CONJUGACY SEPARATION PROBLEM IN BRAIDS: AN ATTACK ON THE ORIGINAL COLORED BURAU KEY AGREEMENT PROTOCOL

MATVEI KOTOV, ANTON MENSHOV, ALEXEY MYASNIKOV,
DMITRY PANTELEEV, AND ALEXANDER USHAKOV

Abstract. In this paper, we consider the conjugacy separation search problem in braid groups. We deeply redesign the algorithm presented in [18] and provide an experimental evidence that the problem can be solved for 100% of very long randomly generated instances. The lengths of tested randomly generated instances is increased by the factor of two compared to the lengths suggested in the original proposal for 120 bits of security.

An implementation of our attack is freely available in CRAG, [6]. In particular, the implementation contains all challenging instances we had to deal with on a way to 100% success. We hope it will be useful to braid-group cryptography community.

Keywords. Algebraic eraser, braid group, colored Burau presentation, conjugacy problem, conjugacy separation, key agreement, cryptography.
2010 Mathematics Subject Classification. 03D15, 20F65, 20F10.

1. Introduction

The Anshel–Anshel–Goldfeld–Lemieux key agreement protocol (or colored Burau KAP, CBKAP, see [2]) was proposed to be used on low-cost platforms that constraint the use of computational resources. The core of the protocol is the concept of an Algebraic Eraser$^{\mathrm{TM}}$(abbreviated AE) which was claimed to be a suitable primitive for use within lightweight cryptography. The AE primitive is based on an idea of using an action of a semidirect product of groups on a (semi)group to obscure involved algebraic structures. The underlying motivation for CBKAP is the need to secure networks that deploy Radio Frequency Identification (RFID) tags used for identification, authentication, tracing and point-of-sale applications.

1.1. **Previous attacks.** There are three attacks on CBKAP. The attack described in [18] was designed to solve the conjugacy separation problem for public data generated by the third trusted party (TTP) for Alice and Bob. Breaking the data generated by TTP allows to use a linear algebra attack to find the shared key as described in [2, Section 6]. The attack worked for

Date: May 22, 2018.

The second author was supported by Russian Science Foundation (project N16-11-10002).

relatively short keys and failed for the parameters suggested for practical use.

KTT attack [11] used linear algebra and attempted to determine a part of the private key data. According to [10] KTT just described a class of weak keys, that would not be used and by choosing the private key data in a specific way, this attack can be defeated.

Recently, the paper [5] improved the KTT attack. The new attack reconstructs the shared secret using all of the public information and performing a large precomputation. It is claimed in [3] that time complexity necessary for the attack to complete grows very fast for appropriately chosen parameters.

1.2. **Our contribution.** In this paper we deeply revise the attack proposed in [18] that operated based on geodesic-braid approximation only. We use ideas of geometric and computational group theory to defeat the TTP algorithm and solve the related conjugacy separation problem with 100% success rate for instances twice as long as originally proposed instances. Our algorithm does not use any specific assumptions on the given input except those described in the original paper [2].

2. Preliminaries: group of braids

In this section we follow the exposition of [17, Section 5.1]. A braid is obtained by laying down a number of parallel pieces of strands and intertwining then, without loosing track of the fact that they run essentially in the same direction. In our pictures the direction is horizontal. We number strands at each horizontal position from the top down. See Figure 1 for example.

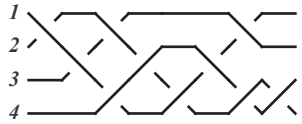

Figure 1. A 4-strand braid.

If we put down two braids u and v in a row so that the end of u matches the beginning of v we get another braid denoted by uv, i.e., concatenation of n-strand braids is a product. We consider two braids equivalent if there exists an isotopy between them, i.e., it is possible to move the strands of one of the braids in space (without moving the endpoints of strands and moving strands through each other) to get the other braid. We distinguish a special n-strand braid which contains no crossings and call it a trivial braid. Clearly the trivial braid behaves as left and right identity relative to the defined multiplication. The set B_n of isotopy classes of n-strand braids has a group structure, because if we concatenate a braid with its mirror image in a vertical plane the result is isotopic to the trivial braid.

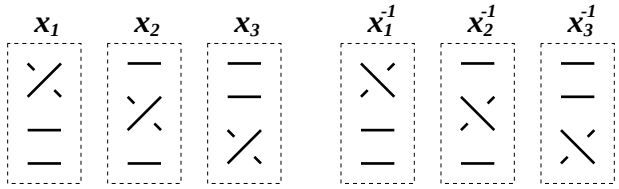

FIGURE 2. Generators of B_4 and their inverses.

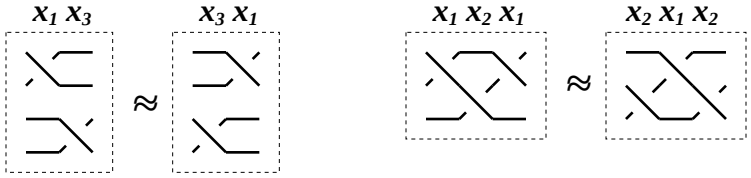

FIGURE 3. Typical relations in braids.

Each braid is uniquely defined by a sequence of strand crossings. A crossing is called positive if the front strand has a positive slope, otherwise it is called negative. There are exactly $n-1$ crossing types for n-strand braids, we denote them by $x_1, \ldots, x_{n-1}$, where x_i is a positive crossing of ith and $i+1$st strands. See Figure 2 for an example for B_4. Since, as we mentioned above, any braid is a sequence of crossings, the set $\{x_1, \ldots, x_{n-1}\}$ generates B_n. It is easy to see that crossings $x_1, \ldots, x_{n-1}$ are subject to the relations

$$[x_i, x_j] = 1$$

for every i, j such that $|i-j| > 1$ and

$$x_i x_{i+1} x_i = x_{i+1} x_i x_{i+1}$$

for every i such that $1 \le i \le n-2$. The corresponding braid configurations are shown in Figure 3. It is more difficult to prove that these two types of relations actually describe the equivalence on braids, i.e., the braid group B_n has the following (Artin) presentation:

$$B_n = \left\langle \; x_1, \ldots, x_{n-1} \;\middle|\; \begin{array}{ll} x_i x_j x_i = x_j x_i x_j & \text{if } |i-j| = 1 \\ x_i x_j = x_j x_i & \text{if } |i-j| > 1 \end{array} \right\rangle.$$

A word $w = w(x_1, \ldots, x_{n-1})$ in the generators of B_n and their inverses is called a *braid word*. Length $|w|$ of a braid word w is the number of letters in w.

From this description, one easily sees that there are many pairs of commuting subgroups in B_n, which makes it possible to use B_n as the platform group for protocols. For example, Ko, Lee et al. [14] used the following two commuting subgroups: $L_n = \langle x_1, \ldots, x_{\lceil \frac{n}{2} \rceil - 1} \rangle$ and $U_n = \langle x_{\lceil \frac{n}{2} \rceil + 1}, \ldots, x_{n-1} \rangle$.

2.1. **Dehornoy handle-free form.** Let w be a word in generators of B_n. An x_i-handle is a subword of w of the form

$$x_i^{-\varepsilon} w(x_1, \ldots, x_{i-2}, x_{i+1}, \ldots, x_n) x_i^{\varepsilon},$$

where $\varepsilon = \pm 1$. Schematically, an x_i-handle can be shown as in Figure 4. An x_i-handle $x_i^{-\varepsilon} w x_i^{\varepsilon}$ where $w = w(x_1, \ldots, x_{i-2}, x_{i+1}, \ldots, x_n)$ is called

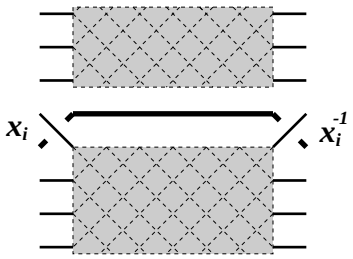

FIGURE 4. A handle.

permitted if w does not contain x_{i+1}-handles. We say that the braid word v' is obtained from the braid word v by a *one step handle reduction* if some subword of v is a permitted x_i-handle $x_i^{-\varepsilon} w x_i^{\varepsilon}$ and v' is obtained from v by applying the following substitutions for all letters in a handle $x_i^{-\varepsilon} w x_i^{\varepsilon}$:

$$x_j^{\pm 1} \to \begin{cases} 1 & \text{if } j = i, \\ x_{i+1}^{\varepsilon} x_i^{\pm 1} x_{i+1}^{\varepsilon} & \text{if } j = i+1, \\ x_j^{\pm 1} & \text{if } j < i \text{ or } j > i+1. \end{cases}$$

Schematically, a reduction of an x_i-handle can be shown as in Figure 5. We

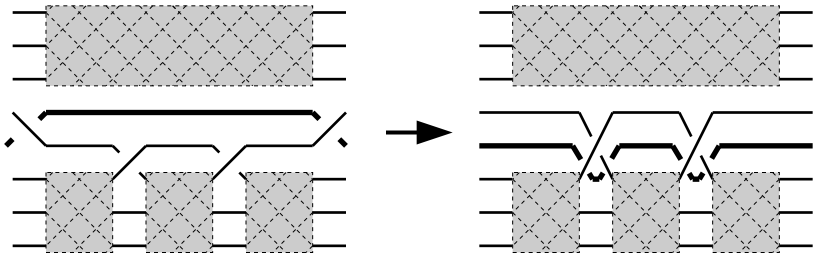

FIGURE 5. Removing a permitted x_i-handle.

say that the braid word v' is obtained from the braid word v by m *step handle reduction* if there exists a sequence of $m+1$ words $v = v_0, v_1, \ldots, v_m = v'$ each of which is obtained from the previous one by a one step handle reduction. A braid word is called *handle-free* if it contains no handles. The next theorem describes the main properties of handle reduction.

Theorem ([7])**.** Let v be a braid word. The following holds:

- Any sequence of handle reductions applied to v eventually stops and produces a handle-free braid word v' (which in general depends on a particular sequence of reductions) representing the same element of the braid group as v.
- The word v represents identity of a braid group if and only if any sequence of handle reductions applied to v produces the trivial word.

Remark 2.1 (Complexity estimates)**.** Even though the handle reduction procedure in practice is very efficient and most of the time works in linear time in terms of the length of a braid word, there is no good theoretical complexity estimate. For more on strategies for handle reduction and the the related discussion on complexity issues see [8, Section 3.3].

2.2. **Garside normal form.** The group B_n has a cyclic center generated by the element Δ^2, where Δ is the element called the half twist and can be expressed in the generators of B_n as follows:

$$\Delta = (x_1 \dots x_{n-1}) \cdot (x_1 \dots x_{n-2}) \cdot \dots \cdot (x_1).$$

Any element $g \in B_n$ can be uniquely represented in the form

$$\Delta^p \xi_1 \dots \xi_p$$

where $\xi_1, \dots, \xi_p$ are *permutation braids* satisfying certain conditions, called the left *Garside normal form*. For more detail see [9, Chapter 9].

2.3. **Geodesic braid approximation.** Let w be a word in generators of B_n. The problem of computing a geodesic word for words in B_∞ is known to be NP-complete (see [19]). It is known however (see e.g. [20, 12, 13]) that many NP-complete problems have polynomial time generic- or average-case solutions or have good approximate solutions.

The following algorithm tries to minimize the given braid word. It exploits the property of Dehornoys form that for a "generic" braid word w one has $|D(w)| < |w|$.

Algorithm 2.2 (Braid Minimization)**.**
INPUT. *A word $w = w(x_1, \dots, x_{n-1})$ in generators of the braid group B_n.*
OUTPUT. *A word w' such that $|w'| \le |w|$ and $w' = w$ in B_n.*
INITIALIZATION. *Put $w_0 = w$ and $i = 0$.*
COMPUTATIONS.

(1) Increment i.
(2) Put $w_i = D(w_{i-1})$.
(3) If $|w_i| < |w_{i-1}|$ then:
  (a) Put $w_i = w_i^\Delta$
  (b) Goto (1).
(4) If i is even then output $w' = w_{i+1}^\Delta$.
(5) If i is odd then output $w' = w_{i+1}$.

The method was introduced in paper [15, 16] where it was successfully used to break protocols [1, 14].

2.4. **Colored Burau group.** Fix a prime p and denote by R_n be the ring of Laurent polynomials in n variables $\{t_1, \ldots, t_n\}$ with coefficients in $\mathbb{F}_p$. Let $\mathbf{GL}_n(R_n)$ be the group of invertible matrices over R_n. The symmetric group S_n naturally acts on $\mathbf{GL}_n(R_n)$ by permuting the variables $\{t_1, \ldots, t_n\}$. The result of action of $\sigma \in S_n$ on $M \in \mathbf{GL}_n(R_n)$ is denoted by M^σ. Recall that a semidirect product of $\mathbf{GL}_n(R_n)$ and S_n is a group

$$\mathbf{GL}_n(R_n) \rtimes S_n = \{(M, \pi) \mid M \in \mathbf{GL}_n(R_n) \text{ and } \pi \in S_n\},$$

equipped with the operation

$$(M_1, \sigma_1) \cdot (M_2, \sigma_2) = (M_1 M_2^{\sigma_1}, \sigma_1 \sigma_2).$$

Define $n-1$ $n \times n$-matrices over polynomials in variables $\bar{t} = \{t_1, \ldots, t_n\}$:

$$C_1(t_1) = \left(\begin{array}{cc|c} -t_1 & 1 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & I_{n-2} \end{array}\right) \quad \text{and} \quad C_i(t_i) = \left(\begin{array}{c|ccc|c} I_{i-2} & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 \\ 0 & t_i & -t_i & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & I_{n-i-1} \end{array}\right)$$

for $2 \leq i \leq n-1$.

Lemma. A map φ on the generators $x_1, \ldots, x_{n-1}$ of B_n:

$$x_i \overset{\varphi}{\mapsto} (C_i(t_i), \pi_i),$$

where $\pi_i = (i, i+1) \in S_n$, extends into a group homomorphism.

The group $\langle (C_1(t_1), \pi_1), \ldots, (C_{n-1}(t_{n-1}), \pi_{n-1}) \rangle$ is called the *colored Burau representation* of B_n and is denoted by CB_n.

2.5. **Action of CB_n on a certain finite set.** Fix n nontrivial elements $\tau_1, \ldots, \tau_n \in \mathbb{F}_p$ termed t-values and define a group homomorphism

$$\epsilon : \mathbf{GL}_n(R_n) \to \mathbf{GL}_n(\mathbb{F}_p),$$

that for each i replaces t_i with the value τ_i. For $(m, \sigma) \in \mathbf{GL}_n(\mathbb{F}_p) \times S_n$ and $(C, \rho) \in CB_n$ define

$$(m, \sigma) \star (C, \rho) = (m \cdot \epsilon(C^\sigma), \sigma\rho),$$

which defines an action of CB_n on $\mathbf{GL}_n(\mathbb{F}_p) \times S_n$ and, hence, an action of B_n on $\mathbf{GL}_n(\mathbb{F}_p) \times S_n$. We say that (m_1, t_1) and (m_2, t_2) in CB_n *$\star$-commute* if the equality

$$(\epsilon(m_1), s_1) \star (m_2, s_2) = (\epsilon(m_2), s_2) \star (m_1, s_1)$$

holds.

3. The protocol

Before the parties perform actual transmissions the following data is being prepared by the Third Trusted Party (TTP).

- A matrix $m_0 \in \mathbf{GL}_n(\mathbb{F}_p)$ which has an irreducible characteristic polynomial over $\mathbb{F}_p$. The choice of m_0 is not relevant for the purposes of this paper, we refer the reader to [2] for more information on how m_0 can be generated randomly.
- $\star$-commuting subgroups $A = \langle w_1, \ldots, w_\gamma \rangle$ and $B = \langle u_1, \ldots, u_\gamma \rangle$ of the group CB_n. We want to point out that the elements w_i and v_j are given to us as products of generators of CB_n and there inverses, i.e., as formal words in group alphabet $\{g_1, \ldots, g_{n-1}\}$. We prefer this form because it allows us to avoid time consuming matrix multiplication in $\mathbf{GL}_n(R_n)$.

Both the matrix m_0 and subgroups A and B can be chosen only once. Now, the public and private keys are chosen as follows:

Alice's Private Key: is a pair which consists of a matrix of the form

$$n_a = l_1 m_0^{\alpha_1} + l_2 m_0^{\alpha_2} + \ldots + l_r m_0^{\alpha_r} \in \mathbf{GL}_n(\mathbb{F}_p)$$

(where $l_1, \ldots, l_r \in \mathbb{F}_p$ and $r, \alpha_1, \ldots, \alpha_r \in \mathbb{Z}^+$) and a random sequence $w_{i_1}^{\varepsilon_1}, \ldots, w_{i_m}^{\varepsilon_m}$ of generators of A and their inverses.

Alice's Public Key: is an element

$$A_{public} = (n_a, id) \star w_{i_1}^{\varepsilon_1} \star \ldots \star w_{i_m}^{\varepsilon_m} \in \mathbf{GL}_n(\mathbb{F}_p) \times S_n.$$

Recall that each w_{i_k} is given as a formal product of the generators of G. To perform the $\star$-operation efficiently one should not directly compute w_{i_k}, but consequently apply the factors of w_{i_k} to the argument.

Bob's Private Key: is a pair which consists of a matrix of the form

$$n_b = l'_1 m_0^{\beta_1} + l'_2 m_0^{\beta_2} + \ldots + l'_{r'} m_0^{\beta_{r'}} \in \mathbf{GL}_n(\mathbb{F}_p)$$

(where $l'_1, \ldots, l'_{r'} \in \mathbb{F}_p$ and $r', \beta_1, \ldots, \beta_{r'} \in \mathbb{Z}^+$) and a random sequence $v_{j_1}^{\delta_1}, \ldots, v_{j_l}^{\delta_l}$ of generators of B and their inverses.

Bob's Public Key: is a pair

$$B_{public} = (n_b, id) \star v_{j_1}^{\delta_i} \star \ldots \star v_{j_l}^{\delta_l} \in \mathbf{GL}_n(\mathbb{F}_p) \times S_n.$$

Again, each v_{j_k} is given as a formal product of the generators of CB_n. To perform the $\star$-operation efficiently one should not directly compute v_{j_k}, but consequently apply the factors of v_{j_k} to the argument.

The shared key: is an element of $\mathbf{GL}_n(\mathbb{F}_p) \times S_n$ obtained by Alice in the form

$$[(n_a, id) \cdot B_{public}] \star w_{i_1}^{\varepsilon_1} \star \ldots \star w_{i_m}^{\varepsilon_m}$$

and by Bob in the form

$$[(n_b, id) \cdot A_{public}] \star v_{j_1}^{\delta_i} \star \ldots \star v_{j_l}^{\delta_l}.$$

It requires a little work to prove that the obtained elements are indeed equal in $\mathbf{GL}_n(\mathbb{F}_p)$. We omit the proof.

3.1. **TTP algorithm.** The cornerstone part of the proposed key exchange is the choice of $\star$-commuting subgroups of the group CB_n. For that one can generate commuting subgroups A and B in B_n and map them into CB_n using the epimorphism φ defined in Section 2.4. The subgroups $\varphi(A)$ and $\varphi(B)$ commute. Moreover, for any choice of ϵ the subgroups $\varphi(A)$ and $\varphi(B)$ $\star$-commute.

Since Δ^2 is a central element, it follows that elements u, w commute in B_n if and only $u\Delta^{2p}$ and $w\Delta^{2r}$ do (for any $p, r \in \mathbb{Z}$). Hence, we may always assume that the normal forms of the generators $\{w_1, \ldots, w_\gamma\}$ and $\{v_1, \ldots, v_\gamma\}$ have the exponent on Δ equal to 0 or -1. When we say that we reduce a braid modulo Δ^2 we mean changing the Δ-power of its Garside normal form to -1 or 0 depending on parity. The algorithm below (originally proposed in [2]) generates two commuting subgroups in B_n.

Algorithm 3.1. (TTP algorithm)

(1) *Choose two secret subsets* $BL = \{x_{l_1}, \ldots, x_{l_\alpha}\}$, $BR = \{x_{r_1}, \ldots, x_{r_\beta}\}$ *of the set of generators of* B_n, *where* $|l_i - r_j| \geq 2$ *for all* $1 \leq i \leq l_\alpha$ *and* $1 \leq j \leq r_\beta$.
(2) *Choose a secret element* $z \in B_n$.
(3) *Choose words* $\{w_1, \ldots, w_\gamma\}$ *of bounded length over the generators* BL.
(4) *Choose words* $\{v_1, \ldots, v_\gamma\}$ *of bounded length over the generators* BR.
(5) *For each* $i = 1, \ldots, \gamma$:
   (a) *calculate the left normal form of* zw_iz^{-1} *and reduce the result modulo* Δ^2;
   (b) *put* w_i' *to be a braid word corresponding to the element calculated in (a);*
   (c) *calculate the left normal form of* zv_iz^{-1} *and reduce the result modulo* Δ^2;
   (d) *put* v_i' *to be a braid word corresponding to the element calculated in (c).*
(6) *Publish the sets* $\{v_1', \ldots, v_\gamma'\}$ *and* $\{w_1', \ldots, w_\gamma'\}$.

TTP algorithm produces generators of two commuting subgroups in B_n. Alice and Bob apply epimorphism φ (defined in Section 2.4) to obtain $\star$-commuting subgroups.

3.2. **Security assumptions.** It was noticed in [2] that if the conjugator z generated randomly by the TTP algorithm is known, then there exists an efficient linear attack on the scheme which is able to recover the shared key of the parties. The problem of recovering the exact z seems like a very difficult mathematical problem because it reduces to solving the system of

equations

$$(1)\qquad \begin{cases} w'_1 = \Delta^{2p_1} z w_1 z^{-1} \\ \dots \\ w'_\gamma = \Delta^{2p_\gamma} z w_\gamma z^{-1} \\ v'_1 = \Delta^{2r_1} z v_1 z^{-1} \\ \dots \\ v'_\gamma = \Delta^{2r_\gamma} z v_\gamma z^{-1} \end{cases}$$

which has too many unknowns, since only left hand sides (i.e., elements $w'_1, \dots, w'_\gamma$, $v'_1, \dots, v'_\gamma$) are known. Hence, it might be difficult to find the original z.

It was noticed in [18] that one does need to use exactly the same z to apply the linear attack [2, Section 6]. In fact, any solution z' of the problem stated below plays a role of a conjugator z.

Simultaneous conjugacy separation search problem in B_n (SCSSP). For tuples $\{v'_1, \dots, v'_\gamma\}$ and $\{w'_1, \dots, w'_\gamma\}$ find any $z' \in B_n$ and

$$p_1, \dots, p_\gamma, r_1, \dots, r_\gamma \in \mathbb{Z}$$

such that the words

$$\{\Delta^{2p_1} z'^{-1} v'_1 z', \dots, \Delta^{2p_\gamma} z'^{-1} v'_\gamma z'\}$$

and

$$\{\Delta^{2r_1} z'^{-1} w'_1 z', \dots, \Delta^{2r_\gamma} z'^{-1} w'_\gamma z'\}$$

can be expressed as words over two disjoint commuting subsets of generators of B_n.

SCSSP has little in common with the *simultaneous conjugacy search problem* often referenced in the papers on the braid group cryptography. The main difference is that in the conjugacy search problem both conjugate elements are available and the goal is to recover the secret conjugator. In case of SCSSP, only the left hand side of the equation is known. The problem was analyzed in [18] with some limited success.

4. The attack

In this paper we improve the results of [18]. The original form of the attack is deeply revised to work for very long keys. We updated all key components to achieve 100% success rate on the tested instances of the protocol. We also introduced several new components. The new attack works as follows for the given instance $\{v'_1, \dots, v'_\gamma\}$, $\{w'_1, \dots, w'_\gamma\}$ of the separation problem:

(1) Multiply each word in the tuples $\{v'_1, \dots, v'_\gamma\}, \{w'_1, \dots, w'_\gamma\}$ by an *anticipated* Δ^2-power. See Section 4.2 for more detail.
(2) Enumerate conjugates of the instance $\{v'_1, \dots, v'_\gamma\}, \{w'_1, \dots, w'_\gamma\}$ using length-based heuristic:
   (a) On each iteration pick an unchecked instance of the least total length.

(b) Conjugate the instance by all generators $x_i^{\pm 1}$ of B_n and save new instances (see Sections 4.3 and 4.4).
(c) Termination condition (see Section 4.1):
- Claim success if separated tuples are found.
- Accept failure when can not find separated tuples in an allocated time (set to 12 hours in our experiments).

(d) If stuck (20 iterations with no length decrease):
- Apply Δ-fix if it is applicable. (see Section 4.5).
- If not, then restart enumeration of conjugates (see Section 4.6).

Below we describe each component of the attack in detail.

4.1. **Termination condition.** We say that tuples $\{v_1, \ldots, v_\gamma\}$ and $\{w_1, \ldots, w_\gamma\}$ are *separated* if the letters involved in $v_1, \ldots, v_\gamma$ and the letters involved in $w_1, \ldots, w_\gamma$ commute. One can compute the number of occurrences of each letter in each words and output the result as a table (with the total length on the right), below is an example for B_8 with $\gamma = 3$:

```
v1: [ 34. 52. 34.  0.  0.  0.  0.] -> 120
v2: [ 33. 70. 33.  0.  0.  0.  0.] -> 136
v3: [ 36. 46. 46.  0.  0.  0.  0.] -> 128

w1: [  0.  0.  0.  0. 28. 53. 39.] -> 120
w2: [  0.  0.  0.  0. 21. 43. 36.] -> 100
w3: [  0.  0.  0.  0. 36. 52. 28.] -> 116
```

It is clear that the tuples with a table as above are separated. Clearly, the program can declare success and stop when it gets an instance with two separated tuples.

Unfortunately, this condition turned out to be weak for tuples of long braid words. We apply braid-minimization to each word that often fails to remove some letters even when those letter can be removed. The original attack often was stuck with tuples of the following form:

```
v1: [  0.  0.  0.  0.150.241.157.] -> 548
v2: [  0.  0.  0.  0.168.199.125.] -> 492
v3: [  0.  0.  0.  0.147.230.153.] -> 530

w1: [139.226.133.  0.  0.  0.  0.] -> 498
w2: [120.245.160.  3.  0.  0.  0.] -> 528
w3: [133.212.151.  0.  0.  0.  0.] -> 496
```

where the words appear to be "almost separated" with an exception of a single braid word w_2. But, experimenting with braid-minimization we noticed the following:

- Dehornoy handle-free form $D(w)$ "favors" cancellations of lower-indexed letters in w;

- Dehornoy handle-free form $\Delta D(\Delta^{-1}w\Delta)\Delta^{-1}$ "favors" cancellations of higher-indexed letters in w.

Our new termination condition uses this observation and checks if the following two conditions hold:

- $\{D(v_1),\ldots,D(v_\gamma)\}$ and $\{\Delta D(\Delta^{-1}w_1\Delta)\Delta^{-1},\ldots,\Delta D(\Delta^{-1}w_\gamma\Delta)\Delta^{-1}\}$ are separated;
- $\{D(w_1),\ldots,D(w_\gamma)\}$ and $\{\Delta D(\Delta^{-1}v_1\Delta)\Delta^{-1},\ldots,\Delta D(\Delta^{-1}v_\gamma\Delta)\Delta^{-1}\}$ are separated.

4.2. **Initial Δ^2-power guess.** Each word $v'_i = \Delta^{2p_i} z v_i z^{-1}$ [and w_i' resp.] is obtained from zv_iz^{-1} [zw_iz^{-1}' resp.] by multiplying it by a Δ^2-power. The initial step in our attack is an attempt to reconstruct the original values p_i and r_i.

There are several tools available for this task. The original paper [18] used a length-aided heuristic to find the value of the dropped Δ^2-power in v_i's and w_i's. Our new attack uses abelianization of B_n instead. Abelianization can be computed in linear time which is a huge advantage over braid-minimization. Recall that abelianization of B_n is a homomorphism

$$\alpha : B_n \to B_n/B'_n = \langle x_1,\ldots,x_{n-1} \mid x_1 = \ldots = x_{n-1}\rangle \simeq \mathbb{Z},$$

that in a given braid-word w counts the number of positive letters minus the number of negative letters. Observe that:

- $\alpha(\Delta^2) = n(n-1)$.
- $\alpha(\Delta^{2p} z w z^{-1}) = pn(n-1) + \alpha(w)$.
- The expected value of the abelianization value is:

$$\mathbb{E}(|\alpha(w)|) \approx \sqrt{|w|},$$

  for a randomly uniformly generated braid-word w.

For instance, if $n = 16$ and $|w| = 1000$ we get $\mathbb{E}(|\alpha(w)|) \approx 33$ and $\alpha(\Delta^{2p} z w z^{-1}) = 240 \cdot p + \alpha(w)$, which is a sum of the value $\alpha(w)$ (expected to be relatively small) and a multiple of 240.

Using the observation above, it is natural to define an *anticipated* Δ^2-*power* for w'_i (and, similarly, for v'_i) to be the number p'_i that minimizes the value of the abelianization $|pn(n+1) + \alpha(w'_i)|$. Hence, the initial step of the attack performs the following:

- Compute $p'_i \in \mathbb{Z}$ that minimizes the value $|p'_i n(n+1) + \alpha(w'_i)|$ and multiply w'_i by $\Delta^{2p'_i}$.
- Compute $r'_i \in \mathbb{Z}$ that minimizes the value $|r'_i n(n+1) + \alpha(v'_i)|$ and multiply v'_i by $\Delta^{2r'_i}$.

This procedure works as expected when $|\alpha(w'_i)|, |\alpha(v'_i)| \leq \frac{1}{2}n(n-1)$. Otherwise it produces a slightly incorrect value (fixed as described in Section 4.5). The chance that the above inequality holds is higher for large values of the rank n.

4.3. **Producing new instances.** On each iteration the program chooses an unchecked instance $\{w_1, \ldots, w_\gamma\}$ and $\{v_1, \ldots, v_\gamma\}$ of the least total length. The instance is conjugated by generators and their inverses $x_1^{\pm 1}, \ldots, x_{n-1}^{\pm 1}$:

$$\{x_i^{\pm 1} w_1 x_i^{\mp 1}, \ldots, x_i^{\pm 1} w_\gamma x_i^{\mp 1}\}, \{x_i^{\pm 1} v_1 x_i^{\mp 1}, \ldots, x_i^{\pm 1} v_\gamma x_i^{\mp 1}\},$$

to produce new equivalent instances of the conjugacy separation problem. The obtained words are minimized using braid-minimization and new instances are saved as unchecked.

4.4. **Fast descend for long conjugators z.** Each iteration applies braid-minimization to $2\gamma \cdot 2(n-1)$ braid words (number of words in an instance times the number of generators and their inverses). For instance, if $\gamma = 10$, $n = 16$, and $|v_i| = |w_i| = |z| \approx 1000$, then each iteration applies braid-minimization to 600 words of length at least 1000 which is very time consuming. A heuristic search procedure that conjugates an instance by a single generator is expected to take at least $|z| \approx 1000$ steps to reach the answer.

To make search more efficient we use the following trick. For an instance $\{w_1, \ldots, w_\gamma\}$ and $\{v_1, \ldots, v_\gamma\}$ we take the terminal segment x of w_1 of length $|w_1|/10$, conjugate by x and add a new instance $\{xw_1x^{-1}, \ldots, xw_\gamma x^{-1}\}$ and $\{xv_1x^{-1}, \ldots, xv_\gamma x^{-1}\}$ to the set of unchecked instances. This approach often decreases the total length of the instance by as much as 10–20% on several iterations in the beginning of heuristic descend.

4.5. **Fixing wrong Δ^2-powers.** As we mentioned in Section 4.2, the initial Δ^2-power guess can produce an incorrect value and we do not have any tools to immediately recognize the mistake. Fortunately, we can recognize the problem deep inside our enumeration of conjugate instances. But first we need to introduce concepts of a crossing number and tuples separated modulo Δ^2.

Recall that the *crossing number* c_{ij} of ith and jth strands in a braid w is the algebraic number of positive crossings of ith and jth strands minus the negative number of crossings of the same strands. The next lemma allows to find the value p in a braid of the form $w\Delta^{2p}$, where $w \in \langle x_1, \ldots, x_k \rangle$, by observing the crossing numbers. Its proof is trivial.

Lemma 4.1. *If $w \in \langle x_1, \ldots, x_k \rangle$, then for $w\Delta^{2p}$ we have:*

$$c_{ij} = 2p \text{ for every } 1 \le i \le n \text{ and every } j \ge k+2. \tag{2}$$

We say that tuples $\{v_1, \ldots, v_\gamma\}$ and $\{w_1, \ldots, w_\gamma\}$ are *separated modulo* Δ^2 if

$$v_i = \Delta^{2p_i} v_i^* \text{ and } w_i = \Delta^{2r_i} w_i^*,$$

where $\{v_1^*, \ldots, v_\gamma^*\}$ and $\{w_1^*, \ldots, w_\gamma^*\}$ are separated tuples. Experimenting with the program, we noticed that length-based search of the conjugator achieved its goal even when Δ^2-powers of some words are incorrect, i.e., the length-based heuristic separated the tuples modulo Δ^2. That allows to use the formula (2) and reconstruct the original Δ^2-powers.

This behavior is not surprising since Δ^2 is a central element. Multiplication of a word in an instance by Δ^2 and conjugating the instance by a generator are commuting actions, which means that recovery of Δ^2-powers and finding a secret conjugator can be performed in any order.

4.6. **Restarting enumeration of conjugates.** Testing the program we found several special instances with correct Δ^2-powers that could not be minimized any further and did not produce separated tuples. Letter-distribution for those examples looked as in the example below:

```
[  8.  8.192.291.186.  9.  4.] -> 698
[  6.  7.194.262.193.  9.  5.] -> 676
[  0.  0.196.290.168.  0.  0.] -> 654

[  4.  7.173.280.192. 15.  5.] -> 676
[  4.  7. 10. 10.163.283.219.] -> 696
[  4.  7. 10. 10.173.252.204.] -> 660
```

Letters in the first tuple concentrate in the middle instead of being concentrated on the left or on the right. (Also, we have wrong values of Δ^2-powers in the example above.) This type of distribution could be obtained by conjugating the instance by δ (the fundamental braid in Birman-Ko-Lee form).

To solve those instances our current implementation aborts computation, conjugates the best obtained instance by a random word of length 500, and starts enumeration of conjugates again.

4.7. **Parallel computations.** The attack described above operates on tuples of braids. Most stages of the attack can be easily parallelized using basic tools of the standard C++ library. For instance, the attack performs numerous conjugations of tuples by generators the of B_n followed by braid-minimization. In our implementation braid-minimization for each word $w_1, \ldots, w_\gamma, v_1, \ldots, v_\gamma$ is run in parallel using std::thread. Using parallel computations we achieved eight-twelve times improved running time for some stages of the algorithm. See section 5 for effective CPU usage numbers.

5. Experimental results

To test success of our attack we performed series of experiments for randomly generated instances with $|v_i| = |w_i| = |z| \approx 20, 50, 100, 200, 500, 1000$ in braid groups $B_8, B_{12}, B_{16}, B_{20}$ with $\gamma = 10$. The program was 100% successful for each parameter set, see Table 1.

All braids were generated uniformly randomly as braid-words on standard generators $x_1 ... x_{n-1}$ and their inverses. (We do not use any particular properties of braids, such as order of corresponding permutation, hence uniform approach is fair.) Using an appropriate conjugator, we may always assume that the sets BL and BR split the set of standard generators in two halves.

Experiments were performed on a machine with two 8-core 3.1 GHz Intel Xeon CPU E5-2687W and 64GB RAM. On average our implementation was

using 3 to 5 cores, with up to 12 cores at its peak. Memory usage was less than 2 GB for parameters $|v_i| = |w_i| = |z| \approx 1000$ in B_{20}.

	B_8	B_{12}	B_{16}	B_{20}
$\|v_i\| = \|w_i\| = \|z\| \approx 20$	100%	100%	100%	100%
$\|v_i\| = \|w_i\| = \|z\| \approx 50$	100%	100%	100%	100%
$\|v_i\| = \|w_i\| = \|z\| \approx 100$	100%	100%	100%	100%
$\|v_i\| = \|w_i\| = \|z\| \approx 200$	100%	100%	100%	100%
$\|v_i\| = \|w_i\| = \|z\| \approx 500$	100%	100%	100%	100%
$\|v_i\| = \|w_i\| = \|z\| \approx 1000$	100%	100%	100%	100%

TABLE 1. Success rate.

Table 2 shows time (in minutes) our program spent on 100 randomly generated instance of the problem.

Remark 5.1. Numbers in Table 2 require some explanation. In the row for $|v_i| = |w_i| = |z| \approx 200$ the value for B_8 is greater than the values for B_{12}, B_{16}, B_{20}. It happens because the chance of a wrong Δ^2-power guess is higher for smaller values of the rank n. On the other hand, for $|v_i| = |w_i| = |z| \approx 500$ complexity of applying braid-minimization overwhelms the time the program spends to fix wrong Δ^2-powers. We observe increasing values in the corresponding row.

	B_8	B_{12}	B_{16}	B_{20}
$\|v_i\| = \|w_i\| = \|z\| \approx 20$	2	0.1	0.3	0.2
$\|v_i\| = \|w_i\| = \|z\| \approx 50$	2	1	1	1
$\|v_i\| = \|w_i\| = \|z\| \approx 100$	5	8	6	5
$\|v_i\| = \|w_i\| = \|z\| \approx 200$	42	26	34	16
$\|v_i\| = \|w_i\| = \|z\| \approx 500$	246	387	586	897
$\|v_i\| = \|w_i\| = \|z\| \approx 1000$	1330	3949	5342	9855

TABLE 2. Time required to solve 100 random instances of the problem (in minutes). See Remark 5.1 for discussion.

	B_8	B_{12}	B_{16}	B_{20}
$\|v_i\| = \|w_i\| = \|z\| \approx 20$	3986	6244	9573	12586
$\|v_i\| = \|w_i\| = \|z\| \approx 50$	10005	15842	21968	28576
$\|v_i\| = \|w_i\| = \|z\| \approx 100$	20258	31485	43089	56229
$\|v_i\| = \|w_i\| = \|z\| \approx 200$	41361	64291	86824	110392
$\|v_i\| = \|w_i\| = \|z\| \approx 500$	103100	159551	218255	275611
$\|v_i\| = \|w_i\| = \|z\| \approx 1000$	206835	320829	438238	556195

TABLE 3. Average total length of an instance after length reduction.

	B_8	B_{12}	B_{16}	B_{20}
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 20$	17708	31392	52102	72553
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 50$	44449	79648	119565	164729
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 100$	90000	158295	234520	324138
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 200$	183754	323233	472556	636368
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 500$	458041	802167	1187895	1588793
$\lvert v_i\rvert = \lvert w_i\rvert = \lvert z\rvert \approx 1000$	918904	1613018	2385195	3206254

TABLE 4. Average bit-size of instances.

Table 3 shows average total length of the public tuples after length reduction: $|v_1| + \ldots + |v_\gamma| + |w_1| + \ldots + |w_\gamma|$. Table 4 shows average total length of the public tuples after length reduction: $(|v_1| + \ldots + |v_\gamma| + |w_1| + \ldots + |w_\gamma|) \cdot (\log_2(n) + 1)$.

6. Conclusion

We showed that 100% of randomly generated instances of the conjugacy separation problem of length twice as long as originally proposed can be broken in less than two hours by a heuristic algorithm that operates based on "geometric tools" such as geodesic-length approximation, abelianization, and cross-numbers for braids. Our tools are not perfect (especially geodesic-length approximation that often gives far from optimal results), yet, they are sufficient for the task. We believe our algorithm has a lot of potential and can work for much longer instances of conjugacy separation problem (of course, some sufficient time should allocated to accommodate the increase). Our algorithm does not use any special assumptions on the given inputs except those described in the original paper [2]. This shows that the original protocol described in [2] cannot provide a significant level of security.

Our attack gives a very important tool. Most group-theoretic protocols (even those that use non-abelian platform groups) require a choice of commuting elements (that's the case for the "ancient" Ko-Lee protocol, for CBKAP, and for very recent protocols such as Kayawood [4]. Typically, choice of commuting elements is done by picking two "naturally commuting" subgroups (L_n and U_n in braids) and then conjugating them with some mixing conjugator. "Naturally commuting" subgroups are typically bad as they nontrivially act on some disjoint spaces and the action of the shared key can be reconstructed based on public keys (that's the case for E-multiplication). Our paper shows that using a mixing conjugator does not help: given two conjugated (braid-)tuples we can always conjugate them back to "naturally commuting" subgroups and recover the shared key.

References

[1] I. Anshel, M. Anshel, and D. Goldfeld. An algebraic method for public-key cryptography. *Math. Res. Lett.*, 6(3-4):287–291, 1999.

[2] I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux. Key agreement, the algebraic eraser^{TM}, and lightweight cryptography. In *Algebraic Methods in Cryptography*, volume 418 of *Contemporary Mathematics*, pages 1–34. American Mathematical Society, 2006.

[3] I. Anshel, D. Atkins, D. Goldfeld, and P. Gunnells. Defeating the Ben-Zvi, Blackburn, and Tsaban Attack on the Algebraic Eraser. preprint. Available at arXiv:1601.04780v1 [cs.CR], 20126.

[4] I. Anshel, D. Atkins, and P. Goldfeld, D. Gunnels. Kayawood, a Key Agreement Protocol. Preprint. Available at `https://eprint.iacr.org/2017/1162` (version: 30-Nov-2017), 2017.

[5] A. Ben-Zvi, S. Blackburn, and B. Tsaban. A practical cryptanalysis of the algebraic eraser. In *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *Lecture Notes Comp. Sc.*, pages 179–189, Berlin, 2016. Springer.

[6] CRyptography And Groups (CRAG) C++ Library. Available at `https://github.com/stevens-crag/crag`.

[7] P. Dehornoy. A fast method for comparing braids. *Adv. Math.*, 125:200–235, 1997.

[8] P. Dehornoy, I. Dynnikov, D. Rolfsen, and B. Wiest. *Why are braids orderable?* Societe Mathematique De France, 2002.

[9] D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, 1992.

[10] D. Goldfeld and P. Gunnells. Defeating the Kalka-Teicher-Tsaban linear algebra attack on the Algebraic Eraser. preprint. Available at eprint 1202.0598, 2012.

[11] A. Kalka, M. Teicher, and B. Tsaban. Short expressions of permutations as products and cryptanalysis of the algebraic eraser. *Adv. Appl. Math.*, 49:57–76, 2012.

[12] I. Kapovich, A. G. Miasnikov, P. Schupp, and V. Shpilrain. Generic-case complexity, decision problems in group theory and random walks. *J. Algebra*, 264:665–694, 2003.

[13] I. Kapovich, A. Myasnikov, P. Schupp, and V. Shpilrain. Average-case complexity and decision problems in group theory. *Adv. Math.*, 190:343–359, 2005.

[14] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. Kang, and C. Park. New public-key cryptosystem using braid groups. In *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes Comp. Sc.*, pages 166–183, Berlin, 2000. Springer.

[15] A. G. Miasnikov, V. Shpilrain, and A. Ushakov. A practical attack on some braid group based cryptographic protocols. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes Comp. Sc.*, pages 86–96, Berlin, 2005. Springer.

[16] A. G. Miasnikov, V. Shpilrain, and A. Ushakov. Random subgroups of braid groups: an approach to cryptanalysis of a braid group based cryptographic protocol. In *Advances in Cryptology – PKC 2006*, volume 3958 of *Lecture Notes Comp. Sc.*, pages 302–314, Berlin, 2006. Springer.

[17] A. G. Miasnikov, V. Shpilrain, and A. Ushakov. *Non-Commutative Cryptography and Complexity of Group-Theoretic Problems*. Mathematical Surveys and Monographs. AMS, 2011.

[18] A. D. Myasnikov and A. Ushakov. Cryptanalysis of Anshel-Anshel-Goldfeld-Lemieux key agreement protocol. *Groups Complex. Cryptol.*, 1:263–275, 2009.

[19] M. Paterson and A. Razborov. The set of minimal braids is co-NP-complete. *J. Algorithms*, 12:393–408, 1991.

[20] J. Wang. Average-case completeness of a word problem for groups. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 325–334. ACM, 1995.

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA
E-mail address: mkotov@stevens.edu

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA, Institute of Mathematics and Information Technologies, Dostoevskii Omsk State University, Omsk, Russia
E-mail address: manton@stevens.edu

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA
E-mail address: amyasnik@stevens.edu

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA
E-mail address: dpantel1@stevens.edu

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA
E-mail address: aushakov@stevens.edu