Braid Group Cryptography Untangled

Andrew Bolstad

Professor Nigel Boston
Math/ECE 842
University of Wisconsin

December 15, 2004

Recently, the class of non-abelian infinite groups known as the braid groups, B_n, has attracted attention as a possible source of cryptographic schemes, including key exchange and user verification. The braid groups have very complicated structure, yet have a very nice geometrical interpretation. There are well known solutions to the word problem, and fast algorithms for implementation on digital computers. Though the braid groups have been known and studied for many years, the first braid group cryptosystems appeared in 2000. Shortly thereafter, a polynomial time attack to existing systems was discovered. Despite this attack, there may be some hope for braid groups. There may be some problems that are still hard and some application specific schemes that are still good enough for large enough n. This report will cover an introduction to braid groups including solutions to the word problem, theoretical advantages, computational advantages, proposed systems, and attacks. In order to examine these cryptosystems, it is first necessary to define and introduce the braid groups.

1. Introduction to Braid Groups

The natural way to think about the braid groups is through their geometric interpretation. Picture a set of n parallel strings hanging in a line. Number the strings $1,2,\ldots,n$ starting on the left. An *n-braid* is obtained by intertwining the strings and fixing the lower ends in a line. Notice that a pair of strings can be intertwined in two ways: by passing the string on the left over or under the string on the right. Figure 1 illustrates a few braids. Braids will be considered to start at the top and end at the bottom throughout.

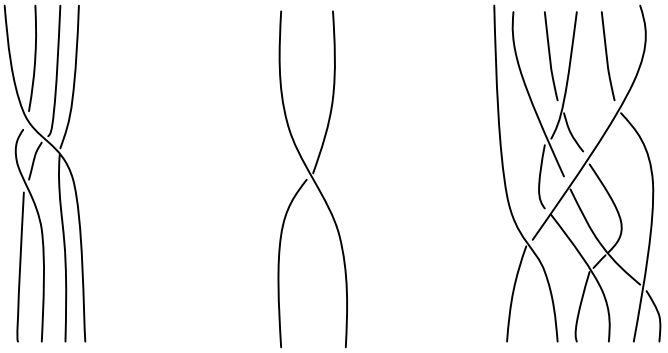

Figure 1: A few braids.

For a given n, called the *braid index*, the set of all possible n-braids forms a group called the n-braid group, B_n. The law of composition for two braids is to match up the ends of the strings on the first braid to the beginnings of strings on the second braid. The identity element is simply the braid formed by letting all strings run parallel with no crossings. The inverse of any braid is its mirror image with the face of the mirror perpendicular to the strings. Two braids are considered equal if one can be obtained from the other by sliding crossings past one another and canceling inverses without adding or removing any other crossings. Examples of composition, inverse, and equality are given in Figure 2.

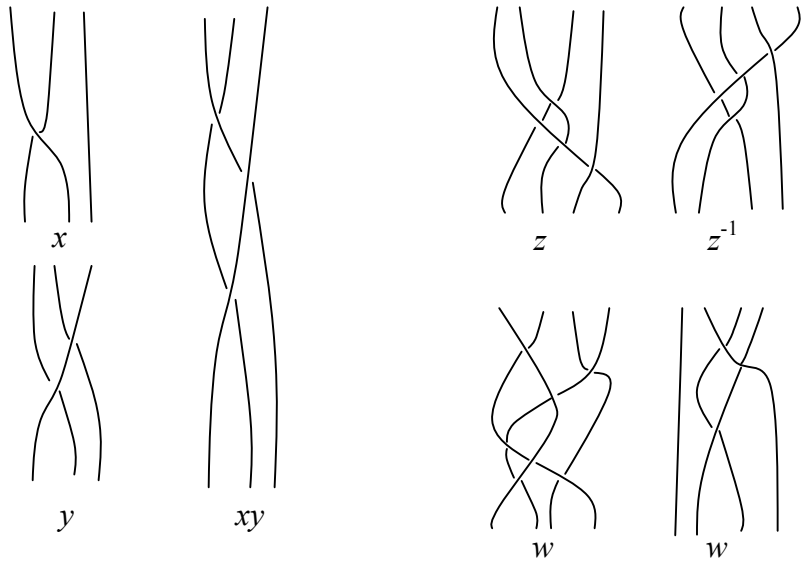

Figure 2: Composition, inversion, and equality.

With this basic understanding, some remarks about braid groups can immediately be made. First, the 1-braid group is isomorphic to the trivial group. Also, the 2-braids are isomorphic to the integers under addition, where a positive integer k is equivalent to k half-twists of the pair of strings. Likewise $-k \in \mathbb{Z}$ is equivalent to k half-twist with the opposite string crossing over the top of each half-twist. For any n, the infinite set of n-braids can be collapsed onto the finite group of permutations of n elements, S_n, as follows. If $i \in [1, n]$ is the position of a string at the top of the braid, then $\pi(i)$ is the position of the string at the bottom of the braid. As noted in [1], B_n, is "a resolution of the permutation group," S_n, in which the path between i and $\pi(i)$ is specified. The mapping of braids into permutations is thus surjective and plays an important role in the word problem through a bijection with a subset of the braids.

Artin Representation

In order to discuss further properties of braid groups, it is necessary to introduce a representation that allows easier manipulation. In the first work on braid groups [2], Emil Artin[1] represented the n-braid group with n-1 generators (plus the identity e), denoted σ_i for $i = 1,2,\ldots n\text{-}1$, and the defining relationships:

$$\sigma_i \sigma_j = \sigma_j \sigma_i \qquad |i-j| > 1 \qquad (1)$$

$$\sigma_i \sigma_j \sigma_i = \sigma_j \sigma_i \sigma_j \qquad |i-j| = 1 \qquad (2)$$

The *Artin generators*, as they are now known, also have a very nice geometric interpretation. The generator σ_i is the braid formed by crossing strings i and (i+1). In this report, the convention will be to pass string i under string (i+1) for σ_i and to pass string i

[1] The author of the book *Algebra* used by the University of Wisconsin Mathematics Department is Michael Artin, Emil's son.

over string (i+1) for σ_i^{-1}. With this representation, the braids in Figure 2 may be labeled: $x = \sigma_1^{-1}$, $y = \sigma_2\sigma_1$, $xy = \sigma_1^{-1}\sigma_2\sigma_1$, $z = \sigma_2^{-1}\sigma_1^{-1}\sigma_2^{-1}\sigma_3$, $z^{-1} = \sigma_3^{-1}\sigma_2\sigma_1\sigma_2$, $w = \sigma_1^{-1}\sigma_3\sigma_2^{-1}\sigma_1^{-1}\sigma_2\sigma_3^{-1}\sigma_1 = \sigma_2^{-1}\sigma_3^{-1}\sigma_2$. As shown in the last example, the defining relationships can be expanded by inversion and some thought to include:

$$\sigma_i^{-1}\sigma_j^{-1} = \sigma_j^{-1}\sigma_i^{-1} \qquad |i-j| > 1$$
$$\sigma_i\sigma_j^{-1} = \sigma_j^{-1}\sigma_i$$

$$\sigma_i^{-1}\sigma_j^{-1}\sigma_i^{-1} = \sigma_j^{-1}\sigma_i^{-1}\sigma_j^{-1} \qquad |i-j| = 1$$
$$\sigma_i^{-1}\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j^{-1}$$
$$\sigma_i^{-1}\sigma_j^{-1}\sigma_i = \sigma_j\sigma_i^{-1}\sigma_j^{-1}$$

It is immediately obvious that there are many ways to write the same braid using the Artin generators and the relations above. Furthermore, it is not always obvious if two words written in the Artin generators represent the same braid or different braids (the word problem). For example, it is not obvious that $\sigma_1^{-1}\sigma_3\sigma_2\sigma_3\sigma_1 = \sigma_3\sigma_2\sigma_1\sigma_2^{-1}\sigma_3$, but it is certainly true. Fortunately, there is a well known unique decomposition (first discovered by Garside [3] and refined by Thurston and others [4]) known as (Artin) left-canonical form. A few preliminaries are necessary to establish this representation, including the permutation braids, positive braids, the fundamental braid, starting and finishing sets, and left-weighted decomposition.

As illustrated above, every n-braid defines a permutation on n objects. We can write this as a surjection from B_n to S_n: $\varphi(b) = \pi$. This mapping can be made bijective between a subset $\hat{S}_n \subset B_n$ and S_n. By constructing φ^{-1}, this subset will be made clear. For each $\pi \in S_n$, draw n dots in a horizontal line labeled 1, 2, … n from left to right. Draw a similar line of dots below and parallel to this line labeled the same way. Now starting with dot n in the top line, draw a string to the dot labeled $\pi(n)$ in the lower line. Continue by connecting dot n-1 in the top line to dot $\pi(n$-1) in the bottom row with a string, crossing under the first string if a crossing occurs. Repeat until all dots are connected, remembering always to pass new strings under existing strings. Some examples are illustrated below. The image of this mapping from the n-symmetric group to the n-braid group defines the subset mentioned above: $\hat{S}_n = \varphi^{-1}(S_n)$. An element of this subset is called a *permutation braid* and is, in a sense, the simplest of all braids that map to the corresponding permutation. Notice that a braid is an element of $\hat{S}_n$ if and only if it has at most a single crossing between any two pairs of strings, and, in any crossing, the string starting on the left passes under the string starting on the right. Such a crossing is termed a positive crossing.

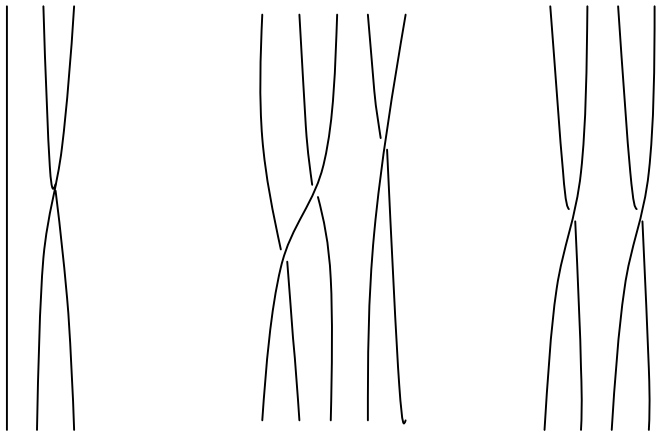

Figure 3: Some permutation braids.

The permutation braids lead to the notion of positive braids: a braid is said to be positive if it can be written as a product of the generators σ_i raised only to positive powers. Braid *y* in Figure 2 and all the braids in Figure 3 are examples of positive braids. Keeping in mind that a braid is considered to start at the top and end at the bottom, positive braids have the following geometric interpretation. Every crossing in a positive braid (when all possible cancellations are made, i.e. the braid is pulled tight) has the string starting on the left side passing under the string starting on the right side. The positive braids form a semigroup called B_n^+ [4]. By definition, the permutation braids (except the identity) are positive. There is one very important positive braid known as the *fundamental n-braid*, Δ_n. This braid is show for $n = 4$ in Figure 4. The fundamental braid can be written with $n(n\text{-}1)/2$ Artin generators as: $\Delta_n = (\sigma_{n-1}\sigma_{n-2}\ldots\sigma_1)(\sigma_{n-1}\sigma_{n-2}\ldots\sigma_2)\ldots\sigma_{n-1}$. Geometrically, the fundamental braid is obtained by lifting the bottom ends of the identity braid and flipping (right side over left) while keeping the ends of the strings in a line. Flipping in the other direction gives Δ_n^{-1}. Notice the fundamental braid is the permutation braid in which all pairs of strings cross once.

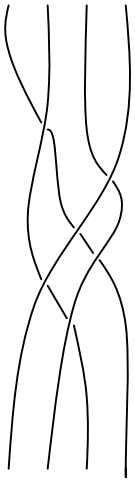

Figure 4: Fundamental braid, Δ_4

The fundamental braid has three useful properties. First, Δ_n^2 commutes with every other element in B_n. In other words, Δ_n^2 is an element of the center of B_n: $\Delta_n^{2k} \in Z(B_n)$, $k \in \mathbb{Z}$. Second, odd powers of the fundamental braid "almost" commute with every element of B_n. That is, $\Delta_n \sigma_i = \sigma_{n-i} \Delta_n$ for all σ_i. For simplicity in later explanations, let $\tau(x) = \Delta_n^{-1} x \Delta_n = x'$ for all $x = \sigma_a \sigma_{b \ldots} \sigma_c$ in B_n where x' = $\sigma_{n-a} \sigma_{n-b \ldots} \sigma_{n-c}$. Finally, some thought reveals $\Delta_n = \sigma_i A_i = B_i \sigma_i$ for all $i = 1, 2, \ldots n\text{-}1$ where A_i and B_i are permutation braids [1].

Each positive braid P has a starting $S(P)$ and finishing set $F(P)$ defined as follows:

$$S(P) = \left\{ i \mid P = \sigma_i P' \quad for \quad some \quad P' \in B_n^+ \right\}$$
$$F(P) = \left\{ i \mid P = Q' \sigma_i \quad for \quad some \quad Q' \in B_n^+ \right\}$$

For example, $S(\Delta_4) = F(\Delta_4) = \{1, 2, 3\}$ by the third property of fundamental braids, $S(e) = F(e) = \{\}$. Also, for the braid y in Figure 2, $S(y) = \{2\}$ and $F(y) = \{1\}$.

With the notion of starting and finishing sets, we can define left-weighted decomposition of a positive braid $P \in B_n^+$.

$$P = A_1 P_1 \quad A_1 \in \hat{S}_n \quad P_1 \in B_n^+ \cup \{e\} \quad F(A_1) \supset S(P_1)$$

The meaning of this decomposition and the fact that it is unique become clear with some thought about the geometry of the situation. Starting at the top of a given positive braid, move down past (necessarily positive) crossings until a pair of strings cross for a second time. Stop here before this second crossing (if no pairs cross twice, this will be the end of the braid). The braid up to this point will be a permutation braid, A_1. The rest of the braid is P_1. The condition that the starting set of P_1 be a subset of the finishing set of A_1 means that no two strings in A_1 contain a full-twist, geometrically speaking. This guarantees that A_1 is a permutation braid.

The promised unique Artin left-canonical form is summarized in a theorem due to Elrifai and Morton [5]:

Theorem 1: For every $W \in B_n$, there is a unique decomposition given by:

$$W = \Delta^u A_1 A_2 \ldots A_l \quad u \in \mathbf{Z} \quad A_i \in \hat{S}_n \setminus \{e, \Delta\} \tag{3}$$

The avoid confusion later, this decomposition will be referred to as Artin canonical form. The proof can be constructed using the machinery given above as follows. For any $W \in B_n$ written in the Artin generators, first replace all negative powers of any σ_i with $\Delta_n^{-1} B_i$, where B_i, is a permutation braid. This is possible due to the third property of the fundamental braid. Next, move all occurrences of Δ_n and Δ_n^{-1} to the extreme left using the fact that even powers of the fundamental braid commute with every element and odd powers "almost" commute as described above. At this point, the word consists of the fundamental braid raised to a power, followed by a string of Artin generators raised to

positive powers only, i.e. a positive braid. This positive braid has a unique left-weighted decomposition into permutation braids. Since the fundamental braid and the identity are permutation braids, they could be part of this decomposition. Collecting fundamental braids to the extreme left as before and deleting identity braids yields the unique Artin canonical form of W. Note that a braid is not positive if and only if its Artin canonical form has the fundamental braid raised to a negative power (i.e. $u < 0$ in (3)).

Implementations of braid group systems using the Artin representation will be discussed in Section 3.

Band Representation

In 1998, Birman, Ko, and Lee introduced an alternative representation of the braid groups using what are called the *band generators* [3]. Though a bit more difficult to visualize, their method allows some computational advantages over the Artin presentation. The band generators are a generalization of the Artin generators, so many properties of the latter carry through to the former with slight modifications in some cases.

Whereas each Artin generator represents a transposition of adjacent strings i and i+1, each band generator represents a transposition of any two strings i and j. Specifically, the generator a_{ts}, where $n \geq t > s \geq 1$, represents the braid formed by lifting strings t and s above all the others, crossing string t over string s, and then setting the strings down again. Figure 5 gives a few examples. The band generators are related to the Artin generators by the formula: $a_{ts} = (\sigma_{t-1}\sigma_{t-2}\ldots\sigma_{s+1})\sigma_s(\sigma_{s+1}^{-1}\ldots\sigma_{t-2}^{-1}\sigma_{t-1}^{-1})$. If $t = s + 1$, then $a_{ts} = a_{(s+1)s} = \sigma_s$, so the Artin generators are indeed a subset of the band generators. Also, it is important to note the condition on the subscripts and how it is related to inversion. The inverse of a_{ts} is *not* written as a_{st}, but is instead a_{ts}^{-1}. A generator with the first subscript smaller than the second is meaningless. Also, the identity is still denoted e.

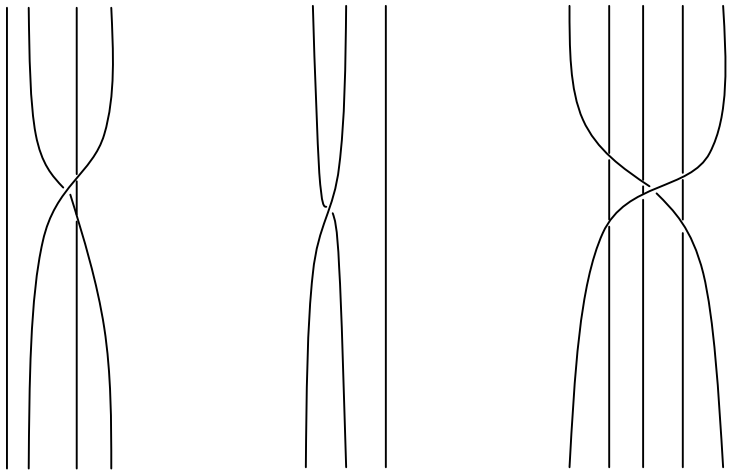

Figure 5: Band generators.

As with Equations (1) and (2) involving the Artin generators, the band generators have two defining relationships, where it is important to note the conditions on t, s, r, and q:

$$a_{ts}a_{rq} = a_{rq}a_{ts} \qquad \text{if } (t-r)(t-q)(s-r)(s-q) > 0 \qquad (4)$$

$$a_{ts}a_{sr} = a_{tr}a_{ts} = a_{sr}a_{tr} \qquad \text{for all } t, s, r \text{ with } n \geq t > s > r \geq 1 \qquad (5)$$

These relationships may be proven by drawing the various cases. In [3], it is shown that these defining relationships imply those of (1) and (2), so the band generators are a faithful representation of the braid group as expressed by the Artin generators.

The band generators for the n-braid can be related to a subset of the permutations of n elements (rather than all the permutations). This subset contains permutations consisting of products of *parallel descending cycles*. A cycle is called descending if it is of the form $(t_j, t_{j-1}, \ldots, t_1)$ where $t_j > t_{j-1} > \ldots > t_1$. Two cycles $(t_j, t_{j-1}, \ldots, t_1)$ and $(s_i, s_{i-1}, \ldots, s_1)$ are called parallel if $(t_a - s_c)(t_a - s_d)(t_b - s_c)(t_b - s_d) > 0$ for all $1 \leq a < b \leq j$ and $1 \leq c < d \leq i$. For any permutation π that is a product of parallel descending cycles, there is a corresponding braid in the band generators given by $a_{t_j t_{(j-1)}} a_{t_{(j-1)} t_{(j-2)}} \ldots a_{t_2 t_1}$. Braids of this type are canonical factors for the band generator representation. For a given n, the number of canonical factors is the n^{th} Catalan number $C_n = (2n)!/(n!(n+1)!)$ [3]. For $n \geq 3$, $C_n < n!$, so there are fewer canonical factors in the band generators than in the Artin generators.

As in the Artin presentation, positive braids are those which can be described by positive powers of the generators. It should be noted, however, that although a positive braid in the Artin generators is positive in the band generators, a positive braid in the band generators is not necessarily positive in the Artin generators.

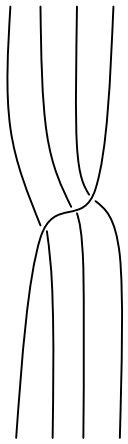

Figure 6: Fundamental braid, δ_4.

There is a fundamental braid in the band generators, denoted δ, formed by crossing string n over all other strings to the first position (see Figure 6). It is given by $\delta_n = a_{n(n-1)} a_{(n-1)(n-2)} \ldots a_{21} = \sigma_{n-1}\sigma_{n-2}\ldots\sigma_1$. Note that $\Delta_n^2 = \delta_n^n$ is an element of the center of B_n. Like Δ_n, δ_n "almost" commutes with every element of B_n as well, but in a different way. For $t > s > 1$, $\delta_n a_{ts} = a_{(t-1)(s-1)}\delta_n$. When $t > s = 1$, $\delta_n a_{t1} = a_{(t-1)n}\delta_n$. As above, future discussion will

benefit from the definition: $\tau(x) = \delta_n^{-1} x \delta_n = x'$ for $x = a_{ab} \ldots a_{cd}$ and $x' = a_{(a+1)(b+1)} \ldots a_{(c+1)(d+1)}$. Although the same symbol τ is used for conjugation by the inverse of the fundamental braid in both the Artin representation and the band generator representation, there will be no ambiguity in future use of τ, because it will only be used when either one of the two operations makes sense in its place.

The band generators allow a unique left-canonical form analogous to the Artin canonical form described above. That this decomposition is always possible and unique follows from similar arguments to those given in the proof of Theorem 1 above with some additional details which are omitted for brevity. The details were worked out in [3], which provides Theorem 2:

Theorem 2: For every $W \in B_n$, there is a unique decomposition given by:

$$W = \delta^u A_1 A_2 \ldots A_l \quad u \in \mathbf{Z} \quad A_i \in \{canonical \quad factors\}$$

Note that the canonical factors of Theorem 2, which are isometric to products of parallel descending cycles, are very different braids than the canonical factors of Theorem 1, the permutation braids. This decomposition will be known as band canonical form.

Some interesting comparisons between the Artin and band representations can be made. First, notice there are n-1 Artin generators versus $n(n-1)/2$ band generators. Also Δ_n is composed of $n(n-1)/2$ Artin generators while δ_n is composed of n-1 generators in either presentation. Since the band generators contain the Artin generators, a word is never longer when written with band generators than with Artin generators. Also, even though there are fewer canonical factors in the band canonical form, words tend to be shorter. One explanation is that the band canonical factors are less restrictive in a sense.

There is also a normal form developed by Dehornoy. Although performing the Dehornoy reduction algorithm on a braid written in Artin generators seems to be faster than finding the band canonical form, there does not appear to be a proof that it is always faster [6].

2. Theoretical Advantages in the Braid Groups

Cryptosystems in which parties can only share information through a public medium rely on so called "one-way" functions to hide private information. A one-way function is computationally easy to apply in one direction, but very difficult (in terms of the operations required) to invert without some additional information, i.e. the key. The classical example of such a function is multiplication of large primes. A similar example is the discrete log problem in Diffie-Hellman key exchange.

The Braid groups offer a variety of potentially difficult problems. The reference [1] lists some examples which are paraphrased below with additional discussion.

Conjugate Search Problem - Given a pair $(x, y) \in B_n \times B_n$ where $y = axa^{-1}$, find a.

The equivalence classes of conjugates provide a sort of structure to non-abelian groups which is often not obvious. As such, problems based on conjugates are often considered for cryptosystems. A brute force attack to this problem searches over (seemingly infinitely many) braids until a conjugator is found. It a practical system, the brute force attack must try only all the braids up to a certain canonical length. If $|a|$ is the canonical length (i.e. number of canonical factors) of a in either canonical form, then there are at least $(\lfloor (n-1)/2 \rfloor !)^{|a|}$ candidates [1]. It should be mentioned that the conjugate search problem for any two elements of B_n can be reduced to the conjugate search problem in B_n^+ [4]. Another attack (discussed in Section 4) solves a related problem in polynomial time by transforming the braid group to a linear group. It is shown in that work that it is usually good enough to find a' (not necessarily equal to a) which conjugates a [7].

Generalized Conjugate Search Problem - Given a pair $(x, y) \in B_n \times B_n$ where $y = axa^{-1}$ and $a \in B_m$ for $m < n$, find a.

This problem differs from the first in that a comes from a subgroup of B_n and so a brute force attack requires fewer iterations.

p^{th} Root Problem - Given $(x, p) \in B_n \times \mathbb{Z}$ such that $x = y^p$ for some $y \in B_n$, find $y' \in B_n$ such that $x = y'^p$.

This problem is essentially to find a repeating pattern in a braid. Though it may seem easy to spot such a pattern given a picture of a braid, this could prove difficult for braids in a canonical form, especially for large n.

Markov Problem - Given $y \in B_n$ where y is conjugate to a braid $w\sigma_{n-1}^{\pm 1}$ with $w \in B_{n-1}$, find $(z, x) \in B_n \times B_{n-1}$ such that $zyz^{-1} = x\sigma_{n-1}^{\pm 1}$.

In this problem, y is conjugate to a braid that is "mostly" in B_{n-1}. This problem is similar to the conjugate search problem, but now x is unknown and comes from a specific subgroup.

Conjugate Decision Problem - Given a pair $(x, y) \in B_n \times B_n$, determine whether or not x and y are conjugate.

This problem is similar to the conjugate search problem only easier because the conjugator a need not be specified. In fact, this problem is useful in digital signature schemes precisely because it can be solved. In such a scheme, a signature is considered valid if certain braids are conjugates. One the other hand, if the conjugate could be found, a forger could produce a valid signature. A method for solving this problem is discussed below in Section 4.

These problems can be used to set up cryptographic schemes such as key exchange and authentication. Such schemes will be discussed in Section 4. Attacks on these schemes can be found in Section 5.

3. Computational Advantages

In order to use a promising cryptographic scheme based on a group, there must be efficient ways to represent group elements and perform operations on these elements in a computer. Fortunately, this is possible for the braid groups.

Since a braid has a unique decomposition in either presentation, this decomposition provides a nice way to store a particular braid in the memory of a computer. To store a braid in a computer's memory, the power of the fundamental braid (u in Theorem 1 or 2) must be stored, as well as the sequence of canonical factors following it (A_i). The power of the fundamental braid can be stored as an integer. There is of course some limit to the size of this integer based on the computer's memory, but typically speed rather than memory is the limiting factor in cryptosystems. Now consider storing a canonical factor. Using the fact that there are $n!$ canonical factors in the Artin representation and C_n canonical factors in the band generator representation, each canonical factor could be stored by an integer between one and $n!$ for the former case and one and C_n for the latter case. In fact it is better (in terms of implementing algorithms) to store canonical factors as arrays instead.

In the Artin representation, the canonical factors are the permutation braids. Thus to store a canonical factor, an array representing the permutation may be used. Let A be an array representing a permutation table. If a permutation sends i to $\pi(i)$, then $A(i) = \pi(i)$.

In the band generators, the canonical factors are products of parallel descending cycles. These can be stored as an array of length n called a descending cycle decomposition table. Suppose X is such a table. Then $X(i)$ is the maximum in the cycle containing i.

Now consider some operations on canonical factors alone. Assume all canonical factors are stored as either permutation tables or descending cycle decomposition tables. Conversion between the two can be accomplished in $\mathcal{O}(n)$ operations according to [8]. The same reference claims that comparison, products, and inverses of canonical factors can be done in $\mathcal{O}(n)$ operations as well. While comparison can be done in either representation, products and inverses of canonical factors are easiest to compute on permutation tables. Although [8] claims linear running time for many of these operations, it is not always shown how this is accomplished. Still, it seems likely that these claims hold with possible modifications to the given algorithms. Also, there are a few errors to Algorithms 1 and 2 presented in [8], but with some modifications, they run correctly in linear time. Please see the Appendix for details.

Moving on to operations on entire braids, recall that in both representations, the particular fundamental braid "almost" commutes with any other braid. Letting D be either Δ_n or δ_n

depending on the representation used, this almost commutativity was defined as $xD = D\tau(x)$. Probably the most basic operation required (aside from comparison) is the inversion of a group element. If the element consists of l canonical factors, inversion can be done in $\mathcal{O}(ln)$ time using the following formula [8]:

$$\left(D^u A_1 A_2 \ldots A_l\right)^{-1} = D^{-u-l}\tau^{-u-l}\left(A_l^{-1}D\right)..\tau^{-u-l}\left(A_2^{-1}D\right)\tau^{-u-l}\left(A_1^{-1}D\right)$$

The next most basic operation is composition of elements. This can be done in $\mathcal{O}(ln)$ time as well where l is the length of the first element. This can be seen by the formula [8]:

$$\left(D^u A_1 A_2 \ldots A_l\right)\left(D^v A_1 A_2 \ldots A_m\right) = D^{u+v}\tau^v\left(A_1\right)\tau^v\left(A_2\right)..\tau^v\left(A_l\right)A_1 A_2 \ldots A_m$$

Notice in both formulas that powers of τ may be reduced modulo 2 in the Artin generators or modulo n in the band generators.

Of course, it is desirable that the element obtained by composing two other elements be in left-canonical form. According to [8], this can be done in $\mathcal{O}(l^2 n \log n)$ in the Artin representation and $\mathcal{O}(l^2 n)$ in the band generators. Also, comparison of braids in canonical form takes $\mathcal{O}(ln)$ operations since each canonical form can be compared in $\mathcal{O}(n)$ time. If the two braids are not in canonical form, some savings are possible by comparing factors while converting to canonical form simultaneously [8].

4. Cryptographic Schemes

The first cryptographic schemes explicitly using braid groups appeared in about 1999 – 2000. Two main key exchange protocols using the braid groups were proposed: one by Ko, S J Lee, Cheon, Han, Kang, and Park at Crypto 2000 [1] and one by I Anschel, Anschel, and Goldfeld [10]. Ko, et al's work also includes a public key cryptosystem. In a later work, Ko, Choi, Cho, and J W Lee propose the Braid Signature Scheme, a method of digital verification/authentication [9]. These schemes are considered in this section.

Central to the schemes in [1] discussed below is the division of B_n into two subgroups $LB_l \approx B_l$ and $RB_r \approx B_r$ where $n = l + r$. The subgroup LB_l is obtained by using only the leftmost l strings to form braids leaving the other r strings alone, while RB_r uses only the right most r strings. Since these subgroups use no common strings, elements of LB_l commute with elements of RB_r. This commutativity plays a key role in the protocols described.

The users of the cryptographic schemes will be Alice (A) and Bob (B). The attacker will be Oscar (O).

Key Agreement (Ko, et al)

Public Information: $l, r, B_{l+r}, x \in B_{l+r}$

Exchange: A chooses a secret $a \in LB_l$ and publishes $y_a = axa^{-1}$.
B chooses a secret $b \in RB_r$ and publishes $y_b = bxb^{-1}$.
A computes the secret key $K = ay_ba^{-1} = abxb^{-1}a^{-1} = abxa^{-1}b^{-1}$.
B computes the secret key $K = by_ab^{-1} = baxa^{-1}b^{-1} = abxa^{-1}b^{-1}$.

Once both parties know K, they can use it to encode secret messages. In order for Oscar to break the code, he must find either a or b given x, y_a, and y_b (allowing him to synthesize K). Clearly this is an instance of the generalized conjugate search problem. Also, note the similarity to Diffie-Hellman key exchange.

In the Anschel et al scheme, each user is assigned a list of complicated braids that are used to generate subgroups. Because of the complexity of the braid groups, it is unlikely that these subgroups will generate the entire group B_n. The notation $\langle s_i \rangle$ is used to denote the subgroup generated by s_i.

Key Agreement (Anschel, et al)

Public Information: B_n, subgroups $S_A = \langle s_1, s_2, \ldots, s_m \rangle$, $S_B = \langle t_1, t_2, \ldots, t_n \rangle$

Exchange: A chooses a secret $a = s_{a1}s_{a2}\ldots s_{ak} \in S_A$ and publishes $a^{-1}t_1a$, $a^{-1}t_2a$, $\ldots$ $a^{-1}t_na$.
B chooses a secret $b = t_{b1}t_{b2}\ldots t_{bl} \in S_B$ and publishes $b^{-1}s_1b$, $b^{-1}s_2b$, $\ldots$, $b^{-1}s_mb$.
A computes secret key $K = a^{-1}(b^{-1}s_{a1}bb^{-1}s_{a2}b\ldots b^{-1}s_{ak}b) = a^{-1}b^{-1}ab$.
B computes secret key $K = (b^{-1}(a^{-1}t_{b1}aa^{-1}t_{b2}a\ldots a^{-1}t_{bl}a))^{-1} = (b^{-1}a^{-1}ba)^{-1} = a^{-1}b^{-1}ab$.

In this case, Oscar must find a or b given $a^{-1}t_1a$, $a^{-1}t_2a$, $\ldots$ $a^{-1}t_na$, $b^{-1}s_1b$, $b^{-1}s_2b$, $\ldots$, $b^{-1}s_mb$. This seems at least as easy as breaking Ko, et al's scheme because if Oscar could find a given x and $a^{-1}xa$, he could find a using just one pair $(s_i, a^{-1}s_ia)$. In fact, there is an attack that relies on multiple conjugate pairs (with the same conjugator), which will be discussed in Section 5.

Public Key Cryptosystem

Public Information: l, r, B_{l+r}, conjugates $(x, y) \in B_{l+r} \times B_{l+r}$, hash function H

Private Key: $a \in LB_l$ such that $y = axa^{-1}$.

Encryption: Choose $b \in RB_r$. Send (c, d) where $c = bxb^{-1}$ and $d = H(byb^{-1}) \oplus m$.

Decryption: Calculate $m = H(aca^{-1}) \oplus d$. To see that this equation holds, note that:

$$
\begin{aligned}
H\left(aca^{-1}\right) \oplus d &= H\left(abxb^{-1}a^{-1}\right) \oplus H\left(byb^{-1}\right) \oplus m \\
&= H\left(abxb^{-1}a^{-1}\right) \oplus H\left(baxa^{-1}b^{-1}\right) \oplus m \\
&= H\left(abxb^{-1}a^{-1}\right) \oplus H\left(abxb^{-1}a^{-1}\right) \oplus m \\
&= m
\end{aligned}
$$

Here, $\oplus$ denotes the bitwise exclusive or operation. The hash function takes a braid as input and gives a fixed length binary representation as output. Oscar's job in this case is to find a given x and y (or equivalently b given c and x), the generalized conjugate search problem.

The Braid Signature Scheme (BSS) is a bit more difficult to present. In this situation, Alice wants to send a message to Bob. Bob wants to make sure that the message is from Alice and not from Oscar. The message may be public or private; the BSS does nothing to hide the message, but this message could already be encoded by another scheme. The hash function in this scheme outputs a braid of fixed length.

Braid Signature Scheme

Public Information:	conjugate pair (x, x')
Private Key:	a such that $x' = a^{-1}xa$
Signing:	A chooses a random braid b and calculates $\alpha = b^{-1}xb$, $y = H(m \oplus \alpha)$, $\beta = b^{-1}yb$, $\gamma = b^{-1}aya^{-1}b$. A sends the message, m, and (α, β, γ) to B.
Verification:	B calculates $y = H(m \oplus \alpha)$ and accepts the message if and only if $\alpha \sim x$, $\beta \sim \gamma \sim y$, $\alpha\beta \sim xy$, and $\alpha\gamma \sim x'y$.

This scheme requires that Bob solve the conjugate decision problem. This can be accomplished by finding the Burau representation, Φ, of the braids in question. For a given n-braid x, $\Phi(x)$ is an n-1 by n-1 matrix over polynomials in t (specifically the Laurent polynomial ring) [6, 9]. The Alexander polynomial, $\det(\Phi(x) - I)$, is invariant to conjugation, i.e. $\det(\Phi(x) - I) = \det(\Phi(axa^{-1}) - I)$, and has degree at most the length of x. Two braids are declared conjugate if their Alexander polynomials agree at sufficiently many points [9].

The key to this signature scheme is that the conjugate decision problem is relatively easy, while the (generalized) conjugate search problem is hard. To forge a message, Oscar must learn the secret key a given x, x', α, β, γ, and m. Though he can feasibly test whether a pair of these braids is conjugate, he can not easily find the specific conjugators.

5. Attacks to Braid Group Cryptosystems

As with any cryptosystem, brute force attacks exist. The common solution to avoid these is to make the algebraic backbone behind the cryptosystem sufficiently complex (e.g.

large) that brute force attacks would take too long to be of any use. In braid groups, this is accomplished by increasing the index of the group n, or by increasing the length, i.e. number of canonical factors, of specific braids. This section focuses on methods to reduce the time required of a brute force attack for a given n.

The attacks to the cryptosystems presented here are essentially attacks on the (generalized) conjugate search problem. As mentioned, brute force attacks can find a conjugator by trying all possible braids up to a fixed word length. To make a useful cryptosystem based on conjugates, there typically must be additional constraints on the conjugators or base braids. This added structure could lead to quicker attacks.

In the Ko, et al's key agreement and public key cryptosystems, the n-braid is divided into two sub-braids B_l and B_r where $n = l + r$. It will be assumed that n is even and $l = r = n/2$. This assumption seems justified because the brute force attack is harder on a braid with more strings. If one sub-braid has fewer strands than the other, the attacker, Oscar, can focus on this sub-braid only. One may argue that in the public key cryptosystem, l should be larger than r because Oscar has more time to work with x and y than with c and x (because he must wait for a user to transmit c). This might be the case if x and y remain constant for long periods of time, but a practical system would likely change keys periodically, say once a day. In either case, the ideas behind attacking the $l = r$ case can be adapted.

The creators of these two schemes were aware of some potential pitfalls and enumerated these in [1]. They give essentially three conditions to avoid when choosing $x \in B_n$. These attacks are described with regard to the key agreement scheme, though the ideas apply to the public key cryptosystem as well. First, x should not reduce to x_1x_2z, where $x_1 \in LB_{n/2}$, $x_2 \in RB_{n/2}$, and z commutes with $LB_{n/2}$ and $RB_{n/2}$. If so, Oscar could find x^{-1} and then use y_a and y_b as follows: $y_a x^{-1} y_b = (ax_1x_2za^{-1})x^{-1}(bx_1x_2zb^{-1}) = (ax_1a^{-1})x_2z(z^{-1}x_2^{-1}x_1^{-1})x_1(bx_2b^{-1})z = (ax_1a^{-1})(bx_2b^{-1})z = abx_1x_2za^{-1}b^{-1} = baxa^{-1}b^{-1} = K$. Second, suppose Oscar could find any pair $(a', a'') \in LB_{n/2} \times LB_{n/2}$ such that $y_a = a'xa''$. Clearly this is easier than the conjugate search problem, which is a subset. Then $a'y_ba'' = a'bxb^{-1}a'' = ba'xa''b^{-1} = by_ab^{-1} = K$. One way to avoid this is to have $xcx^{-1} \notin B_{n/2}$ for all nontrivial $c \in B_{n/2}$. Third, since every n-braid leads to a permutation on n objects, Oscar could find $\varphi(x)$, $\varphi(y_a) \in S_n$. Since $\varphi(y_a) = \varphi(axa^{-1}) = \varphi(a)\varphi(x)\varphi(a)^{-1}$, Oscar could potentially deduce $\varphi(a)$, or a subset of S_n containing $\varphi(a)$, and concentrate his search on the image of this subset. (Obviously, the same can be done with b.) To avoid this problem, choose only $x \in B_n$ such that $\varphi(x) = e$. Then $\varphi(y_a) = \varphi(y_b) = e$ as well regardless of a.

In Anschel, et al's key agreement scheme, the use of multiple conjugate pairs with the same conjugator allows the so called length based attack first proposed by Hughes and Tannenbaum and applied in [11]. The basic idea behind the attack is that composing to braids to form a longer braid typically results in a braid of greater length in canonical form, especially if the braids are complicated. So $a^{-1}t_ia$ is usually longer than t_i. Since a is composed of elements $s_1, s_2, \ldots s_m$, each $a^{-1}t_ia = (s_{ak}^{-1}\ldots s_{a2}^{-1}s_{a1}^{-1})t_i(s_{a1}s_{a2}\ldots s_{ak})$. To attack the system, the attacker, Oscar, uses the set of m generators to form $s_j^{-1}a^{-1}t_ia_1s_j$ for each i and j. If $j = s_{ak}$, the length of the braid $s_j^{-1}a^{-1}t_ia_1s_j$ is likely to be less than the length

of $a^{-1}t_i a$ for a particular i, though this is not guaranteed. If a particular choice of s_j shortens the braid for many values of i, it is likely to be correct. The process can be repeated until all k factors are found. This attack is linear in both k and l. On the other hand, increasing k and l seem to make the length property more likely, i.e. it is more likely that aba is longer than b.

Recently, Cheon and Jun published a polynomial time algorithm for solving Diffie-Hellman type conjugate search problem. That is, it requires knowledge of x, $y_a = axa^{-1}$, and $y_b = bxb^{-1}$, to find $abxa^{-1}b^{-1}$. The algorithm is based on a the Lawrence-Krammer representation which takes B_n to the linear group $GL_{n(n-1)/2}(\mathbb{Z}[t^{\pm 1},q^{\pm 1}])$, which has been proven faithful to the braid group for any n "several times in independent ways by several authors" [7]. Though the computations can be quite messy, the algorithm performs in approximately $\mathcal{O}(2^{-2}l^3n^{13.2}\log n)$ time. By using the Lawrence-Krammer representation, any braid cryptosystem could be potentially attacked, though the attack in [7] does not solve the pure conjugate search problem.

6. Concluding Remarks

In the four years since the braid groups were proposed as a source of cryptographic schemes, systems have been created, modified, and broken. The non-commutative braid groups seem to be a good basis for cryptographic systems because of their complexity. There are several normal forms for writing a braid and easy ways to perform inversions, group operations, and other functions in a computer. A good cryptosystems, however, must be robust against attacks in the Lawrence-Krammer representation.

Appendix

In [8], multiple algorithms for performing typical braid group operations are given. Algorithms 1 and 2 of that work, however, are flawed. This algorithms are supposed to convert between Artin canonical form and band canonical form. The main problem with this idea is that there is no isomorphism between the two sets. The number of band canonical factors, *Cn*, is less than the number of Artin canonical factors, $n!$, for $n > 2$, so transforming between the two is tricky from the start. Also, even though a braid from the Artin canonical factors and a braid from the band canonical factors may map to the same permutation, they are not necessarily the same braid. Thus, they will have different inverses and combine differently with other braids. Still it seems that conversion between the two may be possible in linear time, but in some cases a canonical factor from one representation may need to map to two or more canonical factors in the other representation. Due to the small size of $n!$, the number of input/output pairs, the problems are illustrated with $n = 3$.

Algorithm 1 converts an array representing an Artin canonical factor (i.e. permutation braid) to an array representing a band canonical factor. There are three main problems with this algorithm. Consider the input/output pairs given by the program implemented in Matlab:

Input	*Output*
[1 2 3]	[1 2 3]
[1 3 2]	[1 3 3]
[2 1 3]	[2 2 3]
[2 3 1]	[3 2 3] (*)
[3 1 2]	[3 3 3] (*)
[3 2 1]	[3 2 3] (*)

The outputs marked with (*) are problematic. The input [2 3 1] should produce an output of [3 3 3] since this braid is the fundamental braid in the band representation. This problem seems to be due to a typographical error in the algorithm which can be easily fixed. The other two errors are more fundamental. For $n = 3$, $C_n = 5$, while $n! = 6$, so there is one permutation with no corresponding band canonical factor. The braid (or permutation) represented by [3 1 2] is not a product of parallel descending cycles. Thus it cannot be converted directly to a band canonical factor. As mentioned above, there are fewer band canonical factors than Artin canonical factors for $n > 2$. Finally, the input braid represented by [3 2 1] is the fundamental braid in the Artin generators. The braid represented by the output [3 2 3] maps to the same permutation, but the strings are interlaced in a different way. The problem here is the lack of isomorphism between Artin canonical factors and band generators.

Algorithm 2 converts band canonical factors to Artin canonical factors (actually permutations representing them). Although the lack of an isomorphism between the two is a problem, the algorithm can still convert a descending cycle decomposition table to

the corresponding permutation. The pseudo-code given in [8], however, is flawed. Consider the input/output table:

Input	*Output*
[1 2 3]	[1 2 3]
[1 3 3]	[1 3 2]
[2 2 3]	[2 1 3]
[3 2 3]	[3 2 1]
[3 3 3]	[3 1 2] (*)

The last pair is incorrect. Since [3 3 3] means that the third string crosses over the other two, the permutation should be [2 3 1]. Below is the Matlab code implementing Algorithm 2 of [8] and a corrected version.

Algorithm 2 of [8]

```
n=length(X);
Z=zeros(n,1);

for c=1:n
   if Z(X(c))==0
      A(c)=X(c);
   else
      A(c)=Z(X(c));
   end
   Z(X(c))=c;
end
```

Corrected Algorithm 2

```
n=length(X);
Z=zeros(n,1);

for c=1:n
   if Z(X(c))==0
      A(c)=c;
   else
      A(c)=A(Z(X(c)));
      A(Z(X(c)))=c;
   end
   Z(X(c))=c;
end
```

Though these algorithms are flawed and there is a problem taking one representation to the other, it seems likely that there is a solution which works in linear time. How this error in translating between representations affects the remaining algorithms in [8] is not clear, though being able to convert between representations only seems to increase the speed of other algorithms by log *n* time.

Bibliography

[1] K. H. Ko, S. J. Lee, J. H. Cheon, J. H. Han, J. S. Kang, C. Park, *New Public-Key Cryptosystems Using Braid Groups*, Advances in Cryptography, Proceedings of Crypto 2000, Lecture Notes in Computer Science 1880, ed. M. Bellare, Springer-Verlag (2000), 166-183.

[2] Emil Artin, *Theory of Braids*, Annals of Math., v. 48 (1947), 101-126.

[3] J. S. Birman, K. H. Ko, and S. J. Lee, *A New Approach to the Word and Conjugacy Problem in the Braid Groups*, Advances in Mathematics 139 (1998), 322-353.

[4] D. Epstein, J. Cannon, D. Holt, S. Levy, M. Paterson, and W. Thurston, *Word Processing in Groups*, Jones & Bartlett, 1992.

[5] E. A. Elrifai and H. R. Morton, *Algorithms for Positive Braids*, Quart. J. Math. Oxford v. 45 (1994), no. 2, 479-497.

[6] I. Anshel, M. Anshel, B. Fisher, D. Goldfeld, *New Key Agreement Protocols in Braid Group Cryptography*, ed. D. Naccache, Springer-Verlag (2001), 13-27.

[7] J. H. Cheon and B. Jun, *A Polynomial Time Algorithm for theBraid Diffie-Hellman Conjugacy Problem*, Preprint, Available at http://eprint.iacr.org/2003/019/.

[8] J. C. Cha, K. H. Ko, S. J. Lee, J. W. Han, and J. H. Cheon, *An Efficient Implementations of Braid Groups*, Proc. of Asiacrypt 2001, Lexture Notes in Computer Science, Vol. 2248, Springer-Verlag, pp. 144–156, 2001.

[9] K. H. Ko, D. H. Choi, M. S. Cho, and J. W. Lee, *New Signature Scheme Using Conjugacy Problem*, Preprint, Available at http://eprint.iacr.org/2002/168/.

[10] I. Anshel, M. Anshel, and D. Goldfeld, *An Algebraic Method for Public-Key Cryptography*, Math. Res. Lett., Vol. 6, No. 3-4, pp. 287-291, 1999.

[11] D. Garber, S. Kaplan, M. Teicher, B. Tsaban, and U. Vishne, *Length-Based Conjugacy Search in the Braid Group*, Preprint, Available at http://arxiv.org/abs/math.GR/0209267.