L'exposé de ce travail s'articulera comme suit :

Dans le Chapitre 1, après une brève description des tresses géométriques, nous introduisons les groupes de Garside qui généralisent les propriétés de divisibilité observées par Garside sur le monoïde de tresses défini par Artin en 1925. Nous présentons deux monoïdes de Garside que possède un groupe de tresses : le monoïde d'Artin et le monoïde de Birman, Ko et Lee. Pour finir, nous introduisons la forme Δ-normale.

Le Chapitre 2 présente la problématique actuelle de la cryptographie basée sur les tresses. Cela passe par la numérisation d'une tresse, les problèmes algorithmiques sur les groupes de tresses, la présentation de quelques protocoles et un rappel sur les attaques existantes. Nous terminons en présentant le rôle des générateurs aléatoires de tresses dans la sécurité des cryptosystèmes.

Le Chapitre 3 nous donne plusieurs propriétés de divisibilité reliées à la forme Δ-normale. Ces résultats sont nouveaux ou généralisent d'autres résultats existants. Ils sont établis dans les groupes de Garside.

Dans le Chapitre 4, nous présentons les algorithmes réduisant le secret d'une instance de conjugaison dans les groupes de Garside. L'algorithme principal exploite la densité de l'écriture des éléments sous leur forme Δ-normale. La forme Δ-normale d'un produit peut s'avérer avoir des similitudes avec le produit des formes Δ-normales des éléments. Plus finement, nous établissons des relations de divisibilité que nous utilisons sur la forme Δ-normale du conjugué, axa^{-1}, par rapport à celle du secret, a.

Le Chapitre 5 détaille plus clairement comment cet algorithme, défini sur une structure de Garside, s'applique dans les groupes de tresses au problème de conjugaison, mais aussi à ses variantes. La double structure de Garside des groupes de tresses est mise en évidence pour développer les techniques, et ainsi, améliorer les résultats. L'algorithme de réduction considère une instance de conjugaison et requiert la connaissance de l'infimum du secret et de sa longueur canonique. Nous proposons une justification du fait que garder ces deux données secrètes n'améliore pas sensiblement la sécurité, et donc, que nous pouvons les supposer connues.

Le Chapitre 6 présente une étude de l'efficacité de l'algorithme de réduction dans les groupes de tresses. Nous considérons différents générateurs aléatoires de tresses et étudions les paramètres qui influent sur l'efficacité. L'analyse de la situation permet de déterminer de nouveaux critères sur les bonnes instances, et de proposer un nouveau générateur aléatoire de tresses résistant à l'attaque. Ensuite, nous considérons l'impact de la double structure de Garside des groupes de tresses sur l'algorithme et introduisons l'ordre lexicographique pour optimiser le recouvrement du secret restant. Enfin, nous concluons en proposant différentes ouvertures quant à l'utilisation ultérieure des groupes de tresses comme plateforme cryptographique.

Le Chapitre 7 propose une résolution du problème des cyclages, en donnant un algorithme polynomial qui inverse la fonction *cyclage*. Ayant défini quatre types d'antécédents possibles, nous poursuivons notre travail en donnant des algorithmes d'inversion locale de la fonction.

Notations

Dans l'ordre d'apparition :

Σ_n	groupe des permutations d'un ensemble à n éléments
1	élément neutre
e	mot vide
$\prec$	division à gauche
$\succ$	division à droite
$\wedge$	pgcd à gauche
$\vee$	ppcm à droite
$\tilde{\wedge}$	pgcd à droite
$\tilde{\vee}$	ppcm à gauche
G	groupe de Garside
Δ	élément de Garside
S	ensemble des éléments simples
$\tau(\ )$	$\tau(x) = \Delta^{-1} x \Delta$ où $x \in G$
$\mathcal{E}$	exposant de Δ
B_n^+	monoïde d'Artin
$\mathcal{A}(\ )$	ensemble des atomes d'un monoïde
BKL_n^+	monoïde de Birman, Ko et Lee
B_n	groupe de tresses (sur l'une ou l'autre présentation)
$l(\ )$	longueur d'un mot : nombre d'atomes dans l'écriture
$\inf(\ )$	infimum
$\sup(\ )$	supremum
$l_c(\ )$	longueur canonique
$c(\ )$	cyclage $c(x) = \Delta^r s_2 \cdots s_p \tau^{-r}(s_1)$ où $x \in G$
$d(\ )$	décyclage $d(x) = \Delta^r \tau^r(s_p) s_1 \cdots s_{p-1}$ où $x \in G$
LB_l	sous-groupe de B_n sur les l premiers brins
RB_r	sous-groupe de B_n sur les r derniers brins
*	dual de $x \in G$: $x^* = x^{-1}\Delta^{\sup(x)}$
$\partial(\ )$	$\partial(x) = x^{-1}\Delta(x^* =)$ où $x \in S$
b_{GA}	biais du générateur aléatoire de tresses GA
$\mathcal{S}(\ )$	Starting Set
$\mathcal{F}(\ )$	Finishing Set
$\mathcal{R}(\ )$	complément à droite
$\mathfrak{a}$	$\Delta \tilde{\wedge} \Delta^{-\inf(a)} a$
$\mathfrak{A}$	$\Delta^{-\inf(a)} a \mathfrak{a}^{-1}$

Chapitre 1

Groupes de tresses et groupes de Garside

L'objet de ce chapitre est d'introduire les groupes de tresses et les groupes de Garside. Nous commençons par donner un rapide aperçu de l'aspect géométrique des tresses puis définissons les groupes de Garside. Ces derniers sont une généralisation des propriétés de divisibilité observées dans les monoïdes de tresses introduits par Artin. Notre travail repose essentiellement sur ces propriétés ; ainsi, nous donnons de nombreux résultats dans les groupes de Garside même si la motivation initiale et certaines applications concernent particulièrement les groupes de tresses. Ce chapitre reprend les travaux de Picantin et Sibert [37, 35].

La première introduction explicite du groupe de tresses à n brins date de 1925 ; [3] E. Artin définit le groupe par la présentation suivante :

$$\left\langle \sigma_1, \sigma_2, \ldots, \sigma_{n-1} \;\middle|\; \begin{array}{rll} \sigma_i\sigma_j\sigma_i &= \sigma_j\sigma_i\sigma_j & \text{si } |i-j|=1 \\ \sigma_i\sigma_j &= \sigma_j\sigma_i & \text{si } |i-j|\geq 2 \end{array} \right\rangle$$

Cette présentation est très proche d'une présentation du groupe des permutations, Σ_n, donnée par :

$$\left\langle \sigma_1, \sigma_2, \ldots, \sigma_{n-1} \;\middle|\; \begin{array}{rcll} \sigma_i\sigma_j\sigma_i &=& \sigma_j\sigma_i\sigma_j & \text{si } |i-j|=1 \\ \sigma_i\sigma_j &=& \sigma_j\sigma_i & \text{si } |i-j|\geq 2 \\ \sigma_i^2 &=& 1 & \end{array} \right\rangle$$

Ces deux groupes ont de fortes connexions ; en particulier, les permutations sont utilisées dans l'élaboration d'une forme normale dans les groupes de tresses.

Les groupes de tresses jouent un rôle important dans différents domaines tels que la théorie des groupes combinatoires, la théorie de la représentation et la théorie des nœuds. Un avantage déterminant des tresses sur les nœuds est la possibilité de construire une structure de groupe.

1.1 Les tresses géométriques

Une vulgarisation intéressante et intuitive des tresses est donnée par P. Dehornoy dans [11]. Une tresse peut se représenter par un ensemble orienté de brins qui se croisent.

Une définition géométrique est :

Définition 1.1.1. Soit f une permutation du sous-ensemble $P = \{1, \ldots, n\}$ du plan complexe. Une *tresse géométrique à n brins associée à f* est l'union de n arcs disjoints de $\mathbb{C} \times [0,1]$ joignant chaque point $(i, 0)$ au point $(f(i), 1)$ et telle que chaque plan horizontal $\mathbb{C} \times \{z\},\ z \in [0,1]$ coupe chaque arc exactement une fois.

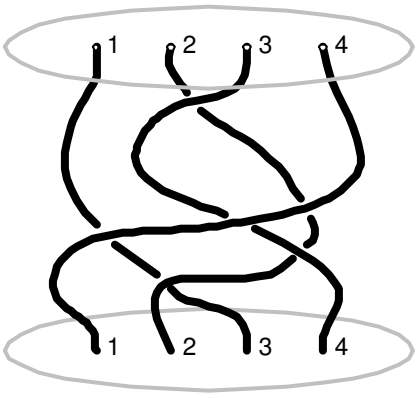

FIG. 1.1 – Une tresse géométrique à quatre brins

La Figure 1.1 représente une tresse géométrique à quatre brins. Il est pratique de considérer que l'objet reste le même si l'on étire ou déforme un arc tout en laissant ses extrémités fixées ($\mathbb{C} \times \{0,1\}$). Ceci se formalise par :

Définition 1.1.2. Deux tresses géométriques, associées à une même permutation f, sont dites *isotopes* si l'une peut être transformée continûment en l'autre dans l'ensemble des tresses géométriques associées à f.

Structure de groupe

Considérant l'ensemble des tresses géométriques à n brins, on le munit d'une *loi de composition interne* associative : pour a, b deux tresses géométriques, le produit ab est la tresse géométrique obtenue en contractant les tresses a et b respectivement dans $\mathbb{C} \times [\frac{1}{2}, 1]$ et $\mathbb{C} \times [0, \frac{1}{2}]$ et en concaténant les morceaux obtenus. La Figure 1.2 nous donne un exemple du produit de deux tresses géométriques.

L'image "miroir" d'une tresse géométrique obtenue par réflexion par rapport au plan $\mathbb{C} \times \{0\}$ constitue un *inverse* pour ce produit. Voir la Figure 1.3.

Définition 1.1.3. Une *tresse à n brins* est une classe d'isotopie de tresses géométriques.

Par exemple, l'élément σ_i [resp. σ_i^{-1}] correspond à la classe d'isotopie de la tresse géométrique permutant uniquement les brins i et $i+1$ et telle que le brins d'indice $i+1$ passe devant [resp. derrière] celui d'indice i.

La classe d'isotopie du produit de deux tresses géométriques ne dépend que des classes d'isotopie de ces tresses. Le produit de tresses geométriques passe au quotient par la relation d'isotopie, et induit un produit sur l'ensemble des tresses à n brins, lui conférant une structure de groupe.

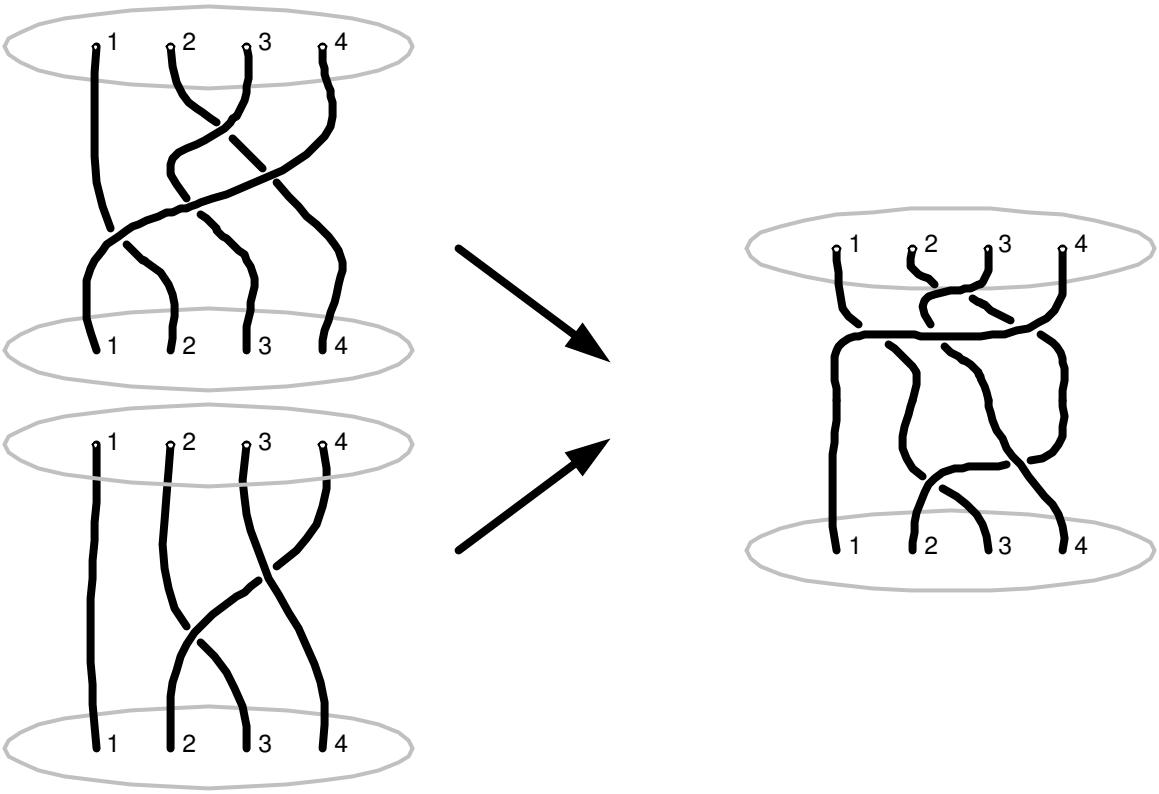

FIG. 1.2 – Le produit de deux tresses géométriques

1.2 Les groupes de Garside

Les groupes de Garside sont une généralisation des propriétés de divisibilité observées par F.A. Garside [20] dans les monoïdes de tresses donnés sous la présentation d'Artin. D'autres personnes ont prolongé cette étude pour arriver au niveau actuel de formalisme : E.A. Elrifai, H.R. Morton, W. Thurston [16, 17]. Ses travaux avaient pour but de résoudre le problème des mots et le problème de conjugaison dans les groupes de tresses.

Les groupes de Garside sont une généralisation, parmi plusieurs, des groupes de tresses. En effet, les groupes de tresses ont une grande richesse de structure : aspect topologique, théorie de la représentation, propriétés algébriques. De nombreux travaux ont été menés afin d'étendre certaines propriétés des groupes de tresses aux groupes de Garside [15, 13, 35, 36, 37]

Appelés aussi *Petits groupes gaussiens* [35, 36], c'est en 2002 que le standard de *groupes de Garside* se généralise dans [13].

1.2.1 Définitions

Définition 1.2.1. Un *monoïde* est un couple $(M, \bullet)$, où M est un ensemble, $\bullet$ une loi de composition interne associative et M admet un élément neutre pour $\bullet$, noté 1.

Notation 1.2.2. On note e le mot vide, afin de le différencier de l'élément neutre 1.

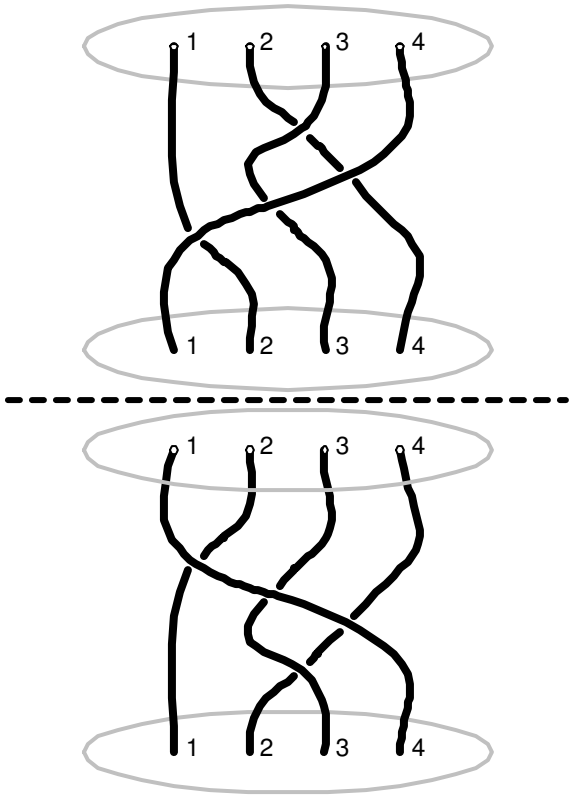

FIG. 1.3 – L'inversion de tresses géométriques

Définition 1.2.3. Un monoïde M est dit *simplifiable* à gauche si
$$\forall x, y, z \in M, \quad xy = xz \quad \Rightarrow \quad y = z$$

Notation 1.2.4. Soit M un monoïde simplifiable (à gauche et à droite) ne possédant pas d'élément inversible autre que son élément neutre ; nous pouvons définir deux ordres partiels $\prec$ et $\succ$: $\quad a \prec b \; [b \succ a] \quad$ si $\quad \exists c \in M; \quad ac = b \; [b = ca]$. L'élément a est appelé *diviseur à gauche* [à droite] de b, et b est appelé *multiple à droite* [à gauche] de a.

Remarquons que dans le cas de mots, le premier ordre partiel représente exactement l'idée de préfixe : un mot est plus petit à gauche qu'un autre, s'il en est un préfixe.

Définition 1.2.5. Soit M un monoïde. L'élément $x \in M$ est un *atome* si $x \neq 1$ et si $x = yz \quad \Rightarrow \quad (y = 1$ ou $z = 1)$. Le monoïde M est dit *atomique* s'il est généré par ses atomes et si pour tout élément $x \in M$, il existe un entier naturel $N_x \in \mathbb{N}$ tel que x ne peut pas s'écrire comme un produit de plus de N_x atomes.

Définition 1.2.6. Un monoïde M est dit *gaussien* s'il est un monoïde atomique simplifiable et si toute paire d'éléments de M admet un plus grand commun diviseur et un plus petit commun multiple.

Notation 1.2.7. Nous notons $\wedge$ le plus grand commun diviseur à gauche, $\vee$ le plus petit commun multiple à droite ; $\tilde{\wedge}$ et $\tilde{\vee}$ pour les notions relatives à la division à droite (à gauche comme à droite).

Le fait que le monoïde soit simplifiable nous donne l'unicité du pgcd et du ppcm. Nous avons :

$$\begin{array}{lccl} \text{Soit } a,b \in M, & d = a \wedge b & \Leftrightarrow & \forall c \in M \quad c \prec a \text{ et } c \prec b \text{ ssi } c \prec d \\ & m = a \vee b & \Leftrightarrow & \forall c \in M \quad a \prec c \text{ et } b \prec c \text{ ssi } m \prec c \end{array}$$

Définition 1.2.8. Soit M un monoïde Gaussien. Un *élément de Garside* est un élément Δ de M, dont les diviseurs à gauche coïncident avec les diviseurs à droite, et forment un ensemble fini et générateur pour le monoïde. Un *monoïde de Garside* est un monoïde gaussien qui possède un *élément de Garside*.

On note qu'il n'y a pas unicité de l'élément de Garside dans un monoïde gaussien. En particulier, chaque puissance d'un élément de Garside est un élément de Garside. La structure d'un monoïde de Garside est déterminée par le choix de l'element de Garside. Quand nous parlons de l'élément de Garside, nous désignons celui correspondant au monoïde de Garsides considéré.

Notation 1.2.9. Les diviseurs de Δ sont appelés *éléments simples*, ou encore *facteurs canoniques*. L'ensemble des éléments simples est noté S.

Un monoïde de Garside satisfait les conditions d'Ore [9] et donc se plonge dans son groupe de fractions. Le groupe de fractions à gauche coïncide avec celui à droite ; il n'y pas d'ambiguïté.

Définition 1.2.10. Un groupe est appelé *groupe de Garside* s'il est le groupe de fractions d'un monoïde de Garside.

FIG. 1.4 – Représentation des générateurs du monoïde B_n^+ : σ_i, σ_i^{-1}

Introduction de la fonction $\tau(\ )$ et propriétés de Δ

Notation 1.2.11. Soit G un groupe de Garside ; on note τ la fonction de G dans G telle que : $x \mapsto \tau(x) = \Delta^{-1} x \Delta$

Proposition 1.2.12. *[13, Lemme 2.3] Soit M un monoïde de Garside ; la restriction de la fonction τ à S est une permutation de S.*

Le lecteur pourra vérifier de lui-même que τ est un automorphisme de G qui laisse invariant M et S et qui respecte $\prec$ et $\succ$.

Corollaire 1.2.13. *Soit G un groupe de Garside ; alors il existe $k \in \mathbb{N}^*$ tel que $\forall x \in G,\ \tau^k(x) = x$ ce qui est équivalent à : $\exists k \in \mathbb{N}^*;\ \Delta^k \in Z(G)$, le centre de G.*

Définition 1.2.14. Soit G un groupe de Garside d'élément de Garside Δ. On note $\mathcal{E}$ et appelle *exposant* le plus petit entier naturel non nul, k, tel que Δ^k appartienne au centre de G. En d'autres mots, $\mathcal{E}$ est l'ordre de τ.

1.2.2 Présentations d'Artin et de Birman, Ko et Lee

Les groupes de tresses sont des exemples typiques de groupes de Garside. Cependant, le fait que les groupes de tresses aient été le modèle étudié pour définir les groupes de Garside n'est pas le seul intérêt. La description d'une structure de Garside dans les groupes de tresses a aussi participé à mieux connaître la structure même des groupes de tresses : chaque groupe de tresses possède au moins deux structures de Garside différentes. Pour le moment, cet aspect a été utilisé pour optimiser des algorithmes en tenant compte des spécificités de chacune des deux structures. Nous proposons une nouvelle approche qui utilise cette double structure afin de mieux contrôler la manipulation des éléments dans les groupes de tresses.

Ainsi un groupe de tresses peut être considéré comme le groupe de fractions de deux monoïdes de Garside : celui de la présentation d'Artin et celui de la présentation de Birman, Ko et Lee. La question de savoir s'il existe d'autres présentations de Garside des groupes de tresses dans le cas général est ouverte. Par exemple, on sait que le groupe de tresses à 3 brins possède au moins quatre présentations de Garside différentes.

Présentation du monoïde d'Artin [3, 4]

$$B_n^+ = \left\langle \sigma_1, \sigma_2, \ldots, \sigma_{n-1} \;\middle|\; \begin{array}{rll} \sigma_i\sigma_j\sigma_i & = \sigma_j\sigma_i\sigma_j & \text{si } |i-j| = 1 \\ \sigma_i\sigma_j & = \sigma_j\sigma_i & \text{si } |i-j| \geq 2 \end{array} \right\rangle \tag{1.1}$$

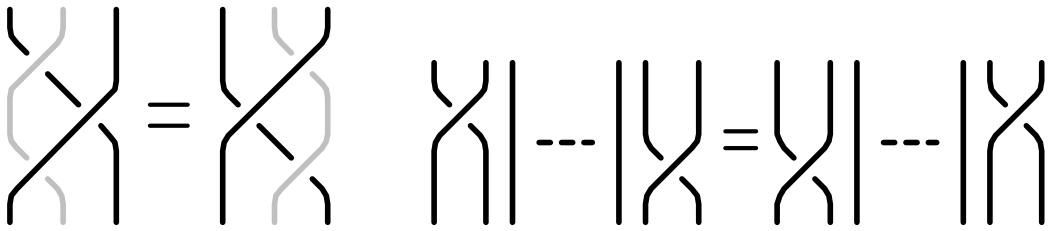

FIG. 1.5 – Représentation des relations du monoïde B_n^+

Ce monoïde est un monoïde de Garside ; ses caractéristiques sont :

$$\left| \begin{array}{rl} \#\mathcal{A}(B_n^+) & = n-1 \quad \text{nombre d'atomes} \\ \Delta_{B_n^+} & = (\sigma_1\sigma_2\ldots\sigma_{n-1})(\sigma_1\sigma_2\ldots\sigma_{n-2})\ldots(\sigma_1\sigma_2)\sigma_1 \\ \#S_{B_n^+} & = n! \\ Z(B_n) & = \langle \Delta^2_{Bn^+} \rangle \text{ centre de } B_n, n \geq 3 \text{ [20]} \\ \tau(\sigma_i) & = \sigma_{n-i} \end{array} \right.$$

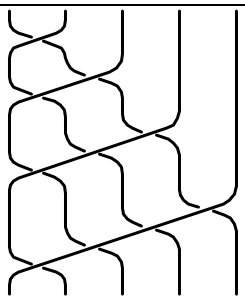

FIG. 1.6 – Représentation de l'élément de Garside $\Delta_{B_5^+}$

La représentation géométrique des générateurs de cette présentation est donnée à la Figure 1.4. Les relations de la présentation sont clairement identifiables géométriquement, voir la Figure 1.5. Comme le montre la Figure 1.6 dans le cas des tresses à cinq brins, l'élément $\Delta_{B_n^+}$ peut être défini comme une tresse positive (tresse dans B_n^+) dans laquelle deux brins quelconques se croisent exactement une fois. Les éléments simples sont les tresses positives dans lesquelles deux brins quelconques se croisent au plus une fois.

Proposition 1.2.15. *[16] L'ensemble des éléments simples, $S_{B_n^+}$, est en bijection avec le groupe des permutations d'un ensemble à n éléments, Σ_n.*

Présentation du monoïde de Birman, Ko and Lee [5]

$$BKL_n^+ = \left\langle a_{ts} \quad (n \geq t > s \geq 1) \;\middle|\; \begin{array}{ll} a_{ts}a_{rq} = a_{rq}a_{ts} & \text{si } (t-r)(t-q)(s-r)(s-q) > 0 \\ a_{ts}a_{sr} = a_{tr}a_{ts} = a_{sr}a_{tr} & \text{si } n \geq t > s > r \geq 1 \end{array} \right\rangle \tag{1.2}$$

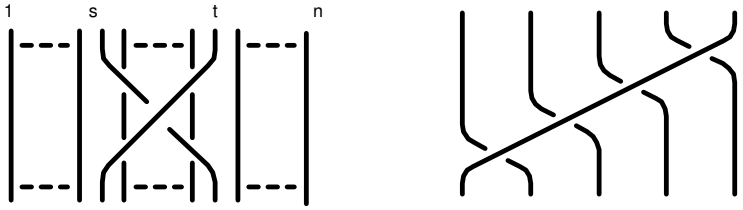

FIG. 1.7 – Représentation dans le monoïde BKL_n^+ du générateur $a_{t,s}$ et de $\Delta_{BKL_5^+}$

Ce monoïde est un monoïde de Garside ; ses caractéristiques sont :

$$\left| \begin{array}{rl} \#\mathcal{A}(BKL_n^+) & = \frac{n(n-1)}{2} \\ \Delta_{BKL_n^+} & = a_{n,n-1}a_{n-1,n-2}...a_{2,1} \\ \#S_{BKL_n^+} & = \frac{(2n)!}{n!(n+1)!} = C_n \in [3^n, 4^n] (\in o(n!)) \\ \Delta^n_{BKL_n^+} & = \Delta^2_{B_n^+} \\ \tau(a_{ts}) & = a_{t+1,s+1} \end{array} \right.$$

La Figure 1.7 nous donne la représentation géométrique des générateurs de cette présentation et celle de $\Delta_{BKL_5^+}$.

Définition 1.2.16. Un *cycle descendant* est une permutation de la forme $T_j = (t_j t_{j-1} \cdots t_1)$ où $t_j > t_{j-1} > \cdots > t_1$. Deux cycles descendants T_j et S_k sont *parallèles* si $(s_a - t_c)(s_a - t_d)(s_b - t_c)(s_b - t_d) > 0$ pour tout $1 \leq a < b \leq k$ et $1 \leq c < d \leq j$.

Proposition 1.2.17. *[5, Théorème 3.4] Il existe une bijection de $S_{BKL_n^+}$ dans le sous-ensemble de Σ_n formé par les permutations qui sont des produits de cycles descendants parallèles.*

Remarque 1.2.18. Il est facile de passer d'une présentation à l'autre avec une complexité polynomiale [7]

$$\sigma_i = a_{i+1,i} \quad \text{et} \quad a_{t,s} = (\sigma_{t-1} \cdots \sigma_{s+1})\, \sigma_s \left(\sigma_{s+1}^{-1} \cdots \sigma_{t-1}^{-1}\right)$$

Remarque 1.2.19. Certains algorithmes sont plus efficaces dans la présentation d'Artin car le nombre de générateurs est inférieur. D'autres mettent en avant le nombre d'éléments simples ou la longueur de l'élément de Garside ; ils sont plus performants sur la seconde présentation.

Notation 1.2.20. Nous noterons B_n le groupe de tresses à n brins, que ce soit le groupe de fractions de B_n^+ ou de BKL_n^+.

Les générateurs de ces présentations correspondent aux atomes de la Définission 1.2.5. Les éléments de Garside proposé, sont ceux employés dans la littérature.

1.2.3 La forme Δ-normale

Dans cette section, nous définissons une forme normale dans les groupes de Garside ; elle associe à chaque élément une écriture canonique. Nous travaillons avec l'élément de Garside Δ du monoïde de Garside M dans le groupe G ; nous sous-entendons parfois ces notations quand il n'y a pas d'ambiguïté.

La forme Δ-normale dépend de l'élément de Garside et donc est propre à la structure du groupe de Garside considéré. Nous allons voir que chaque élément du groupe admet une écriture unique comme le produit d'une puissance de Δ et d'éléments simples ; le mode opératoire est fondé sur l'utilisation du pgcd.

Notation 1.2.21. Ici, la longueur signifie le nombre d'atomes utilisés pour écrire un élément. Dans le cas des deux monoïdes de tresses donnés précédemment, on observe que les relations préservent la longueur des mots. Ainsi, sur ces monoïdes, cette longueur a un sens ; elle dépend de l'élément et non du mot. Ceci n'est plus vrai dans les groupes. On note $l(\ )$ cette longueur.

Lemme 1.2.22. *[37, Lemme 6.2.1] Soit s un élément simple. Alors, pour tout élément x, y dans M, $s \prec xy \quad \Leftrightarrow \quad s \prec x(y \wedge \Delta)$.*

Définition 1.2.23. Soit M un monoïde de Garside et Δ l'élément de Garside de M. On dit qu'une suite $(x_1, \ldots x_n)$ est normale à gauche si x_i est un élément simple différent de 1 pour tout i, et si, pour $1 \leq i < n$, on a $x_i x_{i+1} \wedge \Delta = x_i$.

Proposition 1.2.24. *[37, Proposition 6.2.4] Soit M un monoïde de Garside. Tout élément x de M, différent de 1, admet une unique décomposition $x = s_1 \dots s_p$ avec p un entier naturel et $(s_1, \dots, s_p)$ une suite normale à gauche.*

Proposition 1.2.25. *[37, Proposition 6.2.6] Pour tout élément x de G, il existe un unique entier r et un unique élément y de M vérifiant $x = \Delta^r y$ et $\Delta \not\prec y$.*

Proposition-Définition 1.2.26. *[37, Proposition 6.2.7] Tout élément x de G admet une unique décomposition $x = \Delta^r s_1 \dots s_p$ où r et p sont deux entiers avec $p \geq 0$, et où $(s_1, \dots, s_p)$ est une suite normale à gauche vérifiant $s_1 \neq \Delta$. Cette décomposition est appelée forme Δ-normale à gauche. On appelle respectivement infimum, supremum et longueur canonique de x les entiers $\inf(x) = r$, $\sup(x) = r + p$ et $l_c(x) = p$.*

Corollaire 1.2.27. *Soit x un élément de G. L'infimum et le supremum sont caractérisés par :*

$$\inf(x) = max\{i \in \mathbb{Z};\ \Delta^i \prec x\} \qquad \sup(x) = min\{i \in \mathbb{Z};\ x \prec \Delta^i\}$$

Remarque 1.2.28. Quelquefois, si nous traitons des éléments du monoïde M et qu'il n'y a pas de confusion, nous dirons qu'un élément $x = s_1 \dots s_p \in M$ est dans sa forme Δ-normale à gauche pour exprimer que la suite $(s_1, \dots, s_p)$ est une suite normale à gauche et qu'il existe un entier naturel q tel que la forme Δ-normale à gauche de x est $\Delta^q s_{q+1} \dots s_p$, au sens de la Proposition-Définition 1.2.26, c'est à dire : $s_1 = s_2 = \dots = s_q = \Delta$.

Remarque 1.2.29. Une démarche exactement similaire utilisant les mêmes notions à droite existe ; elle donne une *forme Δ-normale à droite*. Dans la suite, l'orientation par défaut sera celle à gauche.

Remarque 1.2.30. La forme Δ-normale (à gauche), présentée ici, se retrouve dans d'autres papiers sous les noms de *Left-greedy canonical form* [17], *greedy normal form* [14], *left normal form* [18], *left-canonical form* [16, 28], *normal form* [21] et *forme normale à gauche* [13, 37]. De plus les suites normales présentées ici ont des liens avec la *left-weighted factorisation* [16] et la *left-weighted decomposition* [28].

Remarque 1.2.31. Nous avons maintenant deux longueurs pour mesurer les mots : $l(\ )$ qui représente le nombre d'atomes utilisés dans l'écriture et $l_c(\ )$ qui est la longueur canonique du mot mis sous forme Δ-normale.

Cette forme normale donne des informations précises sur les éléments et permet de les manipuler efficacement. En particulier, la complexité de son calcul est polynomiale dans les groupes de tresses (voir la table de complexité 2.1).

Considérant le pgcd à gauche et le ppcm à droite, il apparaît que $(M, \prec)$ possède une structure de treillis et S devient un treillis fini avec un minimum, 1, et un maximum, Δ. La connaissance du diagramme de Hasse du treillis de S et l'existence de la forme Δ-normale donne aux groupes de Garside une structure automatique ; en particulier, cela permet de décrire entièrement les éléments du groupe. Le groupe de tresses à quatre brins dans sa présentation d'Artin est donc totalement exprimable à partir de la Figure 1.8.

Remarque 1.2.32. Pour savoir si un couple d'éléments simples, (s_1, s_2), est une suite normale à gauche, il suffit d'essayer de prolonger dans S l'écriture du premier élément simple par les atomes qui divisent à gauche le second : la suite est normale si et seulement si ce prolongement est impossible.

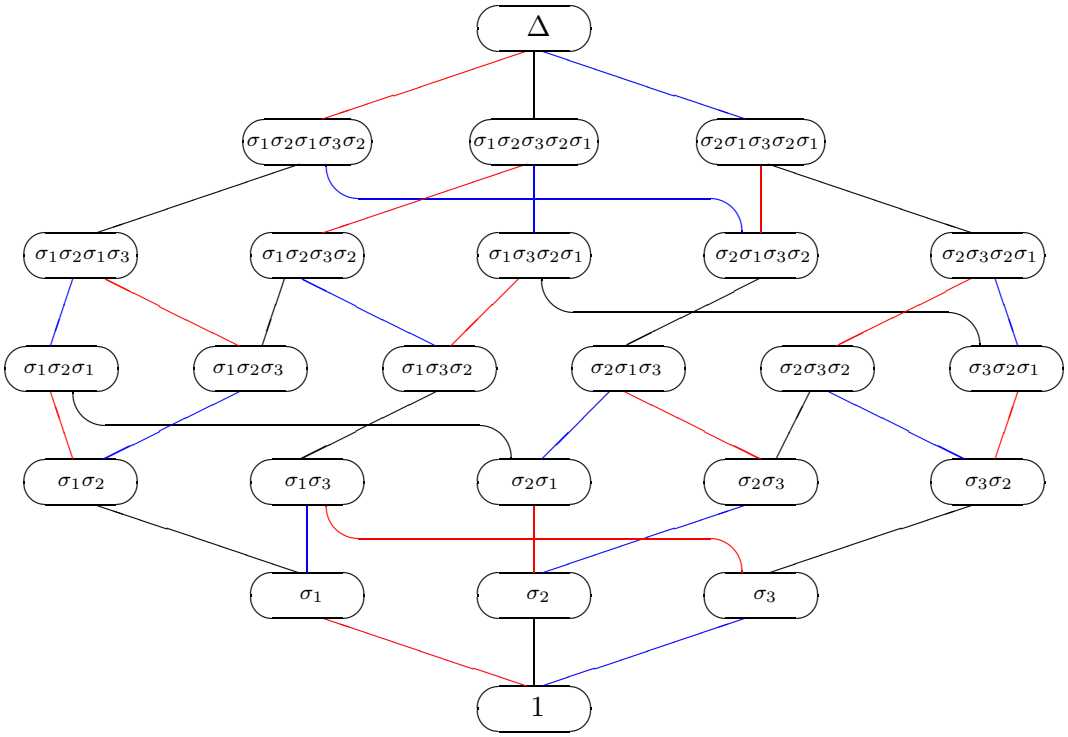

FIG. 1.8 – Treillis des éléments simples de B_4^+ : $\left(S_{B_4^+}, \prec\right)$.

Exemple 1.2.33. Le couple $(\sigma_1\sigma_2\sigma_3, \sigma_2\sigma_1\sigma_3\sigma_2)$ n'est pas une suite normale à gauche : les atomes qui peuvent compléter $\sigma_1\sigma_2\sigma_3$ en un autre élément simple sont σ_1 et σ_2 ; tandis que les atomes qui divisent à gauche $\sigma_2\sigma_1\sigma_3\sigma_2$ sont réduits à σ_2 (ces atomes sont les points de départ, après e, d'un chemin montant vers $\sigma_2\sigma_1\sigma_3\sigma_2$). Donc le premier facteur peut être complété par le deuxième.

Le couple $(\sigma_1\sigma_3\sigma_2\sigma_1, \sigma_1\sigma_2\sigma_1\sigma_3)$ est une suite normale à gauche : les atomes pouvant compléter $\sigma_1\sigma_3\sigma_2\sigma_1$ en un autre élément simple sont réduits à σ_3, tandis que les atomes divisant à gauche $\sigma_1\sigma_2\sigma_1\sigma_3$ sont σ_1 et σ_2. On ne peut pas compléter le premier facteur !

Introduction des fonctions *cyclage* et *décyclage*

La fonction *cyclage* sera l'objet d'une étude particulière au Chapitre 7.

Définition 1.2.34. Soit $\Delta^r s_1 \cdots s_p$ la forme Δ-normale de x, un élément de G. On définit le *cyclé* de x par :

$$c(x) = \Delta^r s_2 \cdots s_p \tau^{-r}(s_1)$$

On appelle *cyclage* la fonction de G dans G qui à tout élément associe son cyclé.

Définition 1.2.35. Soit $\Delta^r s_1 \cdots s_p$ la forme Δ-normale de x, un élément de G. On définit le *décyclé* de x par :

$$d(x) = \Delta^r \tau^r(s_p) s_1 s_2 \cdots s_{p-1}$$

On appelle *décyclage* la fonction de G dans G qui à tout élément associe son décyclé.

Le théorème suivant est un résultat important utilisé dans la résolution du problème de conjugaison. Nous le citons comme complément culturel.

Théorème 1.2.36. *[6, 35] Soit x un élément de G. S'il existe un élément y conjugué de x dans G tel que* $\inf(y) > \inf(x)$ *[resp.* $\sup(y) < \sup(x)$ *], alors il existe* $k \in [1, l(\Delta) - 1]$ *tel que* $inf\left(c^k(x)\right) > \inf(x)$ *[resp.* $sup\left(d^k(x)\right) < \sup(x)$ *].*

Chapitre 2

Les tresses et la cryptographie

Les groupes de tresses ont été définis dans le chapitre précédent. Ici, nous nous intéressons à la problématique de la cryptographie basée sur les tresses en rappelant des résultats existants.

La représentation des éléments et la génération aléatoire de tresses jouent un rôle essentiel. La cryptographie basée sur les tresses rencontre un problème de taille dans la difficulté de générer de bonnes clés. Cet aspect constitue une part importante de notre travail de réduction de la conjugaison.

Ce chapitre n'a pas pour but d'établir un état de l'art pour lequel nous référons à [14]. Nous y abordons les aspects de numérisation des tresses et les problèmes algorithmiques dans les groupes de tresses. Ensuite, nous présentons quelques protocoles et un rappel des attaques existantes. Pour finir, nous abordons le problème de la sécurité et présentons les contraintes de la génération aléatoire de tresses.

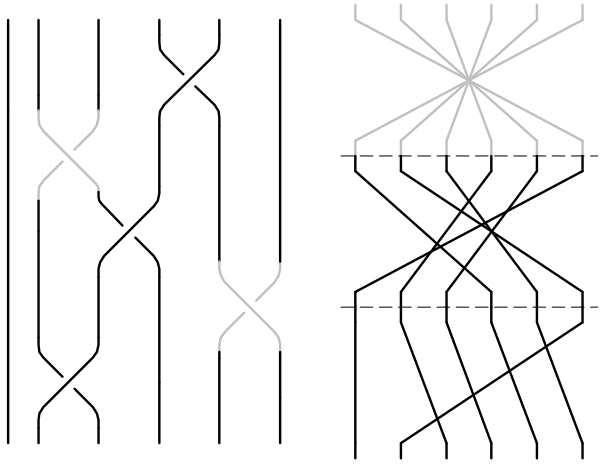

FIG. 2.1 – Illustration de l'Exemple 2.1.1

Nous considérons le groupe de tresses à n brins, B_n, d'élément de Garside Δ. Les éléments Δ et S dépendent du monoïde choisi dont B_n est le groupe de fractions : il peut être B_n^+ ou BKL_n^+.

2.1 Représentation

En général, l'implémentation de protocoles cryptographiques ne pose pas de problème en terme de représentation des éléments ; c'est le cas des cryptosystèmes basés sur la théorie des nombres. Cependant, l'implémentation des tresses nécessite de pouvoir les écrire à l'aide de nombres, comme une suite finie de digits. Le choix effectué n'est pas sans incidence sur ce que l'on entend par : taille d'une tresse, générateur aléatoire, efficacité pratique d'une procédure, [37]. Dans tous les cas, ce n'est pas un élément que nous stockons mais un mot ou son image. Il y a essentiellement deux méthodes de représentation des tresses ; nous pouvons les considérer comme :

- le produit de générateurs issus de la présentation du monoïde
- le produit d'éléments simples (en rapport avec la forme Δ-normale)

Exemple 2.1.1. Exemple de représentation d'une tresse à six brins issue de la présentation d'Artin (1.1), utilisant les générateurs et les éléments simples convertis en permutations.

$$\begin{aligned} a &= \sigma_4\sigma_2^{-1}\sigma_3\sigma_5^{-1}\sigma_2 \rightarrow [4,-2,3,-5,2] \\ &= \Delta^{-1}\underbrace{\sigma_2\sigma_3\sigma_2\sigma_1\sigma_4\sigma_3\sigma_2\sigma_5\sigma_4\sigma_3\sigma_2\sigma_1}_{\pi_1=(1624)(35)}\underbrace{\sigma_5\sigma_4\sigma_3\sigma_2}_{\pi_2=(26543)} \text{ forme } \Delta\text{-normale } \rightarrow [-1,[\pi_1,\pi_2]] \end{aligned}$$

La figure 2.1 illustre cet exemple en présentant à gauche la représentation en terme de produit de générateurs et à droite en terme de produit d'éléments simples. Les croisements négatifs, relatifs à un inverse, sont en gris.

Table de complexité

La table 2.1 nous donne la complexité des opérations de base utilisées dans le groupe de tresses à n brins [7]. Le paramètre l représente la longueur canonique.

Opération	Complexité B_n^+	BKL_n^+
ÉLÉMENT SIMPLE		
Produit	$O(n)$	$O(n)$
Inverse	$O(n)$	$O(n)$
τ^k	$O(n)$	$O(n)$
PGCD	$O(n\log n)$	$O(n)$
Tirage aléatoire	$O(n)$	$O(n\log n)$
n-TRESSE		
Produit	$O(ln)$	$O(ln)$
Inverse	$O(ln)$	$O(ln)$
Forme Δ-normale	$O(l^2n\log n)$	$O(l^2n)$

TAB. 2.1 – Table de complexité des opérations dans le groupe de tresses à n brins.

2.2 Problèmes algorithmiques

2.2.1 Problème des mots

Le problème des mots est directement lié d'une part à la propriété d'isotopie rencontrée dans le cas des tresses géométriques, et d'autre part aux règles de réécriture induites par les relations de la présentation du monoïde et donc du groupe. Clarifions la terminologie utilisée :

Notation 2.2.1. Un *élément* du groupe B_n est une tresse à n brins au sens de la Définition 1.1.3. Un *mot* du groupe B_n est l'écriture d'un élément du groupe B_n.

Définition 2.2.2. Deux mots sont dits *équivalents* s'ils représentent le même élément.

Problème 2.2.3. (problème des mots) : Etant donnés deux mots du groupe, déterminer s'ils sont équivalents.

Ce problème, dans les groupes *présentés* (définis par générateurs et relations), a été proposé en 1985 comme primitive cryptographique par Wagner et Magyarik [39]. De nombreuses suites, dont ceratines récentes [34], ont été données à ce travail.

Dans le groupe de tresses, le problème des mots est efficacement résolu avec un algorithme de complexité polynomiale depuis 1990, grâce aux travaux de Jacquemard [24].

En pratique, il existe deux outils pour résoudre le problème des mots :

- Les formes normales : elles donnent à chaque élément une écriture canonique. Ainsi, normalisant l'écriture des deux mots à comparer, il suffit de regarder si l'on obtient ou non la même écriture finale.

  Naturellement, la *forme Δ-normale* est une forme normale. La complexité polynomiale du calcul de cette forme permet de résoudre efficacement le problème, voir la table de complexité 2.1. Il existe d'autres formes normales, dont celle utilisée par Jacquemard qui est basée sur l'ordre lexicographique.

- Les formes réduites : elles transforment l'écriture d'un mot, respectant les relations de la présentation, et réduisent tout mot représentant l'élément neutre au mot vide. Ainsi, considérant deux mots x et y, il suffit d'appliquer une forme réduite au mot $x^{-1}y$ et regarder si la réduction conduit au mot vide.

  Il existe plusieurs formes réduites. Citons-en une particulièrement intéressante : la réduction de poignées de Dehornoy [12]. Elle repose sur l'existence d'un ordre total sur l'ensemble des tresses [10, 25].

L'ordre lexicographique

L'ordre lexicographique a été introduit par Garside dans le but de résoudre le problème des mots ; il a été rendu exploitable en pratique par Jacquemard [20, 24]. Cet ordre ne s'applique qu'à des mots positifs, c'est-à-dire un produit uniquement de générateurs et non de leurs inverses. Les générateurs du monoïde étant indexés, l'*ordre lexicographique* consiste à classer les mots positifs conformément à l'alphabet de l'indexation des générateurs.

Définition 2.2.4. Un mot positif respecte l'ordre lexicographique s'il n'existe pas de mot positif équivalent ayant une écriture lexicographique antérieure.

La *forme normale de Garside* consiste à considérer un mot x, le mettre sous la forme $\Delta^{\inf(x)}X$ (voir Proposition 1.2.25), puis à écrire X tel qu'il respecte l'ordre lexicographique.

2.2.2 Problèmes de conjugaison

Les problèmes suivants sont issus d'un des papiers fondateurs de la cryptographie basée sur les tresses [28]. Le problème fondamental est l'inversion de la conjugaison. Nous donnerons dans les Sections 2.4 et 2.5 une idée sur la sécurité actuelle de ces problèmes.

Problème 2.2.5. (problème de conjugaison) : Etant donnés deux éléments conjugués dans B_n, déterminer un conjugant : c'est-à-dire, considérant une instance de conjugaison (x, y), trouver un élément $z \in B_n$ tel que $y = zxz^{-1}$.

Problème 2.2.6. (problème décisionnel de conjugaison) : Etant donnés deux éléments dans B_n, déterminer s'ils sont conjugués.

Problème 2.2.7. (problème de conjugaison de type Ko-Lee) : Soit H un sous-groupe de B_n. Etant donnés deux éléments conjugués dans B_n par un élément de H, déterminer un conjugant dans H.

Problème 2.2.8. (problème de conjugaison de type Diffie-Hellman) : Soit H et J deux sous-groupes de B_n qui commutent. Etant donnés les éléments $p, p' = xpx^{-1}, p'' = ypy^{-1}$, avec $x \in H$ et $y \in J$, déterminer l'élément $xp''x^{-1}$ $(= yp'y^{-1})$.

Le lien entre le problème de conjugaison de type Diffie-Hellman et le problème de conjugaison de type Ko-Lee n'est pas sans rappeler celui entre le problème de Diffie-Hellman et le problème du logarithme discret.

Problème 2.2.9. (problème de conjugaison multiple simultanée) : Soit H un sous-groupe de B_n, $(a_i)_{i\in I}$ un ensemble générateur donné de H et x un élément de B_n. Etant donnés les couples $(b_i = xa_ix^{-1}, a_i)$ pour $i \in I$, déterminer un élément y dans B_n vérifiant $b_i = ya_iy^{-1}$ pour $i \in I$.

Les groupes de tresses sont sans torsion, c'est-à-dire que toute puissance non nulle d'une tresse non triviale est non triviale. Ainsi, il est possible d'introduire un problème sur l'extraction des racines. En pratique, ce problème semble plus difficile que le problème de conjugaison [37].

Problème 2.2.10. (problème des racines) : Etant donnés un entier naturel k et un élément p qui soit une puissance $k^{\text{ième}}$ dans B_n, déterminer un élément $s \in B_n$ vérifiant $s^k = p$.

Il existe d'autres problèmes algorithmiques liés au problème de conjugaison, notamment le problème de Markov et le problème des cyclages. Le Chapitre 7 donne une présentation et une résolution du problème des cyclages.

Une intéressante variante est toutefois à signaler : c'est le problème de décomposition. Il peut de façon analogue être dérivé en problème de décomposition de type Ko et Lee, problème de décomposition de type Diffie-Helmann, problème de décomposition multiple simultanée. La sécurité relative à cette primitive est encore mal évaluée même si elle a des liens avec le problème de conjugaison.

Problème 2.2.11. (problème de décomposition) : Etant donnés deux éléments x, y tels que $y = axa'$, avec a, a' deux éléments de B_n, déterminer $b, b' \in B_n$ vérifiant $y = bxb'$.

La structure de morphisme introduite par la conjugaison n'est plus présente ; ainsi, la difficulté s'apparente davantage à un problème de réécriture.

2.3 Exemples de protocoles

Depuis l'introduction des groupes de tresses comme support pour la cryptographie, de nombreux travaux ont réalisé une gamme variée de cryptosystèmes. Parmi ceux non développés ici, il y a entre autres :

Echange de clés : Anshel, Anshel et Goldfeld [1], basé sur le problème de conjugaison multiple simultanée.

Echange de clés multi-parties : Lee, Lee et Lee [30].

Authentification : Sibert, Dehornoy et Girault [37, 38], basé sur le problème de conjugaison de type Diffie-Hellman et le problème de conjugaison. Kim et Kim [26], basé sur une autre variante du problème de conjugaison.

Signature : Ko, Choi, Cho et Lee [27], basé sur une autre variante du problème de conjugaison.

2.3.1 Echange de clés et cryptosystème à clé publique

Nous présentons les protocoles de Ko, Lee, Cheon, Han, Kang et Park [28]. Ils sont basés sur le problème de conjugaison de type Diffie-Hellman. Les deux sous-groupes de B_n choisis sont :

Soit $l, r \in \mathbb{N}$ tel que $l + r \leq n$, typiquement $l = r = \lfloor \frac{n}{2} \rfloor$ on pose :

LB_l	le sous-groupe de B_n concernant les l premiers générateurs
RB_r	le sous-groupe de B_n concernant les r derniers générateurs

Les sous-groupes LB_l et RB_r commutent.

Protocole 2.3.1. (Echange de clés Ko-Lee [28]) Les données publiques sont deux entiers l et r et une tresse p de B_{l+r}.

- Choix des clés :
  - Alice choisit un élément secret s_A dans LB_l
  - Bob choisit un élément secret s_B dans RB_r
- Echange :
  - Alice calcule et transmet $y_A = s_A p s_A^{-1}$
  - Bob calcule et transmet $y_B = s_B p s_B^{-1}$

- Calcul de la clé commune :
  - Alice calcule $K = s_A y_B s_A^{-1}$
  - Bob calcule $K = s_B y_A s_B^{-1}$

Directement dérivé de ce protocole, les mêmes auteurs proposent le cryptosystème suivant où le symbole $\oplus$ désigne l'addition bit à bit et h est une fonction de hachage supposée idéale, c'est-à-dire sans collision et à sens unique, du groupe de tresses dans l'espace des messages.

Protocole 2.3.2. (Cryptosystème à clé publique [28]) Les données publiques sont deux entiers l et r et une tresse p de B_{l+r}.

- Choix des clés :
  - Alice choisit un élément secret s dans LB_l
  - La clé publique d'Alice est (p, p') avec $p' = sps^{-1}$, sa clé secrète est s
- Chiffrement : Bob veut envoyer le message $m \in \{0,1\}^k$ à Alice
  - Bob tire un élément r de RB_r au hasard
  - Bob calcule $x = rpr^{-1}$ et $m' = h(rp'r^{-1}) \oplus m$, et transmet (x, m')
- Déchiffrement :
  - Alice calcule $m = h(sxs^{-1}) \oplus m'$

2.3.2 Authentification

Les deux protocoles d'authentification suivants sont à divulgation nulle de connaissance dans un cadre idéal. Ils consistent en un schéma en trois passes répété plusieurs fois.

Le premier protocole conserve de bonnes propriétés de divulgation nulle de connaissance dans une implémentation pratique ; cependant il nécessite un oracle aléatoire, représenté par une fonction de hachage idéale h. Il est basé sur le problème de conjugaison.

Protocole 2.3.3. (Protocole d'authentification basé sur le problème de conjugaison [37]) Les données publiques sont un entier n et une tresse p dans B_n.

- Choix des clés :
  - Alice choisit un élément secret s dans B_n
  - La clé publique d'Alice est (p, p') avec $p' = sps^{-1}$, sa clé secrète est s
- Phase d'authentification : engagement
  - Alice tire deux éléments r_0 et r_1 dans B_n au hasard
  - Alice calcule et envoie $x_0 = h(r_0 p' r_0^{-1})$ et $x_1 = r_1 p r_1^{-1}$ à Bob
- Phase d'authentification : défi
  - Bob tire un bit aléatoire ε et l'envoie à Alice
- Phase d'authentification : réponse
  - Si $\varepsilon = 0$, Alice envoie $y_0 = r_0$ et $y_1 = r_1$ à Bob, et Bob vérifie $h(y_0 p' y_0) = x_0$ et $y_1 p y_1^{-1} = x_1$
  - Si $\varepsilon = 1$, Alice envoie $y = r_0 s r_1^{-1}$ à Bob, et Bob vérifie $h(y x_1 y) = x_0$

Ce protocole repose sur le fait qu'Alice est capable de résoudre les problèmes de conjugaisons (p', x_0), (p, x_1) et (x_1, x_0). Le deuxième protocole repose sur le problème de conjugaison et le problème des racines.

Protocole 2.3.4. (Protocole d'authentification basé sur le problème de conjugaison et le problème des racines [38]) Les données publiques sont un entier n et une tresse p dans B_n.

- Choix des clés :
  - Alice choisit un élément secret s dans B_n et calcule $p = s^2$.
  - La clé publique d'Alice est p, sa clé secrète est s
- Phase d'authentification : engagement
  - Alice tire un élément r dans B_n au hasard
  - Alice calcule et envoie $x = rpr^{-1}$ à Bob
- Phase d'authentification : défi
  - Bob tire un bit aléatoire ε et l'envoie à Alice
- Phase d'authentification : réponse
  - Si $\varepsilon = 0$, Alice envoie $y = r$ à Bob, et Bob vérifie $x = ypy^{-1}$
  - Si $\varepsilon = 1$, Alice envoie $y = rsr^{-1}$ à Bob, et Bob vérifie $x = y^2$

Ce protocole repose sur le fait qu'Alice est capable de résoudre le problème de conjugaison (p, x) et le problème des racines $(2, x)$.

2.4 Attaques existantes

Les techniques diverses et variées permettant d'attaquer les schémas basés sur les tresses peuvent être classifiées en trois familles : celles qui travaillent dans la classe de conjugaison et cherchent à résoudre le problème de conjugaison, celles qui exploitent une différence de longueur ou plus généralement une différence de complexité structurelle entre les deux conjugués, et enfin celles qui utilisent une représentation linéaire.

Ces attaques sont essentiellement des attaques contre les problèmes et non contre les protocoles. Ainsi c'est le problème de conjugaison et ses variantes qui sont étudiés.

2.4.1 Classe de conjugaison

L'attaque par excellence consiste à résoudre pratiquement le problème de conjugaison. Plusieurs améliorations ont été apportées aux travaux de Garside [20, 16, 18, 21] :

• Le travail est réalisé dans un sous-ensemble fini de la classe de conjugaison, le *Super Summit Set* : c'est l'ensemble des éléments de la classe ayant une longueur canonique minimale. Il ne dépend pas d'un élément mais seulement d'une classe de conjugaison, ainsi deux éléments conjugués ont le même Super Summit Set. Le seul problème algorithmique contraignant dans la construction d'un tel ensemble est sa taille. Toutes les opérations ont une complexité polynomiale abordable. L'hypothèse est que la taille du Super Summit Set est exponentielle en le nombre de brins et polynomiale en la longueur de l'élément considéré.

Conjecture 2.4.1. *[6] Soit n un entier fixé et soit a une tresse positive de longueur l. Il existe une borne supérieure sur la taille du Super Summit Set de a qui soit polynomiale en l.*

• Une amélioration du Super Summit Set est un de ses sous-ensembles : l'*Ultra Summit Set*. Cet ensemble est l'union des orbites cycliques dans le Super Summit Set pour

l'opérateur *cyclage* (Définition 1.2.34). Expérimentalement, sa taille semble linéaire en la longueur du mot initial.

Remarque 2.4.2. [14, 37] Il semble sans espoir d'énumérer tout le Super Summit Set dans B_{50} pour des mots de longueur 500, tandis qu'il est annoncé que l'Ultra Summit Set est accessible dans B_{100} pour des mots de longueur 1000.

2.4.2 Longueur et complexité

L'objectif de ce genre de méthode n'est pas de résoudre le problème de conjugaison mais de déterminer une heuristique, le plus souvent probabiliste, permettant d'attaquer un schéma : il suffit que la probabilité de succès ne soit pas négligeable pour mettre en danger un schéma.

Le principe partagé par ces attaques est d'essayer de retrouver un conjugant à partir de (x, x'). Considérant que x' est dérivé de x, on travaille de façon itérative à minimiser la "complexité" de conjugués de la forme $t^{-1}x't$.

• [23, 19, 29] propose une simple vérification si $t^{-1}x't$ est égal à x.

• [22] l'approche est similaire, mais les auteurs rajoutent une étape supplémentaire. Si le secret restant entre $t^{-1}x't$ et x est de longueur canonique au plus 1, ils proposent une technique pour retrouver la "permutation" manquante. Cela donne à leur algorithme un taux de réussite de 80 % dans B_{80} pour des tresses de longueur canonique 12.

Remarque 2.4.3. Ce type d'attaque est particulièrement efficace sur le problème de conjugaison multiple simultanée et donc sur le schéma d'échange de clé de [1]. Un taux de réussite non négligeable est avancé dans B_{50} sur ce schéma.

2.4.3 Approche linéaire

Cette dernière technique consiste à résoudre les problèmes difficiles non dans les groupes de tresses, mais dans une représentation linéaire. Il existe plusieurs représentations linéaires des groupes de tresses ; deux sont utilisées :

• La *représention de Burau* n'est pas fidèle pour $n \geq 5$ mais possède un noyau petit : c'est à dire, deux tresses peuvent être représentées par la même matrice. Mais la probabilité que deux tresses différentes aient la même image est négligeable. Cependant, il n'y a pas de raison que le conjugant matriciel trouvé appartienne à l'image du groupe de tresses, voir [29].

$$\rho \ : \ B_n \ \longrightarrow \ GL_n(\mathbb{Z}[t^{\pm 1}]), \quad \rho(\sigma_i) = \begin{pmatrix} Id_{i-1} & & & \\ & 1-t & t & \\ & 1 & 0 & \\ & & & Id_{n-i-1} \end{pmatrix}$$

Exemple 2.4.4. Dans B_3 :

$$\rho(\sigma_1\sigma_2\sigma_1) \ = \ \begin{pmatrix} 1-t & t(1-t) & t^2 \\ 1-t & t & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Remarque 2.4.5. La représentation de Burau est utilisée dans [27, 26] pour résoudre pratiquement le problème décisionnel de conjugaison.

• La *représentation de Lawrence-Krammer* est fidèle mais encore une fois, il est difficile de ramener le conjugant matriciel dans le groupe de tresses :

$$B_n \longrightarrow GL_{\frac{n(n-1)}{2}}(\mathbb{Z}[t^{\pm 1}, p^{\pm 1}])$$

Remarque 2.4.6. Les auteurs de [8] proposent un algorithme basé sur la représentation de Lawrence-Krammer de faible complexité pour résoudre le problème de conjugaison de type Diffie-Hellman dans les groupes de tresses. Ils exploitent la structure creuse des matrices de cette représentation ; pour les tailles considérées, la recherche ne serait pas infaisable pour les instances proposées dans [28]. D'autre part, sa complexité serait de l'ordre de 2^{100} dans B_{90} pour des tresses de longueur canonique 12.

2.5 Problématique de la sécurité

Les attaques présentées semblent très dangereuses et presque tous les schémas proposés ont été attaqués pour les valeurs suggérées des paramètres. Cependant, il est à souligner deux points essentiels : ces attaques sont efficaces sur des variantes du problème de conjugaison et elles utilisent une méthode spécifique de génération aléatoire des clés.

A l'image de la richesse structurelle des groupes des tresses, la problématique de la sécurité de la cryptographie basée sur les tresses repose sur de nombreux paramètres qui ne sont pas tous liés entre eux. Nous proposons une approche en deux temps. Tout d'abord, nous présentons les aspects de sécurité inhérents aux schémas cryptographiques en général. Ensuite, nous spécifions davantage les paramètres liés à la génération aléatoire de clés.

2.5.1 Présentation

• Le mode opératoire de la cryptographie nécessite la gestion de données, et en particulier de données répondant à des critères de taille ou de forme. Afin de pallier à l'infinitude des groupes de tresses, il est proposé dans [38] de définir un *sous-ensemble fini*, $\mathbb{A}^{\leq l}$: ensemble des mots s'écrivant sur l'alphabet $\mathbb{A}$ et de longueur inférieure à l. On définit $B_n^{\leq l}$ comme l'ensemble de tresses à n brins qui possèdent une écriture dans $\mathbb{A}^{\leq l}$.

Nous avons plusieurs solutions pour le choix de $\mathbb{A}$: l'ensemble des générateurs et leurs inverses, ou encore l'ensemble des éléments simples et leurs inverses. Nous avons déjà un premier résultat quant à la taille de ce type d'ensemble :

Théorème 2.5.1. *[28] Considérant le groupe B_n et la longueur canonique l_c, nous avons :*

$$card\left(B_n^{\leq l_c}\right) \geq \left(\left\lfloor \frac{n-1}{2} \right\rfloor !\right)^{l_c}$$

• Considérant une instance de conjugaison $(x, x' = axa^{-1})$, il n'y a jamais unicité du conjugant. En particulier, tout élément $b \in \left\{ax^i\Delta^{k\mathcal{E}}; i, k \in \mathbb{Z}\right\}$ est aussi solution. Ceci n'a pas d'incidence majeure en cryptographie puisque la recherche de n'importe quel conjugant suffit à casser l'instance.

• L'ensemble des problèmes présentés dans les groupes de tresses sont décidables [37]. Mais comme nous l'avons vu, les algorithmes existants ne sont en général pas assez efficaces pour les paramètres utilisés en cryptographie.

• Le problème de conjugaison multiple simultanée est particulièrement affecté par les attaques existantes ; en particulier le schéma de [1] requiert une révision avant d'être utilisé sur les tresses. La structure de morphisme de la conjugaison et le fait qu'un secret soit commun à plusieurs instances de conjugaisons réduisent grandement sa difficulté en pratique.

• Il est inévitable, lors des échanges dans les protocoles, de transformer les mots échangés afin que les opérations effectuées ne puissent pas simplement être lues. Deux principes sont utilisés : des fonctions de hachage, des systèmes de réécriture. En effet, il est aisé d'effectuer un brouillage des opérations en utilisant des formes normales ou reduites. Notons que l'utilisation de fonctions de hachage dans les groupes de tresses fait nécessairement appel à une forme normale. Certains schémas proposés sont utilisables uniquement avec des formes réduites, mais il ne faut pas négliger que les formes normales existent et qu'un attaquant peut les utiliser.

2.5.2 Générateur aléatoire

Il est clair que toutes les instances de conjugaison ne sont pas également sûres. La sécurité d'un schéma cryptographique n'est pas conditionnée par le fait que toutes ses instances soient difficiles, mais par le fait qu'il existe un sous-ensemble définissable et assez grand, d'instances qui le soient. Le problème décisionnel de conjugaison illustre cela dans l'Exemple 2.5.2.

Exemple 2.5.2. Le problème décisionnel de conjugaison est un problème qui s'avère facile sur de nombreuses instances et difficile sur d'autres. En effet, considérant deux tresses, il suffit que l'infimum ou le supremum de leur Super Summit Set soit différents pour pouvoir conclure qu'ils ne sont pas conjugués. En revanche, pour d'autres instances, le seul algorithme connu est la résolution du problème de conjugaison en utilisant la construction du Super Summit Set ou de l'Ultra Summit Set. Ceci ne tient pas compte du fait, qu'en cryptographie, la méthode de la Remarque 2.4.5 est suffisante pour résoudre le problème décisionnel de conjugaison.

A ce jour, aucun critère connu ne permet de choisir une instance du problème de conjugaison ou de ses variantes de sorte à garantir un niveau de difficulté du problème ; il n'y a que des spécifications de bon sens.

• Un avis partagé est que la difficulté du problème de conjugaison sur une instance repose sur le choix de la tresse porteuse x [28, 7, 14]. Ainsi, un bon choix de x rendrait les problèmes difficiles pour un bon nombre de conjugants. Cela pourrait au moins être le cas pour les attaques dans la classe de conjugaison.

Définition 2.5.3. Une tresse est dite *pure* si la permutation qu'elle opère sur les brins est triviale.

• Les auteurs de [29] recommandent de prendre des instances avec des tresses pures. Si x est choisie pure, on remarque que la tresse conjuguée x' est aussi pure quel que soit le conjugant. Ainsi, aucune information sur la permutation associée au conjugant ne filtre de l'instance (x, x').

• Les attaques basées sur la longueur ou la complexité du conjugué semblent génériques ; en réalité, elles reflètent uniquement la façon dont les tresses sont générées.

Il est évident, considérant la loi de concaténation des mots, que la longueur d'un produit sera en général assez proche de la somme des longueurs des mots intervenants, ou encore que l'on peut retrouver un grand préfixe de a dans $x' = axa^{-1}$. Cela n'est pas automatique, il est possible de se prémunir contre de telles attaques en choisissant judicieusement les clés. Sibert, dans [37], propose une construction symétrique de x et x' ce qui les rend indistingables en terme de longueur ou de complexité.

Exemple 2.5.4. On peut se prémunir de l'attaque de [22] en prenant x' et x dans le Super Summit Set de x et en veillant à ce que le secret soit de longueur canonique supérieure ou égale à 2 [37].

• L'attaque basée sur l'Ultra Summit Set peut, elle aussi, être contournée en choisissant une tresse x telle que son Ultra Summit Set soit de grande taille.

• Sibert, dans [37], propose d'utiliser de petites permutations, à savoir des éléments simples de courte longueur. La longueur moyenne d'un facteur canonique dans la présentation d'Artin est de $\frac{n(n-1)}{4}$; il propose de la réduire à $\mathcal{O}(n)$. Ceci apporte deux intérêts vérifiés expérimentalement : le premier est que cela limite le phénomène de préfixe et le deuxième est que cela semble favoriser une grande taille du Super Summit Set.

• Considérant l'état de l'art, il semble infaisable de résoudre le problème de conjugaison pour des instances bien choisies dans le groupe de tresses à 50 brins et des mots de longueur 1000 [14].

Voici maintenant quatre générateurs aléatoires. Les trois premiers correspondent directement aux types de représentations d'une tresse. Il y a l'écriture utilisant les atomes et celle utilisant les éléments simples.

Algorithme 2.5.5. GA-Mots (l, n)

Entrée :	Soit $l, n \in \mathbb{N}$. (l : longueur, n : nombre de brins)
	Faire l tirages aléatoires sur l'ensemble des atomes et de leurs inverses en évitant les tirages consécutifs de type (g, g^{-1})
Sortie :	le produit de ces l éléments

Algorithme 2.5.6. GA-MotsP (l, n)

Entrée :	Soit $l, n \in \mathbb{N}$. (l : longueur, n : nombre de brins)
	Faire l tirages aléatoires sur l'ensemble des atomes
Sortie :	le produit de ces l éléments

Algorithme 2.5.7. GA-CANO (p, n)

Entrée :	Soit $p, n \in \mathbb{N}$. (p : longueur canonique, n : nombre de brins)
	Faire un tirage aléatoire sur l'ensemble des entiers naturels inférieurs à l'exposant ($\mathcal{E}$) du groupe : r.
	Faire p tirages aléatoires sur l'ensemble des éléments simples : $(s_1, \cdots, s_p) \quad \rightarrow y = s_1 \cdots s_p$.
	Tant que $l_c(y) < p$ faire
	Soit $k \in [1, p]$ et $(s_1, \cdots, s_k)$ la suite normale à gauche associée à y
	Faire $p - k$ tirages aléatoire sur l'ensemble des éléments simples : $(s_{k+1}, \cdots, s_p) \quad \rightarrow y = s_1 \cdots s_p$
	FinTantQue
	Soit $(s_1, \cdots s_p)$ la suite normale à gauche associée à y.
Sortie :	$[r, [s_1, \cdots, s_p]]$

Le générateur aléatoire suivant est issu de [37]. Il a pour objectif de masquer la différence de complexité entre les deux conjugués ; cela permet de se mettre hors de portée de certaines attaques spécifiques à la complexité [19, 22, 23, 29]. L'auteur précise que l'aspect symétrique n'est pas le but véritable ; ce dernier est de faire en sorte que la distance dans le Super Summit Set entre les conjugués soit grande.

On remarque que dans le cas de ce générateur, c'est la personne qui génère le secret qui doit aussi générer la tresse porteuse. Ainsi, certains protocoles demanderaient à être révisés pour pouvoir utiliser ce générateur aléatoire.

Algorithme 2.5.8. GA-COMPC (p, n)

Entrée :	Soit $p, n \in \mathbb{N}$. (p : longueur canonique, n : nombre de brins)
	Soit $a :=$ GA-CANO(p, n)
	Quitte à utiliser le Théorème 1.2.36, on suppose que $a \in SSS(a)$
	Effectuer une suite aléatoire de $2p$ opérations de { *cyclage*, *décyclage* } sur a on obtient $b \in SSS(a)$ et c tel que $b = cac^{-1}$.
	Soit $t_1 :=$ GA-CANO(p, n) et $t_2 :=$ GA-CANO(p, n)
Sortie :	$(p = t_1 a t_1^{-1}, p = t_2 b t_2^{-1})$ de secret $s = t_2 c t_1^{-1}$

Chapitre 3

Propriétés de divisibilité

Nous présentons, dans ce chapitre, plusieurs résultats de divisibilité en relation avec la forme Δ-normale dans les groupes de Garside. Ces résultats, qui sont nouveaux ou généralisent d'autres déjà existants, sont des outils de base utilisés dans la suite.

Comme cette étude en donne un aperçu, la majeure partie des opérations ou manipulations sont faites dans le monoïde et non dans le groupe. En fait, nous avons plus de moyens pour travailler dans le monoïde, et donc, nous nous y ramenons régulièrement. La forme Δ-normale est propice à cette approche et les éléments simples sont un outil adapté pour manipuler les éléments. Ainsi, la plupart des résultats présentés ici concernent des mots positifs ou des éléments du monoïde.

Nous nous plaçons dans le groupe de Garside G de monoïde de Garside M et d'élément de Garside Δ. On rappelle que S est l'ensemble des éléments simples de M.

3.1 Notations

Nous attirons l'attention du lecteur sur la notation suivante qui diffère légèrement de celle donnée en [31] :

Définition 3.1.1. Soit x un élément du monoïde M ; on appelle *dual* de x l'élément $x^\star$ défini par $xx^\star = \Delta^{\sup(x)}$.

Afin de simplifier les notations ultérieures, nous introduisons une petite redondance. La fonction suivante peut être vue comme la restriction de la notation * sur l'ensemble des éléments simples. On rappelle que le *supremum* d'un élément simple différent de l'élément neutre est 1.

Notation 3.1.2. $\partial : \begin{array}{ccc} S & \longrightarrow & S \\ x & \longmapsto & x^\star = x^{-1}\Delta \end{array}$

Il existe un lien entre τ et ∂ :

Proposition 3.1.3. *Dans S, nous avons $\partial^2 = \tau$ et $\tau \circ \partial = \partial \circ \tau$*

Preuve : Soit x un élément simple ; la définition de ∂ donne :

$$\partial(\partial(x)) = \partial(x^{-1}\Delta) = (\Delta^{-1}x)\Delta = \tau(x)$$

De plus τ étant une puissance de ∂, τ commute avec ∂.

□

3.2 Propriétés

La première proposition nous donne la forme Δ-normale de l'inverse d'un élément :

Proposition 3.2.1. *[16] Soit $\Delta^r s_1 s_2 \cdots s_p$ la forme Δ-normale de $x \in G$; alors celle de x^{-1} est donnée par :*

$$x^{-1} = \Delta^{-(r+p)} \tau^{-r-p}\left(\partial(s_p)\right) \tau^{-r-p+1}\left(\partial(s_{p-1})\right) \cdots \tau^{-r-1}\left(\partial(s_1)\right)$$

Corollaire 3.2.2. *Soit $x \in M$ de forme Δ-normale $\Delta^r s_1 s_2 \cdots s_p$; alors la forme Δ-normale de son dual est :*

$$x^\star = \partial(s_p) \tau\left(\partial(s_{p-1})\right) \cdots \tau^{p-1}\left(\partial(s_1)\right)$$

De plus, $l_c(x) = l_c(x^{-1}) = l_c(x^\star)$ et $\inf(x^\star) = 0$.

Preuve : D'après la définition du dual, la Proposition 3.2.1 nous donne l'expression voulue car $xx^\star = \Delta^{r+p}$ implique :

$$x^\star = x^{-1}\Delta^{r+p} = \tau^{r+p}\Big(\tau^{-r-p}\left(\partial(s_p)\right) \tau^{-r-p+1}\left(\partial(s_{p-1})\right) \cdots \tau^{-r-1}\left(\partial(s_1)\right)\Big)$$

Comme τ laisse stable $\prec$, il laisse aussi stable $\wedge$; ainsi l'expression donnée est aussi sous forme Δ-normale. Les relations supplémentaires en découlent.

□

Corollaire 3.2.3. *Soit $s_1 s_2 \cdots s_k$ une écriture de $x \in M$ comme produit d'éléments simples ; nous avons :*

$$x^\star = \partial(s_k) \tau\left(\partial(s_{k-1})\right) \cdots \tau^{k-1}\left(\partial(s_1)\right) \Delta^{\sup(x)-k}$$

Preuve : Il suffit d'effectuer $xx^\star$. En partant du produit, on utilise la définition de ∂ pour faire apparaître successivement k éléments Δ que l'on décale vers la droite en utilisant la définition de τ.

□

La propriété qui suit est essentielle dans la suite ; elle met en relation la division à gauche d'un élément et la division à droite de son dual. Par exemple : la connaissance d'un multiple à droite d'un élément nous donne un diviseur à droite de son dual.

Lemme 3.2.4. *Soit $a, b \in M$ tels que $a = a_1 a_2 \cdots a_k$ et $b = b_1 b_2 \cdots b_k$ soient des produits d'éléments simples ; nous avons :*

$$a \prec b \quad \Leftrightarrow \quad \partial(a_k)\tau(\partial(a_{k-1})) \cdots \tau^{k-1}(\partial(a_1)) \succ \partial(b_k)\tau(\partial(b_{k-1})) \cdots \tau^{k-1}(\partial(b_1))$$

Preuve : On utilise les définitions de $\prec$, $\succ$, τ et ∂. C'est une généralisation du fait :

$$\forall p, q \in S, \qquad p \prec q \quad \Leftrightarrow \quad \partial(p) \succ \partial(q)$$

.

$$
\begin{aligned}
a \prec b \;&\Leftrightarrow\quad a_1 a_2 \cdots a_k \prec b_1 b_2 \cdots b_k \\
&\Leftrightarrow\quad a_k^{-1} a_{k-1}^{-1} \cdots a_1^{-1} b_1 b_2 \cdots b_k \;\in M \\
&\quad \text{↥ définition de } \prec \\
&\Leftrightarrow\quad \partial(a_k)\Delta^{-1}\partial(a_{k-1})\Delta^{-1} \cdots \partial(a_1)\Delta^{-1} b_1 b_2 \cdots b_k \;\in M \\
&\quad \text{↥ définition de } \partial : \forall q \in S,\ q\partial(q) = \Delta \\
&\Leftrightarrow\quad \partial(a_k)\tau(\partial(a_{k-1})) \cdots \tau^{k-1}(\partial(a_1))\Delta^{-k} b_1 b_2 \cdots b_k \;\in M \\
&\quad \text{↧ définition de } \tau \\
&\Leftrightarrow\quad \partial(a_k)\tau(\partial(a_{k-1})) \cdots \tau^{k-1}(\partial(a_1))\tau^{k-1}(\Delta^{-1} b_1)\tau^{k-2}(\Delta^{-1} b_2) \cdots \Delta^{-1} b_k \;\in M \\
&\Leftrightarrow\quad \partial(a_k)\tau(\partial(a_{k-1})) \cdots \tau^{k-1}(\partial(a_1))\tau^{k-1}(\partial(b_1)^{-1})\tau^{k-2}(\partial(b_2)^{-1}) \cdots \partial(b_k)^{-1} \;\in M \\
&\quad \text{↥ définition de } \partial \\
&\Leftrightarrow\quad \partial(a_k)\tau(\partial(a_{k-1})) \cdots \tau^{k-1}(\partial(a_1)) \succ \partial(b_k)\tau(\partial(b_{k-1})) \cdots \tau^{k-1}(\partial(b_1)) \\
&\quad \text{↥ } \tau \text{ est un morphisme et définition de } \succ
\end{aligned}
$$

□

Proposition 3.2.5. *Soit a, b deux éléments de M, nous avons :*

$$a \prec b \quad \Rightarrow \quad \sup(a) \leq \sup(b) \tag{3.1}$$

$$a \prec b \quad \Leftrightarrow \quad a^\star \Delta^{\sup(b)-\sup(a)} \succ b^\star \tag{3.2}$$

Preuve : Afin de traiter (3.1), discutons suivant les deux cas suivants :

- Soit $\inf(a) = 0$: nous procédons par l'absurde et supposons que $l_c(a) = \sup(a) > \sup(b)$. Soit $a_1 a_2 \cdots a_{\sup(a)}$ et $b_1 b_2 \cdots b_{\sup(b)}$ les formes Δ-normales de a et b.

  $$
  \begin{aligned}
  a \prec b \quad &\Rightarrow && a \prec 1^{\sup(a)-\sup(b)} b \\
  &\Rightarrow_{\text{Lemme 3.2.4 et Corollaire 3.2.3}} && a^\star \succ b^\star \Delta^{\sup(a)-\sup(b)} \\
  &\Rightarrow && a^\star \succ \Delta^{\sup(a)-\sup(b)} \\
  &\Rightarrow && \inf(a^\star) \geq \sup(a) - \sup(b) > 0
  \end{aligned}
  $$

  Ce qui est contradictoire avec le Corollaire 3.2.2 ; alors $\sup(a) \leq \sup(b)$.

- Soit $\inf(a) > 0$: comme $a \prec b \Rightarrow \inf(a) \leq \inf(b)$; nous pouvons appliquer le premier point après avoir simplifié à gauche la relation par $\Delta^{\inf(a)}$.

(3.2) : Considérant a et b écrits sous forme Δ-normale et le résultat (3.1), il suffit d'appliquer le Lemme 3.2.4 à : $1^{\sup(b)-\sup(a)} a \prec b$.

□

Les propriétés suivantes sont le cœur de l'algorithme présenté au chapitre 4. La première nous permettra de construire un multiple de même supremum que le secret, et la seconde sera utilisée pour affiner le multiple.

Proposition 3.2.6. *Soit a, b, c des éléments de M tels que $\sup(b) \geq \sup(a)$ et soit $b_1 b_2 \cdots b_{\sup(b)}$ la forme Δ-normale de b. Nous avons :*

$$a \prec cb \quad \Rightarrow \quad a \prec cb_1 \cdots b_{\sup(a)}$$

En particulier $\quad a \prec b \quad \Rightarrow \quad a \prec b_1 \cdots b_{\sup(a)}$

Preuve : Nous procédons par récurrence sur $\sup(a)$. Si $a = a_1$ alors le Lemme 1.2.22 donne le résultat. Supposons que la relation est vraie pour $a = a_1 \cdots a_k$ avec $1 \leq k < \sup(a)$, alors :

$$a_1 \cdots a_{k+1} \prec cb \qquad \Rightarrow \qquad a_{k+1} \prec \underbrace{(a_1 \cdots a_k)^{-1} c b_1 \cdots b_k}_{\in M} b_{k+1} \cdots b_{\sup(b)}$$
$$\Rightarrow_{\text{Lemme 1.2.22}} \qquad a_{k+1} \prec (a_1 \cdots a_k)^{-1} c b_1 \cdots b_{k+1}$$

Le cas $k+1$ étant vrai, la récurrence est terminée.

□

Corollaire 3.2.7. *Soit $a, b, c \in M$ tel que $\sup(b) \geq \sup(a)$ et soit $a_1 a_2 \cdots a_{\sup(a)}$ et $b_1 b_2 \cdots b_{\sup(b)}$ les formes Δ-normales de a et b. Nous avons :*

$$\forall k \in [1, \sup(a)], \qquad \begin{cases} a \prec cb \\ a_1 \cdots a_k \prec c \end{cases} \quad \Rightarrow \quad a \prec cb_1 \cdots b_{\sup(a)-k}$$

Corollaire 3.2.8. *Soit $C = c_1 \cdots c_l \in M$ un produit d'éléments simples et $C_1 \cdots C_{\sup(C)}$ la forme Δ-normale de C. Nous avons :*

$$\forall k \in [1, \sup(C)], \qquad c_k \cdots c_l \succ C_k \cdots C_{\sup(C)}$$

En particulier, si $a = cb$ avec $a, b, c \in M$ et $a_1 \cdots a_{\sup(a)}$ la forme Δ-normale de a, alors :

$$b \succ a_{\sup(c)+1} a_{\sup(c)+2} \cdots a_{\sup(a)}$$

Preuve : Le cas $k = 1$ est vrai, on a même l'égalité. Supposons $k \in [2, \sup(C)]$, nous avons :

$$c_1 \cdots c_{k-1} \prec c_1 \cdots c_l = C_1 \cdots C_{\sup(C)} \quad \Rightarrow_{\text{Proposition 3.2.6}} \quad c_1 \cdots c_{k-1} \prec C_1 \cdots C_{k-1}$$

$$\text{Alors } c_k \cdots c_l = \underbrace{(c_1 \cdots c_{k-1})^{-1} C_1 \cdots C_{k-1}}_{\in M} C_k \cdots C_{\sup(C)}$$

□

Lemme 3.2.9. *Soit $a_1 \cdots a_{\sup(a)}$ et $b_1 \cdots b_{\sup(b)}$ les formes Δ-normales de a et b. Nous avons :*

$$\forall k \in [1, \sup(a)], \qquad b \succ a \quad \Rightarrow \quad b_k \cdots b_{\sup(b)} \succ a_k \cdots a_{\sup(a)}$$

Preuve : Nous procédons par récurrence sur $k \in [1, \sup(a)]$. Le cas $k = 1$ est vrai. Supposons $b_k \cdots b_{\sup(b)} \succ a_k \cdots a_{\sup(a)}$; alors il existe $m \in M$ tel que

$$m a_k \cdots a_{\sup(a)} = b_k \cdots b_{\sup(b)} \quad \Rightarrow_{\text{Lemme 1.2.22}} \quad b_k \prec m a_k$$

Ce qui donne $b_{k+1} \cdots b_{\sup(b)} \succ a_{k+1} \cdots a_{\sup(a)}$ et termine la récurrence.

□

Proposition 3.2.10. *Soit $a, b, c \in M$. Si $\sup(a) > \sup(b)$, nous avons :*

$$bc \succ a \quad \Rightarrow \quad c \succ a_{\sup(b)+1} \cdots a_{\sup(a)}$$

Preuve : Soit $d = bc$ et soit $d_1 d_2 \cdots d_{\sup(d)}$ son écriture sous forme Δ-normale :

$$d = bc \quad \Rightarrow_{\text{Corollaire 3.2.8}} \quad c \succ d_{\sup(b)+1} \cdots d_{\sup(d)}$$

et si $\sup(a) > \sup(b)$, nous avons

$$d \succ a \quad \Rightarrow_{\text{Lemme 3.2.9}} \quad c \succ d_{\sup(b)+1} \cdots d_{\sup(d)} \succ a_{\sup(b)+1} \cdots a_{\sup(a)}$$

□

Remarque 3.2.11. Le Corollaire 3.2.8 généralise le Lemme 2.9 de [18]. De plus, la Proposition 3.2.6 est une généralisation du Lemme 4.10 de [33], qui donne aussi (3.1).

Chapitre 4

Algorithme de réduction de la conjugaison

Ce chapitre présente un nouvel algorithme de réduction du problème de conjugaison dans les groupes de Garside [31, 32]. Il permet de déterminer un diviseur à gauche et un multiple à droite du secret d'une instance de conjugaison. Le but est naturellement de l'appliquer, dans la suite, aux groupes de tresses.

4.1 Présentation

Considérons une instance de conjugaison, $(x, y = axa^{-1})$, dans le groupe de Garside G, de monoïde M. Le but de notre algorithme est de récupérer des informations sur le secret a. En fait, l'algorithme suivant nous permet de construire un diviseur à gauche et un multiple à droite du secret.

Trouver $(d, m) \in G$ tel que $d \prec a \prec m$

La figure 4.1 permet de visualiser la réduction effective du secret. En effet, ayant déterminé un diviseur à gauche et un multiple à droite du secret, (d, m), l'instance initiale de conjugaison (x, axa^{-1}) est réduite en deux nouvelles instances dont les nouveaux secrets sont de taille plus petite :

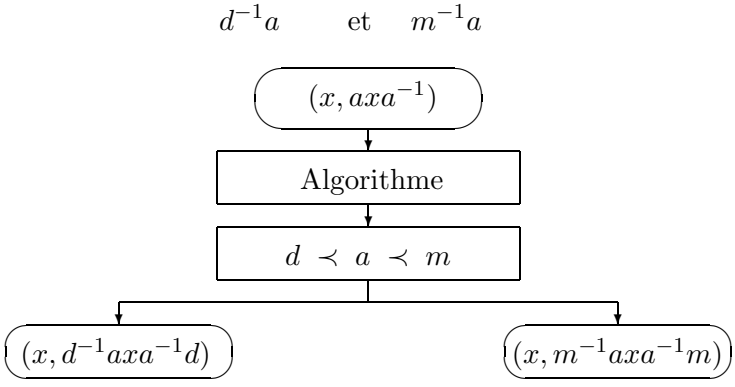

FIG. 4.1 – Réduction d'une instance de conjugaison

Remarque 4.1.1. Nous considérons que garder secrétes certaines informations sur le secret n'améliore pas la sécurité d'une clé et donc, nous les supposons connues pour établir l'algorithme. Ces paramètres sont l'*infimum* et la *longueur canonique* du secret. Nous traitons cet aspect dans le Chapitre 5, Section 5.2.

La réduction que nous obtenons concerne la longueur définie par le nombre de générateurs utilisés dans l'écriture du secret. Cette longueur n'a de sens véritable que dans le monoïde (pas dans le groupe) et la Remarque 4.1.1 permet justement de ramener le travail dans le monoïde. L'algorithme MULTIPLE, présenté dans la section suivante, travaille donc sur des mots positifs et permet de construire un multiple à droite du secret. Une légère variante, l'algorithme DIVISEUR, permet de construire un diviseur à gauche du secret.

Voici, en quelques mots, la description du fonctionnement de MULTIPLE :

L'algorithme est une boucle itérative sur la longueur canonique du secret. Il exploite la dépendance de l'écriture sous forme Δ-normale du conjugué, axa^{-1}, par rapport à celle du secret, a. En effet, pour un k donné, les k premiers éléments simples et les k derniers éléments simples du conjugué sont reliés aux k premiers éléments simples du secret. Cette idée est naturelle, comme on le voit dans l'écriture normalisée de a^{-1} : le j ième élément simple de la forme Δ-normale de a est directement lié au $l_c(a)-j$ ième de a^{-1}. Ainsi, la réduction du conjugué peut se faire en même temps à gauche et à droite.

Considérons un cas simple, soit $a = a_1a_2\cdots a_{l_c(a)}$ sous sa forme Δ-normale. La première étape de l'algorithme est de construire un multiple à droite de a_1 ; la seconde est de construire un multiple à droite de a_1a_2 et ainsi de suite jusqu'à la $l_c(a)$ ième étape. A chaque étape, la construction se fait tout d'abord en considérant les éléments simples du conjugué qui sont à gauche afin d'obtenir un premier multiple. On utilise pour cela un résultat du type :

soit $(c_1, c_2, c_3) \in S^3$ et (s_1, s_2, s_3) une suite normale à gauche, alors :

$$c_1c_2 \prec s_1s_2s_3 \quad \Rightarrow_{\text{Proposition 3.2.6}} \quad c_1c_2 \prec s_1s_2$$

Ensuite, si nous avons un "bon" multiple, on essaye de le réduire afin d'avoir un meilleur multiple. Ainsi, on considère le dual du multiple et les éléments simples du conjugué qui sont à droite dans son écriture normalisée et on utilise un résultat du type :

$$c_1c_2c_3 \succ s_1s_2s_3 \quad \Rightarrow_{\text{Proposition 3.2.10}} \quad c_3 \succ s_3$$

Ayant augmenté la taille du dual du multiple, cela nous donne, au final, un multiple plus petit, donc meilleur.

4.2 Algorithmes Multiple et Diviseur

Format des données d'entrée des algorithmes MULTIPLE et DIVISEUR :

$$\text{ENTRÉE :} \qquad (X, l_1, l_2, \alpha, \beta) \in M \times \mathbb{N}^2 \times \mathbb{Z}^2$$

$$\text{tel que } \exists T, y \in M; \quad \begin{cases} X = \tau^{\alpha}(T) y \tau^{\alpha+\beta}(T^{\star}) \\ l_1 = l_c(y),\ l_2 = l_c(T) \\ \inf(T) = \inf(y) = 0 \end{cases}$$

Algorithme 4.2.1. MULTIPLE $(X, l_1, l_2, \alpha, \beta)$

Entrée :	Soit $X \in M$, $l_1, l_2 \in \mathbb{N}$ et $\alpha, \beta \in \mathbb{Z}$.	
Init :	$A := e$, $Y := \tau^{-\alpha}(X)$, $l := 0$, $cond := vrai$	(m1)
	Calculer la *forme* Δ-*normale* $Y_1 Y_2 \cdots Y_{\sup(Y)}$ de Y	(m2)
	$B := Y_1 Y_2 \cdots Y_{l_2}$	(m3)
Boucle :	Tant que *cond* et $l < l_2$ faire	(m4)
	$l := l + 1$, $A := AY_1$,	(m5)
	$Z := Y_1^{-1} Y \tau^{\beta + l_2 - l}\left(\partial(Y_1)^{-1}\right)$	(m6)
	$Y := Y_1^{-1} Y$	(m7)
	Si $Z \notin M$ alors $cond := faux$	(m8)
	Sinon Calculer la *forme* Δ-*normale* $Z_1 Z_2 \cdots Z_{\sup(Z)}$ de Z	(m9)
	Si $\sup(Z) = l_1 + 2(l_2 - l) + 1$ alors	(m10)
	$A := A\tau^{-\beta - l_2 + l}(Z_{\sup(Z)})^{-1}$	(m11)
	$Z := \tau^{-\beta - l_2 + l}(Z_{\sup(Z)}) Z Z_{\sup(Z)}^{-1}$	(m12)
	FinSi	
	$Y :=$ *forme* Δ-*normale de* Z	(m13)
	FinSi	
	FinTantQue	
	Si $l < l_2$ alors $A := AY_2 Y_3 \cdots Y_{l_2 - l + 1}$ FinSi	(m14)
Sortie :	$A \wedge B$	

Proposition 4.2.2. *Considérons l'algorithme 4.2.1,* MULTIPLE*, avec pour entrée le 5-uplet* $(X, l_1, l_2, \alpha, \beta)$ *tel qu'il existe* $\exists T, y \in M$ *tels que* $X = \tau^{\alpha}(T) y \tau^{\alpha+\beta}(T^{\star})$, $l_1 = l_c(y)$, $l_2 = l_c(T)$ *et* $\inf(T) = \inf(y) = 0$.

Alors la tresse de sortie, m*, vérifie* $T \prec m$ *et* $\sup(T) = \sup(m)$.

La preuve de cette proposition constitue la Section 4.5.

Remarque 4.2.3. En pratique, l'algorithme effectue une sorte de réécriture de X sous la forme :

$$X = \tau^{\alpha}(T) y \tau^{\alpha+\beta}(T^{\star}) \quad \longrightarrow \quad \tau^{\alpha}(m) \tilde{y} \tau^{\alpha+\beta}(m^{\star}) \text{ avec } T \prec m$$

Remarque 4.2.4. Il est possible d'améliorer l'algorithme en transformant la boucle "`Si` $\sup(Z) = l_1 + 2(l_2 - l) + 1$ `alors`" ligne (m10) en "`Tant que` $\sup(Z) \geq l_1 + 2(l_2 - l) + 1$ `faire`". Ceci est justifié par la Proposition 3.2.10. Mais cela complique les preuves sans apporter clairement de meilleurs résultats dans les cas que nous avons traités.

Algorithme 4.2.5. DIVISEUR $(X, l_1, l_2, \alpha, \beta)$

Entrée :	Soit $X \in M$, $l_1, l_2 \in \mathbb{N}$ et $\alpha, \beta \in \mathbb{Z}$.	
Init :	$A := e$, $Y := \tau^{-\alpha}(X)$, $l := 0$, $cond := vrai$	(d1)
	Calculer la *forme* Δ*-normale à droite* $Y_1 Y_2 \cdots Y_{\sup(Y)}$ de Y	(d2)
	$B := Y_{\sup(Y)-l_2+1} \cdots Y_{\sup(Y)}$	(d3)
Boucle :	Tant que *cond* et $l < l_2$ faire	(d4)
	$l := l + 1$, $A := Y_{\sup(Y)} A$,	(d5)
	$Z := \tau^{l-\beta-l_2}\left(\partial^{-1}(Y_{\sup(Y)})^{-1}\right) Y Y_{\sup(Y)}^{-1}$	(d6)
	$Y := Y Y_{\sup(Y)}^{-1}$	(d7)
	Si $Z \notin M$ alors $cond := faux$	(d8)
	Sinon Calculer la *forme* Δ*-normale à droite* $Z_1 Z_2 \cdots Z_{\sup(Z)}$ de Z	(d9)
	Si $\sup(Z) = l_1 + 2(l_2 - l) + 1$ alors	(d10)
	$A := \tau^{\beta+l_2-l}(Z_1)^{-1} A$	(d11)
	$Z := Z_1^{-1} Z \tau^{\beta+l_2-l}(Z_1)$	(d12)
	FinSi	
	$Y :=$ *forme* Δ*-normale à droite* de Z	(d13)
	FinSi	
	FinTantque	
	Si $l < l_2$ alors $A := Y_{\sup(p)-l_2+l+1} \cdots Y_{\sup(Y)} A$ FinSi	(d14)
	$C := \tau^{-\beta}(A \wedge B)$	(d15)
Sortie :	$\Delta^{\sup(C)} C^{-1}$	

Proposition 4.2.6. *Considérons l'algorithme 4.2.5,* DIVISEUR*, avec pour entrée le 5-uplet* $(X, l_1, l_2, \alpha, \beta)$ *vérifiant* $\exists T, y \in M$ *tels que* $X = \tau^{\alpha}(T) y \tau^{\alpha+\beta}(T^{\star})$, $l_1 = l_c(y)$, $l_2 = l_c(T)$ *et* $\inf(T) = \inf(y) = 0$.

Alors la tresse de sortie, d, vérifie $d \prec T$.

Preuve : Considérons les lignes de (d1) à (d15) de l'algorithme 4.2.5. Une preuve similaire à celle de la Proposition 4.2.2 utilisant les notions à droite (forme Δ-normale à droite), nous donne : $C \succ T^{\star}$ et $\sup(C) = \sup(T)$.

La Proposition 3.2.5 permet de conclure :

$$\begin{cases} C \succ T^{\star} \\ \sup(T^{\star}) = \sup(C) \end{cases} \Rightarrow \Delta^{\sup(C)} C^{-1} \prec T$$

□

4.3 Problème de conjugaison - Conj

Nous allons montrer dans cette section comment appliquer les algorithmes MULTIPLE et DIVISEUR sur une instance quelconque de conjugaison afin de déterminer un diviseur à gauche et un multiple à droite du secret.

Rappelons que l'on se place dans le cadre de la Remarque 4.1.1 et donc que l'infimum et la longueur canonique du secret sont connus.

Algorithme 4.3.1. CONJ (x, x', r, p)

Entrée :	Soit $x, x' \in G, \qquad r, p \in \mathbb{Z}$.
Init :	$X := \Delta^{p-\inf(x)} x'$
	$l_1 := l_c(x), \quad l_2 := p$
	$\alpha = \inf(x) - r - p, \quad \beta = -\inf(x)$
	$M := \text{MULTIPLE}(X, l_1, l_2, \alpha, \beta)$
	$D := \text{DIVISEUR}(X, l_1, l_2, \alpha, \beta)$
Sortie :	$(\Delta^r D, \Delta^r M)$

Proposition 4.3.2. *Soit a, x deux éléments de G. Considérons l'instance de conjugaison $(x, x' = axa^{-1})$ et notons (d, m) la sortie de l'algorithme* CONJ *ayant en entrée le 4-uplet $(x, x', \inf(a), l_c(a))$. Alors d est un diviseur à gauche de a et m est un multiple à droite de a de même supremum : $d \prec a \prec m$.*

Preuve : Posons $T = \Delta^{-\inf(a)} a$ et $z = \Delta^{-\inf(x)} x$ donc $\inf(T) = 0 = \inf(z)$ et $\sup(T) = \sup(a) - \inf(a)$. On remarque que $a^{-1} = T^\star \Delta^{-\sup(a)}$. Maintenant considérons x' :

$$\begin{aligned} x' &= axa^{-1} \\ &= \Delta^{\inf(a)} T \Delta^{\inf(x)} z T^\star \Delta^{-\sup(a)} \\ &= \Delta^{\inf(x)-l_c(a)} \tau^{\inf(x)-\sup(a)}(T) \tau^{-\sup(a)}(z) \tau^{-\sup(a)}(T^\star) \end{aligned}$$

Posons $y = \tau^{-\sup(a)}(z)$, donc $\inf(z) = \inf(y) = 0$. Considérons le 5-uplet $(X, l_1, l_2, \alpha, \beta)$ suivant :

$$\begin{cases} X = & \Delta^{l_c(a)-\inf(x)} x' & = \tau^{\inf(x)-\sup(a)}(T) y \tau^{-\sup(a)}(T^\star) \\ l_1 = & l_c(x) & = l_c(y) \\ l_2 = & l_c(a) & = l_c(T) \\ \alpha = & \inf(x) - \inf(a) - l_c(a) & = \inf(x) - \sup(a) \\ \beta = & -\inf(x) & \end{cases}$$

Ce 5-uplet est entièrement défini par les données d'entrée : $x, x', \inf(a)$ et $l_c(a)$. De plus, il vérifie le format des données d'entrée des algorithmes MULTIPLE et DIVISEUR. D'après les Propositions 4.2.2, et 4.2.6, nous avons :

$$D \prec T \prec M \quad \text{et} \quad \sup(T) = \sup(M)$$

Par définition de T on obtient le résultat : $d \prec a \prec m$ et $\sup(a) = \sup(m)$.

□

4.4 Complexité

Proposition 4.4.1. *La complexité des algorithmes* MULTIPLE, DIVISEUR *et* CONJ *est*

$$\mathcal{O}\left(l * \mathcal{O}(\textit{ forme } \Delta\textit{-normale })\right)$$

où l est la longueur canonique du secret.

Preuve : Les trois algorithmes MULTIPLE, DIVISEUR et CONJ ont la même complexité théorique ; en effet, l'opération subsidiaire dans DIVISEUR est négligeable. Travaillons donc à la complexité de l'algorithme de base : celle de MULTIPLE, Algorithme 4.2.1.

Toutes les opérations sont ordinaires ; la plus coûteuse est de mettre un mot sous sa forme Δ-normale. L'algorithme est linéaire à l'exception d'une boucle *Tant que*. Cette dernière est conditionnée par deux paramètres : une incrémentation de la variable l sur au maximum l_2 valeurs, et une condition *cond*. Rappelons que l_2 joue le rôle de la longueur canonique du secret. Ainsi la complexité de MULTIPLE est $\mathcal{O}\left(l_c(secret) * \mathcal{O}(\ forme\ \Delta\text{-}normale\ )\right)$.

□

Remarque 4.4.2. Ces algorithmes ont une très faible complexité si l'on considère que dans les groupes de tresses, le calcul de la forme Δ-normale est efficace, voir la table de complexité 2.1.

4.5 Preuve

Cette section est dévolue à prouver la Proposition 4.2.2 : nous nous intéressons uniquement au fait que MULTIPLE fournisse bien un multiple. L'efficacité des algorithmes sera l'objet du Chapitre 6 : la qualité du multiple vient du fait soit le plus petit possible.

Nous conserverons les notations de l'algorithme. Nous procédons par récurrence sur le paramètre l qui contrôle la boucle principale, *Tant que*, et nous repérons le nombre d'itérations dans l'exposant. Ainsi nous avons :

$$l = 0 \quad \longrightarrow \quad A^0 = e \qquad Y^0 = \tau^{-\alpha}(X) = T\tau^{-\alpha}(y)\tau^{\beta}(T^\star)$$

Soit $t_1 \cdots t_{l_2}$ la forme Δ-normale de T. La proposition de récurrence est :

$$\text{pour } l \in [1, l_2] \qquad \mathcal{P}(l) : \begin{cases} 1 : & t_1 t_2 \cdots t_l \prec A^l \prec Y_1^0 Y_2^0 \cdots Y_l^0 \\ 2 : & T \prec A^l Y_1^l \cdots Y_{l_2-l}^l \prec Y_1^0 Y_2^0 \cdots Y_{l_2}^0 \end{cases} \tag{4.1}$$

Le cas $l = 0$ n'est pas pris en compte mais est vrai :

$$(2) \qquad T \prec Y^0 \quad \Rightarrow_{\text{Proposition 3.2.6}} \quad T \prec Y_1^0 Y_2^0 \cdots Y_{l_2}^0$$

Avant de commencer la récurrence, remarquons quelques points qui aideront à la compréhension générale de la preuve :

- $\mathcal{P}(l)[1] \Rightarrow \sup(A^l) = l$
- Par construction, le lecteur pourra vérifier qu'à chaque étape, le relation suivante est vraie :
$$l \in [1, l_2], \qquad Y^0 = A^l Y^l \tau^{\beta + l_2 - l}((A^l)^\star)$$
- La condition $\mathcal{P}(l)[2]$ est là afin de gérer les cas où la boucle principale est arrêtée : ligne (m8) "`cond :=faux`". Elle garantit que l'élément de sortie de l'algorithme est un multiple à droite de T de même supremum.

Initialisation $l = 1$: Considérant T, $T^\star$ et Y^0 sous leur forme Δ-normale, nous avons :

$$Y^0 = t_1 t_2 \cdots t_{l_2} \tau^{-\alpha}(y) \tau^{\beta}(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-2}(\partial(t_2)) \tau^{\beta+l_2-1}(\partial(t_1)) = Y_1^0 \cdots Y_{\sup(Y^0)}^0$$

Alors $t_1 \prec Y^0$ et $Y^0 \succ \tau^{\beta+l_2-1}(\partial(t_1))$

$$t_1 \prec Y^0 \quad \Rightarrow_{\text{Proposition 3.2.6}} \quad t_1 \prec Y_1^0 \quad \Rightarrow_{\text{Proposition 3.2.5}} \quad \partial(t_1) \succ \partial(Y_1^0)$$

$$\Big(\Rightarrow \quad Y^0 \succ \tau^{\beta+l_2-1}(\partial(t_1)) \succ \tau^{\beta+l_2-1}(\partial(Y_1^0)) \Big)$$

La vérité de l'assertion $Y_2^0 \cdots Y_{\sup(Y^0)}^0 \not\succ \tau^{\beta+l_2-1}(\partial(Y_1^0))$ est équivalente au test booléen $(m8)$: "$Z \notin M$" avec $Z := {Y_1^0}^{-1} Y^0 \tau^{\beta+l_2-1}(\partial(Y_1^0))^{-1}$. Il y a deux cas :

- VRAI : La condition $(m8)$ est vraie, alors $A^1 = Y_1^0$ et $Y^1 = Y_2^0 \cdots Y_{\sup(Y^0)}^0$; le processus sort de la boucle. Ainsi $\mathcal{P}(1)$ est vraie.

- FAUX : Nous remarquons

$$\underbrace{t_2 \cdots t_{l_2} \tau^{-\alpha}(y) \tau^{\beta}(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-2}(\partial(t_2))}_{\sup(") \le l_1+2l_2-2} \underbrace{\tau^{\beta+l_2-1}(\partial(t_1)) \tau^{\beta+l_2-1}(Y_1^0)^{-1}}_{\in S} \succ Z$$

  Soit $Z_1 \cdots Z_{\sup(Z)}$ la forme Δ-normale de Z ; nous avons :

$$\sup(Z) = l_1 + 2l_2 - 1 \quad \Rightarrow_{\text{Proposition 3.2.10}} \quad \tau^{\beta+l_2-1}(\partial(t_1)) \tau^{\beta+l_2-1}(Y_1^0)^{-1} \succ Z_{\sup(Z)}$$

  Maintenant, nous posons :

$$\partial(t_1) \succ \partial(A^1) \quad \text{avec} \quad \partial(A^1) := \begin{cases} \tau^{1-\beta-l_2}(Z_{\sup(Z)}) \partial(Y_1^0) & \text{si } \sup(Z) = l_1 + 2l_2 - 1 \\ \partial(Y_1^0) & \text{sinon} \end{cases}$$

  Ainsi nous avons : $\partial(t_1) \succ \partial(A^1) \succ \partial(Y_1^0)$; le Lemme 3.2.4 donne :

$$\mathcal{P}(1)[1] \ : \ t_1 \prec A^1 \prec Y_1^0$$

  Maintenant, posons $Y_1^1 \cdots Y_{\sup(Y_1)}^1$ la forme Δ-normale de

$$Y^1 := {A^1}^{-1} Y^0 \tau^{\beta+l_2-1}(\partial(A^1))^{-1}$$

  Le Corollaire 3.2.7 nous donne que $\mathcal{P}(1)[2]$ est vraie :

$$T \prec A^1 Y_1^1 \cdots Y_{l_2-1}^1 \prec Y_1^0 \cdots Y_{l_2}^0$$

Hérédité $l = k + 1$: Nous supposons que $\mathcal{P}(k)$ est vraie, avec $1 \le k < l_2$.

$$\begin{array}{ll} \mathcal{P}(k) \Rightarrow_{\text{Proposition 3.2.6}} & t_1 \cdots t_{k+1} \prec A^k Y_1^k \\ \Rightarrow_{\text{Lemme 3.2.4}} & \partial(t_{k+1}) \cdots \tau^{k-1}(\partial(t_1)) \succ \partial(Y_1^k) \tau(\partial(t_k^k)) \cdots \tau^{k-1}(\partial(t_1^k)) \\ \Rightarrow & Y^0 = A^k Y^k \tau^{\beta+l_2-k}((A^k)^\star) \succ \tau^{\beta+l_2-k-1}(\partial(Y_1^k)) \tau^{\beta+l_2-k}(\partial(t_k^k)) \cdots \tau^{\beta+l_2-1}(\partial(t_1^k)) \\ \Rightarrow & A^k Y^k \succ \tau^{\beta+l_2-k-1}\left(\partial(Y_1^k)\right) \end{array}$$

La véracité de l'assertion $Y_2^k \cdots Y_{\sup(Y^k)}^k \not\succ \tau^{\beta+l_2-k-1}(\partial(Y_1^k))$ est équivalente au test booléen $(m8)$: "$Z \notin M$" avec $Z := {Y_1^k}^{-1} Y^k \tau^{\beta+l_2-k-1}(\partial(Y_1^k))^{-1}$. Il y a deux cas :

– VRAI : La condition de $(m8)$ est vraie ; alors $A^{k+1} = A^k Y_1^k$ et $Y^{k+1} = (Y_1^k)^{-1} Y^k$. Le processus sort de la boucle . $\mathcal{P}(k)$ et la Proposition 3.2.6 donnent :

$$t_1 \cdots t_{k+1} \prec A^{k+1} \prec Y_1^0 \cdots Y_{k+1}^0 \quad \Rightarrow \quad \mathcal{P}(k+1)[1] \text{ est vraie}$$

De plus $A^k Y_1^k \cdots Y_{l_2-k}^k = A^{k+1} Y_1^{k+1} \cdots Y_{l_2-k-1}^{k+1} \quad \Rightarrow \quad \mathcal{P}(k+1)[2]$ est vraie.

– FAUX : Nous remarquons que $Z = (A^k Y_1^k)^{-1} Y^0 \tau^{\beta+l_2-k-1}((A^k Y_1^k)^\star)^{-1}$; comme $\sup(A^k Y_1^k) = k+1$, le Corollaire 3.2.2 donne l'écriture suivante :

$$t_{k+2} \cdots t_{l_2} \tau^\beta(y) \tau^\beta(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-k-2}(\partial(t_{k+2}) c \succ Z$$

avec $c = \tau^{\beta+l_2-k-1}\left((t_1 \cdots t_{k+1})^\star ((A^k Y_1^k)^\star)^{-1}\right) \in M$.
La Proposition 3.2.10 donne :

$$(t_1 \cdots t_{k+1})^\star \succ A^{k+1^\star}$$

$$\text{avec } A^{k+1^\star} := \begin{cases} \tau^{k+1-l_2-\beta}(Z_{\sup(Z)})(A^k Y_1^k)^\star & \text{si } \sup(Z) = l_1 + 2(l_2-k) - 1 \\ (A^k Y_1^k)^\star & \text{sinon} \end{cases}$$

Ainsi nous obtenons $(t_1 \cdots t_{k+1})^\star \succ (A^{k+1})^\star \succ (A^k Y_1^k)^\star \succ (Y_1^0 \cdots Y_{k+1}^0)^\star$.
D'après la Propostion 3.2.5, $\mathcal{P}(k+1)[1]$ est vraie. Posons

$$Y^{k+1} = (A^{k+1})^{-1} Y^0 ((A^{k+1})^\star)^{-1}$$

Le Corollaire 3.2.8 donne $\mathcal{P}(k+1)[2]$.
Ce qui termine la récurrence.

Remarque 4.5.1. Le supremum du multiple retourné par l'algorithme est le même que celui du secret. Ceci est garanti par le second argument de $\mathcal{P}$.

Chapitre 5

Application de l'algorithme aux groupes de tresses

Le problème de conjugaison est une primitive cryptographique fondamentale dans les groupes de tresses. Dans le chapitre 4, nous avons présenté un algorithme opérant une réduction de ce problème. Il est défini dans les groupes de Garside et est donc directement applicable aux groupes de tresses sur chacune de ses structures de Garside.

La cryptographie basée sur les tresses utilise aussi comme primitive plusieurs variantes du problème de conjugaison. Ce chapitre sera l'occasion de considérer ses différents aspects sans négliger un travail concernant la Remarque 4.1.1 sur les paramètres d'entrée dans le cas du problème de conjugaison.

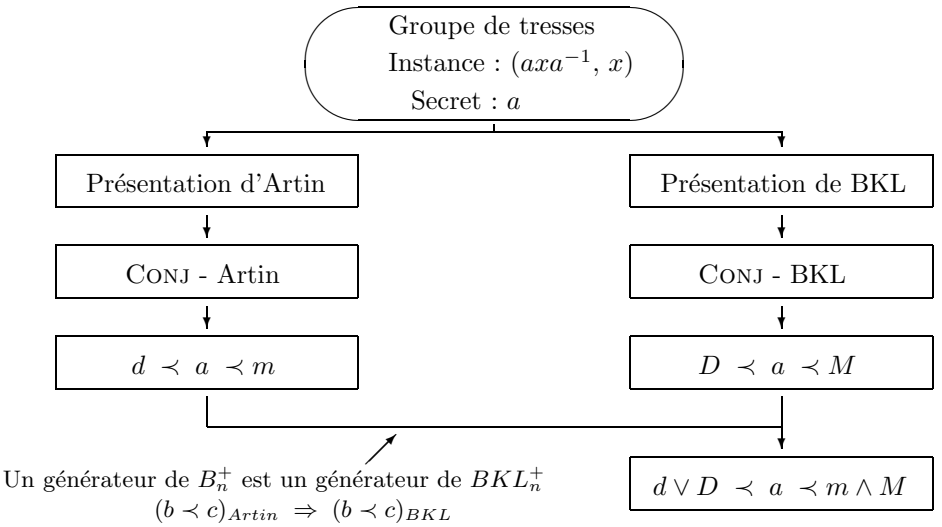

Fig. 5.1 – Utilisation en parallèle de Conj

Nous considérons le groupe de tresses à n brins, B_n, de monoïde M, d'élément de Garside Δ et d'exposant $\mathcal{E}$. En particulier, le choix de l'élément de Gariside Δ dans le monoïde $M \in \{B_n^+,\ BKL_n^+\}$ définit la structure de Garside que nous considérons dans le groupe ; les valeurs de $\mathcal{E}$ et S en découlent.

5.1 Modes parallèle / séquentiel

Afin de profiter au maximum de la double structure de Garside des groupes de tresses, nous proposons deux procédés permettant de cumuler, en partie, les réductions du problème de conjugaison obtenues dans chacune des deux présentations. En pratique, notre réduction fournit deux nouvelles instances de conjugaison dont la taille du secret est réduite. C'est sur ces deux nouvelles instances que nous pouvons travailler à nouveau.

L'algorithme Conj exploite la densité de l'écriture sous forme Δ-normale du conjugué. Une fois son travail terminé, il semble inutile de vouloir le refaire tourner sur les deux nouvelles instances, car l'information contenue dans cette "densité" a déjà été exploitée. C'est là qu'intervient la deuxième présentation. La structure de cette dernière est sensiblement différente et donc la densité de ses écritures n'est pas totalement liée à celle de l'autre présentation ; en conclusion, il reste de l'information à utiliser.

- **En parallèle** : Nous considérons une instance de conjugaison et utilisons Conj pour la réduire dans les deux présentations en même temps. Ensuite nous rapportons les quatre nouvelles instances dans la présentation de Birman, Ko et Lee et, utilisant le pgcd à gauche et le ppcm à droite, nous obtenons deux nouvelles instances dont le secret est encore plus réduit.
  Nous ne pouvons rapporter les quatre nouvelles instances dans la présentation d'Artin, car l'ordre partiel, $\prec$, ne serait pas respecté. En effet un élément positif dans la présentation de Birman, Ko et Lee, même un générateur de BKL_n^+, n'est pas nécessairement un élément positif dans la présentation d'Artin. Voir la figure 5.1

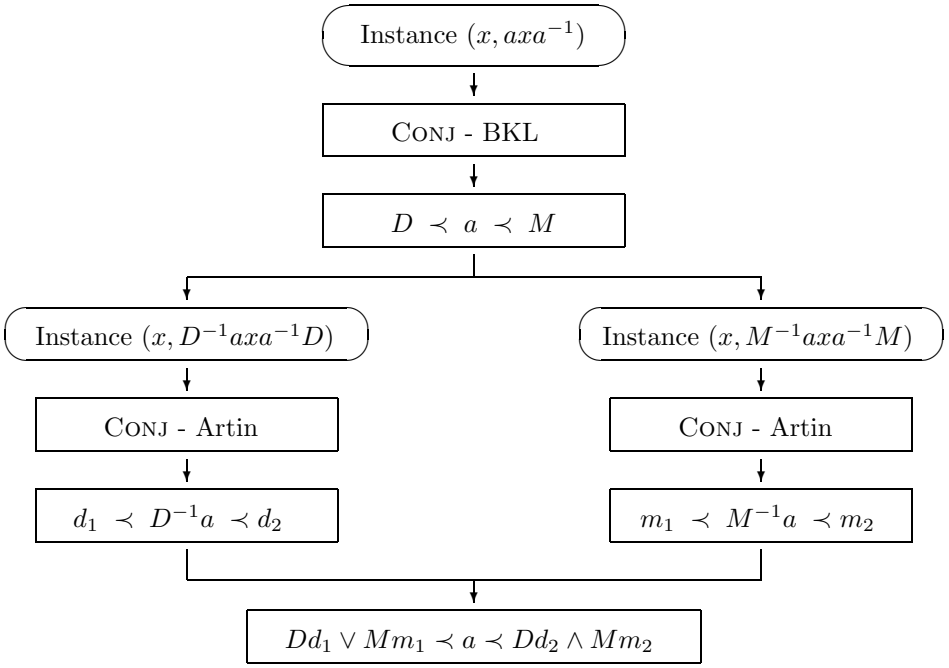

Fig. 5.2 – Utilisation en séquentiel de Conj

- **En séquentiel** : Cette méthode exploite les informations successivement dans les deux présentations. Nous utilisons les deux nouvelles instances produites dans une présentation pour les réduire dans l'autre. Voir un exemple à la figure 5.2.

5.2 Evaluation des paramètres d'entrée

Soit a, x deux éléments de B_n, considérons l'instance de conjugaison $(x, x' = axa^{-1})$. L'algorithme Conj requiert en entrée : $x, x', \inf(a)$ et $l_c(a)$. Ici, notre objectif est de montrer en quoi l'infimum du secret et sa longueur canonique ne sont pas des informations liées à la sécurité, et donc qu'ils peuvent être supposés connus sans que cela soit trop restrictif.

Remarque 5.2.1. Ces paramètres sont de petite taille ; ainsi, il est généralement entendu que l'infimum et la longueur canonique du secret ne sont pas des paramètres de sécurité d'un protocole. Ils sont parfois directement fixés par le protocole lui-même.

5.2.1 Infimum du secret

Etudions l'influence de l'infimum sur la recherche du secret d'une instance de conjugaison. Nous avons déjà un premier résultat intéressant :

Proposition 5.2.2. *Soit (x, x') une instance de conjugaison dans le groupe B_n ; alors il existe un élément $a \in B_n$ tel que $\inf(a) \in [0, \mathcal{E} - 1]$ et $x' = axa^{-1}$.*

> Preuve : Considérant l'instance de conjugaison (x, x'), nous posons $\Delta^r s_1 \cdots s_p$ la forme Δ-normale d'un conjugant de cette instance. Comme $\Delta^{\mathcal{E}}$ appartient au centre du groupe, il commute avec n'importe quel élément ; ainsi $\Delta^{r \bmod \mathcal{E}} s_1 \cdots s_p$ est aussi un conjugant de l'instance et cet élément a bien son infimum dans $[0, \mathcal{E} - 1]$.
>
> □

Remarque 5.2.3. On rappelle ici les exposants dans le cas des groupes de tresses pour les éléments de Garside fixé au Chapitre 1 :

$$\mathcal{E}_{B_n^+} = 2 \qquad\qquad \mathcal{E}_{BKL_n^+} = n$$

Corollaire 5.2.4. *Soit (x, x') une instance de conjugaison dans le groupe B_n ; alors il existe $r \in [0, \mathcal{E} - 1]$ tel que l'infimum du secret, relatif à la Proposition 5.2.2, de l'instance de conjugaison $(\tau^{-r}(x), x')$ soit nul. Soit a le secret de l'instance $(\tau^{-r}(x), x')$, alors $a\Delta^r$ est un secret de l'instance initiale (x, x').*

> Preuve : Le corollaire formalise la situation suivante : soit $r \in [0, \mathcal{E} - 1]$ et b un élément de B_n d'infimum nul tels que $\Delta^r b$ soit un secret de l'instance (x, x') :
>
> $$x' = \Delta^r bxb^{-1}\Delta^{-r} = \tau^{-r}(bxb^{-1})$$
>
> Ainsi l'instance $(\tau^{-r}(x), x')$ admet pour secret $\tau^{-r}(b)$ qui est d'infimum nul, tandis que l'instance de départ admet comme secret $\Delta^r b = \tau^{-r}(b)\Delta^r$
>
> □

Application à Conj

Le corollaire précédent s'applique directement à notre algorithme. Supposant que l'infimum n'est pas connu, nous considérons l'instance $(x, x' = axa^{-1})$ et appelons l'algorithme :

$$(d, m) := \textsc{Conj}\ (x, x', 0, l_c(a))$$

$$d \prec \tau^{-\inf(a)}(\Delta^{-\inf(a)}a) = a\Delta^{-\inf(a)} \prec m \quad \text{et} \quad \sup(m) = l_c(\Delta^{-\inf(a)}a)$$

Afin de déterminer l'infimum du secret et de terminer le recouvrement du secret de l'instance initiale, il suffit de trouver l'entier $r \in [0, \mathcal{E} - 1]$ tel que le couple d'instances $\left(\tau^{-r}(x), d^{-1}x'd\right)$ et $\left(\tau^{-r}(x), m^{-1}x'm\right)$ admette un secret d'infimum nul, qu'il conviendra de déterminer aussi.

Le réduction de la longueur du secret initial est effective, même sans connaître son infimum. En pratique, le fait de ne pas connaître l'infimum entraîne une augmentation, d'un facteur linéaire, de la complexité correspondant à l'attaque complémentaire à mener pour déterminer le secret restant.

L'évaluation de l'efficacité de la réduction est l'objet du Chapitre 6. Cependant nous pouvons déduire que si la réduction est efficace, à savoir, s'il existe un moyen, nécessitant des ressourses abordables, qui permet de récupérer le secret restant, le fait de ne pas connaître l'infimum n'a donc pas d'incidence du point de vue cryptanalytique, car la recherche simultanée sur un nombre limité d'intances (soit 2, soit n) est clairement envisageable.

En conclusion, il n'est pas contraignant de supposer que l'infimum est connu car ce n'est pas un paramètre de sécurité pour le problème de conjugaison tel qu'il est défini.

5.2.2 Longueur canonique du secret

Entre approche intuitive et théorique, nous allons donner un algorithme permettant d'estimer la longueur canonique du secret d'une instance de conjugaison dans B_n. Dans ce cadre aussi, le choix du générateur aléatoire de tresses influe sur la qualité du résultat ; cependant, la lecture globale de cette section permet de relativiser cette influence et montre la faisabilité de l'estimation. Commençons par donner un premier résultat théorique :

Proposition 5.2.5. *Soit $(x, x' = axa^{-1})$ une instance de conjugaison dans le groupe B_n de secret a ; alors nous avons la relation suivante :*

$$l_c(a) \geq max\left(\inf(x) - \inf(x'), \sup(x') - \sup(x)\right) \tag{5.1}$$

Preuve : Considérant l'instance de conjugaison $(x, x' = axa^{-1})$, nous avons :

$$\begin{aligned} x' &= axa^{-1} \\ &= \Delta^{\inf(x)-l_c(a)}\ \underbrace{\tau^{-\sup(a)}\left(\tau^{\inf(x)}(\Delta^{-\inf(a)}a)\ \Delta^{-inf(x)}x\ a^\star\right)}_{\in M} \\ &\Rightarrow \inf(x') \geq \inf(x) - l_c(a) \end{aligned}$$

De plus,

$$sup\left(\tau^{-\sup(a)}\left(\tau^{\inf(x)}(\Delta^{-\inf(a)}a)\ \Delta^{-\inf(x)}x\ a^\star\right)\right) \leq l_c(a) + l_c(x) + \sup(a^\star)$$

D'après le Corrolaire 3.2.2, nous avons :

$$\sup(x') \leq 2l_c(a) + l_c(x) + (\inf(x) - l_c(a)) = \sup(x) + l_c(a)$$

Ainsi, la relation (5.1) est démontrée.

□

Remarque 5.2.6. La portée de la Proposition 5.2.5 peut s'avérer nulle dans certains cas. En effet, si nous considérons un générateur aléatoire produisant des clés *symétriques*, comme le GA-CompC, les deux conjugués ont les mêmes infimums et supremums, donc aucune information n'est dévoilée sur la longueur canonique du secret.

Maintenant, considérons une approche plus intuitive, voire expérimentale, afin de trouver des informations sur la longueur canonique du secret. Nous allons introduire la notion de *biais* pour un générateur aléatoire et proposer un algorithme permettant d'estimer la longueur canonique du secret.

Définition 5.2.7. Considérant une instance de conjugaison $(x, x' = axa^{-1})$ dans le groupe B_n, produite par le générateur aléatoire de tresses GA, nous définissons le *biais* de ce générateur aléatoire par la valeur suivante :

$$b_{\mathrm{GA}} = l_c(a) + \inf(x') - \inf(x)$$

En pratique, le *biais* d'un générateur aléatoire peut être interprété comme l'écart du cas réel à l'égalité induite de (5.1). Bizarrement, on parle de biais d'un générateur aléatoire alors que la valeur du biais ne "dépend" que de l'instance de conjugaison considérée. En fait, les valeurs du biais pour des instances générées par un même générateur sont assez similaires, d'où l'abus de notation. En pratique, on l'estime par une étude statistique sur le générateur aléatoire : ceci permet d'anticiper en partie le biais.

Le choix de présenter les résultats suivants par des *faits expérimentaux* traduit exactement leur valeur. Ce ne sont pas des propositions ; en effet, établir la preuve de ces résultats demanderait plus de rigueur sans avoir beaucoup d'intérêt. Ainsi, par choix, il ne sera présenté qu'une idée directrice.

Fait expérimental 5.2.8. *Considérant une instance de conjugaison $(x, x' = axa^{-1})$ dans le groupe B_n, on note $\Delta^r s_1 \cdots s_p$ la forme Δ-normale de x'. Alors, pour des petites valeurs de j, posant $y = s_1 \cdots s_j$, nous avons :*

$$s_1 \cdots s_p \succ \tau^{-r-j}(y^\star)$$

Idée : Considérons l'instance de conjugaison $(x, x' = axa^{-1})$, soit $a_1 \cdots a_{l_c(a)}$ la forme Δ-normale de $\Delta^{-\inf(a)}a$. Nous avons :

$$x' = \Delta^{\inf(x)-l_c(a)}\tau^{-\sup(a)}\left(\underbrace{\tau^{\inf(x)}(\Delta^{-\inf(a)}a)\ \Delta^{-\inf(x)}x\ a^\star}_{Y}\right)$$

Posons $\Delta^b y_1 \cdots y_p$ la forme Δ-normale de $Y = \tau^{\inf(x)}(\Delta^{-\inf(a)}a)\ \Delta^{-\inf(x)}x\ a^\star$. Alors, le *biais* du générateur aléatoire utilisé pour générer (x, x') est b.

Soit $j \in \mathbb{N}^*$ tel que $b+j \leq l_c(a)$; cela suppose que $\inf(Y) < l_c(a)$. Considérons maintenant l'élément $\Delta^b y_1 \cdots y_j$; la Proposition 3.2.6 nous donne :

$$\tau^{\inf(x)}(a_1 \cdots a_{b+j}) \prec \Delta^b y_1 \cdots y_j$$

Ainsi, le Lemme 3.2.4 nous permet de déduire :

$$\partial(a_{b+j}) \cdots \tau^{b+j-1}(\partial(a_1)) \succ \tau^{-\inf(x)}\left(\partial(y_j) \cdots \tau^{j-1}(\partial(y_1))\right)$$

De plus, il existe $i \in [0, \mathcal{E}-1]$ tel que :

$$a^\star \succ \tau^{l_c(a)-b-j}(\partial(a_{b+j})) \cdots \tau^{l_c(a)-1}(\partial(a_1)) \succ \tau^{i-\inf(x)}\left(\partial(y_j) \cdots \tau^{j-1}(\partial(y_1))\right)$$

Considérant dans ces deux divisibilités à droite, les puissances de τ du facteur de droite, on en déduit l'égalité suivante :

$$l_c(a)-1-(b+j-1) \equiv i-\inf(x)+j-1-(j-1-\inf(x)) \; mod\, \mathcal{E} \quad \Leftrightarrow \quad l_c(a) \equiv b+i+j \; mod\, \mathcal{E}$$

La Définition 5.2.7 nous donne :

$$i - \inf(x) \equiv l_c - b - j - \inf(x) = -\inf(x') - j \; mod\; \mathcal{E}$$

Donc, nous obtenons : $Y \succ a^\star \succ \tau^{-\inf(x')-j}((y_1 \cdots y_j)^\star)$
Ainsi, si le *biais* est inférieur à la longueur canonique du secret, nous obtenons :

$$\forall j; \quad j \leq l_c(a) - b, \qquad y_1 \cdots y_p \succ \tau^{-\inf(x')-j}((y_1 \cdots y_j)^\star)$$

□

L'algorithme suivant permet de donner une valeur approchée de la longueur canonique :

Algorithme 5.2.9. ESTIMATEUR_l_c (x, x')

Entrée :	Soit $x, x' \in B_n$.
Init :	$X := \Delta^{-\inf(x')}x'$, $i := 0$, $cond := vrai$
	Calculer la *forme* Δ*-normale* $X_1X_2 \cdots X_{l_c(X)}$ de X
Boucle :	Tant que *cond* et $i < l_c(X)$ faire
	$Z := X_1 \cdots X_{i+1}$
	Si $X \not\succ \tau^{-\inf(x')-i-1}(Z^\star)$ alors $cond := faux$
	Sinon $i := i+1$
	FinSi
	FinTantQue
Sortie :	i

Fait expérimental 5.2.10. *Considérons une instance de conjugaison $(x, x' = axa^{-1})$ dans le groupe B_n ; alors l'algorithme 5.2.9,* ESTIMATEUR_l_c*, prenant en entrée le couple (x, x') ressort une valeur approchée de la longueur canonique du secret.*

Idée : Considérant une instance de conjugaison $(x, x' = axa^{-1})$, il s'agit à nouveau de travailler sur l'élément :

$$Y = \tau^{\inf(x)}(\Delta^{-\inf(a)}a)\ \Delta^{-\inf(x)}x\ a^{\star}$$

D'après le Fait 5.2.8, nous connaissons le "décalage" existant entre les multiples à droite des premiers facteurs de Y (donc de "a") et leurs duaux qui divisent à droite Y ("donc $a^{\star}$"). On retourne simplement la longueur canonique du plus grand multiple vérifiant cela.

□

$n = 5 \quad l = 1000 \quad l_c = 10$ Nombre de simulations : 1000			
Générateur	$l_c(a)$	$l_c(a) - e^{*1}$	Autres performances
GA-MOTS	28.72	2.46	$\mathcal{P}(e = l_c(a) - i)_{i\in[1,3]} \approx 0.20,\ \mathcal{P}(\lvert l_c(a) - e - m^{*2}\rvert \leq 5) = 0.99$
GA-MOTSP	28.66	0.31	$\mathcal{P}(e = l_c(a)) = 0.73,\ \mathcal{P}(e = l_c(a) - 1) = 0.23$
GA-CANO	10	0	$\mathcal{P}(e = l_c(a)) = 1$
GA-COMPC	23.97	13.97	$\mathcal{P}(e = l_c(a) - 14) = 0.25,\ \mathcal{P}(\lvert l_c(a) - e - m\rvert \leq 5) = 0.96$

^{*1} : e estimation de $l_c(a)$; ^{*2} : m la moyenne des "$l_c(a) - e$"

TAB. 5.1 – Estimation de la longueur canonique - présentation d'Artin

La Table 5.1 nous donne une idée de l'efficacité de cet algorithme, sur la présentation d'Artin, pour les génératateurs aléatoires de tresses introduits au Chapitre 2.

On remarque que l'estimation est assez bonne pour les générateurs GA-MOTSP, GA-CANO et GA-TRONC ; mais que le générateur GA-COMPC dissimule bien la longueur canonique du secret.

Remarque 5.2.11. Nous pouvons faire la remarque fataliste suivante : les cas où l'estimation de la longueur canonique n'est pas bonne sont inclus dans les cas où notre algorithme de réduction ne donne pas de bons résultats exploitables même en supposant la longueur canonique connue. Ainsi, si notre algorithme doit marcher, c'est que l'on peut par ailleurs estimer la longueur canonique de secret. Donc, ce paramètre n'est pas un paramètre de sécurité pour notre algorithme de réduction.

$n = 30 \quad l = 500 \quad l_c = 10$ Nombre de simulations : 1000		
Générateur	b	Autres performances
GA-MOTS	4.07	$\mathcal{P}(b = 4) = 0.2,\ \mathcal{P}(\lvert b - m^*\rvert \leq 2) = 0.68,\ \mathcal{P}(\lvert b - m\rvert \leq 5) = 0.99$
GA-MOTSP	0,31	$\mathcal{P}(b = 0) = 0.73,\ \mathcal{P}(b = 1) = 0.23,\ \mathcal{P}(b = 2) = 0.04$
GA-CANO	0	$\mathcal{P}(b = 0) = 1$
GA-COMPC	$l_c(a)$	$\mathcal{P}(b = l_c(a)) = 1$

* : m est la valeur moyenne du biais

TAB. 5.2 – Statistiques du biais - présentation d'Artin

La connaissance du biais d'une instance est équivalente à la connaissance de la longueur canonique du secret, d'après la définition du biais. La Table 5.2 donne une idée du

comportement du biais, toujours sur la présentation d'Artin. L'anticipation du biais est faisable pour les trois générateurs, précédemment cités, qui dissocient assez bien la valeur du biais de celle de la longueur canonique.

Il est intéressant de soulever un aspect non négligeable : les groupes de tresses possèdent deux présentations de Garside sur lesquelles nous pouvons appliquer l'algorithme de réduction ; étant donné la différence de structure induite par ces présentations, ce qui est vrai pour l'une en terme de complexité ne l'est pas forcément pour l'autre. A la question *est-il intéressant de changer de présentation pour estimer ce paramètre ?*, la réponse semble négative dans les cas que nous avons considérés.

En annexe se trouvent des tables plus complètes qui montrent notamment que cette démarche conduit parfois à un étalement des valeurs possibles et donc qu'il n'y a vraisemblablement pas de gain statistique.

En conclusion, la longueur canonique ne sera pas considérée comme un paramètre de sécurité dans l'évaluation de l'algorithme de réduction présenté dans cette étude.

Remarque 5.2.12. La recherche de la longueur canonique n'a nullement besoin de supposer l'infimum du secret connu. Ceci est important puisque nous supposons que la longueur canonique est connue pour calculer l'infimum.

5.3 Problème de conjugaison multiple simultanée - ConjMS

Rappelons que ce problème consiste à déterminer un secret commun à plusieurs instances de conjugaison. Le but de cette partie n'est pas de décrire un moyen pour attaquer une instance du problème de conjugaison multiple simultanée, mais de proposer une amélioration de l'algorithme Conj basée sur quelques observations du problème de conjugaison multiple simultanée : le but est à nouveau d'attaquer une instance du problème de conjugaison.

Considérons une instance du problème de conjugaison multiple simultanée ; nous avons :

$$(x_i, ax_ia^{-1})_{i\in I} \in \left(B_n^2\right)^I$$

• La première idée est d'appliquer l'algorithme Conj sur chacune des instances ; cela nous donne :

$$(d_i, m_i)_{i\in I}; \qquad \forall i \in I, \quad d_i \prec a \prec m_i \tag{5.2}$$

On en déduit immédiatement un meilleur diviseur à gauche et un meilleur multiple à droite :

$$\bigvee_{i\in I} d_i \prec a \prec \bigwedge_{i\in I} m_i \tag{5.3}$$

• La deuxième approche considère que la conjugaison est un morphisme ; ainsi la donnée de cette instance du problème de conjugaison multiple simultanée nous donne en réalité l'ensemble des instances de conjugaison formées par un élément du groupe engendré par les $(x_i)_{i\in I}$ et de conjugant a :

$$\left\{(x, axa^{-1}), \quad x \in \langle x_i,\ i \in I\rangle\right\}$$

Application à Conj

Considérant une instance de conjugaison $(x, x' = axa^{-1})$, le deuxième point nous donne l'ensemble suivant d'instances :

$$\left\{(x^i, x'^i = ax^ia^{-1}), \quad i \in \mathbb{Z}\right\} \tag{5.4}$$

Ainsi, appliquant le premier point, nous pouvons facilement améliorer la qualité du diviseur et du multiple dans le cas du problème de conjugaison. En pratique, il suffira de considérer les instances (x^i, x'^i) pour des $|i|$ petits.

Algorithme 5.3.1. CONJMS (x, x', r, p)

Entrée :	Soit $x, x' \in G, \qquad r, p \in \mathbb{Z}$.
Init :	$cond := vrai, \quad i := 1$
	$(d_1, m_1) :=$ CONJ (x, x', r, p)
	$(d_2, m_2) :=$ CONJ (x^{-1}, x'^{-1}, r, p)
	$d := d_1 \vee d_2 \qquad m := m_1 \wedge m_2$
Boucle :	Tant que *cond* faire
	$i := i + 1$
	$(d_1, m_1) :=$ CONJ (x^i, x'^i, r, p)
	$(d_2, m_2) :=$ CONJ (x^{-i}, x'^{-i}, r, p)
	Si $d \vee d_1 \vee d_2 \neq d$ ou $m \wedge m_1 \wedge m_2 \neq m$ alors
	$d := d \vee d_1 \vee d_2 \qquad m := m \wedge m_1 \wedge m_2$
	Sinon $cond := faux$
	FinSi
	FinTantQue
Sortie :	(d, m)

Proposition 5.3.2. *Soit a, x deux éléments de G. Considérons l'instance de conjugaison $(x, x' = axa^{-1})$ et notons (d, m) la sortie de l'algorithme 5.3.1,* CONJMS*, ayant en entrée le 4-uplet $(x, x', \inf(a), l_c(a))$. Alors d est un diviseur à gauche de a et m est un multiple à droite de a de même supremum : $d \prec a \prec m$.*

Preuve : Considérant la Proposition 4.3.2, le seul point restant à éclaicir est que la boucle se termine. Le fait que d est bien un diviseur à gauche de a et que m est bien un multiple à droite de a ne pose pas de problème.
Nous notons que les opérations se font dans le monoïde car l'infimum est connu. De plus, le diviseur à gauche d de a ou le multiple à gauche m de a ont leur taille qui varie à chaque tour. Celle de d augmente vers celle de a, tandis que celle de m diminue vers celle de a. Le nombre d'itérations possible est majoré par la valeur d'initialisation de $l(d^{-1}a) + l(a^{-1}m)$; cette quantité étant finie, la boucle se termine.

□

En pratique, nous utiliserons CONJMS à la place de CONJ afin d'améliorer les résultats.

Remarque 5.3.3. Il est évident que cette démarche peut être reportée intégralement dans la gestion d'une instance générique du problème de conjugaison multiple simultanée ; nous ne détaillons pas cet aspect, même si nous l'utilisons en pratique dans nos simulations.

5.4 Problème de conjugaison de type Ko-Lee - ConjKL

Le problème de conjugaison de type Ko-Lee est un cas particulier du problème de conjugaison ; la seule originalité vient de la forme spécifique que possède le secret. Il est possible d'aborder ce *nouveau* problème de deux façons :

- On peut repenser les algorithmes MULTIPLE et DIVISEUR afin qu'ils tiennent compte de la forme particulière de secret.
- On peut considérer ces instances comme des intances du problème de conjugaison, les traiter comme telles et seulement ensuite tenir compte de la forme du secret.

Les deux approches ont été programmées et elles sont par ailleurs cumulables ; mais l'amélioration n'est pas significative. Ainsi, dans une optique de simplification, le choix a été fait de présenter uniquement la deuxième méthode.

Remarque 5.4.1. Nous présenterons notre travail sur le sous-groupe LB_l ; les mêmes résultats sont directement transposables sur RB_r.

Voici un algorithme dérivé de CONJ, permettant de réduire une instance du problème de conjugaison de type Ko-Lee :

Algorithme 5.4.2. CONJKL (x, x', r, p)

Entrée :	Soit $x, x' \in B_n$, $\quad r, p \in \mathbb{Z}$.
	$(d, m) :=$ CONJ $(x, \Delta_{LB_l}^{-r} x' \Delta_{LB_l}^{r}, 0, p)$
	$m := \Delta_{LB_l}^{p} \wedge m$
Sortie :	$(\Delta_{LB_l}^{r} d, \Delta_{LB_l}^{r} m)$

Proposition 5.4.3. *Soit x un élément de B_n et a un élément de LB_l. Considérons l'instance de conjugaison $(x, x' = axa^{-1})$ et notons (d, m) la sortie de l'algorithme 5.4.2,* CONJKL*, ayant en entrée le 4-uplet $(x, x', \inf(a), l_c(a))$ où $\inf(a)$ et $l_c(a)$ sont respectivement l'infimum et la longueur canonique de a dans LB_l. Alors d est un diviseur à gauche de a et m est un multiple à droite de a de même supremum : $d \prec a \prec m$.*

> Preuve : Considérons l'instance du problème de conjugaison de type Ko-Lee $(x, x' = axa^{-1})$. La première opération sur x' permet d'annuler l'infimum du secret ; ainsi la Proposition 4.3.2 donne que le premier couple (d, m) vérifie $d \prec \Delta_{LB_l}^{-r} a \prec m$. Comme $d \prec \Delta_{LB_l}^{-r} a$ alors $d \in LB_l$, mais ce n'est vraisemblablement pas le cas pour m ; ainsi le pgcd ramène le multiple dans LB_l sachant que $e \prec \Delta_{LB_l}^{-r} a \prec \Delta_{LB_l}^{p}$. La dernière opération consiste à restaurer l'infimum au diviseur et multiple du secret.
>
> □

Infimum et longueur canonique d'un élément de LB_l

L'entrée de l'algorithme CONJKL nécessite de connaître l'infimum et la longueur canonique du secret dans son groupe de départ : cette approche n'est pas prise en compte dans la Section 5.2. Le but de cette partie ne sera pas de donner un moyen exact de les déterminer, mais seulement de montrer qu'elles ne sont pas des paramètres de sécurité et donc, que nous pouvons les supposer connues.

Remarque 5.4.4. Encore une fois, il est fait ici un choix de simplification dans la présentation. En effet, on pourrait directement écrire une variante de CONJ pour le problème de conjugaison de type Ko-Lee en supposant que ce sont l'infimum et la longueur canonique dans B_n qui sont connues ; c'est d'ailleurs ce que l'on utilise plus loin pour déterminer ces mêmes paramètres dans LB_l. Mais il semble plus clair pour la compréhension globale de distinguer les deux aspects : résolution et recherche des paramètres dans le sous-groupe.

Comme le montre la fonction suivante, il est facile de déterminer ces deux valeurs dans le groupe B_n si nous les connaissons dans LB_l ; c'est en quelque sorte l'inverse de cette fonction dont nous avons besoin. L'élément $\mathcal{E}$ est l'exposant du groupe B_n.

$$\begin{array}{rccc} \text{PASSAGE\_}LB_l - B_n : & \mathbb{Z}^2 & \longrightarrow & \mathbb{Z}^2 \\ & (r,p) & \longmapsto & \begin{cases} \text{si } r \geq 0 \text{ alors } (0, r+p) \\ \text{si } r \leq 0 \text{ et } p \geq |r| \text{ alors } (r \ mod\ \mathcal{E}, p) \\ \text{si } r \leq 0 \text{ et } p \leq |r| \text{ alors } (r \ mod\ \mathcal{E}, -r) \end{cases} \end{array}$$

Voici un algorithme qui permet d'estimer ces valeurs en partant de l'instance dans B_n :

Algorithme 5.4.5. ESTIMATEUR_LB_l (x, x', r, p)

Entrée : Soit $x, x' \in B_n$ et $r, p \in \mathbb{Z}$.

$r' := r, \quad cond := vrai$
Si $r = 0$ alors
  $(d, m) :=$ CONJ $(x, x', 0, p)$
  Si $d \in LB_l^+$ alors
    $r' = \left[max\{k; \Delta^k_{LB_l} \prec d\}, max\{k; \Delta^k_{LB_l} \prec m\} \right]$
    $p' := [p - r'[1], p - r'[2]]$
    $cond := faux$
  FinSi
FinSi
Si $cond = vrai$ alors
  Tant que $cond$ faire
    $(d, m) :=$ CONJ $(x, \Delta^{-r'}_{LB_l} x' \Delta^{r'}_{LB_l}, 0, p)$
    Si $d \in LB_l^+$ alors $cond := faux$
    Sinon $r' := r' - \mathcal{E}$
    FinSi
  FinTantQue
  $p' := \sup(m)$
FinSi

Sortie : (r', p')

Fait expérimental 5.4.6. *Considérons une instance de conjugaison $(x, x' = axa^{-1})$ dans le groupe B_n dont le secret, a, est dans LB_l ; alors l'algorithme 5.4.5,* ESTIMATEUR_LB_l, *prenant en entrée le couple $(x, x', \inf(a), l_c(a))$ où $\inf(a)$ et $l_c(a)$ sont respectivement l'infimum et la longueur canonique du secret dans B_n, ressort une estimation de l'infimum et de la longueur canonique du secret dans LB_l.*

Idée : On distingue le cas où l'infimum du secret dans LB_l est positif ou nul du cas où il est négatif.

• S'il est positif, alors $r = 0$, le test $d \in LB_l$ est vrai et les intervalles de sortie donnent un encadrement de la valeur de l'infimum et de la longueur canonique dans LB_l du secret.

• S'il est négatif ; r peut aussi être nul (modulo $\mathcal{E}$) et cela est détecté avec la valeur de la variable *cond* : *cond* est vrai à l'entrée de la seconde boucle conditionnelle. La boucle itérative *Tant Que* permet d'obtenir un infimum dans LB_n donnant un diviseur dans LB_l ; ensuite la valeur $\sup(m)$ permet d'estimer la longueur canonique du secret.

□

Cet algorithme a été testé sur les générérateurs GA-Mots, GA-MotsP et GA-Cano dans les deux présentations. L'estimation est correcte en moyenne, voire même très souvent exacte.

Chapitre 6

Analyse de l'efficacité

L'objectif de ce chapitre est de déterminer les paramètres qui rendent efficace l'algorithme de réduction de la conjugaison. Nous commençons par proposer une mesure de l'efficacité de l'algorithme : la complexité de la méthode exhaustive et grossière de recouvrement du secret restant. Nous utiliserons cette mesure de l'efficacité pour considérer l'influence des générateurs aléatoires, de la taille des éléments ou du groupe, sur des variantes du problème de conjugaison.

Nous donnons une explication des résultats obtenus et proposons un nouveau générateur aléatoire de tresses permettant a priori de se mettre hors de portée de cette attaque.

Puis, nous mettons en relief l'impact de la double structure de Garside dans l'utilisation de notre algorithme de réduction. Enfin, avant de conclure, nous présentons un moyen d'optimiser le recouvrement du secret restant basé sur l'ordre lexicographique.

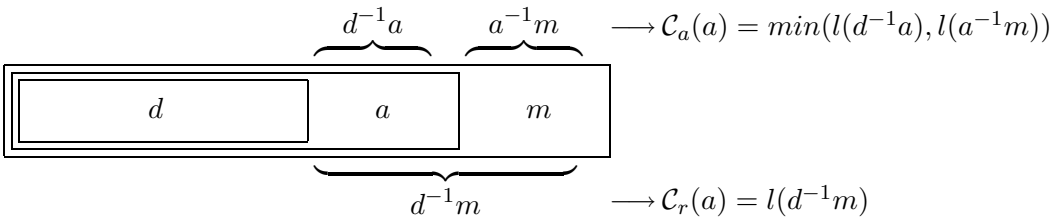

FIG. 6.1 – Réduction du secret

Dans tout ce chapitre, nous nous plaçons sous l'hypothèse de la Remarque 4.1.1 et considérons que l'infimum et la longueur canonique du secret sont connus. Nous concentrons notre rédaction sur la présentation d'Artin ; en annexe se trouvent des tables plus complètes portant sur d'autres générateurs aléatoires et/ou sur la présentation de Birman, Ko et Lee.

Nous notons PCMS le problème de conjugaison multiple simultanée et PC le problème de conjugaison.

6.1 Evaluation du secret restant

L'algorithme Conj présenté au Chapitre 4 considère en entrée une instance de conjugaison $(x, x' = axa^{-1})$ et ressort un diviseur à gauche et un multiple à droite du secret : (d, m). Il est très efficace en termes de complexité et de rapidité d'exécution : $\mathcal{O}(l^3 n \log n)$ dans la présentation d'Artin et $\mathcal{O}(l^3 n)$ dans la présentation de Birman, Ko et Lee.

Nous définissons deux nouveaux paramètres afin d'évaluer la taille du secret restant : la *connaissance absolue* et la *connaissance relative* (voir la Figure 6.1)

- **Connaissance absolue**, $\mathcal{C}_a = min(l(m) - l(a), l(a) - l(d))$,
- **Connaissance relative**, $\mathcal{C}_r = l(m) - l(d)$.

En général, la *connaissance absolue* n'est pas connue car nous ne connaissons pas $l(a)$. Elle représente le nombre d'atomes nécessaires au recouvrement du secret, soit en complétant le diviseur, soit en simplifiant le multiple. Ainsi, ce paramètre conditionne directement la complexité effective des méthodes de recouvrement du secret, basées sur la longueur.

La *connaissance relative* est connue et donne une majoration de la *connaissance absolue* :

$$\mathcal{C}_a \leq \frac{\mathcal{C}_r}{2}$$

Considérons une méthode grossière de recouvrement du secret : essayer tous les mots possibles de longueur comprise entre 0 et $\mathcal{C}_a$. Ainsi, nous pouvons estimer la complexité logarithmique d'une telle méthode par :

$$\mathcal{L}_{B_n} = \log\big(\#\mathcal{A}(M)\big) * (\mathcal{C}_a + 1) \tag{6.1}$$

Remarque 6.1.1. Dans la présentation d'Artin, nous pouvons optimiser la complexité logarithmique de cette méthode grossière de recouvrement du secret en choisissant les atomes, soit dans $\mathcal{A}(B_n^+)$, soit dans l'ensemble des atomes utilisés dans l'écriture de $d^{-1}m$. En effet, les relations de la présentation laissent stable l'ensemble des atomes utilisés dans l'écriture d'un mot. Ainsi, l'expression devient :

$$\mathcal{L}_{B_n} = \log\Big(min\big(n-1, \mathcal{C}_r\big)\Big) * (\mathcal{C}_a + 1)$$

6.2 Dépendance aux paramètres d'entrée

Dans cette section, nous montrons l'influence sur l'efficacité de paramètres comme le choix du générateur aléatoire, les variantes du problème de conjugaison, les tailles des éléments.

6.2.1 Dépendance aux générateurs aléatoires

Le choix du générateur aléatoire utilisé conditionne directement l'efficacité de l'algorithme de réduction présenté. Par exemple, la Table 6.1 montre que la taille du secret ne suffit pas à expliquer la faiblesse d'une instance : le GA-Mots et GA-CompC donnent des instances où la taille du secret et l'efficacité sont inversées. Le GA-Cano et le GAMP semblent très affectés par la réduction ; en particulier, le diviseur retourné par l'algorithme est de très bonne qualité, ce qui permet de réduire la taille du secret de près de 98% . La

structure de définition ne semble pas être un argument déterminant : que le secret soit défini selon son écriture sous forme Δ-normale, comme produit d'éléments simples ou qu'il soit défini comme produit d'atomes, nous rencontrons des générateurs résistants sur les deux modèles.

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000								
	$l_c(a)$	$l(a)$	^{*2}	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-Mots	28.8	12040.0	1	2042.5	2022.3	1775.6	4064.8	8630
			2	1483.6	1470.6	1263.4	2954.2	6142
GA-MotsP	28.7	500	1	11.7	209.5	11.7	221.2	62
			2	3.8	51.1	3.7	54.9	23
GA-Cano	10	2393.3	1	18.8	18.5	13.4	37.3	70
			2	3.1	3.1	1.8	6.2	7
GA-CompC^{*1}	13.0	5812.7	1	2870.3	2802.9	2710.6	5673.3	13173

^{*1} : $x, x', a :=$ GA-CompC(5), ^{*2} : instance du PCMS

Tab. 6.1 – Dépendance aux générateurs - PCMS - présentation "Artin"

Notons l'efficacité toute particulière du générateur GA-CompC, qui dans le cas de recherche du multiple, retourne presque à chaque fois le multiple trivial $\Delta^{\sup(a)}$. Ce générateur est totalement résistant à notre attaque et la raison en est simple : il n'y a pas de différence de complexité entre les deux conjugués. Cela se traduit par un biais égal à la longueur canonique du secret.

Nombre de simulations : 1000									
$\mathcal{C}_a$, $\mathcal{C}_r$			$lc(a) = lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	n	30	13.2	36.6	13.3	37.3	13.1	37.4	69
		50	17.6	45.5	17.4	46.0	17.4	45.7	101
		100	31.0	75.0	30.2	72.4	30.2	72.7	195
			$lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	$l_c(a)$	5	12.9	36.2	13.6	37.6	12.9	37.1	68
		10	13.0	37.2	13.0	37.0	13.3	37.5	
		20	13.0	36.2	13.1	36.7	13.0	36.5	

Tab. 6.2 – Nombre de brins/Taille des éléments - GA-Cano - présentation "Artin"

6.2.2 Dépendance aux formats des entrées

Afin d'apprécier les paramètres de taille qui influent sur l'efficacité de la réduction, nous considérons un générateur aléatoire de tresses qui est sensible à notre réduction et les faisons varier : choisissons le GA-Cano.

Nous constatons que le problème de conjugaison de type Ko-Lee, sous les hypothèses faites au chapitre précédent, possède une sécurité équivalente à celle du problème de conjugaison au regard de l'algorithme de réduction. Voir un exemple dans la Table 6.3. De plus,

la Table 6.2 montre que les paramètres mesurant l'efficacité, $\mathcal{C}_a$ et $\mathcal{C}_r$, dépendent essentiellement du nombre de brins. Expérimentalement, la dépendance est une expression linéaire en n, [32]. La différence de taille entre le secret et la clé publique ne semble pas être un paramètre de sécurité dans le cas du générateur GA-Cano et la taille elle-même du secret non plus.

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000							
	$l_c(a)$	$l(a)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-Cano	14.2	3748.4	10.3	12.7	7.2	23.0	31

Tab. 6.3 – Problème de conjugaison de type Ko-Lee - GA-Cano - présentation "Artin"

6.3 Proposition d'un générateur aléatoire

Nous allons expliquer en partie les résultats obtenus et proposer un nouveau générateur aléatoire de tresses tenant compte de notre analyse et résistant à l'algorithme de réduction.

6.3.1 Explication des résultats

L'algorithme de réduction, Conj, exploite la densité de l'écriture sous forme Δ-normale. Ainsi il est naturel d'aller chercher dans cette direction des raisons qui font qu'une instance est faible ou non.

Pour plus de simplicité, nous considérons dans cette section une instance de conjugaison $(x, x' = axa^{-1})$ dont le secret est d'infimum nul : $\inf(a) = 0$. Les résultats précédents nous conduisent à considérer chacun des quatre générateurs aléatoires de tresses introduits dans le Chapitre 2 et à faire les observations ci-dessous. Ces observations, faites sur la présentation d'Artin, sont à nuancer dans la présentation de Birman, Ko et Lee.

Notons que nous avons toujours :

$$\text{avec } \inf(a) = 0, \qquad l(a) + l(a^\star) = l(\Delta) * l_c(a) \tag{6.2}$$

- GA-Mots : Ce générateur aléatoire de tresses produit une forte expansion de la longueur d'entrée ; cependant pour des longueurs finales similaires à celles du GA-MotsP, nous avons :
$$l(a) \approx l(a^\star) \approx \frac{l(\Delta) l_c(a)}{2}$$
De plus, les premiers éléments simples de l'écriture sous forme Δ-normale de a et $a^\star$ sont gros ($\gg \frac{l(\Delta)}{2}$), tandis que les derniers sont petits ($\ll \frac{l(\Delta)}{2}$). Avec l'expansion, ces propriétés persistent mais des éléments simples de longueur sensiblement proche de $\frac{l(\Delta)}{2}$ s'intercalent au milieu de l'écriture.
Pour ce générateur, l'algorithme Conj donne un mauvais diviseur et un mauvais multiple.
- GA-MotsP : Ce générateur aléatoire de tresses ne produit pas d'expansion par rapport à la longueur d'entrée ; de plus, l'infimum est petit voire nul très souvent. Nous notons que :
$$l(a) \ll l(a^\star)$$

Les facteurs de l'écriture sous forme Δ-normale de a sont tous petits ($\ll \frac{l(\Delta)}{2}$), tandis que ceux de l'écriture de $a^\star$ sont par conséquent gros ($\gg \frac{l(\Delta)}{2}$).
Avec ce générateur, l'algorithme Conj donne un bon diviseur et un mauvais multiple.

- GA-Cano : Ce générateur aléatoire de tresses ne produit pas d'expansion de la longueur canonique d'entrée ; nous avons :

$$l(a) \approx l(a^\star) \approx \frac{l(\Delta) l_c(a)}{2}$$

Cependant, les éléments simples de l'écriture sous forme Δ-normale de a et de $a^\star$ sont tous de taille sensiblement proche de $\frac{l(\Delta)}{2}$.
Avec ce générateur, l'algorithme Conj donne un bon diviseur et un bon multiple.

- GA-CompC : Ce générateur aléatoire de tresses a été conçu sur deux propriétés : l'une étant d'augmenter la distance entre les conjugués dans le Super Summit Set, l'autre étant de dissimuler la différence de complexité canonique entre les conjugués. Il en résulte un générateur qui possède un biais sensiblement égal à la longueur canonique du secret. L'algorithme Conj est donc totalement inefficace sur ce générateur. Le diviseur retourné est quelquefois l'élément neutre tandis que le multiple retourné est fréquemment $\Delta^{\sup(a)}$.
Notons toutefois que ces caractéristiques sont semblables à celles de GA-Cano :

$$l(a) \approx l(a^\star) \approx \frac{l(\Delta) l_c(a)}{2} \quad \text{et} \quad l(s_i) \approx \frac{l(\Delta)}{2}$$

Nous pouvons donner une explication simple de ce constat :

En l'absence d'une construction spécifique visant à dissimuler la complexité canonique, nous remarquons que l'algorithme Conj fonctionne d'autant mieux que la normalisation du produit entre a, x et $a^\star$ est assez proche du produit de leur écriture normalisée. La présence d'éléments simples de petite taille terminant l'écriture sous forme Δ-normale permet de répondre à cette contrainte.

Nous avons aussi constaté le rôle joué par $a^\star$ dans la recherche du diviseur ; ainsi, les propriétés que l'on impose sur le secret doivent aussi être vérifiées par son inverse.

6.3.2 Le GA-TronC

Notre objectif est de proposer un générateur aléatoire de tresses permettant de contrôler la longueur canonique et étant résistant aux attaques basées sur la complexité canonique. Le fait qu'il soit possible d'y arriver avec peu de moyens montre que la recherche dans ce domaine peut encore évoluer positivement.

Nous proposons simplement d'utiliser le générateur GA-Mots qui possède, semble-t-il, les bonnes propriétés et de tronquer son écriture sous forme Δ-normale ; ainsi, les tresses générées vérifieront les critères de la section précédente :

- le secret et son inverse ont même structure
- leurs formes Δ-normales se terminent par des facteurs petits (et donc commencent par des gros)

Algorithme 6.3.1. GA-TronC (p, n)

Entrée :	Soit $p, n \in \mathbb{N}$. (p : longueur canonique, n : nombre de brins)
	Faire un tirage aléatoire sur l'ensemble des entiers naturels inférieurs à l'exposant ($\mathcal{E}$) du groupe $\rightarrow r$.
	$a :=$ GA-Mots $(20p)$
	Tant que $l_c(a) < p$ faire
	$a :=$ GA-Mots $(20p)$
	FinTantQue
	Soit $s_1 \cdots s_k$ la forme Δ-normale de $\Delta^{-\inf(a)}a$
	$j := \lfloor \frac{p}{2} \rfloor$
	$b := s_1 \cdots s_j s_{k-(p-j)+1} \cdots s_k$
	Tant que $l_c(b) < p$ faire
	j :=j-1
	$b := s_1 \cdots s_j s_{k-(p-j)+1} \cdots s_k$
	FinTantQue
	Soit $s_1, \cdots s_q$ la suite normale à gauche associée à b $(q \geq p)$
Sortie :	$[r, [s_1, \cdots, s_p]]$

Le biais de ce générateur a un comportement assez prévisible et l'estimation de la longueur canonique du secret à partir d'une instance de conjugaison, $(x, x' = axa^{-1})$, est relativement fiable comme le montre la Table 6.4.

$n = 30 \quad l_c = 10 \quad$ Nombre de simulations : 1000		
Estimation l_c	$l_c(a) - e \approx 1.53$	$\mathcal{P}(e = l_c(a) - i)_{i\in[1,2]} \approx 0.2,\ \mathcal{P}(\lvert l_c(a) - e - m\rvert \leq 5) = 0.97$
Biais	$b \approx 3.14$	$\mathcal{P}(b = 3) = 0.26,\ \mathcal{P}(b = 2, 4) \approx 0.2,\ \mathcal{P}(b \in [1, 6]) = 0.95$

Tab. 6.4 – Statistique du GA-TronC sur la présentation d'Artin

Ceci, ajouté à la Table 6.5, permet de compléter notre analyse :

> Nous avons une attaque qui marche très bien sur le générateur GA-Cano, qui peut sembler générique à première vue, tellement sa construction est naturelle. Puis, nous avons un nouveau générateur, GA-TronC, de construction très simple, satisfaisant des critères compréhensibles, qui produit des tresses du même type que le GA-Cano mais qui rend cette attaque et d'autres quasi inefficaces.

$n = 30 \quad l = 500 \quad l_c = 10 \quad$ Nombre de simulations : 1000								
	$l_c(a)$	$l(a)$	*	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-TronC	10	2651.6	1	1113.7	1180.5	868.0	2294.3	4221
			2	826.1	935.6	636.5	1761.7	3097
			3	547.2	650.0	410.3	1197.2	1998

* : instance du PCMS

Tab. 6.5 – Dépendance au GA-TronC - PCMS - présentation "Artin"

Le rôle joué par les générateurs aléatoires de tresses dans les attaques basées sur la complexité est crucial. Cela montre qu'il y a vraisemblablement un travail à faire dans ce domaine et que des résultats concluants peuvent en être attendus.

Remarque 6.3.2. Notons que le GA-TronC satisfait aussi le critère de résistance à l'attaque de Hofheinz et Steinwandt, [22]. La distance des deux conjugués résultants de l'attaque d'une instance de conjugaison générée avec le généréteur GA-TronC est au moins de deux, en pratique.

$n = 30$ $l = 500$ Nombre de simulations : 20						
Instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
Présentation : BKL		$l_c(a_{\text{Artin}}) = 29.1,\ l(a_{\text{Artin}}) = 500,$ $l_c(a_{\text{BKL}}) = 46.4,\ l(a_{\text{BKL}}) = 500$				
1	B	119.8	198.8	118.1	318.7	1044
	B // A	13.1	115.6	13.1	128.7	124
	B+A+B	16.2	36.0	13.9	52.2	130
2	B	107.3	173.9	104.5	281.2	924
	B // A	5.2	33.0	4.7	38.2	50
	B+A+B	5.5	4.7	3.4	10.2	39
3	B	100.9	162.3	97.7	263.2	865
	B // A	2.0	23.1	1.3	25.1	20
	B+A+B	4.6	3.9	8.4	2.7	32

Tab. 6.6 – Mode parallèle/séquentiel - PCMS - GA-MotsP - génération "Artin"

6.4 Utilisation des modes parallèle / séquentiel

L'existence des deux structures de Garside dans les groupes de tresses permet de traiter une même instance sur les deux présentations. Ceci apporte un gain et permet de réduire davantage la taille des secrets résultants. Les Tables 6.6 et 6.7 permettent d'apprécier les différents modes ; en particulier, nous voyons que l'utilisation en parallèle et en sequentiel sont complémentaires et vraisemblablement cumulables pour obtenir de meilleurs résultats.

$n = 30$ $l = 500$ Nombre de simulations : 50						
Instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
Présentation : Artin		$l_c(a_{\text{Artin}}) = 28.4,\ l(a_{\text{Artin}}) = 500,$ $l_c(a_{\text{BKL}}) = 45.4,\ l(a_{\text{BKL}}) = 500$				
1	A	11.2	258.0	11.2	269.2	59
	A+B+A	8.9	147.9	8.9	156.7	48
2	A	3.7	73.1	3.6	76.7	22
	A+B+A	2.8	31.5	2.5	34.3	17
3	A	1.4	38.8	1.3	40.3	11
	A+B+A	1.1	2.4	0.9	3.4	3

Tab. 6.7 – Mode séquentiel - PCMS - GA-MotsP - génération "Artin"

Dans ces deux exemples de simulations, les instances sont générées avec le GA-MotsP dans la présentation d'Artin et nous considérons l'efficacité des différents modes d'utilisation de l'algorithme de réduction dans les deux présentations. Les symboles B et A traduisent respectivement que l'algorithme est appliqué sur la présentation de Birman, Ko et Lee ou sur celle d'Artin.

$n = 30 \quad l = 500 \quad l_c = 10$	Nombre de simulations : 100		
Générateur	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-Mots	$l_c(a_{\text{Artin}}) = 28.4,\ l(a_{\text{Artin}}) = 11860.2,$ $l_c(a_{\text{BKL}}) = 40.2,\ l(a_{\text{BKL}}) = 1141.3$		
A	1752.4	4070.1	8518
A+B+A	799.4	1896.6	3888
A+(B+A)^2	673.3	1587.9	3275
A+(B+A)^3	641.3	1507.7	3120
GA-MotsP	$l_c(a_{\text{Artin}}) = 28.4,\ l(a_{\text{Artin}}) = 500,$ $l_c(a_{\text{BKL}}) = 45.5,\ l(a_{\text{BKL}}) = 500$		
A	12.1	231.0	64
A+B+A	9.6	148.1	52
A+(B+A)^2	8.9	135.8	48
A+(B+A)^3	8.7	134.9	47
GA-Cano	$l_c(a_{\text{Artin}}) = 10,\ l(a_{\text{Artin}}) = 2396.2,$ $l_c(a_{\text{BKL}}) = 74.2,\ l(a_{\text{BKL}}) = 2396.2$		
A	12.0	34.0	63
A+B+A	9.3	28.1	50
A+(B+A)^2	9.0	26.7	47
A+(B+A)^3	9.0	26.7	47
GA-TronC	$l_c(a_{\text{Artin}}) = 10,\ l(a_{\text{Artin}}) = 2589.3,$ $l_c(a_{\text{BKL}}) = 31.2,\ l(a_{\text{BKL}}) = 2589.3$		
A	865.2	2233.5	4208
A+B+A	479.3	1311.4	2333
A+(B+A)^2	384.4	1113.3	1872
A+(B+A)^3	357.4	1052.1	1741

TAB. 6.8 – Mode séquentiel - PC - présentation "Artin"

L'itération du mode séquentiel tend à stabiliser la réduction, mais permet cependant de casser la complexité de l'attaque supplémentaire de recouvrement du secret. La Table 6.8 montre ce phénomène de stabilisation : il suffit de regarder une colonne (celle de la complexité ($\mathcal{L}_{B_n}$) ou celle des paramètres $\mathcal{C}_a$ et $\mathcal{C}_r$). Le processus séquentiel A+B+A donne de meilleurs résultats que la réduction simple A, mais de moins bons résultats que si l'on itère encore la réduction (A+B+A)+B+A (=A+(B+A)^2).

6.5 Ordre lexicographique

Nous proposons ici une méthode moins grossière, mais tout aussi exhaustive, afin de recouvrer le secret restant. Le problème de la recherche exhaustive est en pratique de pouvoir générer l'ensemble suivant :

$$E_{d^{-1}m,\mathcal{C}_a} = \left\{ t \in M;\ l(t) \leq \mathcal{C}_a \text{ et } \left(t \prec d^{-1}m \text{ ou } d^{-1}m \succ t \right) \right\}$$

Ainsi la complexité effective minimale d'une méthode exhaustive est $\#E_{d^{-1}m,\mathcal{C}_a}$.

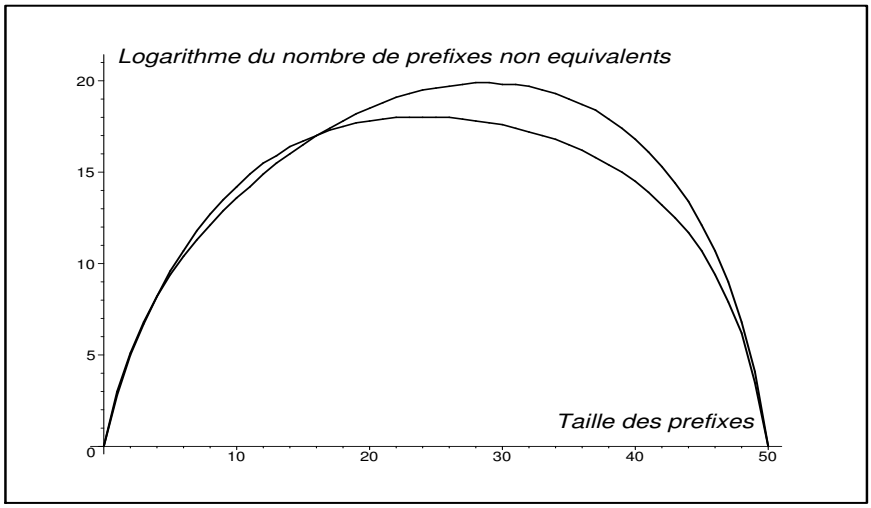

FIG. 6.2 – Nombre de préfixes de mots de longueur 50 dans B_{20}^{+}

Connaissant $d \prec a \prec m$, $E_{d,m,ak}$ est l'ensemble de tous les éléments du monoïde qui sont susceptibles de compléter le diviseur, d, en le secret, a, ou de réduire le multiple, m, en a. Il est facile de parcourir cet ensemble en utilisant l'ordre lexicographique : identifier tous les préfixes non équivalents. Nous utilisons des algorithmes, présentés en annexe, qui sont dérivés des travaux de Jacquemard [24]. La complexité est clairement exponentielle en la taille du secret restant et en le nombre de brins.

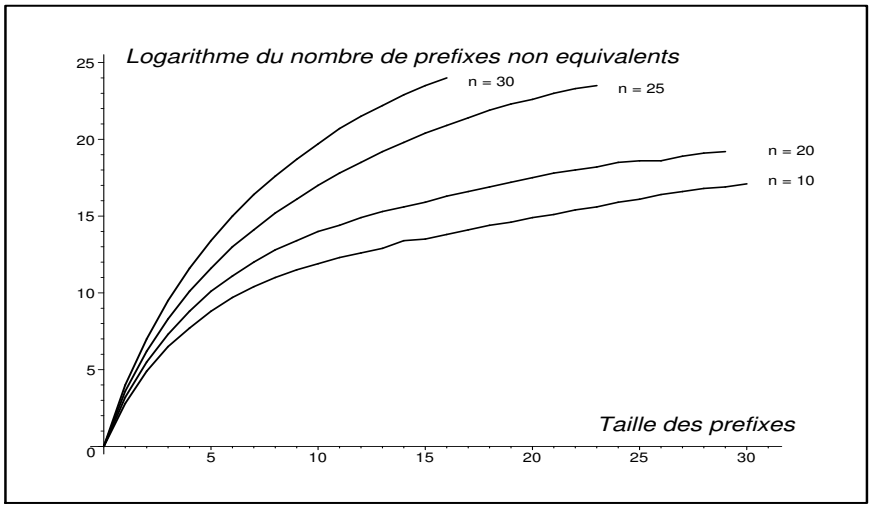

FIG. 6.3 – Préfixes de mots de longueur 60 dans B_n^{+} pour $n \in \{10, 20, 25, 30\}$

Afin de mieux comprendre ce type d'ensemble, nous proposons quelques graphiques permettant d'apprécier plusieurs paramètres relatifs à la recherche de préfixes. La Figure 6.2 propose des courbes représentant le logarithme du nombre de préfixes non

équivalents en fonction de la longueur des préfixes, sur des mots positifs de longueur 50 dans B_{20}^{+}.

Même si cela n'est pas automatique, on peut raisonnablement s'attendre à ce que le nombre de préfixes à gauche soit sensiblement le même que celui des préfixes à droite. Ainsi, nous limitons notre étude à considérer les préfixes de longueur inférieure à la demi-longueur du mot d'origine.

La Figure 6.3 propose d'analyser le comportement du logarithme du nombre de préfixes de mots de longueur fixée à 60 en fonction de la taille des préfixes et de la variation du nombre de brins. La croissance du nombre de préfixes lorsque le nombre de brins augmente est prévisible, mais ce phénomène se stabilise comme le décrit la remarque suivante.

Remarque 6.5.1. Un phénomène se produit lorsque la *connaissance relative* est petite. Si $\mathcal{C}_r \ll n$, alors les atomes servant à écrire $d^{-1}m$ commutent entre eux : en pratique, un tel cas correspond à une distribution assez uniforme des atomes. Ainsi, nous avons :

$$\mathcal{C}_r \ll n \quad \Rightarrow \quad \#E_{d^{-1}m,\mathcal{C}_a} \approx C_{\mathcal{C}_r}^{\mathcal{C}_a} \tag{6.3}$$

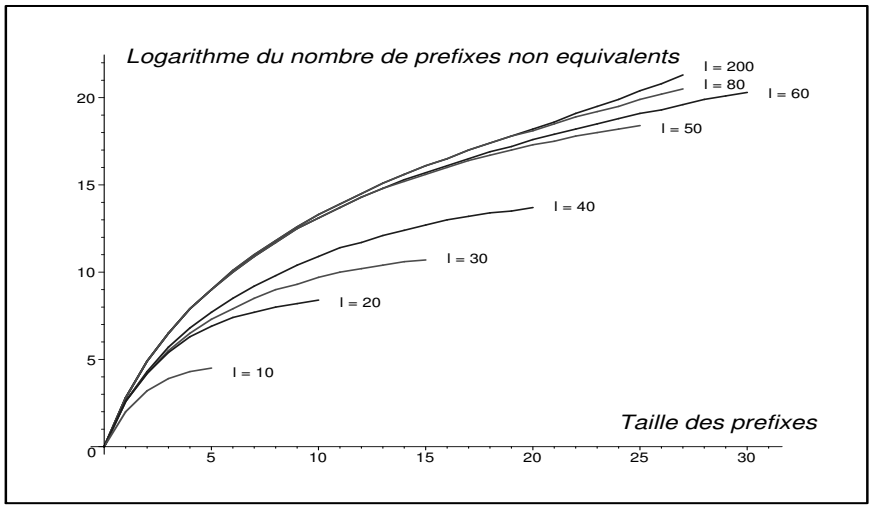

FIG. 6.4 – Préfixes de sous-mots d'un mot donné dans B_{15}^{+}

La dernière figure, 6.4, propose toujours la même sorte de graphique : logarithme du nombre de préfixes en fonction de la taille des préfixes. Nous avons fixé un mot positif d'une grande longueur et nous considérons les sous-mots de longueur $l \in \{10, 20, 30, 40, 50, 60, 80, 200\}$. On observe naturellement une stabilisation des préfixes de petite et moyenne longueur quand on augmente la longueur du mot considéré.

Pour terminer cette section, nous pouvons observer, dans la Table 6.9, la réduction de la complexité lorsque l'on considère la méthode conduisant à l'expression (6.1) et l'approche dénombrant l'ensemble des préfixes non équivalents.

Remarque 6.5.2. L'estimation de $\log\left(\#E_{d^{-1}m,\mathcal{C}_a}\right)$ n'est pas toujours possible : l'ordre de calcul acceptable est de 2^{28}. Quand les paramètres $d^{-1}m$ et $\mathcal{C}_a$ engendrent un ensemble de taille plus grande, nous avons considéré une courbe de référence (celle de la Figure 6.4

pour $l = 50$). Utilisant des affinités horizontale et verticale, nous avons transformé la courbe afin de prolonger l'amorce que nous pouvons générer jusqu'à notre limite calculatoire de 2^{28}.

n	$l_c(a)$	mode	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$	$\log\left(\#E_{d^{-1}m,\mathcal{C}_a}\right)$
30	10	A	13.4	36.3	70	16
		A+(B+A)^2	9.6	27.9	51	14
50	15	A	17.7	46.2	104	22
		A+(B+A)^2	14.1	38.8	80	20
100	15	A	31.7	75.2	204	42*
		A+(B+A)^2	25.3	61.8	156	35*

* estimation expliquée à la Remarque 6.5.2

TAB. 6.9 – Mode séquentiel - PC - GA-CANO - présentation "Artin"

Remarque 6.5.3. L'intérêt de cette section est à relativiser. Même si cette approche nous permet de réduire grandement, voire même de façon décisive, la borne de complexité donnée par (6.1),

$$\log(\#E_{d^{-1}m,\mathcal{C}_a}) \ll \mathcal{L}_{B_n}$$

il faut aussi considérer le coût de génération d'un tel ensemble. En effet, l'algorithme proposé a inévitablement un coût de génération supérieur à la taille de l'ensemble. Ainsi, à moins de déterminer un algorithme plus efficace pour générer des ensembles de type $E_{d^{-1}m,\mathcal{C}_a}$, l'estimation de la taille d'un tel ensemble donne seulement une information théorique sur la sécurité.

6.6 Conclusion

Cette étude sur l'algorithme de réduction de la conjugaison confirme un point important de la cryptographie basée sur les tresses : la sécurité d'une instance du problème de conjugaison dépend essentiellement du générateur aléatoire utilisé.

Les attaques basées sur la complexité peuvent sembler efficaces, mais, bien souvent, leur efficacité reflète uniquement la façon dont l'instance de conjugaison est générée. Dans cette partie, nous avons constaté qu'un générateur aléatoire de tresses, qui semble naturel et générique, rend le problème de conjugaison et ses variantes très faibles du point de vue de la sécurité. Les paramètres de sécurité sont principalement réduits à l'augmentation du nombre de brins, et donc de la taille du groupe. Cette augmentation entraîne un alourdissement des protocoles, avec notamment des clés plus grosses et des temps de calcul plus longs.

Nous avons montré qu'une simple manipulation permettait de remédier aux faiblesses du générateur cité. Nous avons proposé un nouveau générateur aléatoire résistant à l'algorithme de réduction, mais aussi à l'attaque présentée dans [22]. Sans exploiter des arguments de symétrie ou de complexité comme l'a fait finement Sibert dans le cadre de la génération aléatoire de tresses, nous avons montré que la recherche de nouveaux

générateurs aléatoires dans les groupes de tresses n'était pas impossible et qu'elle constituait une ouverture raisonnable pour le développement de l'utilisation du problème de conjugaison dans la cryptographie basée sur les tresses.

De cette étude, nous pouvons exhiber quelques principes supplémentaires favorisant la résistance d'une instance de conjugaison aux attaques basées sur la complexité :

- la structure suggérée pour le secret doit être aussi respectée par son inverse, afin de ne pas favoriser une attaque à gauche ou à droite.
- la structure suggérée ne doit pas dépendre, si possible, de la présentation de Garside, afin de ne pas favoriser une attaque en choisissant une présentation spécifique.
- la forme Δ-normale du secret peut se terminer par des éléments simples de petite taille ; ceci permet une certaine inter-pénétration des formes Δ-normales dans l'écriture du conjugué.
- le biais du générateur aléatoire peut être sensiblement égal à la longueur canonique du secret ; ceci permet de résister aux attaques basées sur la longueur canonique comme celle présentée ici.

Chapitre 7

Fonction *cyclage*

Dans un article de référence de la cryptographie basée sur les tresses, [28], Ko, Lee, Cheon, Han, Kang et Park proposent un problème difficile, potentiellement utilisable comme primitive cryptographique, basé directement sur la fonction *cyclage* : le problème des cyclages.

Le principal résultat de ce chapitre est d'établir que l'inversion de la fonction *cyclage* dans les groupes de Garside admet une solution polynomiale. En particulier, le problème des cyclages ne semble plus en mesure d'apporter une quelconque sécurité dans les groupes de tresses.

Le travail effectué se décompose en deux étapes :

- Nous établissons un algorithme polynomial qui inverse la fonction *cyclage* dans les groupes de Garside. L'algorithme repose sur l'établissement d'une partition de l'ensemble des antécédents en quatre sous-ensembles et sur la donnée de plusieurs critères permettant de caractériser ces sous-ensembles.
- Nous inversons localement la fonction *cyclage* sur chacun des sous-ensembles : nous donnons quatre algorithmes, un pour chaque sous-ensemble.

Nous nous plaçons dans le groupe de Garside G de monoïde de Garside M et d'élément de Garside Δ.

7.1 Présentation

Rappelons la définition de la fonction *cyclage* et introduisons le problème des cyclages :

Définition 7.1.1. Soit G un groupe de Garside et $\Delta^r s_1 \cdots s_p$ la forme Δ-normale de x, un élément de G. On définit le *cyclé* de x par :

$$c(x) = \Delta^r s_2 \cdots s_p \tau^{-r}(s_1)$$

On appelle *cyclage* la fonction de G dans G qui à tout élément associe son cyclé.

Problème 7.1.2. [28]**(problème des cyclages)** : Etant donnés un entier naturel k et un élément y qui soit un cyclé $k^{\text{ième}}$ dans le groupe G, déterminer un élément $x \in G$ vérifiant $c^k(x) = y$.

La fonction *cyclage* intervient à plusieurs niveaux dans la cryptographie basée sur les tresses. En plus d'être l'élément de base du problème des cyclages, elle est un outil utilisé dans la résolution du problème de conjugaison : d'une part elle permet de construire un représentant d'un élément dans son Super Summit Set, [6], et d'autre part, elle est l'outil de base dans la construction de l'Ultra Summit Set, [21]. De plus, cette fonction est utilisée dans le générateur aléatoire GA-CompC afin de garantir une distance suffisante entre les conjugués dans leur Super Summit Set.

On remarque que le cyclé est un conjugué de la tresse de départ :

$$x = \Delta^r s_1 \cdots s_p \qquad c(x) = \tau^{-r}(s_1)^{-1} x \tau^{-r}(s_1)$$

Ainsi, déterminer un antécédent d'un cyclé revient à résoudre une instance particulière de conjugaison : nous connaissons le conjugué et nous devons déterminer un élément qui s'y conjugue en ayant une condition sur le conjugant.

Remarque 7.1.3. La fonction *cyclage* est constante sur l'ensemble $\{\Delta^r,\ r \in \mathbb{Z}\}$.

7.1.1 Partition de l'ensemble des antécédents

Le théorème suivant propose une partition de l'ensemble des antécédents d'un élément pour la fonction *cyclage* en quatre sous-ensembles. Une étude de chaque sous-ensemble est faite plus loin.

Théorème 7.1.4. *Soit a et b deux éléments de G ; nous avons :*

$$c(b) = a \quad \Rightarrow \quad \begin{cases} \inf(b) = \inf(a) \text{ et } l_c(b) \in [l_c(a), l_c(a)+1] \\ \text{ ou} \\ \inf(b) = \inf(a) - 1 \text{ et } l_c(b) \in [l_c(a)+1, l_c(a)+2] \end{cases}$$

Preuve : Soit $\Delta^r a_1 \cdots a_p$ et $\Delta^k b_1 \cdots b_q$ les formes Δ-normales de a et b.

$$c(b) = a \quad \Leftrightarrow \quad \Delta^r a_1 \cdots a_p = \Delta^k b_2 \cdots b_q \tau^{-k}(b_1)$$

• Montrons $q - 1 \leq \sup\left(b_2 \cdots b_q \tau^{-k}(b_1)\right) \leq q$:
La deuxième inégalité vient du Corollaire 1.2.27 et du fait qu'un produit de q éléments simples divise à gauche Δ^q. La première inégalité découle directement de la Proposition 3.2.5, sachant que $(b_2, \cdots, b_q)$ est une suite normale à gauche.
• Montrons $0 \leq \inf\left(b_2 \cdots b_q \tau^{-k}(b_1)\right) \leq 1$:
La première inégalité est immédiate ; l'élément considéré est dans le monoïde donc son infimum est positif. La deuxième inégalité demande légèrement plus de manipulation. Supposons que $\Delta \prec b_2 \cdots b_q \tau^{-k}(b_1)$, comme τ laisse stable le monoïde, nous avons :

$$\begin{aligned} \Delta \prec b_2 \cdots b_q \tau^{-k}(b_1) \quad &\Leftrightarrow \quad e \prec \Delta^{-1} b_2 \cdots b_q \tau^{-k}(b_1) \\ &\Leftrightarrow \quad e \prec \underbrace{b_2 \cdots b_q \overbrace{\tau^{-k}(b_1)\Delta^{-1}}^{\in S^{-1}}}_{\prec b_2 \cdots b_q} \end{aligned}$$

On obtient ainsi $0 \leq \inf\left(b_2 \cdots b_q \tau^{-k}(b_1)\Delta^{-1}\right) \leq \inf(b_2 \cdots b_q) = 0$, ce qui montre la deuxième inégalité.

• Pour pouvoir conclure, il ne reste plus qu'à exprimer ces relations sur $c(b)$, en fonction de a :

$$c(b) = a \quad \Rightarrow \quad k+q-1 \leq r+p \leq k+q \quad \text{et} \quad k \leq r \leq k+1$$

□

Ainsi, on note qu'il y a quatre types différents d'antécédents par la fonction *cyclage* :

Définition 7.1.5. Soit a un élément de G et b un élément de cyclé a ; on dit que b est de :

Type 1 si $\inf(b) = \inf(a)$ et $l_c(b) = l_c(a)$

Type 2 si $\inf(b) = \inf(a)$ et $l_c(b) = l_c(a) + 1$

Type 3 si $\inf(b) = \inf(a) - 1$ et $l_c(b) = l_c(a) + 1$

Type 4 si $\inf(b) = \inf(a) - 1$ et $l_c(b) = l_c(a) + 2$

Etant donné un élément, ces quatres familles forment une partition de l'ensemble de ses antécédents. L'exemple ci-dessous montre que les quatres familles peuvent être non vides simultanément, et en particulier que la fonction *cyclage* n'est pas injective.

Exemple 7.1.6. Considérons $[1, [(1,4,3)(2,5), (1,3,4,2), (1,4,2,5,3)]]$ la forme Δ-normale de l'élément a de B_5^+. Pour $i \in \{1,2,3,4\}$, l'élément b_i est un antécédent de *Type i* relativement à a :

$b_1 : [1, [(1,3,4)(2,5), (1,4,3)(2,5), (2,3,4)]]$
$b_2 : [1, [(1,2,5,3,4), (1,4,3)(2,5), (2,3,4), (3,4)]]$
$b_3 : [0, [(1,5,2,4), (1,4,2,3,5), (1,4,2)(3,5), (2,4)(3,5)]]$
$b_4 : [0, [(1,5,4)(2,3), (1,4,2,5), (1,4,2)(3,5), (2,4)(3,5), (3,4)]]$

7.1.2 Compléments sur la forme Δ-normale

La forme Δ-normale est introduite et définie à la Section 1.2.3 du Chapitre 1. Cette section a pour but d'introduire plusieurs notions et de présenter un autre formalisme des suites normales à gauche et donc de la forme Δ-normale.

Notation 7.1.7. Soit M un monoïde de Garside ; on note $\mathcal{A}(M)$ l'ensemble des atomes du monoïde M.

Définition 7.1.8. Soit x un élément du monoïde M et $s \in S$ un élément simple ; on définit le *Starting Set* et le *Finishing Set* de x, puis le *complémentaire à droite* de s par :

$$\mathcal{S}(x) = \{g \in \mathcal{A}(M);\ g \prec x\}$$
$$\mathcal{F}(x) = \{g \in \mathcal{A}(M);\ x \succ g\}$$
$$\mathcal{R}(s) = \{g \in \mathcal{A}(M);\ sg \in S\} = \mathcal{S}\left(\partial(s)\right)$$

Remarque 7.1.9. Le lecteur prendra note de la différence entre S qui est l'ensemble des éléments simples et $\mathcal{S}(\ )$ qui est le *Starting Set*.

En conséquence directe de la définition, nous avons le corollaire qui suit.

Corollaire 7.1.10. *Soit u et v deux éléments du monoïde M, alors nous avons :*

$$u \prec v \quad \Rightarrow \quad \mathcal{S}(u) \subset \mathcal{S}(v) \qquad et \qquad u \succ v \quad \Rightarrow \quad \mathcal{F}(v) \subset \mathcal{F}(u)$$

Lemme 7.1.11. *Soit u et v deux éléments simples du monoïde M, alors nous avons :*

$$u \succ v \quad \Rightarrow \quad \mathcal{R}(u) \subset \mathcal{R}(v)$$

Preuve : Soit $d \in S$ tel que $u = dv$; nous avons :

$$\Delta = u\partial(u) = v\partial(v) = dv\partial(u)$$

Utilisant le fait que $\partial^2 = \tau$ sur S, Proposition 3.1.3, nous obtenons :

$$\partial\left(v\partial(u)\right) = \tau(d) \quad \text{donc} \quad v\partial(u)\tau(d) = \Delta \quad \text{et} \quad \partial(v) = \partial(u)\tau(d)$$

Ainsi, ayant supposé que $u \succ v$, on a obtenu que $\partial(u) \prec \partial(v)$. Le Corollaire 7.1.10 permet de déduire que $\mathcal{S}\left(\partial(u)\right) \subset \mathcal{S}\left(\partial(v)\right)$ ce qui donne, par définition du *complémentaire à droite* : $\mathcal{R}(u) \subset \mathcal{R}(v)$.

□

Proposition 7.1.12. *[13] L'élément de Garside est le ppcm à droite de l'ensemble des éléments simples : $\Delta = \vee S$*

Remarque 7.1.13. Soit s_1 et s_2 deux éléments simples du monoïde M ; les définitions d'une suite normale à gauche et du *complémentaire à droite* donnent :

$$(s_1, s_2) \text{ est une suite normale à gauche} \quad \Leftrightarrow \quad \mathcal{S}(s_2) \cap \mathcal{R}(s_1) = \emptyset$$

Lemme 7.1.14. *Soit $(s_1, s_2, \cdots, s_n)$ une suite normale à gauche dans M, alors :*

$$\mathcal{S}(s_1 s_2 \cdots s_n) = \mathcal{S}(s_1)$$

Preuve : Le Corollaire 7.1.10 nous donne une première inclusion : $\mathcal{S}(s_1) \subset \mathcal{S}(s_1 s_2 \cdots s_n)$. La Proposition 7.1.12 donne la deuxième inclusion :

$$\mathcal{S}(s_1 s_2 \cdots s_n) \subset \mathcal{S}(s_1 s_2 \cdots s_n \wedge \Delta) = \mathcal{S}(s_1)$$

L'égalité est établie.

□

Remarque 7.1.15. La définition des suites normales à gauche nous donne directement le résultat suivant : si (s_1, s_2) et $(s_2, s_3, \cdots, s_n)$ sont deux suites normales à gauche de M, alors $(s_1, s_2, \cdots, s_n)$ est aussi une suite normale à gauche.

7.2 Inversion de la fonction

Le but de cette section est de montrer le théorème suivant :

Théorème 7.2.1. *Dans les groupes de Garside, il existe un algorithme qui inverse la fonction cyclage et qui possède une complexité polynomiale en le nombre d'atomes et la complexité de la forme Δ-normale.*

Nous donnons un algorithme vérifiant les conditions du théorème ; cela prouve sa véracité.

7.2.1 Les critères

Notre démarche consiste à étudier les différents sous-ensembles d'antécédents. En particulier, nous caractérisons ces sous-ensembles d'antécédents en définissant des critères qui s'appliquent sur l'élément simple de tête de la forme Δ-normale d'un antécédent. La définition même de la fonction *cyclage* fait jouer un rôle essentiel à cet élément simple ; il est donc naturel de baser l'étude sur les propriétés que vérifie ou non cet élément.

Définition 7.2.2. Soit a un élément de G de longueur canonique non nulle et $s \in S\backslash\{1, \Delta\}$ un élément simple de M. Nous définissons les critères suivants :

$$\mathcal{C}1 \qquad \Delta^{-\inf(a)} a \succ s$$

$$\mathcal{C}2 \qquad l_c\left(\Delta^{-\inf(a)} a s^{-1}\right) = l_c(a) - 1$$

$$\mathcal{C}3 \qquad \mathcal{S}\left(\Delta^{-\inf(a)} a s^{-1}\right) \cap \mathcal{R}\left(\tau^{\inf(a)}(s)\right) = \emptyset$$

$$\mathcal{C}4 \qquad l_c\left(\Delta^{-\inf(a)+1} a s^{-1}\right) = l_c(a)$$

$$\mathcal{C}5 \qquad \mathcal{S}\left(\Delta^{-\inf(a)+1} a s^{-1}\right) \cap \mathcal{R}\left(\tau^{\inf(a)-1}(s)\right) = \emptyset$$

Ces critères sont incontournables dans l'étude des antécédents d'un élément par la fonction *cyclage* :

- Le critère $\mathcal{C}1$ s'applique uniquement aux antécédents de *Type 1*et *2* ; il conditionne directement leur existence.
- Les critères $\mathcal{C}2$ et $\mathcal{C}4$ permettent de gérer la longueur canonique des antécédents.
- Les critères $\mathcal{C}3$ et $\mathcal{C}5$ traduisent le fait que l'élément simple considéré est bien celui de tête de la forme Δ-normale de l'antécédent.

La propriété suivante précise l'impact de ces critères :

Corollaire 7.2.3. *Soit $s \in S\backslash\{1, \Delta\}$ un élément simple du monoïde M.*

$$s \text{ vérifie } \mathcal{C}5 \quad \Rightarrow \quad \inf\left(\Delta^{-\inf(a)+1} a s^{-1}\right) = 0$$

Preuve : L'élément $s \in S\backslash\{1,\Delta\}$ vérifie $\mathcal{C}5$ implique

$$\mathcal{R}\left(\tau^{\inf(a)-1}(s)\right) \neq \emptyset \quad \Rightarrow \quad \mathcal{S}\left(\Delta^{-\inf(a)+1} a s^{-1}\right) \neq \mathcal{A}(M)$$

Donc la Proposition 7.1.12 donne $\inf\left(\Delta^{-\inf(a)+1} a s^{-1}\right) = 0$.

□

Lemme 7.2.4. *Soit a un élément de G de longueur canonique non nulle et $s \in S$ un élément simple du monoïde M. Nous avons :*

- *Si s vérifie $\mathcal{C}1$ mais pas $\mathcal{C}2$, alors on a : $l_c(\Delta^{-\inf(a)} a s^{-1}) = l_c(a)$.*
- *Si s vérifie $\mathcal{C}5$ mais pas $\mathcal{C}4$, alors on a : $l_c(\Delta^{-\inf(a)+1} a s^{-1}) = l_c(a) + 1$.*

Preuve : Si s vérifie $\mathcal{C}1$ alors $l_c(\Delta^{-\inf(a)} a s^{-1}) \in [l_c(a) - 1, l_c(a)]$. Comme s ne vérifie pas $\mathcal{C}2$ alors $l_c(\Delta^{-\inf(a)} a s^{-1}) \neq l_c(a) - 1$, d'où le résultat.
Le Corollaire 7.2.3 nous donne que $inf(\Delta^{-\inf(a)+1} a s^{-1}) = 0$, ainsi $l_c(\Delta^{-\inf(a)+1} a s^{-1}) \in [l_c(a), l_c(a) + 1]$; le deuxième point en découle.

□

Proposition 7.2.5. ***(antécédents de la fonction cyclage)*** *Soit a un élément de G de longueur canonique non nulle. Il existe $b \in G$ tel que b soit un antécédent de a de*

$$\begin{array}{lcl}
\textit{Type 1} & \Leftrightarrow & \exists s \in S\backslash\{1,\Delta\};\ s \textit{ vérifie } \mathcal{C}1,\ \mathcal{C}2,\ \mathcal{C}3 \textit{ et } b = sas^{-1} \\
\textit{Type 2} & \Leftrightarrow & \exists s \in S\backslash\{1,\Delta\};\ s \textit{ vérifie } \mathcal{C}1,\ \overline{\mathcal{C}2},\ \mathcal{C}3 \textit{ et } b = sas^{-1} \\
\textit{Type 3} & \Leftrightarrow & \exists s \in S\backslash\{1,\Delta\};\ s \textit{ vérifie } \mathcal{C}4,\ \mathcal{C}5 \textit{ et } b = sas^{-1} \\
\textit{Type 4} & \Leftrightarrow & \exists s \in S\backslash\{1,\Delta\};\ s \textit{ vérifie } \overline{\mathcal{C}4},\ \mathcal{C}5 \textit{ et } b = sas^{-1}
\end{array}$$

Preuve : Tenant compte du Lemme 7.2.4, les preuves des quatre équivalences ont exactement le même schéma ; ainsi, nous nous contentons de prouver la première équivalence.

$\boxed{\Rightarrow}$ Soit $\Delta^r b_1 \cdots b_p$ la forme Δ-normale de b, un antécédent de *Type 1* de a. La relation

$$b_2 \cdots b_p \tau^{-r}(b_1) = \Delta^{-r} a = \Delta^{-\inf(a)} a$$

montre que $s = \tau^{-r}(b_1)$ vérifie $\mathcal{C}1$, $\mathcal{C}2$ et $\mathcal{C}3$.
$\boxed{\Leftarrow}$ Soit s un élément simple vérifiant $\mathcal{C}1$, $\mathcal{C}2$ et $\mathcal{C}3$. Posons $b = sas^{-1}$ et montrons que b est un antécédent de *Type 1* de a. Notons $\Delta^r a_1 \cdots a_p$ la forme Δ-normale de a ; alors $b = \Delta^r \tau^r(s) a_1 \cdots a_p s^{-1}$.

$$\begin{array}{l}
s \text{ vérifie } \mathcal{C}1 \ \Rightarrow a_1 \cdots a_p s^{-1} \in M \\
s \text{ vérifie } \mathcal{C}2 \ \Rightarrow l_c(a_1 \cdots a_p s^{-1}) = p - 1
\end{array}$$

Donc s vérifie $\mathcal{C}1$ et $\mathcal{C}2$ implique qu'il existe une suite normale à gauche de M, $(b_2, \cdots, b_p)$, telle que $a_1 \cdots a_p s^{-1} = b_2 \cdots b_p$. Posons $b_1 = \tau^r(s) \in S\backslash\{1,\Delta\}$. Le Lemme 7.1.14 donne $\mathcal{S}(b_2) = \mathcal{S}(a_1 \cdots a_p s^{-1}) = \mathcal{S}\left(\Delta^{-\inf(a)} a s^{-1}\right)$; nous avons :

$$s \text{ vérifie } \mathcal{C}3 \quad \Leftrightarrow \quad \mathcal{S}(b_2) \cap \mathcal{R}(b_1) = \emptyset$$

Ainsi, (b_1, b_2) est une suite normale à gauche d'après la Remarque 7.1.13 ; de plus, la Remarque 7.1.15 donne que $\Delta^r b_1 \cdots b_p$ est la forme Δ-normale de b. L'élément b vérifie $\inf(b) = \inf(a)$, $l_c(b) = l_c(a)$ et $c(b) = \Delta^r b_2 \cdots b_p \tau^{-r}(b_1) = \Delta^r a_1 \cdots a_p s^{-1} s = a$; donc b est un antécédent de *Type 1* de a.

□

7.2.2 Existence d'antécédent

Dans cette section, nous donnons un algorithme qui satisfait le théorème 7.2.1. Il repose sur la propriété 7.2.10.

Notation 7.2.6. Soit a un élément de G. Nous notons $\mathfrak{a}$ l'élément simple défini par

$$\mathfrak{a} = \Delta\tilde{\wedge}\Delta^{-\inf(a)}a$$

et $\mathfrak{A}$ l'élément défini par $\Delta^{-\inf(a)}a\mathfrak{a}^{-1}$. On remarque que $a = \Delta^{\inf(a)}\mathfrak{A}\mathfrak{a}$ et que $\mathfrak{a}$ est le dernier élément simple de la forme Δ-normale à droite de $\Delta^{-\inf(a)}a$.

En conséquence directe de cette notation, nous avons le corollaire suivant :

Corollaire 7.2.7. *Soit a un élément de G de longueur canonique non nulle et $s \in S\backslash\{1,\Delta\}$ un élément simple de M ; nous avons :*

$$s \text{ vérifie } \mathcal{C}1 \quad \Leftrightarrow \quad \mathfrak{a} \succ s$$

En particulier, $\mathfrak{a}$ vérifie $\mathcal{C}1$.

Lemme 7.2.8. *Soit a un élément de G de longueur canonique non nulle et $s_1,\ s_2 \in S\backslash\{1,\Delta\}$ deux éléments simples de M ; nous avons :*

$$s_2 \text{ vérifie } \mathcal{C}3 \text{ et } s_1 \succ s_2 \quad \Rightarrow \quad s_1 \text{ vérifie } \mathcal{C}3$$

En particulier, s'il existe un élément simple vérifiant $\mathcal{C}1$ et $\mathcal{C}3$, alors $\mathfrak{a}$ vérifie $\mathcal{C}3$.

Preuve : Nous avons les deux inclusions suivantes :

$$s_1 \succ s_2 \quad \Rightarrow_{\text{Corollaire 7.1.10}} \quad \mathcal{S}\left(\Delta^{-\inf(a)}as_1^{-1}\right) \subset \mathcal{S}\left(\Delta^{-\inf(a)}as_2^{-1}\right)$$

$$s_1 \succ s_2 \quad \Rightarrow_{\text{Lemme 7.1.11}} \quad \mathcal{R}\left(\tau^{\inf(a)}(s_1)\right) \subset \mathcal{R}\left(\tau^{\inf(a)}(s_2)\right)$$

Comme s_2 vérifie $\mathcal{C}3$, alors s_1 aussi.

□

Lemme 7.2.9. *Soit a un élément de G de longueur canonique non nulle et $s_1,\ s_2 \in S\backslash\{1,\Delta\}$ deux éléments simples de M ; nous avons :*

$$s_2 \text{ vérifie } \mathcal{C}5 \text{ et } s_1 \succ s_2 \quad \Rightarrow \quad s_1 \text{ vérifie } \mathcal{C}5$$

En particulier, s'il existe un élément simple vérifiant $\mathcal{C}5$, alors il existe un atome, $g \in \mathcal{A}(M)$, tel que $\partial(g)$ vérifie $\mathcal{C}5$.

Preuve : Comme $s_1 \succ s_2$, le Corollaire 7.1.10 et le Lemme 7.1.11 nous donnent les deux inclusions suivantes :

$$\mathcal{S}\left(\Delta^{-\inf(a)+1}as_1^{-1}\right) \subset \mathcal{S}\left(\Delta^{-\inf(a)+1}as_2^{-1}\right)$$

$$\mathcal{R}\left(\tau^{\inf(a)-1}(s_1)\right) \subset \mathcal{R}\left(\tau^{\inf(a)-1}(s_2)\right)$$

Comme s_2 vérifie $\mathcal{C}5$, alors s_1 vérifie aussi $\mathcal{C}5$.
En particulier, comme $s_2 \neq \Delta$, il existe un atome g tel que $g \in \mathcal{S}(\Delta s_2^{-1})$ et $\partial(g) \succ s_2$. D'après ce qui précède, $\partial(g)$ vérifie $\mathcal{C}5$.

□

Proposition 7.2.10. *Soit a un élément de G de longueur canonique non nulle. L'élément a possède un antécédent par la fonction cyclage si et seulement si $s = \mathfrak{a}$ vérifie $\mathcal{C}3$ ou s'il existe un atome $g \in \mathcal{A}(M)$ tel que $s = \partial(g)$ vérifie $\mathcal{C}5$. Alors sas^{-1} est un antécédent de a.*

Preuve : $\boxed{\Leftarrow}$ L'élément $\mathfrak{a}$ vérifie $\mathcal{C}1$ d'après le Corollaire 7.2.7. Si $\mathfrak{a}$ vérifie $\mathcal{C}3$ alors $\mathfrak{a}a\mathfrak{a}^{-1}$ est un antécédent de *Type 1* ou *2* d'après la Proposition 7.2.5. De plus, s'il existe un atome $g \in \mathcal{A}(M)$ tel que $s = \partial(g) \in S\backslash\{1, \Delta\}$ vérifie $\mathcal{C}5$, alors sas^{-1} est un antécédent de *Type 3* ou *4*, toujours d'après la Proposition 7.2.5 ; ceci termine la preuve de la condition suffisante.

$\boxed{\Rightarrow}$ Le Théorème 7.1.4 implique que tout antécédent d'un élément a est forcément de *Type 1*, *2*, *3* ou *4*. Alors, la Proposition 7.2.5 implique que a admet un antécédent si et seulement s'il existe un élément simple vérifiant soit $\mathcal{C}1$ et $\mathcal{C}3$, soit $\mathcal{C}5$. Les Lemmes 7.2.8 et 7.2.9 donnent le résultat voulu.

□

De la Remarque 7.1.3 et de la Proposition 7.2.10, nous pouvons déduire directement l'algorithme 7.2.11, InvCyclage, qui inverse la fonction *cyclage*. Soit a un élément de G. Si a possède un antécédent par la fonction *cyclage* alors InvCyclage appliqué à a en retourne un ; sinon, il retourne "VIDE". Afin d'établir le théorème central de ce chapitre, il nous reste seulement à évaluer la complexité de l'algorithme.

Algorithme 7.2.11. InvCyclage (a)

Entrée : Soit $a \in G$.

Si $l_c(a) = 0$ alors Retourner a FinSi
$\mathfrak{a} := \Delta\tilde{\wedge}\Delta^{-\inf(a)}a$
Si $\mathfrak{a}$ vérifie $\mathcal{C}3$ alors Retourner $\mathfrak{a}a\mathfrak{a}^{-1}$ FinSi
Pour $g \in \mathcal{A}(M)$ faire
    Si $\partial(g)$ vérifie $\mathcal{C}5$ alors Retourner $\partial(g)a\partial(g)^{-1}$ FinSi
Retourner VIDE

Complexité de InvCyclage : Le coût de la construction des ensembles *Starting set* et *complémentaire à droite* peut être majorée simplement par

$$\#\mathcal{A}(M) * \mathcal{O}(\text{forme } \Delta\text{-normale})$$

En effet, il suffit de tester pour chaque atome une divisibilité dans le monoïde ce qui revient à regarder si un élément est dans le monoïde.

Toutes les autres opérations qui interviennent sont de complexité au plus équivalente soit au nombre d'atomes soit à la complexité de la forme Δ-normale ; ainsi, l'algorithme InvCyclage possède une complexité majorée par

$$\mathcal{O}\left(\#\mathcal{A}(M)^2 * \mathcal{O}(\text{forme } \Delta\text{-normale})\right)$$

Cette étude et les sections qui suivent laissent un point sans réponse : la fonction *cyclage* est-elle surjective ?

Dans les groupes de tresses, l'algorithme InvCyclage est très efficace et retourne un résultat quasi immédiat. A ce jour, nous n'avons trouvé aucun contre-exemple de

la surjectivité sur les centaines de milliers de tresses générées aléatoirement dans B_n^+ et BKL_n^+, pour $n \in \{20, 100\}$. Même si la méthode de génération aléatoire des tresses peut être mise en cause, nous considérons que le problème des cyclages dans les groupes de tresses est résolu en pratique.

Remarque 7.2.12. Rappelons que l'Ultra Summit Set est construit à partir des orbites cycliques de la fonction *cyclage* dans le Super Summit Set. Ainsi, dans les cas où le *k-cyclé* d'un problème des cyclages est sur l'une de ces orbites, la résolution peut simplement venir du parcours de l'orbite elle-même. Si sa longueur est supérieure à k, le problème est résolu.

7.3 Inversion locale

Les sections précédentes ont répondu à la problématique initiale qui est d'inverser la fonction *cyclage* et en particulier d'attaquer le problème des cyclages. Comme nous l'avons remarqué, les antécédents d'un élément par la fonction *cyclage* sont de quatre types différents. Nous proposons dans cette section des algorithmes qui inversent la fonction *cyclage*, dans les groupes de Garside, en tenant compte du type des antécédents : nous donnons quatre algorithmes, un pour chaque type ; ils retournent un antécédent de type fixé, s'il en existe.

L'intérêt de cette approche est multiple :

- Satisfaire la curiosité quant à la fonction *cyclage* en étudiant plus en détail ses antécédents.
- Dans l'attente d'une preuve de la surjectivité de la fonction *cyclage*, la résolution du problème des cyclages requiert de pouvoir déterminer tous les antécédents d'un élément. En effet, une instance du problème des cyclages nécessite de pouvoir calculer un antécédent k-ième d'un élément qui en possède un. Or, si la fonction *cyclage* n'est pas surjective, rien ne prouve qu'un antécédent intermédiaire retourné par une itération de InvCyclage possède à nouveau un antécédent. Ainsi, sans l'hypothèse de surjectivité, nous ne pouvons affirmer que la recherche d'un antécédent k-ième ne sera pas bloquée, même si l'élément initial est effectivement un k-cyclé.
- Dans l'hypothétique possibilité de dériver le problème des cyclages afin de construire un nouveau problème difficile, cette étude complémentaire permet d'apprécier davantage cette fonction et ses inversions locales.

Remarque 7.3.1. Tous les algorithmes proposés dans cette section reposent sur des opérations élémentaires des groupes de Garside, dont la complexité est au plus celle de la forme Δ-normale. Ainsi l'étude de la complexité de ces algorithmes se basera essentiellement sur la structure de l'algorithme. Dans les groupes de tresses, ces algorithmes retournent leur résultat de façon quasi immédiate.

7.3.1 Eléments de longueur canonique nulle

Avant de traiter successivement les quatre familles d'antécédents, traitons le cas des éléments de longueur canonique nulle avec la proposition suivante.

Lemme 7.3.2. *Soit M un monoïde de Garside ; la restriction de la fonction τ à $\mathcal{A}(M)$ est une permutation de $\mathcal{A}(M)$.*

Preuve : Comme τ est clairement injective, il ne reste plus qu'à montrer que l'image d'un atome par τ est encore un atome.
Soit $g \in \mathcal{A}(M)$. La Proposition 1.2.12 nous donne que τ est une bijection de S dans S ; ainsi $\tau(g) \in S \subset M$. Supposons que $\tau(g)$ n'est pas un atome ; alors il existe $s_1, s_2 \in S\backslash\{1\}$ deux éléments simples différents de l'élément neutre vérifiant $\tau(g) = s_1 s_2$. Nous avons :

$$\tau(g) = s_1 s_2 \quad \Leftrightarrow \quad g\Delta = \Delta s_1 s_2 = \tau^{-1}(s_1)\tau^{-1}(s_2)\Delta \quad \Leftrightarrow \quad g = \tau^{-1}(s_1)\tau^{-1}(s_2)$$

Or

$$s_1, s_2 \in S\backslash\{1\} \quad \Leftrightarrow_{\text{Proposition 1.2.12}} \quad \tau^{-1}(s_1), \tau^{-1}(s_2) \in S\backslash\{1\}$$

Donc g n'est pas un atome, ce qui est contradictoire. Ce raisonnement par l'absurde donne que $\tau(g) \in \mathcal{A}(M)$.

□

Proposition 7.3.3. ***(longueur canonique nulle)*** *Soit a un élément de G de longueur canonique nulle. L'élément a est lui-même son unique antécédent de Type 1 et ne possède pas d'antécédent de Type 2 ou 3.*

L'élément a possède un antécédent de Type 4 si et seulement si l'infimum de a n'est pas un multiple de l'exposant. De plus, si $\inf(a) \not\equiv 0 \mod \mathcal{E}$, alors il existe un atome $g \in \mathcal{A}(M)$ tel que $g^{-1}ag$ soit un antécédent de Type 4 de a.

Preuve : La Remarque 7.1.3 donne qu'un élément de longueur canonique nulle est son propre cyclé et donc aussi son unique antécédent de *Type 1*. On remarque que le cyclé d'un élément de longueur canonique 1 est aussi de longueur canonique 1 ; donc a ne possède pas d'antécédent de *Type 2* et *3*. Reste le cas des antécédents de *Type 4*. Procédons en deux étapes :

• Montrons que l'ensemble des antécédents de *Type 4* de a est :

$$\mathfrak{E} = \left\{ \Delta^{\inf(a)-1}\tau^{\inf(a)-1}\left(\partial(s)\right) s;\ \mathcal{S}(s) \cap \mathcal{S}\left(\tau^{\inf(a)}(s)\right) = \emptyset \text{ et } s \in S\backslash\{1, \Delta\} \right\}$$

⊃ Soit $b = \Delta^{\inf(a)-1}\tau^{\inf(a)-1}\left(\partial(s)\right) s \in \mathfrak{E}$. Nous avons :
$l_c(b) = 2$ car $\mathcal{R}\left(\tau^{\inf(a)-1}\left(\partial(s)\right)\right) = \mathcal{S}\left(\tau^{\inf(a)-1}\left(\partial^2(s)\right)\right) = \mathcal{S}\left(\tau^{\inf(a)}(s)\right)$.
$\inf(b) = \inf(a) - 1$ car $\inf\left(\tau^{\inf(a)-1}\left(\partial(s)\right) s\right) = 0$.
$c(b) = \Delta^{\inf(a)-1} s\partial(s) = \Delta^{\inf(a)} = a$
Donc les éléments de $\mathfrak{E}$ sont des antécédents de *Type 4* de a.
⊂ Soit $\Delta^{r-1}s_1s_2$ la forme Δ-normale de b, un antécédent de *Type 4* de a. Montrons que $b \in \mathfrak{E}$. Nous avons :

$$c(b) = \Delta^{r-1}s_2\tau^{1-r}(s_1) = \Delta^r \quad \Leftrightarrow \quad s_2\tau^{1-r}(s_1) = \Delta \quad \Leftrightarrow \quad s_1 = \tau^{r-1}\left(\partial(s_2)\right)$$

De plus, nous savons que (s_1, s_2) est une suite normale à gauche donc

$$\mathcal{S}(s_2) \cap \mathcal{R}(s_1) = \emptyset$$

Comme $\partial(s_1) = \tau^{r-1}\left(\partial^2(s_2)\right) = \tau^r(s_2)$, nous obtenons

$$\mathcal{R}(s_1) = \mathcal{S}\left(\partial(s_1)\right) \quad \Rightarrow_{\text{Proposition 3.1.3}} \quad \mathcal{S}(s_2) \cap \mathcal{S}\left(\tau^r(s_2)\right) = \emptyset \quad \Rightarrow \quad b \in \mathfrak{E}$$

Donc les antécédents de *Type 4* de a sont contenus dans $\mathfrak{E}$. Ceci qui termine la preuve du premier point.

• Montrons $\mathfrak{E} \neq \emptyset \quad \Leftrightarrow \quad \inf(a) \not\equiv 0 \mod \mathcal{E}$

$\boxed{\Rightarrow}$ Montrons la contraposée :

$$\inf(a) \equiv 0 \mod \mathcal{E} \quad \Rightarrow \quad \forall s \in S,\ \tau^{\inf(a)}(s) = s \quad \Rightarrow \quad \mathfrak{E} = \emptyset$$

$\boxed{\Leftarrow}$ Nous avons :

$$\begin{array}{lll} \inf(a) \not\equiv 0 \mod \mathcal{E} & \Rightarrow & \Delta^{\inf(a)} \notin Z(G) \\ & \Rightarrow & \exists g \in \mathcal{A}(M);\ g\Delta^{\inf(a)} \neq \Delta^{\inf(a)} g \\ & \Rightarrow & \exists g \in \mathcal{A}(M);\ \tau^{\inf(a)}(g) \neq g \\ & \Rightarrow_{\text{Lemme 7.3.2}} & \exists g \in \mathcal{A}(M);\ \mathcal{S}(g) \cap \mathcal{S}\left(\tau^{\inf(a)}(g)\right) = \emptyset \\ & \Rightarrow & \mathfrak{E} \neq \emptyset \end{array}$$

□

Remarque 7.3.4. Si l'exposant du groupe vaut 1, l'infimum d'un élément est toujours un multiple de l'exposant, et donc les éléments de longueur canonique nulle ne possèdent pas d'antécédent de *Type 4*.

7.3.2 *Type 1*

Dans cette section nous déterminons un algorithme construisant un antécédent de *Type 1* s'il en existe. Le lemme suivant résulte directement de la Notation 7.2.6 et de la Définition 7.2.2 :

Lemme 7.3.5. *Soit a un élément de G de longueur canonique non nulle. L'élément $\mathfrak{a}$ vérifie $\mathcal{C}2$.*

Proposition 7.3.6. ***(CNS d'existence d'antécédents de Type 1)*** *Soit a un élément de G de longueur canonique non nulle. L'élément a admet un antécédent de Type 1 si et seulement si $\mathfrak{a}$ vérifie $\mathcal{C}3$ et alors $\mathfrak{a}a\mathfrak{a}^{-1}$ est un antécédent de Type 1 de a.*

Preuve : $\boxed{\Rightarrow}$ Supposons que a admette un antécédent de *Type 1*. La Proposition 7.2.5 donne qu'il existe un élément simple $s \in S\backslash\{1, \Delta\}$ vérifiant $\mathcal{C}1$, $\mathcal{C}2$ et $\mathcal{C}3$. Ainsi, le Corollaire 7.2.7 et les Lemmes 7.3.5 et 7.2.8 donnent que $\mathfrak{a}$ vérifie aussi $\mathcal{C}1$, $\mathcal{C}2$ et $\mathcal{C}3$. La Proposition 7.2.5 donne alors que $\mathfrak{a}a\mathfrak{a}^{-1}$ est un antécédent de *Type 1* de a.

$\boxed{\Leftarrow}$ Le Corollaire 7.2.7 et le Lemme 7.3.5 donnent que $\mathfrak{a}$ vérifie aussi $\mathcal{C}1$ et $\mathcal{C}2$. Ainsi la Proposition 7.2.5 donne que a possède un antécédent de *Type 1*.

□

Soit a un élément du groupe G. Si a possède un antécédent de *Type 1*, alors l'algorithme suivant appliqué à a en retourne un ; sinon il retourne "VIDE". L'algorithme est l'application directe de la Proposition 7.3.3 et de la Proposition 7.3.6.

Algorithme 7.3.7. TYPE1 (a)

Entrée :	Soit $a \in G$.
Init :	$\mathfrak{a} := \Delta\tilde{\wedge}\Delta^{-\inf(a)}a$
	Si $l_c(a) = 0$ alors Retourner a FinSi
	Si $\mathfrak{a}$ vérifie $\mathcal{C}3$ alors Retourner $\mathfrak{a}a\mathfrak{a}^{-1}$
	Sinon Retourner VIDE
	FinSi

Remarque 7.3.8. Dans le cas particulier d'existence d'un antédédent de *Type 1*, la fonction *cyclage à droite* est un inverse de la fonction *cyclage*. Le lecteur notera la distinction entre la fonction *cyclage à droite*, qui est l'analogue de la fonction *cyclage* sur la forme Δ-normale à droite, et les fonctions *décyclage* [37] et *cyclage inverse* [35] qui correspondent à la fonction *decycling* de [6] dont la définition est donnée à la Définition 1.2.35.

7.3.3 *Type 2*

Dans cette section nous déterminons un algorithme construisant un antécédent de *Type 2* s'il en existe. Nous donnons aussi deux conditions nécessaires d'existence d'antécédents de *Type 2* d'un élément. La première conditionne la longueur canonique d'un antécédent :

Lemme 7.3.9. *Soit a un élément de G de longueur canonique non nulle et $s \in S\backslash\{1, \Delta\}$ un élément simple de M. Si s vérifie $\mathcal{C}1$, nous avons :*

$$s \text{ vérifie } \mathcal{C}2 \quad \Leftrightarrow \quad s \succ \left(\mathfrak{A}^{-1}\Delta^{l_c(a)-1} \wedge \mathfrak{a}\right)^{-1} \mathfrak{a}$$

Preuve : Notons $A_1 \cdots A_p$ la forme Δ-normale à droite de $\Delta^{-\inf(a)}a$; ainsi, $\mathfrak{a} = A_p$ et $\mathfrak{A} = A_1 \cdots A_{p-1}$. On suppose ici que $l_c(a) = p$ et $A_p s^{-1} \in M$; ainsi nous avons :

$$\begin{aligned} s \text{ vérifie } \mathcal{C}2 &\Leftrightarrow l_c(A_1 \cdots A_p s^{-1}) = p - 1 \\ &\Leftrightarrow sA_p^{-1} \cdots A_1^{-1}\Delta^{p-1} \in M \\ &\Leftrightarrow A_p s^{-1} \prec A_{p-1}^{-1} \cdots A_1^{-1}\Delta^{p-1} \\ &\Leftrightarrow A_p s^{-1} \prec A_{p-1}^{-1} \cdots A_1^{-1}\Delta^{p-1} \wedge A_p \\ &\Leftrightarrow s \succ \left(A_{p-1}^{-1} \cdots A_1^{-1}\Delta^{p-1} \wedge A_p\right)^{-1} A_p \end{aligned}$$

Ce qui est le résultat attendu.

□

La proposition suivante montre qu'il ne peut exister d'antécédent de *Type 2* s'il n'existe pas d'antécédent de *Type 1*. Ceci nous donne une nouvelle condition nécessaire :

Proposition 7.3.10. *Soit a un élément de G. Le décyclé d'un antécédent de Type 2 de a est un antécédent de Type 1 de a.*

Preuve : Soit b un antécédent de *Type 2* de a. Posons $A_1 \cdots A_p$ la forme Δ-normale à droite de $\Delta^{-r}a$ et $\Delta^r b_1 \cdots b_{p+1}$ la forme Δ-normale de b.
Nous avons $A_1 \cdots A_p \succ \tau^{-r}(b_1)$ donc $A_p \succ \tau^{-r}(b_1)$. Posons

$$t = A_p \tau^{-r}(b_1)^{-1} \in S$$

Comme $c(b) = a$, on obtient $b_2 \cdots b_p b_{p+1} \tau^{-r}(b_1) = A_1 \cdots A_p$ et

$$b_2 \cdots b_p b_{p+1} = A_1 \cdots A_{p-1} t$$

Le Corollaire 3.2.8 appliqué à la dernière égalité nous donne $t \succ b_{p+1}$, d'où $A_p \succ b_{p+1}\tau^{-r}(b_1)$. On vérifie aisément que l'élément simple $b_{p+1}\tau^{-r}(b_1)$ vérifie les critères $\mathcal{C}1$, $\mathcal{C}2$ et $\mathcal{C}3$: $b_{p+1} b b_{p+1}^{-1} = d(b)$ est un antécédent de *Type 1* de a.

□

Donnons maintenant un algorithme qui retourne un antécédent de *Type 2* :

Algorithme 7.3.11. RoutineType2 $(t, d, m, \mathfrak{a}, a)$

Entrée : Soit $t, d, m, \mathfrak{a} \in M$ avec $t \prec d \prec m \prec \mathfrak{a}$ et $a \in G$.

Si $t = d$ alors Retourner VIDE FinSi
Pour $g \in \mathcal{S}(t^{-1}m) \backslash \mathcal{S}(t^{-1}d)$ faire
  Si $g^{-1}t^{-1}\mathfrak{a}$ vérifie $\mathcal{C}3$ alors Retourner $g^{-1}t^{-1}\mathfrak{a}$ FinSi
FinPour
Pour $g \in \mathcal{S}(t^{-1}d)$ faire
  $r :=$ RoutineType2 $(t\, g, d, m, \mathfrak{a}, a)$
  Si $r \neq$ VIDE alors Retourner r FinSi
FinPour
Retourner VIDE

Algorithme 7.3.12. Type2 (a)

Entrée : Soit $a \in G$.

Init : $\mathfrak{a} := \Delta\tilde{\wedge}\Delta^{-\inf(a)}a$ et $\mathfrak{A} := \Delta^{-\inf(a)}a\mathfrak{a}^{-1}$

Si $l_c(a) = 0$ alors Retouner VIDE FinSi
Si $\mathfrak{a}$ vérifie $\mathcal{C}3$ alors
  $d := \mathfrak{A}^{-1}\Delta^{l_c(a)-1} \wedge \mathfrak{a}$
  $m := \mathfrak{a}$
  $r :=$ RoutineType2 $(1, d, m, \mathfrak{a}, a)$
  Si $r =$ VIDE alors Retourner VIDE
  Sinon Retourner rar^{-1}
  FinSi
Sinon Retourner VIDE
FinSi

Proposition 7.3.13. *Soit a un élément de G. Si a admet un antécédent de Type 2, alors l'algorithme 7.3.12,* Type2*, appliqué à a en retourne un ; sinon il retourne en temps fini "VIDE".*

Preuve : Commençons par décrire le contenu de TYPE2, ensuite nous nous intéresserons à ROUTINETYPE2.

• Le test concerne la condition nécessaire d'existence d'antécédents de *Type 2* introduite par la Proposition 7.3.10 : l'existence d'antécédents de *Type 1*. Le test résulte donc de la Propositon 7.3.6.

• Considérons l'algorithme ROUTINETYPE2. C'est un algorithme récursif dont la profondeur d'arbre est $l(d)$. En pratique, le nombre maximal d'appels est le cardinal de l'ensemble des mots représentant un élément du monoïde, diviseur à gauche de d ; donc le nombre d'appels est fini.

Il ne reste plus qu'à montrer deux points. (1) : si a admet un antécédent de *Type 2*, alors ROUTINETYPE2 $(e, d, m, \mathfrak{a}, a)$ ne retourne pas VIDE. (2) : si ROUTINETYPE2 $(e, d, m, \mathfrak{a}, a)$ retourne un élément, cet élément est un élément simple conjugant d'un antécédent de *Type 2* de a.

(1) Considérant l'arbre des appels de ROUTINETYPE2, pour répondre au point (1), il suffit de montrer que si a admet un antécédent de *Type 2*, alors il existe un antécédent de *Type 2* de a qui réponde aux critères de sélection de ROUTINETYPE2.
Soit b un antécédent de *Type 2* de a et $\Delta^r b_1 \cdots b_{p+1}$ sa forme Δ-normale. Soit $g \in \mathcal{S}(b_{p+1})$. Comme nous l'avons vu dans la preuve de la Proposition 7.3.10 : $\mathfrak{a} \succ b_{p+1}\tau^{-r}(b_1)$. Posons $s = g^{-1}b_{p+1}\tau^{-r}(b_1)$; il reste à vérifier que sas^{-1} vérifie les critères de sélection de ROUTINETYPE2. Nous avons $a = \Delta^r b_2 \cdots b_p g s$ avec $(b_2, \ldots, b_p, g)$ une suite normale à gauche par construction. Comme $s \succ \tau^{-r}(b_1)$, le Lemme 7.2.8 donne que s vérifie $\mathcal{C}3$. Donc $t = \mathfrak{a}s^{-1}g^{-1}$ convient et on a bien $t \prec d$ car $\mathfrak{A}t = b_2 \ldots b_p$.

(2) Soit u l'élément de sortie de ROUTINETYPE2 ; alors il existe t et g vérifiant $tgu = \mathfrak{a}$, $t \prec d$, $t \neq d$ et $g \notin \mathcal{S}(t^{-1}d)$. Donc u vérifie directement $\mathcal{C}1$ et $\mathcal{C}3$. Reste à voir la vérification de $\overline{\mathcal{C}2}$: $g \notin \mathcal{S}(t^{-1}d) \Rightarrow l_c(\Delta^{-\inf(a)} a u^{-1}) = l_c(a)$.

□

Remarque 7.3.14. L'algorithme TYPE2 est minimaliste dans le sens où il retourne un antécédent de *Type 2* tel que le dernier élément simple de sa forme Δ-normale soit un atome.

Remarque 7.3.15. L'algorithme ROUTINETYPE2 peut être amélioré en introduisant une heuristique permettant de ne tester le paramètre t que sur des mots non équivalents. En effet, l'ensemble des valeurs possibles de ce paramètre est en fait tous les sous-mots du paramètre d. On a déjà rencontré ce genre de problématique dans le Chapitre 6 où l'on utilise l'ordre lexicographique pour optimiser la génération de tels ensembles.

7.3.4 *Type 3*

Dans cette section nous déterminons un algorithme construisant un antécédent de *Type 3* s'il en existe. Il est possible de déterminer une condition nécessaire et suffisante d'existence d'un antécédent de *Type 3*. Cela est donné par la proposition suivante :

Proposition 7.3.16. ***(CNS d'existence d'antécédents de Type 3)*** *Soit a un élément de G de longueur canonique non nulle. L'élément a admet un antécédent de Type 3 si et seulement s'il existe un atome g tel que $\partial(g)$ vérifie $\mathcal{C}4$ et $\mathcal{C}5$.*

Preuve : Le sens $\boxed{\Leftarrow}$ découle directement de la Proposition 7.2.5.

$\boxed{\Rightarrow}$ Supposons que b est un antécédent de *Type 3* de a. Notons $\Delta^{r-1}b_1\cdots b_{p+1}$ la forme Δ-normale de b, alors nous avons :

$$c(b) = \Delta^{r-1}b_2\cdots b_{p+1}\tau^{1-r}(b_1) = \Delta^{r-1}\tau^{-1}(\Delta^{-r}a)\Delta = a$$

Comme $b_1 \neq \Delta$, nous pouvons choisir $g \in \mathcal{S}\left(\Delta\tau^{1-r}(b_1)^{-1}\right)$ et poser

$$s = \partial(g) = g^{-1}\Delta.$$

Comme $s \succ \tau^{1-r}(b_1)$ et $\tau^{1-r}(b_1)$ vérifie $\mathcal{C}5$, le Lemme 7.2.9 donne que s vérifie $\mathcal{C}5$. Montrons que s vérifie $\mathcal{C}4$.
Or $\left(\tau^{-1}(\Delta^{-r}a)g\right) = \left(\tau^{-1}(\Delta^{-r}a)\Delta\tau^{1-r}(b_1)^{-1}\right) = b_2\ldots b_{p+1}$, alors

$$l_c(a) \leq \sup\left(\tau^{-1}(\Delta^{-r}a)g\right) \leq \sup(b_2\ldots b_{p+1}) \leq l_c(a)$$

Comme $0 \leq \inf\left(\tau^{-1}(\Delta^{-r}a)g\right) \leq \inf(b_2\ldots b_{p+1})0$, il vient que s vérifie $\mathcal{C}4$.
On remarque que $g \in \mathcal{S}\left(\tau^{-1}(\Delta^{-r}a)^{-1}\Delta^{l_c(a)}\right)$, puisque $l_c\left(\tau^{-1}(\Delta^{-r}a)g\right) = l_c(a)$.

□

Corollaire 7.3.17. *Soit a un élément de G de longueur canonique non nulle. L'élément a admet un antécédent de Type 3 si et seulement s'il existe un atome $g \in \mathcal{S}\left(\tau^{-1}(\Delta^{-\inf(a)}a)^{-1}\Delta^{l_c(a)}\right)$ tel que $\partial(g)$ vérifie $\mathcal{C}5$*

Soit a un élément du groupe G. Si a possède un antécédent de *Type 3*, alors l'algorithme suivant appliqué à a en retourne un ; sinon il retourne "VIDE". L'algorithme est l'application directe de la Proposition 7.3.3 et du Corollaire 7.3.17.

Algorithme 7.3.18. TYPE3 (a)

Entrée :	Soit $a \in G$.
	Si $l_c(a) = 0$ alors Retourner VIDE FinSi
	Pour $g \in \mathcal{S}\left(\tau^{-1}(\Delta^{-\inf(a)}a)^{-1}\Delta^{l_c(a)}\right)$ faire
	Si $\partial(g)$ vérifie $\mathcal{C}5$ alors Retourner $\partial(g)a\partial(g)^{-1}$ FinSi
	FinPour
	Retourner VIDE

7.3.5 *Type 4*

Dans cette section nous déterminons un algorithme qui construit un antécédent de *Type 4* s'il en existe. Le lien entre les antécédents de *Types 3* et *4* semble faussement similaire à celui entre les antécédents de *Types 1* et *2*. Comme nous avons pu le remarquer dans l'étude des éléments de longueur canonique nulle, nous avons moins de contraintes sur le choix d'un conjugant potentiel. En particulier, il n'y a pas de parallèle au critère $\mathcal{C}1$.

Commençons par donner une condition nécessaire d'existence d'antécédent de *Type 4* d'un élément ; elle concerne la longueur canonique d'un antécédent :

Lemme 7.3.19. *Soit a un élément de G de longueur canonique non nulle et $s \in S\backslash\{1, \Delta\}$ un élément simple de M. Supposant que s vérifie $\mathcal{C}5$, nous avons :*

$$s \text{ vérifie } \mathcal{C}4 \quad \Leftrightarrow \quad s \succ \left(\tau^{-1}(\Delta^{-\inf(a)}a)^{-1}\Delta^{l_c(a)} \wedge \Delta\right)^{-1}\Delta$$

Preuve : L'élément s vérifie $\mathcal{C}5$; d'après le Corollaire 7.2.3 nous avons $\inf(\Delta^{-\inf(a)}as^{-1}) = 0$ et :

$$\begin{aligned}
s \text{ vérifie } \mathcal{C}4 &\Leftrightarrow l_c\left(\Delta^{-\inf(a)+1}as^{-1}\right) = l_c(a)\\
&\Leftrightarrow sa^{-1}\Delta^{\inf(a)-1}\Delta^{l_c(a)} \in M\\
&\Leftrightarrow s\Delta^{-1}\tau^{-1}\left(a^{-1}\Delta^{\inf(a)}\right)\Delta^{l_c(a)} \in M\\
&\Leftrightarrow \Delta s^{-1} \prec \tau^{-1}\left(\Delta^{-\inf(a)}a\right)^{-1}\Delta^{l_c(a)}\\
&\Leftrightarrow \Delta s^{-1} \prec \tau^{-1}\left(\Delta^{-\inf(a)}a\right)^{-1}\Delta^{l_c(a)} \wedge \Delta\\
&\Leftrightarrow s \succ \left(\tau^{-1}\left(\Delta^{-\inf(a)}a\right)^{-1}\Delta^{l_c(a)} \wedge \Delta\right)^{-1}\Delta
\end{aligned}$$

Ce qui est le résultat attendu.

□

Certains antécédents de *Type 4* peuvent être réduits en antécédents de *Type 3* par la fonction *décyclage* ; mais tous ne le sont pas. La propostion suivante clarifie la situation :

Proposition 7.3.20. *Soit a un élément de B_n de longueur canonique non nulle. Soit $\Delta^{r-1}b_1 \cdots b_{p+2}$ la forme Δ-normale de b, un antécédent de Type 4 de a tel que*

$$b_{p+2}\tau^{1-r}(b_1) \neq \Delta.$$

Alors le décyclé de b est un antécédent de Type 3 de a.

Preuve : Soit b un antécédent de *Type 4* de a. Posons $\Delta^{r-1}b_1 \cdots b_{p+2}$ la forme Δ-normale de b. Posons $t = \Delta\tau^{1-r}(b_1)^{-1} \in S$,

$$\begin{aligned}
c(b) = a \quad &\Leftrightarrow \quad b_2 \cdots b_{p+2}\tau^{-r}(b_1) = \tau^{-1}(\Delta^{-\inf(a)}a)\Delta\\
&\Leftrightarrow \quad b_2 \cdots b_{p+2} = \tau^{-1}(\Delta^{-\inf(a)}a)t
\end{aligned}$$

Le Corollaire 3.2.8 appliqué à la dernière égalité nous donne $t \succ b_{p+2}$; donc :

$$\Delta \succ b_{p+2}\tau^{1-r}(b_1)$$

Or, par hypothèse, nous avons $b_{p+2}\tau^{1-r}(b_1) \neq \Delta$. On vérifie aisément que l'élément simple non trivial $b_{p+2}\tau^{1-r}(b_1)$ vérifie les critères $\mathcal{C}4$ et $\mathcal{C}5$; donc $b_{p+2}bb_{p+2}^{-1}$ est un antécédent de *Type 3* de a.

□

L'exemple suivant illustre le fait que l'étude les antécédents de *Type 1, 3* et *4* sont incontournables si l'on veut inverser la fonction *cyclage*.

Exemple 7.3.21. On se place dans B_5^+, les éléments sont données sous leur forme Δ-normale :

	possède un antécédent de		
	Type 1	*Type 3*	*Type 4*
$a1 : [0, [(2,4,5),(1,5,2,3,4),(1,4,2,5,3)]]$	$vrai$	$faux$	$faux$
$a3 : [2, [(1,3,4,2,5),(1,3,5,2),(1,2,3)(4,5)]]$	$faux$	$vrai$	$faux$
$a4 : [2, [(1,4,3)(2,5),(2,5,4),(1,4)(2,5),(3,4,5)]]$	$faux$	$faux$	$vrai$

Donnons maintenant un algorithme qui retourne un antécédent de *Type 4* :

Algorithme 7.3.22. RoutineType4 (t, d, m, a)

Entrée : Soit $t, d, m \in M$; $t \prec d \prec m$ et $a \in G$.

Si $t = d$ alors Retourner VIDE FinSi
  Pour $g \in \mathcal{S}(t^{-1}m) \backslash \mathcal{S}(t^{-1}d)$ faire
  Si $g^{-1}t^{-1}\Delta$ vérifie $\mathcal{C}5$ alors Retourner $g^{-1}t^{-1}\Delta$ FinSi
  FinPour
  Pour $g \in \mathcal{S}(t^{-1}m) \cap \mathcal{S}(t^{-1}d)$ faire
    $r :=$ RoutineType4 $(t\ g, d, m, a)$
    Si $r \neq$ VIDE alors Retourner r FinSi
  FinPour
  Retourner VIDE

Algorithme 7.3.23. Type4 (a)

Entrée : Soit $a \in B_n$.

Si $l_c(a) = 0$ alors
  Si $\inf(a) \equiv 0 \mod \mathcal{E}$ Retourner VIDE
  Sinon Pour $g \in \mathcal{A}(M)$ faire
    Si $g \neq \tau^{\inf(a)}(g)$ alors Retourner $g^{-1}ag$ FinSi
    FinPour
  FinSi
FinSi

$d := \tau^{-1}(\Delta^{-\inf(a)}a)^{-1}\Delta^{l_c(a)} \wedge \Delta$
$m := \Delta$
$r :=$ RoutineType2 $(1, d, m, a)$
Si $r =$ VIDE alors Retourner VIDE
Sinon Retourner rar^{-1}
FinSi

Proposition 7.3.24. *Soit a un élément de G. Si a admet un antécédent de Type 4, alors l'algorithme 7.3.23,* Type4*, appliqué à a en retourne un ; sinon il retourne en temps fini "VIDE".*

Preuve : Le cas $l_c(a) = 0$ correspond à la Proposition 7.3.3. Le reste de l'algorithme est similaire à celui de recherche d'antécédent de *Type 2*.

□

Conclusion

Dans cette étude, nous nous sommes intéressés à certaines applications cryptographiques dans les groupes de tresses : le problème de conjugaison et le problème des cyclages. Le travail réalisé repose sur les structures de Garside des groupes de tresses ; ainsi, la plupart des résultats sont formulés dans les groupes de Garside.

D'une part, le problème des cyclages était un problème supposé difficile dans les groupes de tresses, [28]. Nous montrons que l'inversion de la fonction *cyclage* admet une solution polynomiale dans les groupes de Garside, ce qui résout efficacement, en pratique, le problème des cyclages.

D'autre part, nous avons établi un algorithme de réduction d'une instance du problème de conjugaison. Cet algorithme travaille dans les groupes de Garside ; il retourne un diviseur à gauche et un multiple à droite du secret. De plus, il exploite la différence de complexité canonique entre les conjugués et permet d'attaquer le problème de conjugaison et ses variantes.

L'étude de cette attaque a permis de mettre en relief le rôle joué par les générateurs aléatoires de tresses et a ouvert des perspectives encourageantes : non seulement nous avons complété une liste existante de critères pour générer de bonnes instances de conjugaison, mais nous avons également pu construire un nouveau générateur aléatoire capable de résister à l'algorithme de réduction. La recherche de nouveaux générateurs aléatoires dans les groupes de tresses constitue donc une ouverture intéressante pour l'utilisation du problème de conjugaison comme primitive cryptographique. Nous avons montré que les attaques basées sur la complexité ne semblent plus être un obstacle majeur ; il convient donc d'orienter les recherches sur la génération aléatoire de tresses possédant des grands Super Summit Set et Ultra Summit Set.

Les multiples représentations d'un monoïde ou d'un groupe relèvent d'un domaine encore mal connu et qu'il convient d'étudier. Notre démarche montre qu'un attaquant s'autorise à changer de présentation pour travailler afin d'optimiser son attaque. Ainsi, la difficulté d'un problème doit être reliée aux nombreuses possibilités de définir une nouvelle présentation, équivalente ou non, sur laquelle le problème peut être considéré.

Il est cependant à noter que le problème de conjugaison et le problème des cyclages ne sont pas les seuls problèmes difficiles dans les groupes de tresses. Nous pouvons par exemple citer le problème de décomposition qui apporte des outils cryptographiques similaires. Il repose plus sur un problème de réécriture que sur un problème structurel et sa sécurité est encore mal connue, mais il semble prometteur. Tout ceci confirme que les groupes de tresses possèdent une richesse structurelle intéressante pour une utilisation en cryptographie. Ce travail dans les groupes de tresses constitue les prémices d'une nouvelle génération de cryptosystèmes basés sur les groupes présentés. Ainsi toutes les problématiques abordées dans le cas des tresses trouveront vraisemblablement une application intéressante dans un avenir proche.

Chapitre 8

Annexe

Ce chapitre regroupe diverses simulations qui servent de support à certaines remarques présentées aux Chapitres 5 et 6. Nous profitons de cette annexe pour préciser davantage le cadre des simulations et proposer diverses variations des paramètres intervenants : taille des éléments, taille des groupes de tresses, présentation, générateur aléatoire ...

Afin de simplifier les tables, nous écrivons A ou B pour préciser la présentation dans laquelle nous avons travaillé ; la présentation où l'instance a été générée se trouve généralement dans l'intitulé des tables : A pour la présentation d'Artin et B pour la présentation de Birman, Ko et Lee.

8.1 Statistiques du biais et estimations

Dans cette section se trouvent des résultats statistiques plus complets que ceux présentés dans le corps des Chapitres 5 et 6, aux Section 5.2.2 et 6.3. Ils concernent l'évaluation du biais et l'estimation de la longueur canonique du secret d'une instance de conjugaison pour les générateurs aléatoires : GA-Mots, GA-MotsP, GA-Cano, GA-CompC et GA-TronC. Les Tables sont : 8.1, 8.2, 8.3 et 8.4.

$n = 30 \quad l = 500 \quad l_c = 10$ Nombre de simulations : 1000			
Estimation $\bigtriangledown$		b	Autres performances
GA-Mots	A	4.07	$\mathcal{P}(b = 4) = 0.2,\ \mathcal{P}(\lvert b - m^* \rvert \leq 2) = 0.68,\ \mathcal{P}(\lvert b - m \rvert \leq 5) = 0.99$
	B	8.16	$\mathcal{P}(b = i)_{i \in [6,11]} \approx 0.1,\ \mathcal{P}(\lvert b - m \rvert \leq 5) = 0.84$
GA-MotsP	A	0,31	$\mathcal{P}(b = 0) = 0.73,\ \mathcal{P}(b = 1) = 0.23,\ \mathcal{P}(b = 2) = 0.04$
	B	6.53	$\mathcal{P}(b = i)_{i \in [4,8]} \approx 0.15,\ \mathcal{P}(\lvert b - m \rvert \leq 5) = 0.96$
GA-Cano	A	0	$\mathcal{P}(b = 0) = 1$
	B	21.7	$\mathcal{P}(b = i)_{i \in [19,24]} \approx 0.07,\ \mathcal{P}(\lvert b - m \rvert \leq 5) = 0.63,\ \mathcal{P}(\lvert b - m \rvert \leq 10) = 0.95$
GA-CompC	A	$l_c(a)$	$\mathcal{P}(b = l_c(a)) = 1$
	B	$l_c(a)$	$\mathcal{P}(b = l_c(a) - i)_{i \in [-2,3]} \approx 0.05,\ \mathcal{P}(\lvert l_c(a) - b \rvert \leq 10) = 0.74$
GA-TronC	A	3.14	$\mathcal{P}(b = 3) = 0.26,\ \mathcal{P}(b = 2, 4) \approx 0.2,\ \mathcal{P}(b \in [1, 6]) = 0.95$
	B	15.65	$\mathcal{P}(b = i)_{i \in [13,18]} \approx 0.05,\ \mathcal{P}(\lvert b - m \rvert \leq 10) = 0.80$

* : m valeur moyenne du biais

Tab. 8.1 – Statistiques du biais - génération "Artin"

$n=30$ $l=500$ $l_c=10$ Nombre de simulations : 1000			
Estimation $\bigtriangledown$		b	Autres performances
GA-MOTS	B	0.81	$\mathcal{P}(b=0,1)\approx 0.4,\ \mathcal{P}(b=2)=0.17,\ \mathcal{P}(b=3)=0.03$
	A	2.07	$\mathcal{P}(b=2)=0.43,\ \mathcal{P}(b=1,3)\approx 0.25,\ \mathcal{P}(b=0,4)\approx 0.05$
GA-MOTSP	B	0	$\mathcal{P}(b=0)=1$
	A	2.21	$\mathcal{P}(b=2)=0.42,\ \mathcal{P}(b=2)=0.30,\ \mathcal{P}(b=1)=0.21$
GA-CANO	B	0	$\mathcal{P}(b=0)=1$
	A	2.47	$\mathcal{P}(b=2,3)\approx 0.34,\ \mathcal{P}(\lvert b-m\rvert\leq 2)=0.95$
GA-COMPC	B	$l_c(a)$	$\mathcal{P}(b=l_c(a))=0.99$
	A	$l_c(a)$	$\mathcal{P}(b=l_c(a))=0.26,\ \mathcal{P}(\lvert b-l_c(a)\rvert\leq 2)=0.80$
GA-TRONC	B	0.57	$\mathcal{P}(b=0)=0.53,\ \mathcal{P}(b=1)=0.38,\ \mathcal{P}(b=2)=0.08$
	A	2.16	$\mathcal{P}(b=2)=0.45,\ \mathcal{P}(b=1,3)\approx 0.23,\ \mathcal{P}(b\in[0,4])=0.99$

* : m valeur moyenne du biais

TAB. 8.2 – Statistiques du biais - génération "BKL"

$n=30$ $l=500$ $l_c=10$ Nombre de simulations : 1000				
Estimation $\bigtriangledown$		$l_c(a)$	$l_c(a)-e^{*^1}$	Autres performances
GA-MOTS	A	28.72	2.46	$\mathcal{P}(e=l_c(a)-i)_{i\in[1,3]}\approx 0.20,\ \mathcal{P}(\lvert l_c(a)-e-m^{*^2}\rvert\leq 5)=0.99$
	B	41.17	3.70	$\mathcal{P}(e=lc(a)-i)_{i\in[1,6]}\approx 0.1,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.83$
GA-MOTSP	A	28.66	0.31	$\mathcal{P}(e=l_c(a))=0.73,\ \mathcal{P}(e=l_c(a)-1)=0.23$
	B	45.74	2.40	$\mathcal{P}(e=l_c(a)-i)_{i\in[0,4]}\approx 0.15,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.96$
GA-CANO	A	10	0	$\mathcal{P}(e=l_c(a))=1$
	B	73.87	10.78	$\mathcal{P}(e=l_c(a)-i)_{i\in[8,13]}\approx 0.09,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.74$
GA-COMPC	A	23.97	13.97	$\mathcal{P}(e=l_c(a)-14)=0.25,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.96$
	B	146.3	82.8	$\mathcal{P}(e=l_c(a)-i)_{i\in[80,85]}\approx 0.02,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 20)=0.81$
GA-TRONC	A	10	1.53	$\mathcal{P}(e=l_c(a)-i)_{i\in[1,2]}\approx 0.2,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.97$
	B	30.47	7.86	$\mathcal{P}(e=l_c(a)-i)_{i\in[5,10]}\approx 0.05,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 10)=0.77$

$^{*^1}$: e estimation de $l_c(a)$; $^{*^2}$: m valeur moyenne de $l_c(a)-e$

TAB. 8.3 – Estimation de la longueur canonique - génération "Artin"

$n=30$ $l=500$ $l_c=10$ Nombre de simulations : 1000				
Estimation $\bigtriangledown$		$l_c(a)$	$l_c(a)-e^{*^1}$	Autres performances
GA-MOTS	B	251.2	0.4	$\mathcal{P}(e=l_c(a))=0.45,\ \mathcal{P}(e=l_c(a)+1)=0.32$
	A	200.5	1.05	$\mathcal{P}(e=l_c(a)-1)=0.47,\ \mathcal{P}(e=l_c(a)-i)_{i\in\{0,2\}}\approx 0.24$
GA-MOTSP	B	232.0	0	$\mathcal{P}(e=10)=1$
	A	203.2	01.06	$\mathcal{P}(e=l_c(a)-1)=0.49,\ \mathcal{P}(e=l_c(a)-i)_{i\in\{0,2\}}\approx 0.23$
GA-CANO	B	10	0	$\mathcal{P}(e=10)=1$
	A	13.3	1.21	$\mathcal{P}(e=l_c(a)-1)=0.44,\ \mathcal{P}(e=l_c(a)-2)=0.29$
GA-COMPC	B	24.19	14.19	$\mathcal{P}(b=l_c(a)-i)_{i\in[14,16]}\approx 0.2,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.96$
	A	28.14	16.01	$\mathcal{P}(b=l_c(a)-i)_{i\in[14,19]}\approx 0.1,\ \mathcal{P}(\lvert l_c(a)-e-m\rvert\leq 5)=0.91$
GA-TRONC	B	10	0.26	$\mathcal{P}(e=l_c(a))=0.56,\ \mathcal{P}(e=l_c(a)-1)=0.27$
	A	9.83	1.12	$\mathcal{P}(e=l_c(a)-1)=0.29,\ \mathcal{P}(e=l_c(a)-1)=0.38$

$^{*^1}$: e estimation de $l_c(a)$; $^{*^2}$: m valeur moyenne de $l_c(a)-e$

TAB. 8.4 – Estimation de la longueur canonique - génération "BKL"

Ces valeurs sont données dans un cadre précis et illustrent uniquement le fait qu'une anticipation statistique est possible dans certains cas. Il est à prévoir que ces valeurs ne seront pas valables pour des situations ne répondant pas exactement aux mêmes conditions. Pour chaque situation, on considère une instance de conjugaison dans $B_n : (x, x' = axa^{-1})$.

8.2 Dépendance aux générateurs

Dans cette section se trouvent des résultats numériques plus complets que ceux présentés dans les Sections 6.2.1 et 6.3. Ils concernent l'évaluation de l'efficacité de l'algorithme de réduction, Conj (ConjMS), pour le problème de conjugaison multiple simultanée sur les différents générateurs aléatoires de tresses introduits. Les Tables concernées sont 8.5 et 8.6.

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000								
	$l_c(a)$	$l(a)$	^{*2}	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-Mots	28.8	12040.0	1	2042.5	2022.3	1775.6	4064.8	8630
			2	1483.6	1470.6	1263.4	2954.2	6142
			3	1130.2	1122.8	954.7	2252.9	4642
GA-MotsP	28.7	500	1	11.7	209.5	11.7	221.2	62
			2	3.8	51.1	3.7	54.9	23
			3	1.6	22.6	1.4	24.2	11
GA-Cano	10	2393.3	1	18.8	18.5	13.4	37.3	70
			2	3.1	3.1	1.8	6.2	7
			3	0.8	0.7	0.3	1.6	1
GA-CompC^{*1}	13.0	5812.7	1	2870.3	2802.9	2710.6	5673.3	13173
GA-TronC	10	2651.6	1	1113.7	1180.5	868.0	2294.3	4221
			2	826.1	935.6	636.5	1761.7	3097
			3	547.2	650.0	410.3	1197.2	1998

^{*1} : $x, x', a :=$ GA-CompC(5), ^{*2} : instance du PCMS

Tab. 8.5 – Dépendance aux générateurs - PCMS - présentation "Artin"

Remarque 8.2.1. Il est à noter que nous avons utilisé la variante ConjMS pour le problème de conjugaison, mais que nous avons simplement utilisé Conj pour le problème de conjugaison multiple simultanée. Ainsi, il est parfois arrivé que le diviseur ou le multiple résultant d'une instance double du problème de conjugaison multiple simultanée soit moins bon que celui obtenu pour le problème de conjugaison équivalent, ce qui est impossible normalement. Nous avons fait le choix de ne pas aller jusqu'au bout des réductions possibles, car ceci n'était pas utile à la mise en évidence des propriétés recherchées.

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000								
	$l_c(a)$	$l(a)$	[*2]	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-MOTS[*3]	249.3	7155.3	1	52.5	57.4	46.2	109.9	414
			2	30.3	30.9	26.0	61.2	236
			3	19.4	19.2	15.3	38.6	143
GA-MOTSP[*3]	231.9	500	1	0.2	236.0	0.2	236.1	10.5
			2	0	231.9	0	231.9	0
			3	0	231.9	0	231.9	0
GA-CANO	10	564.7	1	4.8	4.7	3.5	9.5	39
			2	1.0	0.9	0.4	1.9	12
			3	0.2	0.2	0.1	0.4	10
GA-COMPC[*1]	13.3	516.5	1	194.8	192.0	182.7	386.8	1610
GA-TRONC	10	565.2	1	28.7	46.5	26.6	75.2	242
			2	13.2	25.2	11.9	38.4	113
			3	6.7	14.5	5.8	21.2	60

[*1] : $x, x', a :=$ GA-COMPC(5), [*2] : instance du PCMS, [*3] : Nombre de simulations : 20

TAB. 8.6 – Dépendance aux générateurs - PCMS - présentation “BKL”

8.3 Dépendance aux formats des entrées

Dans cette section se trouvent des résultats numériques reprenant ceux présentés dans la Section 6.2.2. Ils concernent l'évaluation de l'efficacité de l'algorithme de réduction, CONJ , pour le problème de conjugaison et le problème de conjugaison de type Ko-Lee sur les générateurs aléatoires de tresses GA-CANO et GA-MOTSP. Les Tables sont : 8.7, 8.8, 8.9, 8.10, 8.11, 8.12.

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000							
	$l_c(a)$	$l(a)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-MOTSP	53.0	500	5.9	58.5	5.9	64.4	26
GA-CANO	14.2	3748.4	10.3	12.7	7.2	23.0	31

TAB. 8.7 – Problème de conjugaison de type Ko-Lee - présentation “Artin”

$n = 30$ $l = 500$ $l_c = 10$ Nombre de simulations : 1000							
	$l_c(a)$	$l(a)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
GA-MOTSP	238.5	500	0.1	1.2	0.1	1.3	7
GA-CANO	14.2	282.8	2.6	3.1	1.8	5.7	19

TAB. 8.8 – Problème de conjugaison de type Ko-Lee - présentation “BKL”

Nombre de simulations : 1000									
$\mathcal{C}_a$, $\mathcal{C}_r$			$l(a) = l(x)$						
			500		1000		1500		$\mathcal{L}_{B_n}$
GA-MotsP	n	30	11.6	223.1	11.8	229.6	11.8	218.0	62
		50	18.6	587.9	19.1	567.1	18.7	573.1	110
		100*	36.2	7799.5	36.5	1755.7	37.5	1931.5	250
			$lc(a) = lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	n	30	13.2	36.6	13.3	37.3	13.1	37.4	69
		50	17.6	45.5	17.4	46.0	17.4	45.7	101
		100	31.0	75.0	30.2	72.4	30.2	72.7	195

* : Nombre de simulations : 200

Tab. 8.9 – Nombre de brins/Taille du secret - présentation “Artin”

n = 30, Nombre de simulations : 1000									
$\mathcal{C}_a$, $\mathcal{C}_r$			$l(x)$						
			500		1000		1500		$\mathcal{L}_{B_n}$
GA-MotsP	$l(a)$	500	11.7	225.5	11.7	226.2	11.7	227.8	62
		1000	11.7	235.1	11.6	228.9	11.5	218.8	
		1500	11.7	232.8	11.8	239.8	11.8	217.6	
			$lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	$l_c(a)$	5	12.9	36.2	13.6	37.6	12.9	37.1	68
		10	13.0	37.2	13.0	37.0	13.3	37.5	
		20	13.0	36.2	13.1	36.7	13.0	36.5	

Tab. 8.10 – Taille des éléments - présentation “Artin”

Nombre de simulations : 1000									
$\mathcal{C}_a$, $\mathcal{C}_r$			$lc(a) = lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	n	30	3.2	8.5	3.5	9.5	3.5	9.4	39
		50	4.3	11.2	4.5	11.7	4.7	11.8	56
		100	8.0	19.2	8.3	20.1	8.5	20.2	114

Tab. 8.11 – Nombre de brins/Taille du secret - présentation “BKL”

n = 30, Nombre de simulations : 1000									
$\mathcal{C}_a$, $\mathcal{C}_r$			$lc(x)$						
			5		10		20		$\mathcal{L}_{B_n}$
GA-Cano	$l_c(a)$	5	3.1	8.5	3.3	8.8	3.3	8.9	38
		10	3.4	9.0	3.4	9.3	3.5	9.3	
		20	3.4	9.2	3.5	9.4	3.6	9.5	

Tab. 8.12 – Taille des éléments - présentation “BKL”

8.4 Utilisation des modes

Dans cette section se trouvent des résultats numériques reprenant ceux présentés dans la Section 6.4. Ils concernent l'évaluation de l'efficacité de l'algorithme de réduction, Conj, lors de son utilisation en parallèle et en séquentiel sur les générateurs aléatoires de tresses GA-Cano et GA-MotsP. Les Tables sont : 8.13, 8.14, 8.15, 8.16, 8.17 et 8.18

$n = 30$ $l_c = 10$ Nombre de simulations : 20						
instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
Présentation : BKL		$l_c(a_{\text{Artin}}) = 10$, $l(a_{\text{Artin}}) = 2294.1$, $l_c(a_{\text{BKL}}) = 74.8$, $l(a_{\text{BKL}}) = 2294.1$				
1	B	397.4	419.5	358.9	816.9	3155
	B // A	18.3	17.4	13.5	35.7	127
	B+A+B	16.7	15.4	12.2	32.1	127
2	B	345.5	370.4	312.1	715.9	2744
	B // A	3.0	2.2	1.4	5.2	21
	B+A+B	2.2	2.2	1.2	4.4	19
3	B	315.8	332.3	279.1	648.1	2455
	B // A	1.0	0.6	0.3	1.6	12
	B+A+B	1.9	2.1	1.0	4.0	18

Tab. 8.13 – Mode parallèle/séquentiel - PCMS - GA-Cano - génération "Artin"

$n = 50$ $l = 1000$ Nombre de simulations : 50						
Instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
Présentation : BKL		$l_c(a_{\text{Artin}}) = 83.5$, $l(a_{\text{Artin}}) = 1000$, $l_c(a_{\text{BKL}}) = 54.0$, $l(a_{\text{BKL}}) = 1000$				
1	B	260.3	1312.1	260.3	1572.4	2680
	B // A	17.3	1278.1	17.3	1295.5	188
	B+A+B	117.5	89.9	35.1	207.4	370
2	B	238.2	1224.0	238.2	1462.2	2454
	B // A	5.6	1223.0	5.6	1228.6	68
	B+A+B	53.4	25.3	13.5	78.7	148
3	B	2241.1	1171.5	224.1	1395.6	2309
	B // A	2.3	1147.8	2.3	1150.1	34
	B+A+B	51.4	23.3	11.6	74.7	129

Tab. 8.14 – Mode parallèle/séquentiel - PCMS - n=50 - GA-MotsP - génération "Artin"

$n = 30 \quad l_c = 10$ Nombre de simulations : 20							
Présentation	$*^1$		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
"BKL"	$l_c(a_{\text{Artin}}) = 13.4,\ l(a_{\text{Artin}}) = 4829.4,$ $l_c(a_{\text{BKL}}) = 10,\ l(a_{\text{BKL}}) = 578.7$						
	1	B	4.9	4.7	3.8	9.6	42
		B // A	4.9	4.7	3.8	9.6	42
		B+A+B	3.7	3.8	2.9	7.5	34
	2	B	0.7	1.1	0.5	1.8	13
		B // A	0.7	1.1	0.5	1.8	13
		B+A+B	0.4	0.7	0.4	1.0	12
	3	B	0.2	0.3	0	0.4	0
		B // A	0.2	0.3	0	0.4	0
		B+A+B	0	0.3	0	0.3	0
"Artin"	$l_c(a_{\text{Artin}}) = 10,\ l(a_{\text{Artin}}) = 2395.5,$ $l_c(a_{\text{BKL}}) = 73.4,\ l(a_{\text{BKL}}) = 2395.5$						
	1	A	18.7	18.5	13.1	37.2	69
		A+B+A	15.3	15.0	10.2	30.4	54
	2	A	2.7	2.9	1.5	5.6	6
		A+B+A	1.5	1.6	1.0	3.1	3
	3	A	0.7	0.8	0.3	1.4	1
		A+B+A	0.4	0.4	0.1	0.8	1

$*^1$: instance du PCMS

TAB. 8.15 – Mode parallèle/séquentiel - PCMS - n=30 - GA-CANO

$n = 50 \quad l_c = 15$ Nombre de simulations : 20							
Présentation	$*^1$		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
"BKL"	$l_c(a_{\text{Artin}}) = 21.3,\ l(a_{\text{Artin}}) = 23180.1,$ $l_c(a_{\text{BKL}}) = 15,\ l(a_{\text{BKL}}) = 1614.7$						
	1	B	6.3	7.1	5.7	13.4	69
		B // A	6.3	7.1	5.7	13.4	69
		B+A+B	5.2	6.2	4.4	11.3	55
	2	B	1.7	1.3	0.9	3.0	20
		B // A	1.7	1.3	0.9	3.0	20
		B+A+B	0.9	0.7	0.6	1.6	16
	3	B	0.3	0.3	0.1	0.6	11
		B // A	0.3	0.3	0.1	0.6	11
		B+A+B	0.2	0.1	0.1	0.3	11
"Artin"	$l_c(a_{\text{Artin}}) = 15,\ l(a_{\text{Artin}}) = 9778.6,$ $l_c(a_{\text{BKL}}) = 178.0,\ l(a_{\text{BKL}}) = 9778.6$						
	1	A	20.4	20.8	15.8	41.2	90
		A+B+A	18.4	19.1	13.9	37.6	78
	2	A	3.0	3.6	2.0	6.7	8
		A+B+A	1.9	2.2	1.4	4.1	5
	3	A	0.8	1.0	0.5	1.8	1
		A+B+A	0.4	0.7	0.3	1.2	1

$*^1$: instance du PCMS

TAB. 8.16 – Mode parallèle/séquentiel - PCMS - n=50 - GA-CANO

$n = 30 \quad l = 500$ Nombre de simulations : 50						
instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
$l_c(a_{\text{Artin}}) = 201.5,\ l(a_{\text{Artin}}) = 68963.6,$ $l_c(a_{\text{BKL}}) = 229.4,\ l(a_{\text{BKL}}) = 500$						
1	B	0	232.8	0	232.8	0
	B // A	0	232.8	0	232.8	0
	B+A+B	0	0	0	0	0
2	B	0	229.4	0	229.4	0
	B // A	0	229.4	0	229.4	0
	B+A+B	0	0	0	0	0
3	B	0	229.4	0	229.4	0
	B // A	0	229.4	0	229.4	0
	B+A+B	0	0	0	0	0

TAB. 8.17 – Mode parallèle/séquentiel - PCMS - GA-MOTSP - présentation “BKL”

$n = 50 \quad l = 1000$ Nombre de simulations : 50						
Instance du PCMS		$l(d^{-1}a)$	$l(a^{-1}m)$	$\mathcal{C}_a$	$\mathcal{C}_r$	$\mathcal{L}_{B_n}$
$l_c(a_{\text{Artin}}) = 35.0,\ l(a_{\text{Artin}}) = 1000,$ $l_c(a_{\text{BKL}}) = 56.4,\ l(a_{\text{BKL}}) = 1000$						
1	A	18.3	481.6	18.3	499.9	108
	A+B+A	15.6	276.8	15.6	292.3	93
2	A	6.2	107.0	6.2	113.2	40
	A+B+A	4.6	42.8	4.5	47.4	30
3	A	2.5	44.5	2.5	47.1	20
	A+B+A	2.0	4.9	1.8	6.9	8

TAB. 8.18 – Mode séquentiel - PCMS - n=50 - GA-MOTSP - présentation “Artin”

8.5 L'ordre lexicographique

Voici les trois programmes en langage MAGMA utilisés pour travailler sur l'ordre lexicographique. Ils travaillent dans la présentation d'Artin :

- EXTRACT_GENE (g, k, L) permet de réécrire un mot en extrayant à gauche, si c'est possible, l'atome g à la position k en considérant uniquement le sous-mot de droite commençant à la position k. Un mot positif est décrit par la liste de ses atomes : L.
- LEXICO (a) permet de réécrire un élément du monoïde a tel que le mot qui le représente respecte l'ordre lexicographique.
- LEXICOMPT (a, k) permet de dénombrer tous les sous-mots non équivalents de longueur k qui divisent à gauche l'élément positif a.

Le groupe est initialisé par :

```
B :=BraidGroup(n);
D :=FundamentalElement(B);
```

Procédure 8.5.1. EXTRACT_GENE (g, k, L)

Entrée : Soit g un atome ($\in \mathcal{A}(B_n^+)$), L une liste d'atomes et $k \in [1, \#L]$.

```
extract_gene := function(g,k,L)
        local i,cond,var,LL;
        if k gt #L then return false,L; end if;
        if L[k] eq g then return true,L; end if;

        cond :=k;
        while cond le #L and L[cond] ne g do cond + :=1; end while;
        if cond eq #L+1 then return false,L; end if;
        LL :=L;
        while cond gt k do
            while cond ne k and Abs(LL[cond-1]-g) ne 1 do
                LL[cond] :=LL[cond-1];
                cond - :=1;
                LL[cond] :=g;
            end while;
            if cond eq k then return true,LL; end if;
            var,LL :=$$ (LL[cond-1],cond+1,LL);
            if var then
                LL[cond] :=LL[cond-1];
                LL[cond-1] :=g;
                LL[cond+1] :=g;
                cond - :=1;
            else return false,LL;
            end if;
        end while;
        return true,LL;
        end function;
```

Sortie : *vrai* ou *faux*, et une liste d'atomes *équivalente* à L : LL
Si *vrai* alors $\forall i \in [1, k-1]$, $LL[i] = L[i]$ et $L[k] = g$.

Procédure 8.5.2. LEXICO (b)

Entrée : Soit b un élément du monoïde B_n^+.

```
lexico := function(b)
        local ind,i,j,var,L,bb ;
        L :=WordToSequence(b) ;
        ind :=1 ;
        for i :=1 to #b do
            var,L :=extract_gene(ind,i,L) ;
            while not(var) do
                ind + :=1 ;
                var,L :=extract_gene(ind,i,L) ;
            end while ;
            if ind gt 1 then ind - :=1 ; end if ;
        end for ;
        bb :=B !L ;
        return bb ;
        end function ;
```

Sortie : L'élément b dont l'écriture respecte l'ordre lexicographique

Procédure 8.5.3. LEXICOMPT (b, k)

Entrée : Soit b un élément du monoïde B_n^+ et $k \in [1, \#b]$.

```
Lexicompt := function(b,k)
        local ind,i,j,l,m,compt,var,L,LL,varLL ;
        compt :=0 ;
        L :=WordToSequence(b) ;
        if k le 0 or k gt #b then return 0 ; end if ;
        ind :=1 ;
        for i :=1 to k do
            var,L :=extract_gene(ind,i,L) ;
            while not(var) do
                ind + :=1 ;
                var,L :=extract_gene(ind,i,L) ;
            end while ;
            if ind gt 1 then ind - :=1 ; end if ;
        end for ;
        compt + :=1 ;
        i :=k ;
        while i ge 1 do
            j :=L[i]+1 ;
            while j le n-1 do
                var,L :=extract_gene(j,i,L) ;
                if var and i eq k then
                    LL :=[L[m] : m in [1..k]] ;
                    for m :=1 to k do
                        varLL,LL :=extract_gene(LL[m]-1,m,LL) ;
                        if varLL then break ; end if ;
                    end for ;
```

```
                    if not(varLL) then compt + :=1 ; end if ;
                end if ;
                if var and i ne k then
                    while i ne k do
                        i + :=1 ;
                        ind :=L[i-1]-1 ;
                        for l :=i to k do
                            var,L :=extract_gene(ind,l,L) ;
                            while not(var) and ind lt n-1 do
                                ind + :=1 ;
                                var,L :=extract_gene(ind,l,L) ;
                            end while ;
                            if var and ind gt 1 then ind - :=1 ;
                            else
                                i :=l ;
                                while not(var) and i gt 1 do
                                    i - :=1 ;
                                    ind :=L[i] ;
                                    while not(var) and ind lt n-1 do
                                        ind + :=1 ;
                                        var,L :=extract_gene(ind,i,L) ;
                                    end while ;
                                end while ;
                                if not(var) then return compt ; end if ;
                                break ;
                            end if ;
                        end for ;
                    end while ;
                    LL :=[L[m] : m in [1..k]] ;
                    for m :=1 to k do
                        varLL,LL :=extract_gene(LL[m]-1,m,LL) ;
                        if varLL then break ; end if ;
                    end for ;
                    if not(varLL) then compt + :=1 ; end if ;
                    break ;
                end if ;
                j + :=1 ;
            end while ;
            if j eq n then i - :=1 ; end if ;
        end while ;
        return compt ;
        end function ;
```

Sortie : Le nombre d'éléments du monoïde de longueur k divisant à gauche b

Table des figures

1.1 Une tresse géométrique à quatre brins 18
1.2 Le produit de deux tresses géométriques 19
1.3 L'inversion de tresses géométriques 20
1.4 Représentation des générateurs du monoïde $B_n^+ : \sigma_i,\ \sigma_i^{-1}$ 21
1.5 Représentation des relations du monoïde B_n^+ 22
1.6 Représentation de l'élément de Garside $\Delta_{B_5^+}$ 23
1.7 Représentation dans le monoïde BKL_n^+ du générateur $a_{t,s}$ et de $\Delta_{BKL_5^+}$ . . 23
1.8 Treillis des éléments simples de $B_4^+ : \left(S_{B_4^+}, \prec\right)$. 26

2.1 Illustration de l'Exemple 2.1.1 29

4.1 Réduction d'une instance de conjugaison 47

5.1 Utilisation en parallèle de Conj 55
5.2 Utilisation en séquentiel de Conj 56

6.1 Réduction du secret 67
6.2 Nombre de préfixes de mots de longueur 50 dans B_{20}^+ 75
6.3 Préfixes de mots de longueur 60 dans B_n^+ pour $n \in \{10, 20, 25, 30\}$ 75
6.4 Préfixes de sous-mots d'un mot donné dans B_{15}^+ 76

Liste des tableaux

2.1 Table de complexité des opérations dans le groupe de tresses à n brins. . . . 30

5.1 Estimation de la longueur canonique - présentation d'Artin 61
5.2 Statistiques du biais - présentation d'Artin 61

6.1 Dépendance aux générateurs - PCMS - présentation "Artin" 69
6.2 Nombre de brins/Taille des éléments - GA-Cano - présentation "Artin" . . 69
6.3 Problème de conjugaison de type Ko-Lee - GA-Cano - présentation "Artin" 70
6.4 Statistique du GA-TronC sur la présentation d'Artin 72
6.5 Dépendance au GA-TronC - PCMS - présentation "Artin" 72
6.6 Mode parallèle/séquentiel - PCMS - GA-MotsP - génération "Artin" . . . 73
6.7 Mode séquentiel - PCMS - GA-MotsP - génération "Artin" 73
6.8 Mode séquentiel - PC - présentation "Artin" 74
6.9 Mode séquentiel - PC - GA-Cano - présentation "Artin" 77

8.1 Statistiques du biais - génération "Artin" 99
8.2 Statistiques du biais - génération "BKL" 100
8.3 Estimation de la longueur canonique - génération "Artin" 100
8.4 Estimation de la longueur canonique - génération "BKL" 100
8.5 Dépendance aux générateurs - PCMS - présentation "Artin" 101
8.6 Dépendance aux générateurs - PCMS - présentation "BKL" 102
8.7 Problème de conjugaison de type Ko-Lee - présentation "Artin" 102
8.8 Problème de conjugaison de type Ko-Lee - présentation "BKL" 102
8.9 Nombre de brins/Taille du secret - présentation "Artin" 103
8.10 Taille des éléments - présentation "Artin" 103
8.11 Nombre de brins/Taille du secret - présentation "BKL" 103
8.12 Taille des éléments - présentation "BKL" 103
8.13 Mode parallèle/séquentiel - PCMS - GA-Cano - génération "Artin" 104
8.14 Mode parallèle/séquentiel - PCMS - n=50 - GA-MotsP - génération "Artin" 104
8.15 Mode parallèle/séquentiel - PCMS - n=30 - GA-Cano 105
8.16 Mode parallèle/séquentiel - PCMS - n=50 - GA-Cano 105
8.17 Mode parallèle/séquentiel - PCMS - GA-MotsP - présentation "BKL" . . 106
8.18 Mode séquentiel - PCMS - n=50 - GA-MotsP - présentation "Artin" . . . 106

Index

$1, e$ 19
BKL_n^+ 23
B_n 24
B_n^+ 22
LB_l, RB_r 33, 64
$Z(\,)$ 22
$\mathcal{A}(\,)$ 22, 81
$\mathcal{C}1$, $\mathcal{C}2$, $\mathcal{C}3$, $\mathcal{C}4$, $\mathcal{C}5$ 83
$\mathcal{C}_a$, $\mathcal{C}_r$ 68
Δ, S 21
$\mathcal{S}(\,)$, $\mathcal{F}(\,)$, $\mathcal{R}(\,)$ 81
Σ_n 17, 23, 24
$\inf(\,)$, $l_c(\,)$, $\sup(\,)$ 25
$\prec$, $\succ$ 20
$\mathfrak{a}$, $\mathfrak{A}$ 85
$\tau(\,)$, $\mathcal{E}$ 21
$\wedge$, $\vee$, $\tilde{\wedge}$, $\tilde{\vee}$ 20
*, $\partial(\,)$ 41
b_{GA} 59
$c(\,)$, $d(\,)$ 26
$l(\,)$ 24

A, B 99
Algorithme
    InvCyclage, 86
    Estimateur_LB_l, 65
    Type1, 90
    Type2, 91
    Type3, 93
    Type4, 95
    Conj, 51
    ConjKL, 64
    ConjMS, 63
    Diviseur, 50
    Estimateur_l_c, 60
    Multiple, 49
Atome 20, 81

Biais 59
Burau 36

Complémentaire à droite 81
Connaissance
    Absolue, 68
    Relative, 68
Cyclage 26, 36, 79
Cycle descendant 24

Décyclage 26
Dual 41

Elément de Garside 21
Elément simple 21
Element 31
Equivalence 31
Exposant 22, 57

Facteur canonique 21
Finishing Set 81
Fonction de hachage 34
Forme
    Δ-normale, 25
    Garside, 32
    Normale, 31
    Réduite, 31

Générateur aléatoire 38, 58
    GA-Cano, 40
    GA-CompC, 40
    GA-TronC, 72
    GA-Mots, 39
    GA-MotsP, 39
    Biais, 59
Groupe de Garside 21

Infimum 25
Isotopie 18

Lawrence-Krammer 37

Longueur 24
    Canonique, 25

Monoïde 19
    Artin, 22
    Atomique, 20
    Birman, Ko et Lee, 23
    Garside, 21
    Gaussien, 20
    Simplifiable, 20
Mot 31

Ordre lexicographique 31, 92

Parallèle 56
PC, PCMS 67
Petit groupe gaussien 19
Préfixe 20
Présentation
    Artin, 22
    Birman, Ko et Lee, 23
Presentation 22
Problème
    Conjugaison, 32
    Cyclage, 79
    Décisionnel, 32, 36
    Décomposition, 33
    Mots, 31
    Multiple simultanée, 32, 62
    Racines, 32
    Type Diffie-Hellman, 32
    Type Ko-Lee, 32, 64
Procédure
    Extract_gene, 107
    Lexico, 108
    Lexicompt, 109
Protocole
    Authentification, 34
    Cryptosystème à clé publique, 34
    Echange de clés, 33
Pure 38

Réduction des poignées 31
Représentation linéaire
    Burau, 36
    Lawrence-Krammer, 37

Séquentiel 57
Starting Set 81
Suite normale 24
Super Summit Set 35, 39, 87
Supremum 25

Tresse géométrique 18

Ultra Summit Set 36, 39, 87

Bibliographie

[1] I. Anshel, M. Anshel, D. Goldfeld, *An algebraic method for public-key cryptography*, Mathematical Research Letters 6, 1999, 287-291.

[2] I. and M. Anshel, B. Fischer, D. Goldfeld, *New key agreement protocols in braid group cryptography*, RSA 2001, LNCS 2020 (2001),1-15.

[3] E. Artin, *Theorie of zöpfe*, Abh. Math. Sem. Univ. Hamburg, 4 (1925), 47-72.

[4] E. Artin, *Theory of braids*, Annals of Math. 48 (1947), 101-126.

[5] J.S. Birman, K.H. Ko, S.J. Lee, *A new approach to the word and conjugacy problems in the braid groups*, Advances in Math. 139-2 (1998), 322-353.

[6] J.S. Birman, K.H. Ko, S.J. Lee, *The infimum, supremum and geodesic length of a braid conjugacy class*, Advances in Math. 146 (2001), 41-56.

[7] J.C. Cha, K.H. Ko, S.J. Lee, J.W. Han, J.H. Cheon, *An efficient implementation of braid groups*, Asiacrypt 2001, LNCS 2248 (2001), 144-156.

[8] J.H. Cheon, B. Jun, *A polynomial time algorithm for the braid Diffie-Hellman conjugacy problem*, Crypto 2003.

[9] A.H. Clifford, G.B. Preston, *The algebraic theory of semigroups*, Vol1, AMS surveys 7 (1961).

[10] P. Dehornoy, *Deux propriétés des groupes de tresses*, C.R. Acad. Sér. I 315 (1992), 633-638.

[11] P. Dehornoy, *L'art de tresses*, Pour la science, Dossier hors-série *La science des noeuds* (1997), 68-75.

[12] P. Dehornoy, *A fast method for computing braids*, Adv. in Math. 125-2 (1997), 200-235.

[13] P. Dehornoy, *Groupes de Garside*, Ann. Sc. Ec. Norm. Sup., 35 (2002), 267-306

[14] P. Dehornoy, *Braid-based cryptography*, Contemp. Math., Amer. Math. Soc. 360, 2004.

[15] P. Dehornoy, L. Paris, *Gaussian groups and Garside groups, two generalisations of Artin groups*, Proc. London Math. Soc. 79(3) (1999), 569-604.

[16] E.A. Elrifai, H.R. Morton, *Algorithms for positive braids*, Quart. J. Math. Oxford 45-2, 1994, 479-497.

[17] D. Epstein, J. Cannon, D. Holt, S. Levy, M. Patterson, W. Thurston, *Word processing in groups*, Jones et Barlett Publishers, Boston, chapter 9, 1992.

[18] N. Franco, J. Gonzalez-Meneses, *Conjugacy problem for braid groups and Garside groups*, Journal of Algebra 266-1 (2003), 112-132.

[19] D. Garber, S. Kaplan, M. Teicher, B. Tsaban, U. Vishne, *Length-based conjugacy search in the Braid group*, (2002) : http ://arXiv.org/abs/math.GR/0209267

[20] F.A. Garside, *The braid group and other groups*, Quart. J. Math. Oxford 20-78, 1969, 235-254.

[21] V. Gebhardt, *A new approach to the conjugacy problem in Garside groups*, (2003), à paraitre dans Journal of Algebra : http ://arXiv.org/abs/math.GT/0306199

[22] D. Hofheinz, R. Steinwandt, *A practical attack on some braid group cryptographic primitives*, PKC 2003 Proceedings, LNCS 2567, 2003, 187-198.

[23] J. Hughes, Allen Tannenbaum, *Length-based attacks for certain group based encryption rewriting systems*, Inst. for Math. and its Applic. Minneapolis (2000).

[24] A. Jacquemard, *About the effective classification of conjugacy classes of braids*, Journal of pure and applied algebra, 63-2, 1990, 161-169.

[25] C. Kassel, *L'ordre de Dehornoy sur les tresses*, Séminaire Bourbaki n 865, Astérisque 276, Soc. Math. France (2002), 7-28.

[26] Z. Kim, K. Kim, *Provably-Secure Identification Scheme based on Braid Group*, SCIS 2004.

[27] K.H. Ko, D.H. Choi, M.S. Cho, J.W. Lee, *New Signature Scheme Using Conjugacy Problem* (2002) : http ://eprint.iacr.org/2002/168/

[28] K.H. Ko, S.J. Lee, J.H. Cheon, J.W. Han, J.S. Kang, C. Park, *New public-key cryptosystem using braid groups*, Advances in cryptology - CRYPTO 2000, LNCS 1880 (2000), 166-183.

[29] S.J. Lee, E.K. Lee, *Potential weakness of the commutator key agreement protocol based on braid groups*, Eurocrypt 2002, LNCS 2332 (2002), 14-28.

[30] H.K. Lee, H.S. Lee, Y.R. Lee, *An Authenticated Group Key Agreement Protocol on Braid groups*, (2003) : : http ://eprint.iacr.org/2003/018/

[31] S. Maffre, *Reduction of conjugacy problem in braid groups, using two Garside structures*, WCC 2005, 214-224.

[32] S. Maffre, *A weak key test for braid based cryptography*, à paraître dans Designs, Codes and Cryptography.

[33] J. Michel, *A note on words in braid monoids*, J. of Algebra 215 (1999), 366-377.

[34] L. Perret, *A Chosen Ciphertest Attack on a Public Key Cryptosystem Based on Lyndon Words*, WCC 2005,235-224.

[35] M. Picantin, *Petits groupes quassiens*, Thèse de doctorat, Univ. de Caen (2000).

[36] M. Picantin, *The conjugacy problem in small Gaussian groups*, Communications in Algebra 29-3 (2001), 1021-1039.

[37] H. Sibert, *Algorithmique des groupes de tresses*, Thèse de doctorat, Univ. de Caen (2003).

[38] H. Sibert, P. Dehornoy, M. Girault, *Entity authentification schemes using braid word reduction*, WCC 2003, 153-163.

[39] N.R. Wagner, M.R. Magyarik, *A public-key cryptosystem based on the word problem*, Advances in Cryptology, Proceedings CRYPTO 84, LNCS 196 (1985), 19-36.

[40] The Magma Computational Algebra System for Algebra, Number theory and Geometry : http ://magma.maths.usyd.edu.au/magma

Conjugaison et cyclage dans les groupes de Garside, applications cryptographiques

Résumé

Ce travail s'inscrit dans la thématique de la cryptographie basée sur les tresses. Nous nous intéressons au problème de conjugaison et au problème des cyclages présentés par K.H. Ko, S.J. Lee et al. à CRYPTO 2000 (LNCS 1880) dans *New public-key cryptosystem using braid groups*.

D'une part, nous montrons que l'inversion de la fonction *cyclage* admet une solution polynomiale dans les groupes de Garside, qui sont une généralisation des groupes de tresses ; ceci permet de résoudre efficacement le problème des cyclages.

D'autre part, le travail réalisé sur le problème de conjugaison et ses variantes met en relief le rôle joué par les générateurs aléatoires de tresses. Nous proposons un algorithme qui donne une factorisation du secret sous la forme d'un diviseur et d'un multiple. Ceci permet de définir deux nouvelles instances dont les secrets sont de taille réduite. De plus, nous exploitons la double structure de Garside des groupes de tresses afin d'améliorer l'efficacité de cette réduction. Nous observons que le choix du générateur aléatoire influe grandement sur la sécurité d'une instance et donnons plusieurs éléments constructifs et encourageants pour de futures recherches dans la conception d'un bon générateur aléatoire de tresses.

Mots clés : cryptographie à clé publique, groupes de tresses, groupes de Garside, problème de conjugaison, problème des cyclages, générateur aléatoire de tresses.
Classification AMS : 20F36, 94A60

Abstract

This work deals with braid based cryptography. We study the conjugacy search problem and the cycling problem presented by K.H. Ko, S.J. Lee and al. at CRYPTO 2000 (LNCS 1880) in *New public-key cryptosystem using braid groups*.

On the one hand, we give a polynomial time algorithm to inverse the *cycling* function in Garside group which are a generalization of braid groups ; that allows to solve practically the cycling problem.

On the other hand, our work on the conjugacy search problem and its variants emphasizes the choice of random generator of braids in protocols. We give an algorithm that factorizes the secret into a divisor and a multiple. That allows to define two new conjugacy instances with shorter secrets. Moreover, we exploit the fact that a braid group has two distinct Garside structures to improve the efficiency of the reduction. We observe that the choice of random generators influences greatly the security of an instance and we give several constructive and encouraging elements for further research in the design of good random generator of braids.

Key words : public key cryptography, braid groups, Garside groups, conjugacy search problem, cycling problem, random generator of braids.
AMS Classification : 20F36, 94A60