ブレイド群を利用した公開鍵暗号系の実装

修士学位論文

ブレイド群を利用した公開鍵暗号系の実装

平成１９年度

（２００８ 3月）

東京海洋大学大学院

海洋科学技術研究科

海運ロジスティクス専攻

０６５５００５　　前田 有宏

目 次

第1章　はじめに

ネットワーク化が進んだ今日において，情報を隠蔽するための暗号技術は情報セキュリティ技術の中核をなすものの一つとしてとらえられ，非常に活発に研究がなされている．暗号とは，送信者がメッセージを別の形にすることにより，第三者に対して秘匿し，意図した特定の受信者だけがメッセージを理解できるようにする通信手段のことである．現在，軍事，外交，商業などの幅広い分野で利用されている．

現在最も広く使われている公開鍵暗号は素因数分解の困難性を利用した「RSA 暗号」である．その後,「楕円曲線暗号」や「ElGamal 暗号」などの新しい公開鍵暗号が開発されるなど，現在に至るまで多くの公開鍵暗号が研究されてきている [15].

RSA 暗号や楕円曲線暗号は離散対数問題の困難性に基づいた暗号であるが，2000 年に Ko 達はブレイド群の共役問題の困難性を利用した新たな公開鍵暗号 [10] を提案した．ここでブレイド群とは組み紐群のことであり，x と axa^{-1}から a を求めることを共役問題という．離散対数問題や共役問題はどちらも「群上の関数であって逆問題を解くことが困難なもの」であるから，この点でさらなる一般化が期待されている．Ko 達の暗号が発表されて以来，暗号に対する攻撃の研究も活発に行われ，2003 年に Cheon 達は [7] で上記の暗号に対して攻撃を行う初の多項式時間アルゴリズムを提案した．この攻撃において，Cheon らはブレイド群の線形表現である Lawrence-Krammer 表現 [12] を用いることにより，共役問題を直接解くことなく暗号に対して攻撃を行うことができることを示した．しかし，上記のアルゴリズムには幾つかの誤りがあったため，本論文においては Lukkarila の修正したアルゴリズム [13] を用いることにした．上記の攻撃において，共役問題は解かれておらず，現在においてもブレイド群の共役問題を解くことは難しい．

本論文ではまず，ブレイド群の定義，及び上記の公開鍵暗号に関する総合報告を行う．次にブレイド群の線形表現と暗号に対する攻撃アルゴリズムを示す．実装に関しては Python を用いて暗号を構成するのに必要なアルゴリズムを実装し，Mathematica を用いて攻撃アルゴリズムを実装する．

本論文の構成は以下の通りである．第 2 章でブレイド群に関する諸定義

とブレイドの標準形について述べ，第 3 章で暗号に使われているブレイド群の共役問題，及びブレイド群を利用した公開鍵暗号について述べる．第 4 章ではブレイド群の線形表現と暗号に対する攻撃について述べ，最後に第 5 章で作成したプログラムの構成について述べる．

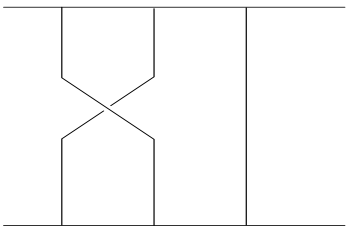
第2章　ブレイド群

2.1　ブレイド群に関する諸定義

この節ではブレイドの積や同値関係といったブレイド群に関する諸定義について述べる [4],[16].

2.1.1　ブレイドとは

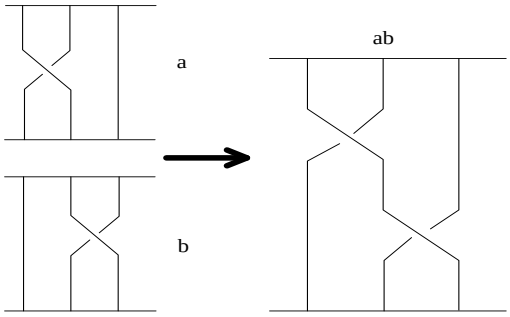
$n-$ ブレイドとは平行な 2 つの面に接続している n 本の紐であり，上の面から出ている紐をたどると常に下へ向かい必ず下の面につくものをいう．$n-$ ブレイド群のことを B_n と表し，n を braid index という．



B_3の例

2.1.2　ブレイドの積

2 つの任意のブレイド a, b の積とは，a を b の上に置くことで得られる．



2 つのブレイドの積

2.1.3 Artin generator

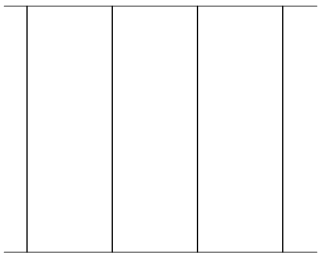
$B_n, n = 1, 2, 3, \ldots$ は $n-1$ 個の生成元 $\sigma_1, \sigma_2 \ldots, \sigma_{n-1}$を持っている. 生成元 σ_i は 左から数えて i 番目と $i+1$ 番目の紐が交差しており, i 番目の紐が $i+1$ 番目の紐の下にきているブレイドを意味している. また，生成元σ_i^{-1}とは,σ_i の紐の重なり方を上下反対にしたものをいい，どの紐も交差していないブレイドを単位元とし，e で表す．上記の生成元は Artin の生成元と呼ばれ，以下のような基本関係式が成り立つ.

$$\sigma_t \sigma_s = \sigma_s \sigma_t \quad \text{if} \quad |t-s| > 1 \tag{2.1}$$

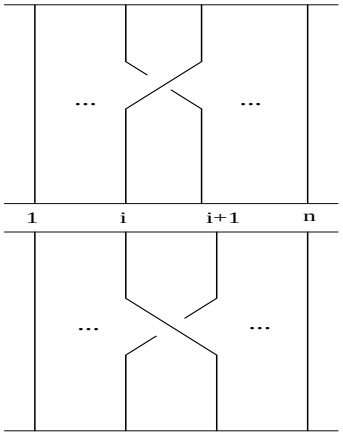
$$\sigma_t \sigma_s \sigma_t = \sigma_s \sigma_t \sigma_s \quad \text{if} \quad |t-s| = 1 \tag{2.2}$$

また，2 つのブレイド a, b があり, a の端点を固定し，上下の平面によって囲まれた空間から紐を出さずに上下関係を維持したまま動かし，b に一致させることができるときを同値という．B_nに対して， $\sigma_i^2 = 1, i = 1, 2, 3, \ldots, n-1$ の関係を加えると，B_nは置換群Σ_nになる. つまり，B_nからΣ_nへの自然な全射準同型写像$\rho : B_n \to \Sigma_n$が定義できる．ここで,σ_iは互換 $(i, i+1)$ に対応している.
例えば $a \in B_4, \pi \in \Sigma_4$とし, $a = \sigma_1\sigma_2\sigma_3$に対する置換は$\pi = 2341$ である.

$$\pi = \begin{pmatrix} 1\ 2\ 3\ 4 \\ 2\ 3\ 4\ 1 \end{pmatrix}$$



単位元 e



生成元σ_i, σ_i^{-1}

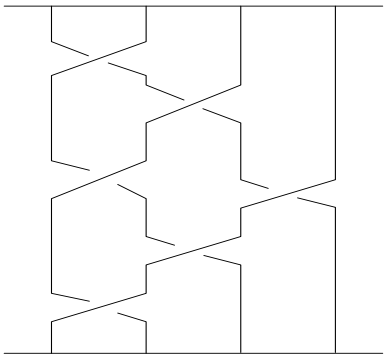
2.1.4 正ブレイド

生成元σ_iと関係式 (2.1), (2.2) によって定義される半群を B_n^+とし，これを正ブレイドと呼ぶ. 正ブレイドの中で，それぞれの紐の交差が高々 1 回であるものを permutation braid と呼び，その集合を$\tilde{\Sigma}_n$と表す．また permutation braid の中で，すべての紐がちょうど一回ずつ交差しているブレイドを fundamental braid と呼び,Δ_nで表す.

$$\Delta_n := \sigma_1\sigma_2\cdots\sigma_{n-1}\Delta_{n-1} \tag{2.3}$$

fundamental braid の例

$$\Delta_4 = \sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1 = \sigma_1\sigma_2\sigma_3\sigma_2\sigma_1\sigma_2 = \sigma_1\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2$$



fundamental braid Δ_4

2.1.5 半順序

B_nの元に対する半順序を次のように定義する. $V \in B_n^+, U = WV$ のとき, $W \leq_L U$ と表す. また, $U = VW$ のとき, $W \leq_R U$ とする. 最後に $V_1, V_2 \in B_n^+, U = V_1WV_2$のとき, $W \leq U$ と表す.

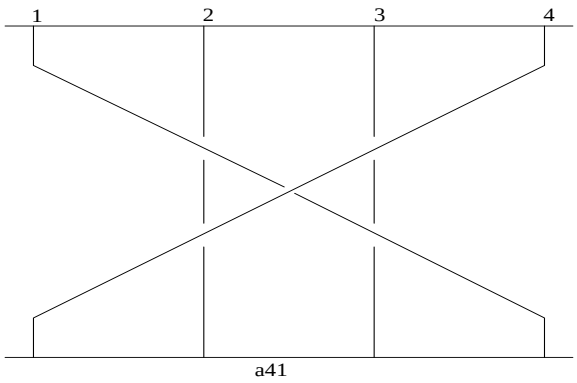
2 つのブレイド U, V が与えられたとき, $W \leq_L U, W \leq_L V$ を満たす最大の元 W を left meet と呼び, $U \wedge_L V$ と表す. 同様に, 関係 $\leq_R$ を用いて right meet も定義でき, $U \wedge_R V$ と表す. また, $U \leq_L W, V \leq_L W$ を満たす最小の元 W を left join と呼び, $U \vee_L V$ と表す. またこれも同様に right join を定義でき, $U \vee_R V$ で表す. ブレイド群において meet は最大公約数, join は最小公倍数と同様の役割を果たす.

2.1.6 Band Generator

Birman らは Artin の生成元に対し, 複数の紐をひとまとめにした band generator [3],[9] と呼ばれる新しい生成元を提案した. $1 \leq s < t \leq n$ のとき, band generator a_{ts} を

$$a_{ts} = (\sigma_{t-1}\sigma_{t-2}\cdots\sigma_{s+1})\sigma_s(\sigma_{s+1}^{-1}\cdots\sigma_{t-2}^{-1}\sigma_{t-1}^{-1}) \tag{2.4}$$

と定義する. これは t 番目の紐と s 番目の紐を他の紐の上で交差させたものである. 例えば, $a_{(i+1)i} = \sigma_i$である.



band generator の例

band generator にも以下のような基本関係式がある.

$$a_{ts} = a_{rq}a_{ts} \quad \text{if} \quad (t-r)(t-q)(s-r)(s-q) > 0 \tag{2.5}$$

$$a_{ts}a_{sr} = a_{tr}a_{ts} = a_{sr}a_{tr} \quad \forall t, s, r \ (1 \leq r < s < t \leq n) \tag{2.6}$$

$$a_{t(t+1)}a_{s(s+1)} = a_{s(s+1)}a_{t(t+1)} \quad \text{if} \quad |t-s| > 1 \tag{2.7}$$

$$a_{t(t+1)}a_{s(s+1)}a_{t(t+1)} = a_{s(s+1)}a_{t(t+1)}a_{s(s+1)} \quad \text{if} \quad |t-s| = 1 \tag{2.8}$$

また，band generator には次のような fundamental word δ と呼ばれるブレイドがある.

$$\delta = a_{n(n-1)}a_{(n-1)(n-2)} \cdots a_{21} = \sigma_{n-1}\sigma_{n-2} \cdots \sigma_2\sigma_1 \tag{2.9}$$

2.2 ブレイドの標準形

fundamental braid の例でもわかるように，ブレイドを生成元σ_iのワードとして書く表し方は一意ではない．そこで, left canonical form という標準形を用いる.

2.2.1 left canonical form の定義

任意の正ブレイド $P \in B_n^+$ に対して，以下の集合が定義される．

$$S(P) = \{i \mid \text{ある } P^{'} \in B_n^+ \text{に対して}, P = \sigma_i P^{'}\} \tag{2.10}$$
$$F(P) = \{i \mid \text{ある } Q^{'} \in B_n^+ \text{に対して}, P = Q^{'}\sigma_i\} \tag{2.11}$$

$S(P), F(P)$ をそれぞれStarting 集合，Finishing 集合という．また，任意の正ブレイドの積 AB が $F(A) \supseteq S(B)$ を満たすとき left weighted という．

定理 1．任意のブレイド $\mathrm{W} \in B_n$ に対して，left canonical form と呼ばれる以下のような表現が一意に存在する．

$$W = \Delta_n^u A_1 A_2 \ldots A_p, \quad u \in \mathbb{Z},\ A_i \in \tilde{\Sigma}_n \diagdown \{e, \Delta_n\} \tag{2.12}$$

ただし，$A_i A_{i+1} (1 \leq i < p)$ は left weighted である．また，上記の p を canonical length，u を infimum，$u + p$ を supremum という [10].

2.2.2 left canonical form 構成アルゴリズム

Starting 集合に関して次の命題が成り立つ．

命題 1．$P \in \tilde{\Sigma}_n$ならば $\forall \sigma_j \in S(P)$ に対して，

$$\sigma_j^{-1} P \in \tilde{\Sigma}_n \tag{2.13}$$

また，Δ_n は次のような性質を持っている．

(a) $1 \leq i \leq n-1$ に対して,$\Delta_n = \sigma_i A_i = B_i \sigma_i$となるような $A_i, B_i \in \tilde{\Sigma}_n$ が存在する．
(b) $1 \leq i \leq n-1$ に対して,$\sigma_i \Delta_n = \Delta_n \sigma_{n-i}$が成り立つ．

上の命題，及び $(a), (b)$ を用いてleft canonical form を構成する．

1. 生成元のワードの形で表された任意のブレイド W において，W がσ_i^{-1} を含んでいる場合，(a) より$\sigma_i^{-1} = \Delta_n^{-1} B_i$を用いて$\sigma_i^{-1}$を置き換える．次に (b) を用いて左にΔ_n^{-1}を集め, $W = \Delta_n^{u^{'}} P$, $P \in B_n^+$を作る．

2-1. 正ブレイド P を左から見ていって, permutation braid の積になるように分解する.

$$P = a_1 a_2 \cdots a_l, \quad a_i \in \tilde{\Sigma}_n$$

2-2
-1. $1 \leq \forall i < l$ に対して, $S(a_{i+1}) - F(a_i) \neq \emptyset$ ならば $\sigma_j \in S(a_{i+1}) - F(a_i)$ を用いて,

$$a_i \leftarrow a_i \sigma_j, \quad a_{i+1} \leftarrow \sigma_j^{-1} a_{i+1}$$

と更新する.

-2. 上の 1 を $S(a_{i+1}) - F(a_i) = \emptyset$ が全ての i について成り立つまで繰り返す.

-3. $a_1 = \Delta_n$のときは, $u' \leftarrow u' + 1, a_1 \leftarrow a_2$と更新する. これを $a_1 \neq \Delta_n$ となるまで繰り返す.

-4. $A_i \leftarrow a_i$を出力する.

2-2 に関するアルゴリズムの詳細については Appendix を参照のこと.

3. 1, 2 を組み合わせることにより, 任意のブレイドを定理 1 の left canonical form に変形することができる.

left canonical form の例

$n = 4, W = \sigma_1 \sigma_3^{-1} \sigma_2 \sigma_1$をleft canonical form に変形する.

1. σ_3^{-1}を$\Delta_4^{-1} B_3$に置き換え,Δ_4^{-1}を左に集める.

$$\begin{aligned} W &= \sigma_1 \Delta_4^{-1} (\sigma_3 \sigma_2 \sigma_1 \sigma_3 \sigma_2) \sigma_2 \sigma_1 \\ &= \Delta_4^{-1} \sigma_3 (\sigma_3 \sigma_2 \sigma_1 \sigma_3 \sigma_2) \sigma_2 \sigma_1 \\ &= \Delta_4^{-1} P \end{aligned}$$

2-1. 正ブレイド $P = \sigma_3 \sigma_3 \sigma_2 \sigma_1 \sigma_3 \sigma_2 \sigma_2 \sigma_1$ を permutation braid の元の積に分解する.

$$\begin{aligned} P &= \sigma_3 \sigma_3 \sigma_2 \sigma_1 \sigma_3 \sigma_2 \sigma_2 \sigma_1 \\ &= (\sigma_3)(\sigma_3 \sigma_2 \sigma_3 \sigma_1 \sigma_2)(\sigma_2 \sigma_1) \\ &= (a_1)(a_2)(a_3) \quad (a_1, a_2, a_3 \in \tilde{\Sigma}_n) \end{aligned}$$

次に上記の $(a_1a_2),(a_2a_3)$ が left weighted になるように分解する．

2-2

1. $F(a_1)=\{3\}, S(a_2)=\{2,3\}$ より，$S(a_2)-F(a_1)=\{2\}$ であるので，
$a_1 \leftarrow (a_1)\sigma_2=\sigma_3\sigma_2,\ a_2 \leftarrow \sigma_2^{-1}(a_2)=\sigma_2^{-1}\sigma_3\sigma_2\sigma_3\sigma_1\sigma_2=\sigma_3\sigma_2\sigma_1\sigma_2$

2. $F(a_1)=\{2\}, S(a_2)=\{1,3\}$ より，$S(a_2)-F(a_1)=\{1,3\}$ であるので，
$a_1 \leftarrow (a_1)\sigma_1=\sigma_3\sigma_2\sigma_1,\ a_2 \leftarrow \sigma_1^{-1}(a_2)=\sigma_1^{-1}\sigma_3\sigma_2\sigma_1\sigma_2=\sigma_3\sigma_2\sigma_1$

3. $F(a_1)=\{1\}, S(a_2)=\{3\}$ より，$S(a_2)-F(a_1)=\{3\}$ であるので，
$a_1 \leftarrow (a_1)\sigma_3=\sigma_3\sigma_2\sigma_1\sigma_3,\ a_2 \leftarrow \sigma_3^{-1}(a_2)=\sigma_3^{-1}\sigma_3\sigma_2\sigma_1=\sigma_2\sigma_1$

4. $F(a_1)=\{1,3\}, S(a_2)=\{2\}$ より，$S(a_2)-F(a_1)=\{2\}$ であるので，
$a_1 \leftarrow (a_1)\sigma_2=\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2,\ a_2 \leftarrow \sigma_2^{-1}(a_2)=\sigma_2^{-1}\sigma_2\sigma_1=\sigma_1$

5. $F(a_1)=\{1,2\}, S(a_2)=\{1\}$ より，$S(a_2)-F(a_1)=\{\ \}$ であるので，
$F(a_1)\supseteq S(a_2)$ が成り立つので，a_1a_2は left weighted である．

6. $F(a_2)=\{1\}, S(a_3)=\{2\}$ より，$S(a_3)-F(a_2)=\{2\}$ であるので，
$a_2 \leftarrow (a_2)\sigma_2=\sigma_1\sigma_2,\ a_3 \leftarrow \sigma_2^{-1}(a_3)=\sigma_2^{-1}\sigma_2\sigma_1=\sigma_1$

7. $F(a_2)=\{2\}, S(a_3)=\{1\}$ より，$S(a_3)-F(a_2)=\{1\}$ であるので，
$a_2 \leftarrow (a_2)\sigma_1=\sigma_1\sigma_2\sigma_1,\ a_3 \leftarrow \sigma_1^{-1}(a_3)=\sigma_1^{-1}\sigma_1=e$

8. $F(a_2)=\{1,2\}, S(a_3)=\{\ \}$ より，$S(a_3)-F(a_2)=\{\ \}$ であるので，
$F(a_2)\supseteq S(a_3)$ が成り立つので，a_2a_3は left weighted である．

9. $F(a_1)=\{1,2\}, S(a_2)=\{1,2\}$ より，$S(a_2)-F(a_1)=\{\ \}$ であるので，
$F(a_1)\supseteq S(a_2)$ が成り立つので，a_1a_2は left weighted である．

上記より，

$$\begin{aligned} P &= (\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2)(\sigma_1\sigma_2\sigma_1)(e) \\ &= (\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2)(\sigma_1\sigma_2\sigma_1) \\ &= A_1A_2 \end{aligned}$$

となる．上記の A_1, A_2において，$F(A_1)\supseteq S(A_2)$ であり，A_1, A_2 は left weighted である．よって，以下の left canonical form を得る．

$$W=\Delta_4^{-1}(\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2)(\sigma_1\sigma_2\sigma_1)$$

上記の left canonical form と同様にして，right canonical form も定義できる．

$$W = A_1 A_2 \dots A_p \Delta_n^u \quad u \in \mathbb{Z},\ A_i \in \tilde{\Sigma}_n \diagdown \{e, \Delta_n\}$$

ここで，A_i, A_{i+1}は $F(A_i) \subseteq S(A_{i+1})$ が成り立ち，right weighted である．

第3章　ブレイド群を利用した公開鍵暗号

この章ではブレイド群の共役問題を紹介し，次にそれを利用した公開鍵暗号について述べる [10].

3.1　共役とは

ブレイド x, y が共役であるとは, $y = axa^{-1}$となるブレイド a が存在するときをいう. 共役に関して，困難であると予想されている計算問題に以下のようなものがある．ここで, B_mを $m < n$ に対して,$\sigma_1, \cdots, \sigma_{m-1}$で生成される B_nの部分群とする.

3.1.1　共役問題

1. 共役決定問題
入力：$(x, y) \in B_n \times B_n$
出力：x と y が共役であるかどうかを決定する.

2. 共役探索問題
入力：$(x, y) \in B_n \times B_n$ （$a \in B_n$が存在して, $y = axa^{-1}$.）
出力：$y = axa^{-1}$を満たす $a \in B_n$を見つけること.

3. 一般共役探索問題
入力：$(x, y) \in B_n \times B_n$ （$m \leq n$ に対して, $y = axa^{-1}$, $a \in B_m$）
出力：$y = axa^{-1}$を満たす, $a \in B_m$を見つけること.

4. 共役分解問題
入力：$(x, y) \in B_n \times B_n$ （$m < n$ に対して, $y = axa^{-1}$, $a \in B_m$）
出力：$y = a^{'}xa^{''}$を満たす $a^{'}$, $a^{''} \in B_m$を見つけること.

次の節以降で具体的に上記の一般共役探索問題を用いた公開鍵暗号について説明する.

3.2 一方向関数

$l, r \leq n$ に対して, B_nの部分群 LB_l, RB_rを次のように定める. LB_lは n 本の紐の中で左側の l 本の紐を, RB_rは右側の r 本の紐を編むことによって構成されるとする. すなわち, LB_lは生成元$\sigma_1, \cdots, \sigma_{l-1}$, RB_rはσ_{n-r+1}, $\cdots, \sigma_{n-1}$から生成される部分群である. $l+r \leq n$ のとき, 任意の $a \in LB_l$, $b \in RB_r$は可換である. 以下 $l+r=n$ とする. 上記の LB_lを用いて以下の一方向関数を定義する.

$$f : LB_l \times B_{l+r} \to B_{l+r} \times B_{l+r}, \qquad f(a,x) = (axa^{-1}, x)$$

(a,x) が与えられたとき, axa^{-1}を計算するのは易しいが, (axa^{-1},x) から a を計算する多項式時間アルゴリズムは知られていない [1]. また, 上記の一方向関数は一般共役探索問題に基づている.

ブレイド群を利用した鍵共有, 公開鍵暗号は以下の問題の計算量的困難性に基づいている.

[Base Problem]
入力：秘密である $a \in LB_l$, $b \in RB_r$に対して, $y_1 = axa^{-1}$, $y_2 = bxb^{-1}$を満たす B_{l+r}の元 (x, y_1, y_2)
出力：$by_1b^{-1} (= ay_2a^{-1} = abxa^{-1}b^{-1})$

Base Problem を計算量的に困難にするためには, 十分に複雑なブレイド $x \in B_{l+r}$を選ばなければならない. ここで, x が十分に複雑であるとは $x_1 \in LB_l$, $x_2 \in RB_r$, $z \in B_{l+r}$であって,「z は a, b, x_1, x_2と可換かつ x が $x = x_1x_2z$ と分解される」ものが存在しないことをいう. もし $x = x_1x_2z$ と分解されたとしたら,

$$\begin{aligned} by_1b^{-1} &= baxa^{-1}b^{-1} \\ &= ab(x_1x_2z)a^{-1}b^{-1} \quad (a, x_1 \in LB_l,\ b, x_2 \in RB_r) \\ &= abx_1x_2a^{-1}b^{-1}z \\ &= (ax_1a^{-1})(bx_2b^{-1})z \end{aligned}$$

ここで,

$$\begin{aligned} y_1 &= axa^{-1} \\ &= a(x_1x_2z)a^{-1} \\ &= (ax_1x_2a^{-1})z \\ &= (ax_1a^{-1})x_2z \end{aligned}$$

$$\begin{aligned} y_2 &= bxb^{-1} \\ &= b(x_1x_2z)b^{-1} \\ &= (bx_1x_2b^{-1})z \\ &= x_1(bx_2b^{-1})z \end{aligned}$$

より, $ax_1a^{-1} = y_1z^{-1}x_2^{-1}$, $bx_2b^{-1} = x_1^{-1}y_2z^{-1}$と分かるので,

$$\begin{aligned} by_1b^{-1} &= (ax_1a^{-1})(bx_2b^{-1})z \\ &= (y_1z^{-1}x_2^{-1})(x_1^{-1}y_2z^{-1})z \end{aligned}$$

となる．これより，a, b を知らなくても by_1b^{-1}が分かってしまう．

Base Problem における x の役割は, g^xと g^yから g^{xy} を求める Diffie-Hellman Problem [8] における g に相当していることから，上記の問題は Diffie-Hellman-Like Conjugacy Problem(DHCP) と呼ばれている．

3.3 鍵共有

A(lice) と B(ob) との間において，以下の鍵共有を定義する．

1. 前処理
適当な整数の組 (l, r)，及び十分に複雑なブレイド $x \in B_{l+r}$を選び公開する．

2. 鍵共有
(a) A はランダムに秘密鍵 $a \in LB_l$ を選び，B に対して $y_1 = axa^{-1}$を送る．
(b) B はランダムに秘密鍵 $b \in RB_r$ を選び，A に対して $y_2 = bxb^{-1}$を送る．
(c) A は y_2 を受け取り，共有鍵 $K = ay_2a^{-1}$を計算する．

(d) B は y_1 を受け取り，共有鍵 $K = by_1b^{-1}$を計算する．

ここで，$a \in LB_l, b \in RB_r$は $ab = ba$ であるので，

$$ay_2a^{-1} = a(bxb^{-1})a^{-1} = b(axa^{-1})b^{-1} = by_1b^{-1}$$

上記より，Alice と Bob は同じブレイドを得ることができる．

3.4 公開鍵暗号

3.3 の 鍵共有の原理を用いて，ブレイド群を利用した公開鍵暗号を定義する．$H : B_{l+r} \to \{0,1\}^k$をブレイド群からメッセージ空間へのハッシュ関数とする．

1. 鍵の生成：
(a) 十分に複雑なブレイド $u \in B_{l+r}$を選ぶ.
(b) $a \in LB_l$を選ぶ
(c) $v = aua^{-1}$を計算し, (u, v) を公開鍵, a を秘密鍵とする．

2. 暗号化：メッセージ $m \in \{0,1\}^k$と公開鍵 (u, v) が与えられているとする．
(a) $b \in RB_r$をランダムに選ぶ.
(b) $w = bub^{-1}$と $d = H(bvb^{-1}) \oplus m$ を計算し，暗号文を (w, d) とする．

3. 復号化：暗号文 (w, d) と秘密鍵 a を用いて, $m = H(awa^{-1}) \oplus d$ を計算する．

a と b は可換であるので, $awa^{-1} = abub^{-1}a^{-1} = baua^{-1}b^{-1} = bvb^{-1}$が成り立ち, $H(awa^{-1}) \oplus d = H(awa^{-1}) \oplus H(bvb^{-1}) \oplus m = m$ となりメッセージを得ることができる．

第4章　Lawrence-Krammer表現を用いた公開鍵暗号に対する攻撃

この章ではブレイド群の線形表現を示し，それを用いた暗号に対する攻撃について説明する．

4.1　ブレイド群の線形表現

ブレイド群の線形表現は幾つかあるが，本論文では Burau 表現 [2] と Lawrence-Krammer [12] 表現の 2 つのみを紹介する．

4.1.1　Burau 表現

Burau 表現は次の写像 ρ_{B}: $B_n \to GL_n$ $(\mathbb{Z}[t, t^{-1}])$ で定義される．

$$\sigma_i \to \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & I_{n-i-1} \end{pmatrix}$$

$n = 4$ のとき，

$$W = \sigma_1\sigma_2 \to \begin{pmatrix} 1-t & t-t^2 & t^2 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Burau 表現は一般的に忠実ではない．また，与えられた Burau 行列に対応するブレイドを返す決定性アルゴリズムは現在のところ存在しない．Burau 表現は $n = 3$ のとき忠実であり，$n \geq 5$ のとき忠実ではない．$n = 4$ のとき，Burau 表現が忠実であるかどうかという問題はまだ明らかではない．

4.1.2 Lawrence-Krammer 表現

Lawrence-Krammer 表現は次の写像 $\rho_{\mathrm{K}}\colon B_n \to GL_{n(n-1)/2}(V)$ によって定義される．ここで，$V = \mathbb{Z}[t^{\pm 1}, q^{\pm 1}]$ である．$\{x_{i,j} | 1 \leq i < j \leq n\}$ を自由加群 V の標準基底とすると，生成元σ_kはLawrence-Krammer 表現のもとで次のように表される．

$$\rho_{\mathrm{k}}(\sigma_k)x_{i,j} = \begin{cases} tq^2 x_{k,k+1} & (i=k, j=k+1) \\ (1-q)x_{i,k} + qx_{i,k+1} & (j=k, i<k) \\ x_{i,k} + tq^{k-i+1}(q-1)x_{k,k+1} & (j=k+1, i<k) \\ tq(q-1)x_{k,k+1} + qx_{k+1,j} & (i=k, k+1<j) \\ x_{k,j} + (1-q)x_{k+1,j} & (i=k+1, k+1<j) \\ x_{i,j} & (i<j<k \ \text{ or } \ k+1<i<j) \\ x_{i,j} + tq^{k-i}(q-1)^2 x_{k,k+1} & (i<k<k+1<j). \end{cases}$$

例えば $n = 4$ のとき,$\rho_{\mathrm{K}}(\sigma_2\sigma_3)$ は

$$\rho_{\mathrm{K}}(\sigma_2\sigma_3) = \begin{pmatrix} 1-q & 1-q & 1 & 0 & 0 & 0 \\ q & 0 & 0 & 0 & 0 & 0 \\ 0 & q & 0 & 0 & 0 & 0 \\ 0 & 0 & -q^2t + q^3t & 0 & q^2t & 0 \\ 0 & 0 & -q^3t + q^4t & 0 & -q^2t + q^3t & q^2t \\ 0 & 0 & -q^3t + 2q^4t - q^5t & q^2 & -q^2t + 2q^3t - q^4t & q^2t - q^3t \end{pmatrix}$$

となる．Lawrence-Krammer 表現は任意の自然数 n に対して忠実である．次の節で上記の表現を用いた公開鍵暗号に対する攻撃について述べる．

4.2 攻撃アルゴリズム

Cheon と Jun は Lawrence-Krammer 表現を用いた暗号を攻撃する多項式時間アルゴリズムを提案した．このアルゴリズムでは暗号化と鍵共有に対して攻撃を行うが，直接的には共役問題を解かない．一般的に，現在においても共役問題を解く多項式時間アルゴリズムは知られていない．まず，アルゴリズムの大まかな流れを記述する [7],[11].

上記の暗号ではブレイド群の共役問題の困難性を利用しており，与えられた $u, v = aua^{-1}, w = bub^{-1} \in B_n (a \in LB_n, b \in RB_n)$ から $baua^{-1}b^{-1}$ を見つけだせれば解読できる.Cheon らは Lawrence-Krammer 表現を用いて，次のような暗号に対する攻撃を提案した.

まず，上記のアルゴリズムにおいてLawrence-Krammer表現はアルゴリズムの計算量を減らすために$q = 1/2$（qを$0 < q < 1$を満たす実数としてもLawrence-Krammer表現は忠実である[12].）とし，$\rho_{\mathrm{k}}^{'}$と表す．$\rho_{\mathrm{k}}^{'}$におけるa, b, u, v, wの像をそれぞれA, B, U, V, Wとすると，$V = AUA^{-1}$から$VA = AU$となり，以下のAに関する連立方程式を得る．

$$VA = AU \tag{4.1}$$

$$A\rho_{\mathrm{k}}^{'}(\sigma_i) = \rho_{\mathrm{k}}^{'}(\sigma_i)A \qquad (l+1 \leq i \leq n-1) \tag{4.2}$$

上記の連立方程式の正則な解を$A^{'}$とする．この$A^{'}$を用いて，次のように$\rho_{\mathrm{k}}^{'}(baua^{-1}b^{-1})$を計算する．

$$\begin{aligned} A^{'}WA^{'-1} &= A^{'}BUB^{-1}A^{'-1} = BA^{'}UA^{'-1}B^{-1} \\ &= BVB^{-1} = \rho_{\mathrm{k}}^{'}(baua^{-1}b^{-1}) \ \in \rho_{\mathrm{k}}^{'}(B_n) \end{aligned} \tag{4.3}$$

ここで，$A^{'} \in \rho_{\mathrm{k}}^{'}(B_n)$とは限らないが，$A^{'}$は暗号における秘密鍵$a$と同じ役割をはたしている．暗号に対する攻撃アルゴリズムは次のようにまとめられる．

Algorithm 1: **DHCPを解くアルゴリズム**
Input: $u, v = aua^{-1}, w = bub^{-1} \in B_n (a \in LB_n, b \in RB_n)$
Output: $baua^{-1}b^{-1}$.

---

1. $q = 1/2$と選び，$\rho_{\mathrm{k}}^{'}$とする．$\rho_{\mathrm{k}}^{'}(u), \rho_{\mathrm{k}}^{'}(v), \rho_{\mathrm{k}}^{'}(w)$を計算する．

2. 連立方程式$VA = AU,\ A\rho_{\mathrm{k}}^{'}(\sigma_i) = \rho_{\mathrm{k}}^{'}(\sigma_i)A\ (l+1 \leq i \leq n-1)$を解く．

3. Aが正則行列であれば，A^{-1}を計算する．そうでなければ，上のステップに戻って別の解を求める．

4. $A\rho_{\mathrm{k}}^{'}(w)A^{-1} = \rho_{\mathrm{k}}^{'}(baua^{-1}b^{-1})$を計算する．

5. Lawrence-Krammer表現に対応するブレイドを返す(ρ_{k}^{-1})アルゴリズムを用いて$baua^{-1}b^{-1}$を出力する．

上記のアルゴリズムにおける連立方程式を解く際，Cheonらはさまざまな高速化を行っているが，本論文においてはMathematicaのSolve命令を使用したのみで，特別な高速化は行っていない．また，5におけるρ_{k}^{-1}を求めるアルゴリズムについては，次の節で述べる．

4.3 ρ_{k}^{-1} を計算するアルゴリズム

cheon らは [12] に基づいて逆写像 ρ_{k}^{-1}: $GL_{n(n-1)/2}(\mathbb{Z}[t^{\pm 1}, q^{\pm 1}]) \to B_n$ を求めるアルゴリズムを提案した．しかし，幾つかの誤りとあいまいな部分があったため，本論文においては Lukkarila の修正したアルゴリズム [13] について扱う．まずアルゴリズムについて述べる．

Algorithm 2: ρ_{k}^{-1} **を計算するアルゴリズム**.
Input: Krammer 行列 $\rho_{\mathrm{k}}(W) \in GL_{n(n-1)/2}(\mathbb{Z}[t^{\pm 1}, q^{\pm 1}])$.
Output: W の left canonical from.

---

1. $d_t \leftarrow$ 行列$\rho_{\mathrm{k}}(W)$ における t の最小次数.

2. $\rho_{\mathrm{k}}(W) \leftarrow \rho_{\mathrm{k}}(\Delta_n)^{-d_t}\rho_{\mathrm{k}}(W)$.

3. $l \leftarrow$ 行列$\rho_{\mathrm{k}}(W)$ における t の最大次数.

4. **for** $k = 1$ to l:
   4.1. $\mathbf{v} \leftarrow \rho_{\mathrm{k}}(W) \cdot (1)_{1\times n(n-1)/2}$.
   4.2. $A \leftarrow \{(i,j) |\ (\mathbf{v}, x_{i,j}) \in t\mathbb{R}[t])\}$.
   4.3. $A_k \leftarrow \mathrm{GB}(A)$.
   4.4. $\rho_{\mathrm{k}}(W) \leftarrow \rho_{\mathrm{k}}(A_k)^{-1}\rho_{\mathrm{k}}(W)$.

5. **return** $\Delta_n^{d_t} A_1 \cdots A_l$.

上記のアルゴリズムにおける $(\mathbf{v}, x_{i,j})$ は $\mathbf{v}$ と基底 $x_{i,j}$の内積 をとることを意味しており，4.2 では定数項を持たない t の多項式になっている行のみを集めることを意味している．また,4.3 の GB（Greatest Braid）は [12] から引用しており，以下の通りである．

$s_{ij} \in \Sigma_n$ $(1 \le i < j \le n)$ を i と j のみを交換し，残りは移動させない置換とする．例えば,

$$s_{14} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

である．また，Ref で s_{ij}の集合 $\mathrm{Ref} = \{s_{ij} \mid 1 \le i < j \le n\}$ を表す.Ref の部分集合

$$\mathrm{HP} = \left\{ A \subset \mathrm{Ref} \;\middle|\; s_{ij}, s_{jk} \in A \Rightarrow s_{ik} \in A \quad 1 \le i < j < k \le n \right\} \quad (4.4)$$

に対して,

$$\mathrm{GB} = rL^{-1}\mathrm{Pro}\colon \mathrm{HP} \to \mho \subset B_n^+ \tag{4.5}$$

を次のように定義する.

$$L\colon \Sigma_n \to \mathrm{HP},\ L(x) = \left\{ s_{ij} \;\middle|\; 1 \le i < j \le n,\ x^{-1}i > x^{-1}j \right\},\quad (x \in \Sigma_n) \tag{4.6}$$

$$\mathrm{Pro} : \mathrm{HP} \to \mathrm{HP} \tag{4.7}$$

ただし，$A \in \mathrm{HP}$ に対して, $B = \mathrm{Pro}(A)$ は $B \subset A$ かつ $B \in L(\Sigma_n)$ となる最大の集合である.

$$r\colon \Sigma_n \to B_n^+,\quad \text{また}\quad \mho = \ r(\Sigma_n) \tag{4.8}$$

例えば,

$$r(s_{14}) = \sigma_1\sigma_2\sigma_3\sigma_2\sigma_1$$

である.

上記より，$L(x) \subset \mathrm{Ref} \subset \Sigma_n$の関係が成り立ち, GB は HP から B_n^+への写像である.

ρ_{k}^{-1} の計算例

$W = \sigma_2\sigma_3 \in B_4$とし, $\rho_{\mathrm{k}}^{-1}(W)$ を計算する.

$$\rho_{\mathrm{k}}(W) = \begin{pmatrix} 1-q & 1-q & 1 & 0 & 0 & 0 \\ q & 0 & 0 & 0 & 0 & 0 \\ 0 & q & 0 & 0 & 0 & 0 \\ 0 & 0 & -q^2t + q^3t & 0 & q^2t & 0 \\ 0 & 0 & -q^3t + q^4t & 0 & -q^2t + q^3t & q^2t \\ 0 & 0 & -q^3t + 2q^4t - q^5t & q^2 & -q^2t + 2q^3t - q^4t & q^2t - q^3t \end{pmatrix}$$

1. $d_t \leftarrow$ 上記の行列の t の最小次数 0.

2. $\rho_{\mathrm{k}}(W) \leftarrow \rho_{\mathrm{k}}(\Delta_4)^0\rho_{\mathrm{k}}(W) = \rho_{\mathrm{k}}(W)$.

3. $l \ \leftarrow$ 上記の行列の t の最大次数 1.

4. **for ループ**:

4.1. $\mathbf{v} \leftarrow \rho_{\mathrm{k}}(W) \cdot (1)_{1\times 6}$

$$= \begin{pmatrix} 3-2q \\ q \\ q \\ q^3t \\ q^4t \\ q^2+q^4t-q^5t \end{pmatrix}$$

4.2. $(\mathbf{v}, x_{i,j})$

$$= \begin{cases} 3-2q & (i,j)=(1,2) \\ q & (i,j)=(1,3) \\ q & (i,j)=(1,4) \\ q^3t & (i,j)=(2,3) \\ q^4t & (i,j)=(2,4) \\ q^2+q^4t-q^5t & (i,j)=(3,4) \end{cases}$$

$A \leftarrow \{(i,j)|\ (\mathbf{v}, x_{i,j}) \in t\mathbb{R}[t])\} = \{(2,3),(2,4)\}.$

4.3. $A = \{s_{23}, s_{24}\} \in \mathrm{HP}$

$$x = \begin{pmatrix} 1\ 2\ 3\ 4 \\ 1\ 3\ 4\ 2 \end{pmatrix} \in \Sigma_n$$

とすると， $L(x) = \{s_{23}, s_{24}\} \subseteqq A$

$$\therefore\ x = L^{-1}\mathrm{Pro}(A)$$

$A_1 \longleftarrow \mathrm{GB} = r(x) = \sigma_2\sigma_3.$

4.4. $\rho_{\mathrm{k}}(W) \leftarrow \rho_{\mathrm{k}}(A_1)^{-1}\rho_{\mathrm{k}}(W)$

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

5. **return** $\Delta_4^0 A_1 = \sigma_2\sigma_3$.

上記より，$\rho_{\mathrm{k}}^{-1}(W) = \sigma_2\sigma_3$を得る. 上記のアルゴリズムは $q = 1/2$ としても同様に動作する.

4.4 暗号への攻撃

この章では上記のアルゴリズムを用いて，暗号への攻撃例を示す．例えば，$u = \sigma_3\sigma_2\sigma_1$, $v = aua^{-1} = \sigma_1\sigma_3\sigma_2\sigma_1\sigma_1^{-1}$, $w = bub^{-1} = \sigma_3\sigma_3\sigma_2\sigma_1\sigma_3^{-1}$ を知っているとする.

1. $\rho_{\mathrm{k}}^{'}(u), \rho_{\mathrm{k}}^{'}(v), \rho_{\mathrm{k}}^{'}(w)$ を計算する.

2. 連立方程式 $VA = AU,\ A\rho_{\mathrm{k}}^{'}(\sigma_3) = \rho_{\mathrm{k}}^{'}(\sigma_3)A$ を解く．ここで,

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{pmatrix}$$

とする.

3. 上の連立方程式を解くと,

$$A = \begin{pmatrix} \frac{t}{4}a_{35} & -\frac{t}{4}a_{35} & -\frac{t}{4}a_{35} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{35} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{35} & 0 \\ 0 & \frac{1}{2}a_{35} & 0 & \frac{1}{2}a_{35} & 0 & 0 \\ 0 & 0 & \frac{1}{2}a_{35} & 0 & \frac{1}{2}a_{35} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{35} \end{pmatrix}$$

を得る．この A は正則行列であるので, A^{-1}を計算する.

4. $A\rho_{\mathrm{k}}^{'}(w)A^{-1} = \rho_{\mathrm{k}}^{'}(baua^{-1}b^{-1})$

$$= \begin{pmatrix} 0 & -\frac{t}{4} & \frac{3t}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{8}+\frac{3t}{32} & -\frac{1}{8}-\frac{9t}{32} & \frac{1}{4}-\frac{3t}{16} & -\frac{1}{4}+\frac{9t}{16} & \frac{2}{t} \\ 0 & \frac{1}{8}+\frac{t}{32} & -\frac{1}{8}-\frac{3t}{32} & \frac{1}{4}-\frac{t}{16} & -\frac{1}{4}+\frac{3t}{16} & \frac{2}{t} \\ \frac{3}{16} & \frac{5}{16}+\frac{3t}{64} & -\frac{5}{16}-\frac{9t}{64} & \frac{1}{8}-\frac{3t}{32} & -\frac{1}{8}+\frac{9t}{32} & \frac{1}{t} \\ \frac{1}{16} & \frac{5}{16}+\frac{t}{64} & -\frac{5}{16}-\frac{3t}{64} & \frac{1}{8}-\frac{t}{32} & -\frac{1}{8}+\frac{3t}{32} & \frac{1}{t} \\ -\frac{t}{64} & -\frac{3t^2}{128} & \frac{5t^2}{128} & \frac{3t^2}{64} & -\frac{5t^2}{64} & 0 \end{pmatrix}$$

を得る.

5. **Algorithm 2** を用いて，4 で計算した $\rho_{\mathrm{k}}^{'}(baua^{-1}b^{-1})$ から $baua^{-1}b^{-1}$ を求める.

$$\begin{aligned} baua^{-1}b^{-1} &= \Delta_4^{-1}A_1A_2A_3 \\ &= \Delta_4^{-1}(\sigma_3\sigma_1)(\sigma_1\sigma_2\sigma_3\sigma_2\sigma_1)(\sigma_3\sigma_2). \end{aligned}$$

上記より，暗号への攻撃が成功した.

第5章　プログラムの仕様

Lukkarila の Mathematica パッケージ [14] などをもとにブレイドの積, left canonical form などの公開鍵暗号を構築するのに必要なアルゴリズムを Python で実装し [4],[5],[6]，4 章で述べた Lawrence-Krammer 表現を用いた公開鍵暗号に対する攻撃を Mathematica で実装した [7],[13].

5.1　Python パッケージの使用方法

Python がインストールされているとし，作成したプログラムの使用方法について述べる．まず，Python を起動した後,

```
>>> import Braid
>>> Braid.Artin_Braid(4).Delta()
[1, 2, 3, 1, 2, 1]
```

と入力することで Δ_4 を返すプログラムを起動することができる．Braid.Artin_Braid(n). 関数において，任意の紐の本数を n に入力し，続いて関数を入力することでプログラムを起動することができる．関数については以下を参照.

5.2　Python パッケージ

Python で実装したプログラムの仕様について述べる．プログラムにおいて引数が w のときは word を，p のときは置換を入力することを意味している．例えば $n = 4$ のとき, $W = \sigma_1\sigma_2\sigma_3^{-1}$を引数 w では配列を用いて w = [1, 2, −3] と入力し，置換では同じく配列を用いて p = [2, 3, 4, 1] と入力する.

Word Operations

- Braid_As_WordQ[w]: Braid_As_WordQ は w が word を表しているときに True を，そうでなければ False を返す.

- Positive_Braid_As_WordQ[w]: Positive_Braid_As_WordQ[w] はwが正ブレイド word を表しているときに True を，そうでなければ False を返す.
- Freely_Reduce_Word[w]: Freely_Reduce_Word[w] はw = [a, $\cdots$, i, $-$ i, $\cdots$, b] のようなとなりあう要素が打ち消しあうことができる場合w = [a $\cdots$ b] を返す. 打ち消しあう要素がない場合はそのままwを返す.
- Delta[ ]： Delta[n] は word としての Δ_nを返す.
- Word_To_Permutation[w]: Word_To_Permutation は w に対応する置換を返す.
- Product_Word[w1,w2]: Product_Word は w1$*$w2 を返す.
- Inverse_Word[w]: Inverse_Word は w^{-1} を返す.
- Reverse_Word[w]: Reverse_Word はwの reverse である $\mathrm{w_r}$ を返す.
- Negation_Word[w]: Negation_Word は $\mathrm{w_r}^{-1}$ を返す.
- Half_Twist_Word[w]: Half_Twist_Word は $\tau(\mathrm{w}) = \Delta_n^{-1}\mathrm{w}\Delta_n$ を返す.
- Word_Left_Meet[p1,p2]: Word_Left_Meet は入力した置換p1,p2の p1 $\wedge_L$ p2 を word として返す.
- Word_Right_Meet[p1,p2]: Word_Right_Meet は入力した置換p1,p2の p1 $\wedge_R$ p2 を word として返す.
- Word_Left_Join[p1,p2]: Word_Left_Join は入力した置換 p1,p2 の p1 $\vee_L$ p2 を word として返す.
- Word_Right_Join[p1,p2]: Word_Right_Join は入力した置換 p1,p2 の p1 $\vee_R$ p2 を word として返す.
- Word_Random_Braid[ ]: Word_Random_Braid はランダムなw $\in B_n$ を返す.
- Word_To_Left_Canonical_Form[w]: Word_To_Left_Canonical_Form はwの left canonical form を返す. 戻り値は [infimum,W(word のリスト)] のリストとする. ここで，[infimum,W] $= \Delta_n^{\mathrm{infimum}}$W $= \Delta_n^{\mathrm{infimum}}$ w1$*$w2$*\cdots$ を表している.

- Word_To_Right_Canonical_Form[w]: Word_To_Right_Canonical_Form は w の right canonical form を返す．戻り値は [W(word のリスト),infimum] のリストとする．ここで，[W,infimum]=W$\Delta_n^{\text{infimum}}$ = w1∗w2∗$\cdots\Delta_n^{\text{infimum}}$ を表している．

- Word_Braid_Comparison[w1,w2]: Word_Braid_Comparison は w1, w2 が同じブレイドを表している場合は True，そうでなければ False を返す．

- Word_Commute_Left[w,s,d]: Word_Commute_Left は w の s 番目の要素を群の同値関係を使って d 番目に移動させる．d 番目まで移動できない場合は，移動できるところまで要素を移動させる．最終的に [s 番目の要素の現在位置, 要素を移動させた後の w] のリストを返す．(1≤ s,d ≤ n かつ $s \geq d$)

- Word_Commute_Right[w,s,d]: Word_Commute_Right は w の s 番目の要素を群の同値関係を使って d 番目に移動させる．d 番目まで移動できない場合は，移動できるところまで要素を移動させる．最終的に [s 番目の要素の現在位置, 要素を移動させた後の w] のリストを返す．(1≤ s,d ≤ n かつ $s \leq d$)

Permutation Operations

- Identity_Permutation[ ]: Identity_Permutation は単位元 e に対応する置換を返す．

- Omega[ ]: Omega は Δ_nに対応する置換Ω_nを返す．

- Braid_As_PermutationQ[p]: Braid_As_PermutationQ は p が置換を表しているときに True を，そうでなければ False を返す．

- Half_Twist_Permutation[p]: Half_Twist_Permutation は p を Ω_n で共役したものを返す．

- Transposition[i]: Transposition は互換 $(i, i+1) \in \Sigma_n$ を返す．

- Permutation_Product[p1,p2]: Permutation_Product は p1,p2 の置換の積を返す．

- Permutation_To_Word[p]: Permutation_To_Word は p に対応する word を返す．

- Permutation_Inverse[p]: Permutation_Inverse は p の逆置換を返す．

- Starting_Set_Permutation[p]: Starting_Set_Permutation はpのStarting集合を返す．ここで，pは正ブレイドを仮定している．
- Finishing_Set_Permutation[p]: Finishing_Set_Permutation はpのFinishing集合を返す．ここで，pは正ブレイドを仮定している．
- Left_Weighted[P]: P=[p1,p2, $\cdots$] は置換 p1,p2 $\cdots$ のリストとする．Left_Weighted は p1,p2 $\cdots$ が left weighted である場合 True, そうでなければ False を返す．
- Right_Weighted[P]: P=[p1,p2, $\cdots$] は置換 p1,p2 $\cdots$ のリストとする．Right_Weighted は p1,p2 $\cdots$ が Right weighted である場合 True，そうでなければ False を返す．
- Permutation_Length[p]: Permutation_Length はpに対応する生成元 σ_i の最少数を返す．
- Permutation_Left_Meet[p1,p2]: Permutation_Left_Meet は入力した置換 p1,p2 の p1 $\wedge_L$ p2 を置換として返す．
- Permutation_Right_Meet[p1,p2]: Permutation_Right_Meet は入力した置換 p1,p2 の p1 $\wedge_R$ p2 を置換として返す．
- Permutation_Left_Join[p1,p2]: Permutation_Left_Join は入力した置換 p1,p2 の p1 $\vee_L$ p2 を置換として返す．
- Permutation_Right_Join[p1,p2]: Permutation_Right_Join は入力した置換 p1,p2 の p1 $\vee_R$ p2 を置換として返す．
- Maximal_Head[p1,p2]: Maximal_Head は p1,p2 の maximal Head p1$((\Omega_n \mathrm{p1}^{-1}) \wedge_L$ p2) を返す．
- Permutation_To_Left_Canonical_Form[inf,P]: Permutation_To_Left_Canonical_Form は $\Delta_n^{\inf}$P= $\Delta_n^{\inf}$p1$*$p2$*\cdots$ の left canonical form を返す．戻り値は [infimum,P] のリストとする．ここで，P=[p1,p2,$\cdots$] は置換のリストを表している．
- Permutation_To_Right_Canonical_Form[inf,P]: Permutation_To_Right_Canonical_Form は P$\Delta_n^{\inf}$ = p1$*$p2$*\cdots\Delta_n^{\inf}$ の Right canonical form を返す．戻り値は [P,infimum] のリストとする．ここで，P=[p1,p2,$\cdots$] は置換のリストを表している．
- Permutation_Left_Canonical_Form_Product[LCF1,LCF2]: Permutation_Left_Canonical_Form_Product は LCF1,LCF2 の積を返す．

ただし，LCF1,LCF2 には left canonical form(LCF=[inf,p1,p2,$\cdots$]) を入力する.

- Permutation_Left_Canonical_Form_Inverse[LCF]: Permutation_Left_Canonical_Form_Inverse は LCF^{-1} を返す．ただし，LCF には left canonical form(LCF=[inf,p1,p2,$\cdots$]) を入力する.

- Permutation_Braid_Comparison[P1,P2]: Permutation_Braid_Comparison は入力された置換のリスト P1,P2 が同じブレイドを表している場合は True，そうでなければ False を返す.

- Permutation_Random_Braid[ ]: Permutation_Random_Braid はランダムな置換 p $\in \Sigma_n$ を返す.

Braid Operations

- Infimum_Left_Canonical_Form[w]: Infimum_Left_Canonical_Form は w の left canonical form における infimum を返す.

- Canonical_Length_Left_Canonical_Form[w]: Canonical_Length_Left_Canonical_Form は w の left canonical form における canonical length を返す.

- Supremum_Left_Canonical_Form[w]: Supremum_Left_Canonical_Form は w の left canonical form における supremum を返す.

- Infimum_Right_Canonical_Form[w]: Infimum_Right_Canonical_Form は w の right canonical form における infimum を返す.

- Canonical_Length_Right_Canonical_Form[w]: Canonical_Length_Right_Canonical_Form は w の right canonical form における canonical length を返す.

- Supremum_Right_Canonical_Form[w]: Supremum_Right_Canonical_Form は w の right canonical form における supremum を返す.

- Cycling[W_LCF]: Cycling は W_LCF= $\Delta_n^u x_1 x_2 \cdots x_k$ が left canonical form のとき，cycling c(W_LCF) = $\Delta_n^u x_2 \cdots x_k \tau^u(x_1)$ を返す.

- Decycling[W_LCF]: Decycling は W_LCF= $\Delta_n^u x_1 x_2 \cdots x_k$ が left canonical form のとき，decycling d(W_LCF) = $\Delta_n^u \tau^u(x_k) x_1 \cdots x_{k-1}$ を返す.

5.3 Mathematica パッケージ

今回，実装した暗号系に対する攻撃のプログラムは Lukkarila の Mathematica パッケージ [14] の関数を利用して作成した．プログラムの詳細については以下を参照.

- LKRAttack[u,v,w,index,n]: LKRAttack は $a \in LB_n$, $b \in RB_n$, $u \in B_n$ のとき，u, $v = aua^{-1}$, $w = bub^{-1}$ から $baua^{-1}b^{-1}$ を計算し，戻り値とする．また，index は (4.2) の $A\rho_{\mathrm{k}}^{'}(\sigma_i) = \rho_{\mathrm{k}}^{'}(\sigma_i)A$ $(l+1 \leq i \leq n-1)$ の i に相当する.

第6章　考察，及び結論

本研究を通して，ブレイド群の定義や共役問題を用いた公開鍵暗号について整理し，まとめることができた．また，Python という無料かつ誰にでも簡単に習得できる言語でそれらを実装したことは価値が高い．一方，攻撃に関しては既存のアルゴリズムをまとめ，Mathematica ではあるが，実装することができた．

これらを通して得た一考察として，今回の公開鍵暗号における暗号化，及び復合化にはブレイド群からメッセージ空間へのハッシュ関数が使われているが，このハッシュ関数をブレイド群の忠実な表現である Lawrence-Krammer 表現からメッセージ空間への関数に変更することにより，暗号プロトコルを高速化できるのではないかと思われる．

今後の課題としては，まず攻撃のプログラムを無料で使用できる言語で作成するということが挙げられる．その他としては，非常に難しい研究ではあるが，上記の暗号プロトコルにそのまま適用でき，かつ忠実な線形表現を持たない新たな群を見つけることなどが挙げられる．

しかし，今回の暗号に対する攻撃は強力ではあるが，非常に大きな行列を扱うため，計算量や実装に関して幾分問題があり，まだ研究の余地は残っている．

謝辞

下野孝一 教授，竹縄知之 准教授には修士論文を作成するにあたり，多くの有意義な助言を頂きました．本論文を作成できたのも両先生の熱心な御指導があったからであると考えております．最後になりましたが，両先生に厚く御礼を申し上げます．

Appendix

Python Program

```
# -*- coding: cp932 -*-

"""
A python-package for braid groups.

start date:   2007 7/5 (Thursday)

last update:   2008 2/20 (Wednesday)

author:  Arihiro_Maeda
"""

############################################################
#                                                          #
# This is a class for braid_groups. (Artin_Generator)      #
#                                                          #
############################################################

import random
import sys

class Artin_Braid:

    def __init__(self,n):
        """
        n is the braid index. (n > 0)
        you must input natural number.
        """
        if n <= 0:
```

```
        print "Braid_Index_Error"
        while n <= 0:
            n = input("input: n = ")
    self.n = n

"""
Word Operations
"""

def Braid_As_WordQ(self,w):
    """
    You must input "Word" into w.
    """
    for i in w:
        if abs(i) >= self.n or i == 0:
            return False
    return True

def Positive_Braid_As_WordQ(self,w):
    """
    You must input "Word" into w.
    """
    for i in w:
        if i <= 0 or i >= self.n:
            return False
    return True

def Freely_Reduce_Word(self,w):
    """
    You must input "Word" into w.
    """
    if len(w) == 0:
        return w
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    i = 0
    while (i >= 0 and i < len(w)-1):
```

```
            if w[i] == -w[i+1]:
                del w[i]
                del w[i]
                i = i-1
            i = i+1
        return w

    def Delta(self):
        a=[]
        n = self.n
        i = 1
        while n >= 1:
            if i < n:
                a.append(i)
                i = i+1
            else:
                i = 1
                n = n-1
        return a

    def Word_To_Permutation(self,w):
        """
        You must input "Word" into w.
        """
        if self.Braid_As_WordQ(w) == False:
            print "Word_Error"
            sys.exit(0)
        a = self.Identity_Permutation()
        i = 0
        while i < len(w):
            j = w[i]
            if j < 0:
                j = -j
            Temp(a,j-1,j)
            i = i+1
        return a
```

```
def Product_Word(self,w1,w2):
    """
    You must input "Word" into w1 and w2.
    """
    if (self.Braid_As_WordQ(w1) == False) or \
     (self.Braid_As_WordQ(w2) == False):
        print "Word_Error"
        sys.exit(0)
    #W = []
    #for i in w1:
    #    W.append(["σ",i])
    #
    for i in w2:
        w1.append(i)
        #W.append(["σ",i])
    # print W
    return w1

def Inverse_Word(self,w):
    """
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    i = len(w)-1
    #r = []
    p = []
    while i >= 0:
        j = -w[i]
        p.append(j)
        #r.append(["σ",j])
        i = i-1
    #print r
    return p

def Reverse_Word(self,w):
    """
```

```
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    i = 0
    n = len(w)
    W = []
    while i < n:
        W.append(w.pop())
        i = i + 1
    return W

def Negation_Word(self,w):
    """
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    i = 0
    while i < len(w):
        w[i] = -w[i]
        i = i+1
    return w

def Half_Twist_Word(self,w):
    """
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    p = []
    for i in w:
        if i > 0:
            p.append(self.n-i)
        else:
```

```
            p.append(-self.n-i)
    return p

def Word_Left_Meet(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    p3 = self.Permutation_Left_Meet(p1,p2)
    w = self.Permutation_To_Word(p3)
    return w

def Word_Right_Meet(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    p3 = self.Permutation_Right_Meet(p1,p2)
    w = self.Permutation_To_Word(p3)
    return w

def Word_Left_Join(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    p3 = self.Permutation_Left_Join(p1,p2)
    w = self.Permutation_To_Word(p3)
    return w

def Word_Right_Join(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    p3 = self.Permutation_Right_Join(p1,p2)
    w = self.Permutation_To_Word(p3)
    return w

def Word_Random_Braid(self):
    p = self.Permutation_Random_Braid()
    w = self.Permutation_To_Word(p)
```

```
    i = 0
    while i < len(w):
        if (random.randint(1,2)%2) == 1:
            w[i] = -w[i]
        i = i+1
    return w

def Word_To_Left_Canonical_Form(self,w):
    """
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    pr = Word_To_Permutation_Representation(w,self.n)
    W_LCF = Permutation_Representation_\
    To_Left_Canonical_Form(pr,self.n)
    i = 1
    while i < len(W_LCF):
        W_LCF[i] = self.Permutation_To_Word(W_LCF[i])
        i = i+1
    return W_LCF

def Word_To_Right_Canonical_Form(self,w):
    """
    You must input "Word" into w.
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    pr = Word_To_Permutation_\
    Representation(w,self.n)
    lcf = Permutation_Representation_To_Left\
    _Canonical_Form(pr,self.n)
    rcf = []
    inf = lcf[0]
    del lcf[0]
    p = lcf
```

```
        if (inf % 2) == 1:
            i = 0
            while i < len(p):
                p[i] = self.Half_Twist_Permutation(p[i])
                i = i+1
        p.reverse()
        rev = []
        i = 0
        while i < len(p):
            rev.append(self.Permutation_Inverse(p[i]))
            i = i+1
        rev.insert(0,0)
        rev = Permutation_Representation_To_Left_\
        Canonical_Form(rev,self.n)
        del rev[0]
        rev.reverse()
        i = 0
        while i < len(rev):
            rev[i] = self.Permutation_Inverse(rev[i])
            i = i+1
        p = rev
        p.append(inf)
        W_LCF = p
        i = 0
        while i < len(W_LCF)-1:
            W_LCF[i] = self.Permutation_To_Word(W_LCF[i])
            i = i+1
        return W_LCF

    def Word_Braid_Comparison(self,w1,w2):
        """
        You must input "Word" into w1 and w2.
        """
        if (self.Braid_As_WordQ(w1) == False) or \
        (self.Braid_As_WordQ(w2) == False):
            print "Word_Error"
            sys.exit(0)
        pr = Word_To_Permutation_Representation(w1,self.n)
```

```
    w1 = Permutation_Representation_To_Left_\
    Canonical_Form(pr,self.n)
    pr = Word_To_Permutation_Representation(w2,self.n)
    w2 = Permutation_Representation_To_Left_\
    Canonical_Form(pr,self.n)
    if w1 == w2:
        return True
    else:
        return False

def Word_Commute_Left(self,w,s,d):
    """
    You must input "Word" into w and
    "natural number" into s,d (s >= d).
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    if (s < d) or (s > len(w)) or (d <= 0):
        return w
    if self.Positive_Braid_As_WordQ(w) == False:
        return w
    s = s-1
    d = d-1
    while s > d:
        k = s
        if w[s] == w[s-1]:
            i = s+1
            return [[i]]+w
        if abs(w[s]-w[s-1]) > 1:
            Temp(w,s-1,s)
            s = s-1
        else:
            j = s+1
            while j < len(w) and j-2 >= 0:
                if w[j-2] == w[j]:
                    if j != s+1:
                        w[j-2] = w[j-1]
```

```
                        w[j-1] = w[j]
                        w[j] = w[j-2]
                        s = s-1
                        break
                    else:
                        w[s-1] = w[s]
                        w[s] = w[s+1]
                        w[s+1] = w[s-1]
                        s = s-1
                        break
                j = j-1
                if j-(s+1) == -2:
                    i = s+1
                    return [[i]]+w
            j = j-1
            if s == d:
                i = s+1
                return [[i]]+w
            if j-2 >= 0 and w[j-2] == w[j]:
                w[j-2] = w[j-1]
                w[j-1] = w[j]
                w[j] = w[j-2]
                s = s-1
        if k == s:
            break
    i = s+1
    return [[i]]+w

def Word_Commute_Right(self,w,s,d):
    """
    You must input "Word" into w and
    "natural number" into s,d (s <= d).
    """
    if self.Braid_As_WordQ(w) == False:
        print "Word_Error"
        sys.exit(0)
    if (s <= 0) or (d > len(w)) or (s > d):
        return w
```

```
if self.Positive_Braid_As_WordQ(w) == False:
    return w
s = s-1
d = d-1
while s < d:
    k = s
    if w[s] == w[s+1]:
        i = s+1
        return [[i]]+w
    if abs(w[s]-w[s+1]) > 1:
        Temp(w,s,s+1)
        s = s+1
    else:
        j = s-1
        while j >= 0 and j+2 < len(w):
            if (w[j] == w[j+2]):
                if j != s-1:
                    w[j] = w[j+1]
                    w[j+1] = w[j+2]
                    w[j+2] = w[j]
                    s = s+1
                    break
                else:
                    w[s-1] = w[s]
                    w[s] = w[s+1]
                    w[s+1] = w[s-1]
                    s = s+1
                    break
            j = j+1
            if j-(s-1) == 2:
                i = s+1
                return [[i]]+w
        j = j+1
        if s == d:
            i = s+1
            return [[i]]+w
        if j+2 < len(w) and w[j] == w[j+2]:
            w[j] = w[j+1]
```

```
                w[j+1] = w[j+2]
                w[j+2] = w[j]
                s = s+1
        if k == s:
            break
    i = s+1
    return [[i]]+w

"""
Permutation Operations
"""

def Identity_Permutation(self):
    p = range(1,self.n+1)
    return p

def Omega(self):
    p = range(1,self.n+1)
    p.reverse()
    return p

def Braid_As_PermutationQ(self,p):
    """
    You must input "Permutation" into p.
    """
    if len(p) != self.n:
        return False
    I = self.Identity_Permutation()
    for i in p:
        if (self.n-i) < 0 or (self.n-i) >= self.n:
            return False
        I[i-1] = 0
    for i in I:
        if i > 0:
            return False
    return True

def Half_Twist_Permutation(self,p):
```

```
    """
    You must input "Permutation" into p.
    """
    if self.Braid_As_PermutationQ(p) == False:
        print "Permutation_Error"
        sys.exit(0)
    i = 0
    P = []
    while i < self.n:
        P.append(0)
        i = i+1
    i = 0
    while i < self.n:
        P[i] = self.n - p[self.n-(i+1)] + 1
        i = i+1
    return P

def Transposition(self,i):
    """
    You must input "natural number" into i.
    """
    if i <= 0 or i >= self.n:
        print "Index_Error"
        sys.exit(0)
    p = self.Identity_Permutation()
    Temp(p,i-1,i)
    return p

def Permutation_Product(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    if (self.Braid_As_PermutationQ(p1) == False) or \
    (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p3 = range(0,self.n)
    i = 0
```

```
        while i < self.n:
            t = p2[i]-1
            t = p1[t]
            p3[i] = t
            i = i + 1
        return p3

    def Permutation_To_Word(self,p):
        """
        You must input "Permutation" into p.
        """
        if self.Braid_As_PermutationQ(p) == False:
            print "Permutation_Error"
            sys.exit(0)
        w = []
        q = self.Identity_Permutation()
        while p != q:
            i = 0
            while i < len(p)-1:
                if p[i] > p[i+1]:
                    w.append(i+1)
                    Temp(p,i,i+1)
                    continue
                i = i+1
        w = self.Reverse_Word(w)
        return w

    def Permutation_Inverse(self,p):
        """
        You must input "Permutation" into p.
        """
        if self.Braid_As_PermutationQ(p) == False:
            print "Permutation_Error"
            sys.exit(0)
        p.insert(0,0)
        q = range(0,self.n+1)
        i = 1
        while i < self.n + 1:
```

```
            q[p[i]] = i
            i = i + 1
        del p[0]
        del q[0]
        return q

    def Starting_Set_Permutation(self,p):
        """
        You must input "Permutation" into p.
        """
        if self.Braid_As_PermutationQ(p) == False:
            print "Permutation_Error"
            sys.exit(0)
        S = self.Finishing_Set_Permutation\
        (self.Permutation_Inverse(p))
        return S

    def Finishing_Set_Permutation(self,p):
        """
        You must input "Permutation" into p.
        """
        if self.Braid_As_PermutationQ(p) == False:
            print "Permutation_Error"
            sys.exit(0)
        all_word = range(1,self.n)
        S = []
        for i in all_word:
            if Is_LeftElement_Big(p,i-1) == True:
                S.append(i)
        return S

    def Left_Weighted(self,P):
        """
        You must input "Permutation List" into P.
        """
        for i in P:
            if self.Braid_As_PermutationQ(i) == False:
                print "Permutation_Error"
```

```
            sys.exit(0)
        i = 0
        while i < len(P)-1:
            S1 = self.Starting_Set_Permutation(P[i+1])
            S2 = self.Finishing_Set_Permutation(P[i])
            for j in S2:
                k = 0
                while (len(S1) > 0) and (k < len(S1)):
                    if S1[k] == j:
                        del S1[k]
                    k = k+1
            if len(S1) != 0:
                return False
            i = i+1
        return True

    def Right_Weighted(self,P):
        """
        You must input "Permutation List" into P.
        """
        for i in P:
            if self.Braid_As_PermutationQ(i) == False:
                print "Permutation_Error"
                sys.exit(0)
        i = 0
        while i < len(P)-1:
            S1 = self.Starting_Set_Permutation(P[i+1])
            S2 = self.Finishing_Set_Permutation(P[i])
            for j in S1:
                k = 0
                while (len(S2) > 0) and (k < len(S2)):
                    if S2[k] == j:
                        del S2[k]
                    k = k+1
            if len(S2) != 0:
                return False
            i = i+1
        return True
```

```
def Permutation_Length(self,p):
    """
    You must input "Permutation" into p.
    """
    if self.Braid_As_PermutationQ(p) == False:
        print "Permutation_Error"
        sys.exit(0)
    n = len(self.Permutation_To_Word(p))
    return n

# The following algorithm is taken straight from
# "Efficient implementation of braid groups"
# (J.C.Cha et al) and "Braid group Cryptography
# untangled" (A.Bolstad).

"""
def Convert_Descending_To_Permutation(self,x):
    a = range(0,(len(x)+1))
    z = [0]*(self.n+1)
    i = 1
    i = 1
    x.insert(0,0)
    while i <= len(x)-1:
        if z[x[i]] == 0:
            a[i] = i
        else:
            a[i] = a[z[x[i]]]
            a[z[x[i]]] = i
        z[x[i]] = i
        i += 1
    del a[0]
    del x[0]
    return a
"""

def Permutation_Right_Meet(self,p1,p2):
    """
```

```
    You must input "Permutation" into p1 and p2.
    """
    if (self.Braid_As_PermutationQ(p1) == False) \
    or (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p1.insert(0,0)
    p2.insert(0,0)
    self.c = range(0,self.n+1)
    Meet_Sub(p1,p2,self.c,1,self.n,self.n)
    i = 1
    r = (self.n+1)*[0]
    while i < self.n+1:
        r[self.c[i]] = i
        i = i+1
    del p1[0]
    del p2[0]
    del r[0]
    return r

def Permutation_Left_Meet(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    if (self.Braid_As_PermutationQ(p1) == False) or\
    (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p1.insert(0,0)
    p2.insert(0,0)
    u = range(0,self.n+1)
    v = range(0,self.n+1)
    i = 1
    while i < len(u):
        u[p1[i]] = i
        v[p2[i]] = i
        i = i + 1
    self.c = range(0,self.n+1)
```

```
    Meet_Sub(u,v,self.c,1,self.n,self.n)
    del p1[0]
    del p2[0]
    del self.c[0]
    return self.c

def Permutation_Left_Join(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    if (self.Braid_As_PermutationQ(p1) == False) or\
    (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p = []
    p.append(self.Permutation_Inverse(p1))
    p.append(self.Permutation_Inverse(p2))
    join = self.Permutation_Right_Join(p[0],p[1])
    return self.Permutation_Inverse(join)

def Permutation_Right_Join(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
    if (self.Braid_As_PermutationQ(p1) == False) or\
    (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p = []
    p.append(Right_Complement(p1,self.n))
    p.append(Right_Complement(p2,self.n))
    join = self.Permutation_Right_Meet(p[0],p[1])
    return Right_Complement(join,self.n)

def Maximal_Head(self,p1,p2):
    """
    You must input "Permutation" into p1 and p2.
    """
```

```
    if (self.Braid_As_PermutationQ(p1) == False) or\
    (self.Braid_As_PermutationQ(p2) == False):
        print "Permutation_Error"
        sys.exit(0)
    p1_inv = self.Permutation_Inverse(p1)
    d = self.Omega()
    c = self.Permutation_Product(d,p1_inv)
    c = self.Permutation_Left_Meet(c,p2)
    M_H = self.Permutation_Product(p1,c)
    return M_H

def Permutation_To_Left_Canonical_Form(self,inf,P):
    """
    You must input "Infimum" into inf and
    "Permutation List" into P.
    """
    if len(P) <= 0:
        return [inf]+P
    for i in P:
        if self.Braid_As_PermutationQ(i) == False:
            print "Permutation_Error"
            sys.exit(0)
    i = 0
    w = []
    while i < len(P):
        a = self.Permutation_To_Word(P[i])
        for j in a:
            w.append(j)
        i = i+1
    pr = Word_To_Permutation_Representation(w,self.n)
    P_LCF = Permutation_Representation_To_Left_\
    Canonical_Form(pr,self.n)
    P_LCF[0] = P_LCF[0]+inf
    return P_LCF

def Permutation_To_Right_Canonical_Form(self,inf,P):
    """
    You must input "Infimum" into inf and
```

```
    "Permutation List" into P.
    """
    if len(P) <= 0:
        return [inf]+P
    for i in P:
        if self.Braid_As_PermutationQ(i) == False:
            print "Permutation_Error"
            sys.exit(0)
    i = 0
    w = []
    while i < len(P):
        a = self.Permutation_To_Word(P[i])
        for j in a:
            w.append(j)
        i = i+1
    P_RCF = self.Word_To_Right_Canonical_Form(w)
    i = 0
    while i < len(P_RCF)-1:
        P_RCF[i] = self.Word_To_Permutation(P_RCF[i])
        i = i+1
    P_RCF[-1] = P_RCF[-1]+inf
    return P_RCF

def Permutation_Left_Canonical_Form_Product(self,LCF1,LCF2):
    """
    You must input Left Canonical Form
    into LCF1 and LCF2.
    (LCF = [inf,"Permutation List"])
    """
    if (len(LCF1) or len(LCF2)) == 1:
        sys.exit(0)
    p = LCF1[0]
    q = LCF2[0]
    inf = p+q
    del LCF1[0]
    del LCF2[0]
    for i in LCF1:
        if self.Braid_As_PermutationQ(i) == False:
```

```
            print "Permutation_Error"
            sys.exit(0)
    for i in LCF2:
        if self.Braid_As_PermutationQ(i) == False:
            print "Permutation_Error"
            sys.exit(0)
    LCF = []
    for i in LCF1:
        if q % 2 == 1:
            LCF.append(self.Half_Twist_Permutation(i))
        else:
            LCF.append(i)
    LCF = LCF + LCF2
    LCF = self.Permutation_To_Left_Canonical_Form(inf,LCF)
    return LCF

def Permutation_Left_Canonical_Form_Inverse(self,LCF):
    """
    You must input Left Canonical Form
    into LCF.   (LCF = [inf,"Permutation List"])
    """
    q = LCF[0]
    l = len(LCF)-1
    inf = -(q+l)
    del LCF[0]
    for i in LCF:
        if self.Braid_As_PermutationQ(i) == False:
            print "Permutation_Error"
            sys.exit(0)
    B = []
    for i in LCF:
        B.append(self.Permutation_Product(\
    self.Permutation_Inverse(i),self.Omega()))
    B.reverse()
    LCF = []
    for i in B:
        if -(q+l) % 2 == 1:
            LCF.append(self.Half_Twist_Permutation(i))
```

```
            else:
                LCF.append(i)
            l = l-1
        LCF = [inf] + LCF
        return LCF

    def Permutation_Braid_Comparison(self,P1,P2):
        """
        You must input [Infimum,["Permutation List"]]
        into P1 and P2.
        """
        if (len(P1) < 2) or (len(P2) < 2):
            sys.exit(0)
        for i in P1[1]:
            if self.Braid_As_PermutationQ(i) == False:
                print "Permutation_Error"
                sys.exit(0)
        for i in P2[1]:
            if self.Braid_As_PermutationQ(i) == False:
                print "Permutation_Error"
                sys.exit(0)
        P1_inf = P1[0]
        P2_inf = P2[0]
        del P1[0]
        del P2[0]
        P1 = self.Permutation_To_Left_\
        Canonical_Form(P1_inf,P1[0])
        P2 = self.Permutation_To_Left_\
        Canonical_Form(P2_inf,P2[0])
        if P1 == P2:
            return True
        else:
            return False

    def Permutation_Random_Braid(self):
        i = 0
        I = self.Identity_Permutation()
        while i < self.n:
```

```
            j = random.randint(0,self.n-1)
            Temp(I,i,j)
            i = i+1
        return I

    """
    Braid Operations
    """

    def Infimum_Left_Canonical_Form(self,w):
        """
        You must input "Word" into w.
        """
        LCF = self.Word_To_Left_Canonical_Form(w)
        return LCF[0]

    def Canonical_Length_Left_Canonical_Form(self,w):
        """
        You must input "Word" into w.
        """
        LCF = self.Word_To_Left_Canonical_Form(w)
        return len(LCF)-1

    def Supremum_Left_Canonical_Form(self,w):
        """
        You must input "Word" into w.
        """
        LCF = self.Word_To_Left_Canonical_Form(w)
        return (LCF[0]+len(LCF)-1)

    def Infimum_Right_Canonical_Form(self,w):
        """
        You must input "Word" into w.
        """
        RCF = self.Word_To_Right_Canonical_Form(w)
        return RCF[-1]

    def Canonical_Length_Right_Canonical_Form(self,w):
```

```
    """
    You must input "Word" into w.
    """
    RCF = self.Word_To_Right_Canonical_Form(w)
    return len(RCF)-1

def Supremum_Right_Canonical_Form(self,w):
    """
    You must input "Word" into w.
    """
    RCF = self.Word_To_Right_Canonical_Form(w)
    return (RCF[-1]+len(RCF)-1)

def Cycling(self,W_LCF):
    """
    You must input "Left Canonical Form(Word)"
    into W_LCF.
    """
    if len(W_LCF) <= 1:
        return W_LCF
    inf = W_LCF[0]
    x1 = W_LCF[1]
    del W_LCF[1]
    cyc = W_LCF
    if inf % 2 == 1:
        x1 = self.Half_Twist_Word(x1)
    cyc.append(x1)
    return cyc

def Decycling(self,W_LCF):
    """
    You must input "Left Canonical Form(Word)"
    into W_LCF.
    """
    if len(W_LCF) <= 1:
        return W_LCF
    inf = W_LCF[0]
    x1 = W_LCF[-1]
```

```
        del W_LCF[-1]
        cyc = W_LCF
        if inf % 2 == 1:
            x1 = self.Half_Twist_Word(x1)
        cyc.insert(1,x1)
        return cyc

# The end of this Artin_Braid class.


############################################################
#                                                          #
#  Following functions is subfunction for main program     #
#                                                          #
############################################################

def Temp(a,i,j):
    temp = a[i]
    a[i] = a[j]
    a[j] = temp

def Is_LeftElement_Big(p,i):
    if p[i] > p[i+1]:
        return True
    else:
        return False

def Longest_Permutation_Braid_Word_From_Left(w,n):
    b = Artin_Braid(n)
    P = b.Identity_Permutation()
    i = 0
    while i < len(w):
        g = w[i]
        if Is_LeftElement_Big(P,g-1) == True:
            W = []
            j = 0
            while j <= i-1:
                W.append(w[j])
```

```
                j = j+1
            return W
        g = abs(g)
        Temp(P,g-1,g)
        i = i+1
    return w

def Word_To_Permutation_Representation(w,n):
    b = Artin_Braid(n)
    seq = [0]
    W = []
    X = []
    i = len(w)-1
    while i >= 0:
        g = abs(w[i])
        if seq[0]%2 == 0:
            g = g
        else:
            g = n-g
        if w[i] < 0:
            seq[0] = seq[0] - 1
            pos = b.Omega()
            Temp(pos,g-1,g)
            W = b.Permutation_To_Word(pos)+W
        else:
            W.insert(0,g)
        i = i-1
    while W != []:
        X = []
        for i in W:
            X.append(i)
        pref = Longest_Permutation_Braid_\
        Word_From_Left(X,n)
        i = 0
        while i < len(pref):
            del W[0]
            i = i+1
        seq.append(b.Word_To_Permutation(pref))
```

```
    del X
    return seq

def Permutation_Representation_To_Left_Canonical_\
Form(PR,n):
    b = Artin_Braid(n)
    if len(PR) == 1:
        return PR
    inf = PR[0]
    A = []
    I = b.Identity_Permutation()
    D = b.Omega()
    i = 1
    while i < len(PR):
        A.append(PR[i])
        i = i+1
    i = 0
    while i < len(A):
        if A[i] == I:
            del A[i]
        i = i+1
    i = 0
    while i < len(A)-1:
        ss = b.Starting_Set_Permutation(A[i+1])
        if ss == []:
            del A[i+1]
            continue
        fs = b.Finishing_Set_Permutation(A[i])
        k = 0
        Complement(ss,fs)
        if ss != []:
            j = ss[0]
            Temp(A[i],j-1,j)
            A[i+1] = \
            Multiply_From_LeftPriv(A[i+1],j-1,j,n)
            if A[i+1] == I:
                del A[i+1]
            if i > 0:
```

```
                i = i-1
        else:
            i = i + 1
    while (len(A) > 0) and (A[0] == D):
        inf = inf+1
        del A[0]
    A.insert(0,inf)
    return A

def Meet_Sub(a,b,c,s,t,n):
    u = (n+1)*[0]
    v = (n+1)*[0]
    w = (n+1)*[0]
    if t <= s:
        return
    m = (s+t)/2
    Meet_Sub(a,b,c,s,m,n)
    Meet_Sub(a,b,c,m+1,t,n)
    u[m] = a[c[m]]
    v[m] = b[c[m]]
    if s < m:
        i = m-1
        while i >= s:
            u[i] = min(a[c[i]],u[i+1])
            v[i] = min(b[c[i]],v[i+1])
            i = i-1
    u[m+1] = a[c[m+1]]
    v[m+1] = b[c[m+1]]
    if t > m+1:
        i = m+2
        while i <= t:
            u[i] = max(a[c[i]],u[i-1])
            v[i] = max(b[c[i]],v[i-1])
            i = i+1
    l = s
    r = m+1
    i = s
    while i <= t:
```

```
            if ((l > m) or ((r <= t) and \
            (u[l] > u[r]) and (v[l] > v[r]))):
                w[i] = c[r]
                r = r+1
            else:
                w[i] = c[l]
                l = l+1
            i = i+1
        i = s
        while i <= t:
            c[i] = w[i]
            i = i+1

    def Complement(SS,FS):
        for i in FS:
            k = 0
            while (len(SS) > 0) and (k < len(SS)):
                if SS[k] == i:
                    del SS[k]
                k = k+1
            if SS == []:
                break

    def Multiply_From_LeftPriv(p,i,j,n):
        b = Artin_Braid(n)
        p = b.Permutation_Inverse(p)
        Temp(p,i,j)
        p = b.Permutation_Inverse(p)
        return p

    def Right_Complement(p,n):
        b = Artin_Braid(n)
        o = b.Omega()
        return b.Permutation_Product(o,p)
```

Mathematica Program

プログラム内で使われている関数，及び使用方法は Lukkarila の Mathematica パッケージを参照.

```
LKRAttack[u_, v_, w_, index_, n_] :=
    Module[{a, A, alist, ans, i, j, ki, KI, m, q, U, VV, W},
      If[n/2 + index ≧ n || index ≦ 0, Return[False]];
      q = 1/2;
      m = n(n - 1)/2;
      U = Krammer[u, n, t, q];
      VV = Krammer[v, n, t, q];
      A = Normal[SparseArray[{m, m} -> 0]];
      alist = Normal[SparseArray[{m*m} -> 0]];
      For[i = 1; k = 1, i ≦ m, i += 1,
        For[j = 1, j ≦ m, j += 1; k += 1, A[[i, j]] =
            StringExpression[a, ToString[i], ToString[j]];
          alist[[k]] = A[[i, j]]]];
      i = n/2 + index;
      ki = {i};
      KI = Krammer[ki, n, t, q];
      (*Solve VA = AU, AK'(σ i) = K'(σ i)A*)
      ans = Solve[{A.U == VV.A, A.KI == KI.A}, alist];
      A1 = A /. ans[[1]];
      W = Krammer[w, n, t, q];
      A2 = Simplify[A1.W.Inverse[A1]];
      A2 = InvertKrammer[A2, t, q];
      Return[A2];];
```

参考文献

[1] I. Anshel, M. Anshel, D. Goldfeld *An algebraic method for public-key cryptography*, Math. Res. Lett.6 287-291 (1999).

[2] S. Bigelow, *The Burau representation is not faithful for n = 5*, *Geometry and Topology*, 3 397-404 (1999).

[3] J. S. Birman, K. H. Ko, S. J. Lee, *A new approach to the word and conjugacy problems in the braid groups*, Advances in Mathmatics, 139, 322-353, (1998).

[4] A. Bolstad, *Braid Group Cryptography Untangled*, http://www.math.wisc.edu/~boston/bolstad.doc, (2004).

[5] J. C. Cha, K. H. Ko, S. J. Lee, J. W. Han, J. H. Cheon, *An efficient implementation of braid groups*, Advances in Cryptology: Proceedings of ASIACRYPT 2001, Lecture Notes in Computer Science, Springer-Verlag, 2248 (2001) 144-156.

[6] J. C. Cha, CBraid: a C++ library for computation in braid groups, http://knot.kaist.ac.kr/ jccha/cbraid/, (2001).

[7] J. H. Cheon, B. Jun, *A polynomial time algorithm for the braid Diffie-Hellman conjugacy problem*, Advances in Cryptology: Proceedings of CRYPTO 2003, Lecture Notes in Computer Science, Springer-Verlag, 2729 (2003) 212-225.

[8] W. Diffie, M. E. Hellman, *New directions in Cryptography*, IEEE Transactions on Information Theory, 22 (1976), 644-654.

[9] H. Fuji, *An implementation of the algorithm to construct the nomal forms of band generator presentation for the braid group*, 日本大学文理学部応用数学科卒業研究報告書, (2003), http://www.tani.cs.chs.nihon-u.ac.jp/g-2003/hanana/.

[10] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. S. Kang, C. S. Park, *New public-key cryptosystem using braid groups*, Advances

in Cryptology： Proceedings of CRYPTO 2000, Lecture Notes in Computer Science, Springer-Verlag, 1880 (2000) 166-183.

[11] K. H. Ko, S. J. Lee, T. Thomas, *TOWARDS GENERATING SECURE KEYS FOR BRAID CRYPTOGRAPHY*, http://eprint.iacr.org/2007/149.pdf, (2007).

[12] D. Krammer, *Braid groups are linear*, Annals of Mathematics, 155 (2002) 131-156.

[13] V. Lukkarila, *A Mathematica-package for algebraic braid groups*, TUCS Technical Report, (2005), http://www.tucs.fi/publications/insight.php?id=tLukkarila05a.

[14] V. Lukkarila, AlgebraicBraids, http://www.tucs.fi/publications/attachments/TR689-Algebraic Braids.m, (2005).

[15] D. R. Stinson 著, 櫻井 幸一監訳, 暗号理論の基礎, 共立出版, (1996)

[16] Y. Tanaka, *Experimental Performance Evaluations of Public-key Cryptosystem Using Braid Groups*, 日本大学文理学部情報システム解析学科卒業研究報告書, (2005), http://www.tani.cs.chs.nihon-u.ac.jp/g-2005/yuko/.