WALNUTDSA$^{\text{TM}}$: A GROUP THEORETIC DIGITAL SIGNATURE ALGORITHM

IRIS ANSHEL, DEREK ATKINS, DORIAN GOLDFELD, AND PAUL E. GUNNELLS

Abstract. This paper presents an in depth discussion of WalnutDSA, a quantum resistant public-key digital signature method based on the one-way function E-multiplication. A key feature of WalnutDSA is that it provides very efficient means of validating digital signatures which is essential for low-powered and constrained devices. This paper presents an in-depth discussion of the construction of the digital signature algorithm, and delves deeply into the underlying mathematics that facilitates analyzing the security of the scheme. When implemented using parameters that defeat all known attacks, WalnutDSA is among the fastest quantum resistant signature verification methods; it performs orders of magnitude faster than ECC, even on low-end embedded hardware. WalnutDSA delivers a 12-25x speed improvement over ECDSA on most platforms, and a 31x speed improvement on a 16-bit microcontroller, making it an ideal solution for low-resource processors found in the Internet of Things (IoT).

Group Theoretic Cryptography, Digital Signature, E-Multiplication, Braids, Internet of Things, IoT

1. Introduction

Digital signatures provide a means for one party to create a document that can be sent through a second party and verified for integrity by a third party. This method ensures that the first party created the document and that it was not modified by the second party. Historically, digital signatures have been constructed using various number-theoretic, public-key methods like RSA, DSA, and ECDSA. These methods are inherently not very efficient when run on platforms with constrained processors (16-bit or even 8-bit), or systems with limited space or energy.

Digital signatures based on algorithmically hard problems in group theory were introduced in 2002 by Ko, Choi, Cho, and Lee [31] and in 2009 by Wang and Hu [47]. The approach of Ko, Choi, Cho, and Lee utilized a variation of the conjugacy problem in non-commutative groups as a basis for security, while Wang and Hu [47] opted for the hardness of the root problem in braid groups (see also [29]). The attacks introduced in [19], [20], [22], and [27] suggest that the schemes by Ko et al. and Wang and Hu may not be practical over braid groups in resource limited settings.

The group-theoretic one-way function E-Multiplication was first introduced in 2005 by Anshel, Anshel, Goldfeld, and Lemieux [6]. Based on a representation of the Artin Braid group, E-Multiplication enables the effective use of a non-abelian infinite group and can serve as a building block for a range of cryptographic protocols which are, by construction, quantum-resistant due to the braid group being an infinite non-abelian group. Other examples of applications of E-Multiplication include the cryptographic hash function AEHash [3], which has been implemented using very little code space on a 16-bit platform [4], and the Ironwood Meta Key agreement protocol [5].

Implementations of E-Multiplication in various instances have shown that code space is small and runtime is extremely rapid, with constructions using E-Multiplication outperforming competing methods, especially in small, constrained devices.

2. Colored Burau Representation of the Braid Group

We begin by recalling the colored Burau representation. For $N \geq 2$, let B_N denote the N-strand braid group with Artin generators $\{b_1, b_2, \ldots, b_{N-1}\}$, subject to the following relations:

$$\begin{aligned} b_i b_{i+1} b_i &= b_{i+1} b_i b_{i+1}, \qquad (i = 1, \ldots, N-2), \\ b_i b_j &= b_j b_i, \qquad (|i-j| \geq 2). \end{aligned}$$

Thus any $\beta \in B_N$ can be expressed as a product of the form

$$\beta = b_{i_1}^{\epsilon_1} \, b_{i_2}^{\epsilon_2} \, \cdots \, b_{i_k}^{\epsilon_k}, \tag{1}$$

where $i_j \in \{1, \ldots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$. Note that β is not uniquely represented by (1) since the braid group is palpably not free.

Let S_N be the group of permutations on N letters. Each braid $\beta \in B_N$ determines a permutation in S_N as follows. For $1 \leq i \leq N-1$, let $\sigma_i \in S_N$ be the i^{th} simple transposition, which maps $i \to i+1$, $i+1 \to i$, and leaves $\{1, \ldots, i-1, i+2, \ldots, N\}$ fixed. Then the map $b_i \mapsto \sigma_i$ extends to a surjective homomorphism $B_N \to S_N$. A braid is called pure if its corresponding permutation is trivial (i.e., the identity permutation). Clearly the set of pure braids coincides with the kernel of the map $B_N \to S_N$.

Let $\mathbb{F}_q$ denote the finite field of q elements, and for variables $t_1, t_2, \ldots, t_N$, let

$$\mathbb{F}_q[t_1, t_1^{-1}, \ldots, t_N, t_N^{-1}]$$

denote the ring of Laurent polynomials in $t_1, t_2, \ldots, t_N$ with coefficients in $\mathbb{F}_q$. We introduce the colored Burau representation

$$\Pi_{CB} \colon B_N \to GL\Big(N, \mathbb{F}_q[t_1, t_1^{-1}, \ldots, t_N, t_N^{-1}]\Big) \rtimes S_N.$$

For each Artin generator b_i we define the $N \times N$ colored Burau matrix $CB(b_i)$ generator as follows [38]: if $i = 1$, we put

$$CB(b_1) = \begin{pmatrix} -t_1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \vdots \\ \vdots & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{2}$$

and for $2 \leq i \leq N-1$, we define $CB(b_i)$ by

$$CB(b_i) = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & t_i & -t_i & 1 & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{3}$$

where the indicated variables appear in row i. We similarly define $CB(b_i^{-1})$ by modifying (3) slightly:

$$CB(b_i^{-1}) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & -\frac{1}{t_{i+1}} & \frac{1}{t_{i+1}} & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix},$$

where again the indicated variables appear in row i, and as above if $i = 1$ the leftmost 1 is omitted.

Recall that each b_i has an associated permutation σ_i. We may then associate to each braid generator b_i (respectively, inverse generator b_i^{-1}) a colored Burau/permutation pair $(CB(b_i), \sigma_i)$ (resp., $(CB(b_i^{-1}), \sigma_i)$). We now wish to define a multiplication of such colored Burau pairs. To accomplish this, we require the following observation. Given a Laurent polynomial $f(t_1, \ldots, t_N)$ in N variables, a permutation in $\sigma \in S_N$ can act (on the left) by permuting the indices of the variables. We denote this action by $f \mapsto {}^{\sigma}f$:

$${}^{\sigma}f(t_1, t_2, \ldots, t_N) = f(t_{\sigma(1)}, t_{\sigma(2)}, \ldots, t_{\sigma(N)}).$$

We extend this action to matrices over the ring of Laurent polynomials in the t_i by acting on each entry in the matrix, and denote the action by $M \mapsto {}^{\sigma}M$. The general definition for multiplying two colored Burau pairs is now defined as follows: given $b_i^{\pm}, b_j^{\pm}$, the colored Burau/permutation pair associated with the product $b_i^{\pm} \cdot b_j^{\pm}$ is

$$(CB(b_i^{\pm}), \sigma_i) \cdot (CB(b_j^{\pm}),\ \sigma_j) = \Big(CB(b_i^{\pm}) \cdot ({}^{\sigma_i}CB(b_j^{\pm})),\ \sigma_i \cdot \sigma_j\Big).$$

We extend this definition to the braid group inductively: given any braid

$$\beta = b_{i_1}^{\epsilon_1} b_{i_2}^{\epsilon_2} \cdots b_{i_k}^{\epsilon_k},$$

as in (1), we can define a colored Burau pair $(CB(\beta), \sigma_\beta)$ by

$$(CB(\beta), \sigma_\beta) = \\ (CB(b_{i_1}^{\epsilon_1}) \cdot {}^{\sigma_{i_1}}CB(b_{i_2}^{\epsilon_2}) \cdot {}^{\sigma_{i_1}\sigma_{i_2}}CB(b_{i_3}^{\epsilon_3})\ \cdots\ {}^{\sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_{k-1}}}CB(b_{i_k}^{\epsilon_k}),\ \ \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_k}).$$

The colored Burau representation is then defined by

$$\Pi_{CB}(\beta) := (CB(\beta), \sigma_\beta).$$

One checks that Π_{CB} satisfies the braid relations and hence defines a representation of B_N.

3. E-Multiplication

E-Multiplication was first introduced in [6] as a one-way function used as a building block to create multiple cryptographic constructions. We recall its definition here. Beginning with an ordered list of entries in the finite field (termed T-values) is defined to be a collection of non-zero field elements:

$$\{\tau_1, \tau_2, \ldots, \tau_N\} \subset \mathbb{F}_q^{\times}.$$

Given a set of T-values, we can evaluate any Laurent polynomial $f(t_1, t_2, \ldots, t_N)$ to obtain an element of $\mathbb{F}_q$:

$$f(t_1, t_2, \ldots, t_N) \downarrow_{t\text{-values}} := f(\tau_1, \tau_2, \ldots, \tau_N).$$

We extend this notation to matrices over Laurent polynomials in the obvious way.

With all these components in place, we can now define E-Multiplication. By definition, E-Multiplication is an operation that takes as input two ordered pairs,

$$(M, \sigma_0), \quad (CB(\beta), \sigma_\beta),$$

where $\beta \in B_N$ and $\sigma_\beta \in S_N$ as before, and where $M \in GL(N, \mathbb{F}_q)$, and $\sigma_0 \in S_N$. We denote E-Multiplication with a star: $\star$. The result of E-Multiplication, denoted

$$(M', \sigma') = (M, \sigma_0) \star (CB(\beta), \sigma_\beta),$$

will be another ordered pair $(M', \sigma') \in GL(N, \mathbb{F}_q) \times S_N$.

We define E-Multiplication inductively. When the braid $\beta = b_i^{\pm}$ is a single generator or its inverse, we put

$$(M, \sigma_0) \star \left(CB(b_i^{\pm}),\ \sigma_{b_i^{\pm}}\right) = \Big(M \cdot {}^{\sigma_0}\big(CB(b_i^{\pm})\big) \downarrow_{t\text{-values}},\ \ \sigma_0 \cdot \sigma_{b_i^{\pm}}\Big).$$

In the general case, when $\beta = b_{i_1}^{\epsilon_1} b_{i_2}^{\epsilon_2} \cdots b_{i_k}^{\epsilon_k}$, we define

$$(M, \sigma_0) \star (CB(\beta), \sigma_\beta) = (M, \sigma_0) \star (CB(b_{i_1}^{\epsilon_1}), \sigma_{b_{i_1}}) \star (CB(b_{i_2}^{\epsilon_2}), \sigma_{b_{i_2}}) \star \cdots \star (CB(b_{i_k}^{\epsilon_k}), \sigma_{b_{i_k}}), \tag{4}$$

where we interpret the right of (4) by associating left-to-right. One can check that this is independent of the expression of β in the Artin generators.

Convention: Let $\beta \in B_N$ with associated permutation $\sigma_\beta, \in S_N$. Let $M \in GL(N, \mathbb{F}_q)$ and $\sigma \in S_n$. For ease of notation, we let $(M, \sigma) \star \beta := (M, \sigma) \star (CB(\beta), \sigma_\beta)$.

The discussion above can be summarized as follows: E-multiplication is an *action* of B_N on $GL(N, \mathbb{F}_q) \times S_N$ via a representation into a semidirect product

$$\big(GL(N, \mathbb{F}_q) \times S_N\big) \star \Pi_{CB}(B_N) \longrightarrow \big(GL(N, \mathbb{F}_q) \times S_N\big).$$

Given $\beta \in B_N$, we define $\mathcal{P}(\beta)$ to be the image of $(\mathrm{Id}_N, \mathrm{Id}_{S_N}) \in \big(GL(N, \mathbb{F}_q) \times S_N\big)$ under E-multiplication by β:

$$\mathcal{P}(\beta) := \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big) \star \beta,$$

where Id_N is the $N \times N$ identity matrix and $\mathrm{Id}_{S_N} \in S_N$ is the identity permutation.

The security of WalnutDSA is based, in part, on the following highly non-linear problem that we perceive to be computationally infeasible for sufficiently large key and parameter sizes.

The REM Problem (Reversing E-Multiplication is hard) *Consider the braid group B_N and symmetric group S_N with $N \geq 10$. Let $\mathbb{F}_q$ be a finite field of q elements,and fix a set of non-zero T-values $\{\tau_1, \tau_2, \ldots, \tau_N\}$ in $\mathbb{F}_q^\times$, the invertible elements of $\mathbb{F}_q$. Given a pair $(M, \sigma) \in (GL(N, \mathbb{F}_q),\ S_N)$ where it is stipulated that*

$$(M, \sigma) = \mathcal{P}(\beta)$$

for some unknown braid $\beta \in B_N$ (with sufficiently long BKL normal form), then it is infeasible to determine a braid β' such that $(M, \sigma) = \mathcal{P}(\beta')$.

Support for the hardness of reversing E-Multiplication can be found in [39] which studies the security of Zémor's [50] hash function $h : \{0,1\}^* \to SL_2(\mathbb{F}_q)$, with the property that $h(u\,v) = h(u)h(v)$, where $h(0), h(1)$ are fixed matrices in $SL_2(\mathbb{F}_q)$ and uv denotes concatenation of the bits u and v. For example a bit string $\{0, 1, 1, 0, 1\}$ will hash to $h(0)h(1)h(1)h(0)h(1)$. Zémor's hash function has not been broken since its inception in 1991. In [39] it is shown that feasible cryptanalysis for bit strings of length 256 can only be applied for very special instances of h. Now E-Multiplication, though much more complex, is structurally similar to a Zémor type scheme involving a large finite number of fixed matrices in $SL_2(\mathbb{F}_q)$ instead of just two matrices $h(0), h(1)$. This serves as an additional basis for the assertion that E-Multiplication is a one-way function.

4. Cloaking Elements

The second component of the security of WalnutDSA is based on our ability to explicitly construct certain braid words which we term cloaking elements. They are defined as follows.

Definition 4.1. (Cloaking element) *Let $\sigma \in S_N$. An element $v \in B_N$ is termed a cloaking element of σ provided v stabilizes (M, σ) under E-muliplication for all $M \in GL(N, \mathbb{F}_q)$, i.e.,*

$$(M, \sigma) \star v = (M, \sigma).$$

Thus a cloaking element is characterized by the property that it essentially disappears when performing E-Multiplication. Remark that, by definition, every cloaking element must itself be a pure braid braid (see §2). Letting $\mathrm{Cloak}_{(M,\sigma)}$ denote the set of all such cloaking elements we have the following proposition:

Proposition 4.2. *The set $\mathrm{Cloak}_{(M,\sigma)}$ forms a subgroup of B_N which is contained in the pure braid subgroup.*

We remark that whether a braid element is a cloaking element is contingent on the T-values, which are used in defining the operation $\star$. It is clear that cloaking elements must exist: the braid group is infinite and any action of an infinite group on a finite set will necessarily have stabilizers. Further, it is clear that generating very long cloaking elements is straightforward: starting with an arbitrary braid v, first raise v to the order of its associated permutation σ_v, yielding a purebraid $\bar{v}$. Then raise $\bar{v}$ to the (generally very large) order of the matrix $(1,1) \star {}^{\sigma}\bar{v}$ (where $(1,1)$ denotes the identity in $GL(N, \mathbb{F}_q) \times S_N$). What is not immediately obvious is how to construct cloaking elements sufficiently short to be useful. The following proposition provides one technique to construct them, and serves as an illustration of the behavior of the action (a paper focusing exclusively on the construction and enumeration of cloaking elements will be forthcoming):

Proposition 4.3. *For $N \geq 10$, suppose $1 \leq x_1 < x_2 < \cdots < x_\mu \leq N$, and let*

$$\begin{aligned} w \;=\; & b_{x_2-1}^{\epsilon_{x_2-1}}\, b_{x_2-2}^{\epsilon_{x_2-2}} \cdots b_{x_1+1}^{\epsilon_{x_1+1}}\, b_{x_1}\, b_{x_1+1}^{-\epsilon_{x_1+1}} \cdots b_{x_2-1}^{-\epsilon_{x_2-1}} \\ & \cdot b_{x_3-1}^{\epsilon_{x_3-1}}\, b_{x_3-2}^{\epsilon_{x_3-2}} \cdots b_{x_2+1}^{\epsilon_{x_2+1}}\, b_{x_2}\, b_{x_2+1}^{-\epsilon_{x_2+1}} \cdots b_{x_3-1}^{-\epsilon_{x_3-1}} \\ & \cdot \; \cdots \; \cdot b_{x_\mu-1}^{\epsilon_{x_\mu-1}}\, b_{x_\mu-2}^{\epsilon_{x_\mu-2}} \cdots b_{x_{\mu-1}+1}^{\epsilon_{x_\mu+1}}\, b_{x_{\mu-1}}\, b_{x_{\mu-1}+1}^{-\epsilon_{x_\mu+1}} \cdots b_{x_\mu-1}^{-\epsilon_{x_\mu-1}}, \end{aligned}$$

where all the exponents $\epsilon_i \in \{+1, -1\}$. Then then we have that,

- *w is braid involving the strands which start at the points $\{x_1, \ldots, x_\mu\}$,*
- *Any strand in w that originates at any $y \in \{x_1 + 1, \ldots, x_2 - 1, x_2 + 1, \ldots, x_\mu - 1\}$ ends at y,*
- *The braid $w^{2\mu}$, cloaks for the identity $(1,1)$ provided the identity*

$$t_{x_1}\, t_{x_2} \cdots t_{x_\mu} = -1,$$

holds. Further, by conjugating w by a braid z whose associated permutation is σ^{-1}, a cloak for σ is given by $z\, w^{2\mu}\, z^{-1}$.

The element w in the above proposition is termed the *core* of the cloak $w^{2\mu}$ with exponent 2μ. There turn out to be various ways of constructing cores of cloaking elements. To facilitate the flow of this paper we defer the lengthy discussion of this topic to Appendices A and B. In all discussions that follow we will assume that all methods of generating cores are used; the combinatorics of the generation methods will contribute to the security of WalnutDSA.

The concept of a cloaking element naturally lends itself to the following observation. Fix a braid β, say

$$\beta = b_{i_1}^{\epsilon_1} \cdots b_{i_\ell}^{\epsilon_\ell},$$

and choose some integer $1 \leq k \leq \ell$. Clearly, $\beta = x_1 \cdot x_2$ where $x_1 = b_{i_1}^{\epsilon_1} \cdots b_{i_{k-1}}^{\epsilon_{k-1}}$ and $x_2 = b_{i_k}^{\epsilon_k} \cdots b_{i_\ell}^{\epsilon_\ell}$, and we hence for any for any matrix/permutation pair (m_0, σ_0), we have that

$$(m_0, \sigma_0) \star \beta = ((m_0, \sigma_0) \star x_1) \star x_2.$$

Using Proposition 4.3 we can generate a cloaking element v for the product of $\sigma_0 \cdot \sigma_{x_1}$ where σ_{x_1} deotes the permutation associated with x_1. By construction, given any matrix M we have that $(M, \sigma_0 \cdot \sigma_{x_1}) \star v = (M, \sigma_0 \cdot \sigma_{x_1})$. Since $(m_0, \sigma_0) \star x_1$ takes the form $(m_0, \sigma_0) \star x_1 = (M, \sigma_0 \cdot \sigma_{x_1})$, we have that

$$\begin{aligned}
(m_0, \sigma_0) \star \beta &= ((m_0, \sigma_0) \star x_1) \star x_2 \\
&= (M, \sigma_0 \cdot \sigma_{x_1}) \star x_2 \\
&= (M, \sigma_0 \cdot \sigma_{x_1}) \star v \star x_2 \\
&= ((m_0, \sigma_0) \star x_1) \star v \star x_2 = (m_0, \sigma_0) \star x_1 \star v \star x_2.
\end{aligned}$$

Hence we have generated a new braid β' which contains v,

$$\beta' = x_1 \cdot v \cdot x_2,$$

which has the property that $(m_0, \sigma_0) \star \beta = (m_0, \sigma_0) \star \beta'$. We shall refer to this inserted cloaking element as a *concealed* cloaking element. The above discussion is summarized in the following proposition:

Proposition 4.4. *Given a braid β and a matrix/permutation pair (m_0, σ_0) it is possible to generate another braid β' so that $(m_0, \sigma_0) \star \beta = (m_0, \sigma_0) \star \beta'$ by randomly inserting a cloaking element for a permutation that is not a priori known, i.e., a concealed cloaking element within β. In the case β is itself a cloaking element for a given permutation, the resulting β' will also be a cloaking element for the same permutation, but will have a distinct structure from β.*

The process of randomly inserting cloaking elements into a braid can be iterated and we introduce the following definition:

Definition 1.5 Given an element $\beta \in B_N$, the output of κ iterations of randomly inserting cloaking elements as described in Proposition 4.4 into the braid β, is defined to be a κ–cloaking of β and is denoted by $\kappa(\beta)$.

5. Key Generation for WalnutDSA

WalnutDSA allows a signer with a fixed private/public-key pair to create a digital signature associated with a given message that can be validated by anyone who knows the public-key of the signer and the verification protocol. To facilitate the method, a central authority generates the system wide parameters using a publicly known parameter generation algorithm, and a signer S generates its own public and private key pair, denoted (Pub(S), Priv(S)), via a key generation algorithm.

Public System Wide Parameters:

• An integer $N \geq 10$ and associated braid group B_N.

• A message encoding algorithm which is the composition of a cryptographically secure 2η–bit hash function $H : \{0,1\}^* \to \{0,1\}^{2\eta}$, and an injective map $E : \{0,1\}^{2\eta} \to H_N$, where H_N is a free subgroup contained in the pure braid group on N strands.

• A finite field $\mathbb{F}_q$.

Signer's Security Parameters:

The Signer requires a set of parameters used to meet the desired security level. These parameters may be public, but the verifier does not not need access to them. They include:

• A method of generating a large search space of cloaking elements, v, for any given pair (M, σ).

• An integer $\kappa > 1$ which is chosen to meet the security level. The signature will utilize κ concealed cloaking elements.

• A rewriting algorithm $\mathcal{R}\colon B_N \to B_N$ which uses the relations of the group to render a rewritten word unrecognizable. Example of such rewriting algorithms, which will serve as the third component of the security of WalnutDSA, can be found in [13] or [17].

Signer's Private Key:

The Signer's Private Key consists of two random, freely-reduced braids:

• $\text{Priv(S)} = (w, w') \in B_N \times B_N$.

Here the three braids w, w' and $w' \cdot w$ are not in the pure braid group. We assume w, w' are sufficiently long to provide the necessary resistance to brute-force searches for the desired security level (see §9).

Signer's Public Key:

The Signer's Public Key consists of two matrix and permutation pairs, each of which is generated from the Private Keys of the signer via E-Multiplication, and a set of T-values:

• $\text{Pub(S)} = \Big(\mathcal{P}(w),\ \mathcal{P}(w')\Big)$.

• T-values $= \{\tau_1, \tau_2, \ldots, \tau_N\}$, where each τ_i is an invertible element in $\mathbb{F}_q$, such that some specified identities involving a subset of the T-values hold. Such identities may take the form, for example, $\tau_a \cdot \tau_x \cdot \tau_b = -1$, where $1 \leq a < x < b \leq N$.

6. Message Encoder Algorithm

In order to generate a secure signature and prevent certain types of merging attacks, one must carefully convert the message to be signed into a braid word. Let $m \in \{0,1\}^*$ be a message. Let

$H : \{0,1\}^* \to \{0,1\}^{2\eta}$ denote a cryptographically secure 2η-bit hash function for $\eta \geq 1$. We now present an injective encoding function $E : \{0,1\}^{2\eta} \to H_N$, where H_N is a free subgroup of the pure braid group generated by the $N-1$ braids defined below. We recall that a group is said to be freely generated by a subset of elements provided a reduced element (a word where the subwords $x \cdot x^{-1}$, and $x^{-1} \cdot x$ do not appear for any generator x) is never the identity.

It is requisite in WalnutDSA that the permutation of the encoded message be trivial, i.e., the encoded message must be a pure braid. In order to ensure that no two messages will be encoded in the same way, we require the message be encoded as unique nontrivial reduced elements in a free subgroup of the pure braid group. This requirement ensures that distinct messages will result in distinct encodings. One possible encoding algorithm is based on the following classical observation: the collection of pure braids given by

$$\begin{aligned} g_{(N-1),N} &= b_{N-1}^2 \\ g_{(N-2),N} &= b_{N-1} \cdot b_{N-2}^2 \cdot b_{N-1}^{-1} \\ g_{(N-3),N} &= b_{N-1}b_{N-2} \cdot b_{N-3}^2 \cdot b_{N-2}^{-1}b_{N-1}^{-1} \\ g_{(N-4),N} &= b_{N-1}b_{N-2}b_{N-3} \cdot b_{N-4}^2 \cdot b_{N-3}^{-1}b_{N-2}^{-1}b_{N-1}^{-1} \\ &\vdots \\ g_{1,N} &= b_{N-1}b_{N-2}\cdots b_2 \cdot b_1^2 \cdot b_2^{-1}b_3^{-1}\cdots b_{N-1}^{-1}, \end{aligned} \tag{5}$$

generate a free subgroup $H_N \leq B_N$ [12]. For simplicity, we will write g_i for $g_{i,N}$ when N is clear from the context.

Message Encoder Algorithm: We determine a braid $E(H(m)) \in B_N$ as follows. The hashed message $H(m)$ consists of η 2-bit blocks. Fix a collection S of η subsets S_η, where each S_η consists of a four-tuple of distinct generators taken from (5):

$$S_k = (g_{k_1}, g_{k_2}, g_{k_3}, g_{k_4}).$$

Each 2-bit block of $H(m)$ determines a unique element of the corresponding tuple S_k, and the output $E(H(m))$ is then the product of these generators of H_N, taken in order over the blocks of $H(m)$. It is clear that this map is injective, since the g_i generate a free subgroup, and since the knowledge of $E(H(m))$ and the sets S_k allows one to recover $H(m)$.

Without the presence of the hash function, the encoding function E would be homomorphic, i.e., $E(m)E(m') = E(mm')$ for all messages m, m'. However, this is not a problem since the input to the encoder is the *digest* of a message. Indeed, for a good cryptographic hash function H, we know that $H(m)H(m')$ will never equal $H(mm')$. We also know it is unlikely to find two classes of hash functions H_1, H_2 such that the output size of H_1 is half the output size of H_2, and then to further find three messages m, m', and m'' such that $H_1(m)$ $H_1(m')$ results in the same output[1] as $H_2(m'')$, and also get a signer to sign both messages m and m' using H_1. We also note that including a hash algorithm identifier in the message after it is hashed would prevent this attack.

[1]For a weak hash H_1 and a strong hash H_2, which has twice the output size of H_1, an attacker would need to find two messages m and m' that are preimages to the halves of H_2 of the desired forgery and then get the signer to use H_1 and sign both m and m'. E.g. the attacker would need to take his or her desired forged message, hash it using SHA2-256, find two preimages with MD5, get the signer to sign those MD5 preimages, and only then can he or she compose a message that would verify with SHA2-256.

A Second Message Encoder Algorithm: There are multiple ways to generate an encoder algorithm. The important requirements are that it must be injective, must result in a pure braid, and should use every strand (in order to generate a large dimension in the E-multiplication vector space). Moreover, both the signer and verifier must agree on the method itself.

In the case of $N = 10$ there is another, more compact encoder that we can use. We note that the braid

$$b_1^{\epsilon_X} b_2^{\epsilon_1} b_3^{\epsilon_2} b_4^{\epsilon_3} b_5^{\epsilon_4} b_6^{\epsilon_5} b_7^{\epsilon_6} b_8^{\epsilon_7} b_9^{\epsilon_8} b_9^{\epsilon_8} b_8^{\epsilon_9} b_7^{\epsilon_{10}} b_6^{\epsilon_{11}} b_5^{\epsilon_{12}} b_4^{\epsilon_{13}} b_3^{\epsilon_{14}} b_2^{\epsilon_{15}} b_1^{\epsilon_{16}}$$

can directly encode 16 bits into 18 generators, where each bit denotes a $+1$ or -1 exponent, and we duplicate ϵ_8 in the middle of the word and ϵ_{16} can carry into ϵ_X in the next word. For a 256-bit hash like SHA2-256, this results in a fixed-size encoder output of 288 generators.

One can clearly see that this method produces an output that is injective, a pure braid, and utilizes all braid strands.

7. Signature Generation and Verification

Fix a hash function H as in §6. To sign a message $m \in \{0,1\}^*$ the Signer performs the following steps:

Digital Signature Generation:

1. Compute the hash of the message $H(m)$.

2. Generate cloaking elements $\{v, v_1, v_2\}$ which cloak, respectively, the identity permutation in S_N, Id_{S_N}, and the permutations associated with w, w' σ_w, and $\sigma_{w'}$.

3. Generate the encoded message $E(H(m))$.

4. Compute $\mathrm{Sig} = \mathcal{R}\big(\kappa(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2)\big)$, which is a rewritten braid.

5. The final signature for the message m is the ordered pair $(H(m), \mathrm{Sig})$.

As addressed earlier, the cloaking elements $v, v_1, v_2 \in B_n$ disappear when the signature is E-Multiplied by the public key Pub(S), and the insertion of κ concealed cloaking elements will, by construction, not impact the verification .

Signature Verification: The signature (m, Sig) is verified as follows:

1. Generate the encoded message $E(H(m))$.

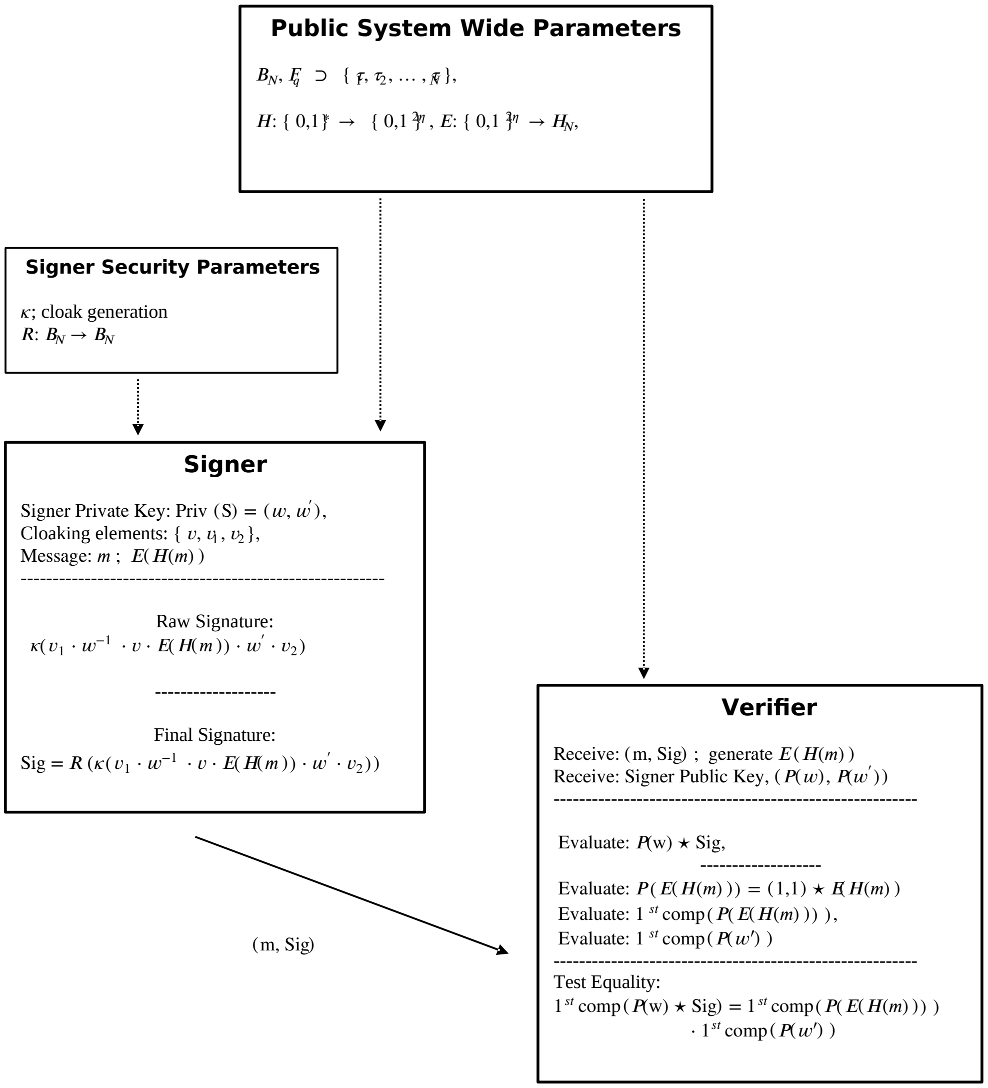
2. Evaluate $\mathcal{P}(E(H(m)))$.

3. Evaluate the E-Multiplication $\mathcal{P}(w) \star \mathrm{Sig}$.

4. Test the equality

$$\mathrm{Matrix}\Big(\mathcal{P}(w) \star \mathrm{Sig}\Big) \overset{?}{=} \mathrm{Matrix}\Big(\mathcal{P}\big(E(H(m))\big)\Big) \cdot \mathrm{Matrix}\Big(\mathcal{P}\big(w'\big)\Big), \tag{6}$$

where Matrix denotes the matrix part of the ordered pair in question, and the multiplication on the right is the usual matrix multiplication.



Public System Wide Parameters
B_N, $F_q \supset \{ \tau_1, \tau_2, \ldots, \tau_N \}$,
$H: \{0,1\}^* \to \{0,1\}^{4n}$, $E: \{0,1\}^{4n} \to H_N$,
Signer Security Parameters
κ; cloak generation
$R: B_N \to B_N$
Signer
Signer Private Key: Priv (S) = (w, w'),
Cloaking elements: $\{ v, v_1, v_2 \}$,
Message: m ; $E(H(m))$
Raw Signature:
$\kappa(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2)$
Final Signature:
Sig = $R(\kappa(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2))$
(m, Sig)
Verifier
Receive: (m, Sig) ; generate $E(H(m))$
Receive: Signer Public Key, $(P(w), P(w'))$
Evaluate: P(w) $\star$ Sig,
Evaluate: $P(E(H(m))) = (1,1) \star E(H(m))$
Evaluate: 1st comp$(P(E(H(m))))$,
Evaluate: 1st comp$(P(w'))$
Test Equality:
1st comp$(P$(w) $\star$ Sig) = 1st comp$(P(E(H(m))))$
$\cdot$ 1st comp$(P(w'))$

FIGURE 1. WalnutDSA Flow Diagram

5. Reject signatures that are longer than 2^{14} Artin generators[2].

The signature is valid if and only if (4), (5) holds.

[2]In practice 128-bit signatures average around 2^{11} generators, but different rewriting techniques could extend that. Because the braid group is infinite there are many ways to represent the same signature, however all those ways are well beyond the 2^{14} limit.

8. Security Discussion

To facilitate the accuracy of the discussion below we recall the following definition of *security level*:

Definition 8.1. (Security Level): *A secret is said to have security level k over a finite field $\mathbb{F}$ if the best known attack for obtaining the secret involves running an algorithm that requires at least 2^k elementary operations (addition, subtraction, multiplication, division) in the finite field $\mathbb{F}$.*

Linear Algebraic, Group Theoretic, and Probabilistic Attacks. Neither the attack of Ben-Zvi–Blackburn–Tsaban [8], based on ideas in [30], or the invalid public key attack of Blackburn–Robshaw [14] (also [1]) target the underlying hard problems on which WalnutDSA is based. This is because the signature is a braid (a cloaked braid word) and the public key is coming from E-Multiplication of the identity element with a braid that has very little algebraic structure.

The more recent work of Hart–Kim–Micheli–Perez–Petit–Quek [26] proposes a practical universal forgery attack on WalnutDSA in the special case where the two private braids w and w' are equal. The attack proceeds by taking a collection of signed messages (M_i, s_i) indexed by a finite set I and using them to produce a valid signature for a new message M. The main idea underlying the attack is finding a short expression in $GL(N, \mathbb{F}_q)$ for the element $h = \text{Matrix}\Big(\mathcal{P}(E(M))\Big)$ in terms of elements $g_i := \text{Matrix}\Big(\mathcal{P}(E(M_i)\Big)$. Namely, one seeks an expression of the form

$$h = \prod_{j=1}^{l} g_{i_j}^{\epsilon_{i_j}}, \quad i_j \in I, \epsilon_{i_j} \in \{\pm 1\}. \tag{7}$$

Then the braid

$$s = \prod_{j=1}^{l} s_{i_j}^{\epsilon_{i_j}}$$

will be a valid signature for M.

Thus the attack relies on both the equality of w and w' and on finding factorizations in nonabelian groups: the former implies that one can appropriately multiply the signatures s_i together in the final step to produce a signature for M, and the latter implies that one can find the correct product of the s_i. This attack fails if $w \neq w'$, since one cannot multiply the s_i together to produce a valid signature. It is observed in [9] that it is possible to modify the attack of [26] so that it reduces to the case $w = w'$ with forged signatures that are expected to be twice as long as forged signatures produced by the attack of [26]. The authors of [26] point out that the forged signatures produced by their method (in the case $w = w'$) are many orders of magnitude longer than the actual signatures produced by WalnutDSA, so the attack is easily thwarted by rejecting long signatures. Further, they also point out that their attack fails with moderate increases in the parameters N, q.

Four additional attacks have appeared recently. A Pollard-Rho type method taken by [15] uses the estimate for the number of braids of a given length in Artin generators (see §9), and assumes the output of E-multiplication is uniformly distributed, to give an exponential algorithm that recovers an equivalent private key of a signature from the corresponding public key. Specifically, [15] shows that to reach a k-bit security level:

$$q^{N(N-3)-1} > 2^{2k} \tag{8}$$

By choosing $N \geq 10$ (and $q = 32$ or 256) this approach becomes ineffective.

Further, the encoding method specified in §6 ensures that the vector space consisting of the matrix component of the signers' public keys has a sufficiently large dimension. It was observed in [10] that the encoding must ensure this property to maintain the specified security level, specifically, to reach a k-bit security level:

$$q^{\text{dimension}} > 2^{2k} \tag{9}$$

As an example of an encoding that yields sufficient security in the case $N = 12$, let S be the periodic sequence of tuples $\{(5, 7, 9, 11),\ (4, 6, 8, 10),\ (3, 5, 7, 9),\ (2, 4, 6, 8),\ (1, 3, 5, 7),\ (2, 4, 6, 8),\ (3, 5, 7, 9),\ (4, 6, 8, 10), \ldots\}$. One can check that this dimension is 122, so using $q = 32$ or 256 results in sufficiently large spaces. For the case of $N = 10$, S can be the sequence $\{(3, 5, 7, 9),\ (2, 4, 6, 8),\ (1, 3, 5, 7),\ (2, 4, 6, 8),\ \ldots\}$ which results in a dimension of 82.

An alternate exponential factoring attack [11] found a more efficient way to find alternate private keys that produce short signatures. The attack was mounted against the WalnutDSA NIST submission, which uses an older version of WalnutDSA where $\tau_1 = \tau_2 = 1$, and suggested parameters $N = 8, q = 32$ for 128-bit security and $N = 8, q = 256$ for 256-bit security. Specifically, the attack in [11] showed that those parameters were too small. Against that older version of WalnutDSA using those parameters the attack runs in $q^{N-5/2}$ time although they claim it can be reduced to $q^{(N/2)-1}$. While the former runtime was verified, the latter runtime was never observed using the attack code made available.

Against this version of WalnutDSA, where $\tau_1 \cdot \tau_a \cdot \tau_N = -1$, their running time is much higher, adding at least a factor of $\sqrt{q}\sqrt{x}$ to their runtime, where x is a parameter in their attack (they set $x = 60$ for $N = 8$, it is unclear what it needs to be for $N = 10$). This results in an (unverified) search time of at least

$$\sqrt{x}\ q^{(N-1)/2} \tag{10}$$

Next, a method for searching for cloaking elements of known permutations has been posited by Kotov–Menshov–Ushakov [33]. It is the presence of κ concealed cloaking elements that blocks this attack. In general, knowing that κ concealed cloaking elements have been placed in a known braid, it would require $(N!)^\kappa$ searches to find them and thus, taking the lack of possible birthday attacks into account, to insure k-bit security we would require

$$(N!)^\kappa > 2^k \tag{11}$$

and hence

$$\kappa > \text{Security Level}/\log_2(N!).$$

We have explored possible birthday attacks and have ruled out obvious ways to use a birthday attack to discover all the concealed cloaking elements. Indeed, multiple cloaking elements could use the same permutation but each would still need to individually be discovered. Without access to a birthday attack, in the case of $N = 10$, and a security level of 128 we can comfortably take $\kappa = 6$ (which results in $2^{130.74}$). Likewise, when $N = 10$ and the security level is 256, taking $\kappa = 12$ is sufficient (resulting in $2^{261.49}$).

Lastly, Merz and Petit [36] proposed a practical forgery attack on WalnutDSA. They found that using the Garside Normal Form of the signature allowed them to find commonalities with the Garside form of the encoded message, and using those commonalities they could create a forgery. As pointed out by the authors, the attack can be defeated by adding cloaking elements into the encoded message. Specifically, they conjecture that each additional cloaking element effectively mutates approximately five (5) permutation braids in the Garside Normal Form, but, when mutated, their attack no longer succeeds.

The Merz and Petit universal forgery attack is a heuristic method that, using knowledge of a valid signature of a message M, aims to generate a signature of a second message M' that will be validated by a receiver. The decomposition algorithm introduced in their paper (which uses the Garside canonical form as its basis) can be applied because a Walnut signature has the form

$$W_1 E(H(M)) W_2$$

and, critically, the braid element $E(H(M))$ is known to everyone. Knowledge of $E(H(M))$ allows the algorithm to try to derive braids W_1', W_2' which satisfy the conditions $W_i \equiv W_i'$ (Mod Δ^2), and $W_1 \cdot W_2 \;=\; W_1' \cdot W_2'$. Once a forger has said elements in place, the braid $W_1' \cdot E\big(H(M')\big) \cdot W_2'$ will verify as a signature of a message M'.

In fact, knowledge of the entire $E(H(M))$ is not actually requisite. Were one to insert a single concealed cloaking element into the encoding $E(H(M))$ it is still possible that the permutation braids in the Garside normal form of said encoding still appear in the Garside normal form for the signature. While the forgery in this case would be longer than the average signature, it might be within the acceptable length range. Thus, in order to completely thwart the heuristic attack, the signer must insert sufficiently many concealed cloaking elements into the braid $E(H(M))$ to completely alter the Garside normal form. We have done significant testing and have concluded that inserting cloaking elements every 5-12 generators will successfully block this attack. It should be noted that the approaches to removing cloaking elements required the attacker to be able to reduce the problem to a conjugacy search problem; finding concealed cloaking elements in the encoded message does not fit into that effort.

9. Brute Force Attacks

Brute force security level for each Private Key: In order to choose private keys of security level = SL that defeat a brute force attack, which enumerates all possible expressions in the braid generators, we need to analyze the set of braids in B_N of a given length ℓ and try to assess how large this set is. Being as conservative as possible, at a minimum, the brute force security level for the signer's private key pair will be the brute force security level of a single private key. Letting $W_N(\ell)$ denote the number of distinct braid words of length ℓ in B_N, the most basic estimate for $W_N(\ell)$ is given by

$$W_N(\ell) \;\le\; (2(n-1))^{\ell}\,.$$

This trivial bound does not take into account the fact that the braid relations, particularly the commuting relations, force many expressions to coincide. Furthermore, the commuting relations $b_i\, b_j \;=\; b_j\, b_i \quad |i-j| \ge 2$, allow us to write products of generators far enough apart in weighted form, i.e., given $b_i\, b_j$ where $|i-j| \ge 2$, we can assume $i > j$.

To start analyzing the situation we work in B_5, we enumerate words of length 2 starting with a given generator: $b_1\, b_2^{\pm 1}$, $b_1\, b_1$, $b_2\, b_3^{\pm 1}$, $b_2\, b_2$, $b_2\, b_1^{\pm 1}$, $b_3\, b_4^{\pm 1}$, $b_3\, b_3$, $b_3\, b_2^{\pm 1}$, $b_3\, b_1^{\pm 1}$, $b_4\, b_4$, $b_4\, b_3^{\pm 1}$, $b_4\, b_2^{\pm 1}$, $b_4\, b_1^{\pm 1}$. Words of length 2 starting with inverses of the generators are of course similar, and thus the number of distinct words of length $\ell = 2$ in B_5 taking the commuting relations into account is $44 < (2(5-1))^2 = 64$. In order to obtain a good bound for $W_N(\ell)$, which eliminates the redundancy arising from the commuting elements, we require the following function:

$$w_k(k') \;=\; \begin{cases} 1 & k = k', \\ 2 & k \neq k' \text{ and } k' < N-1, \\ 0 & k' > N-1. \end{cases}$$

Using this notation, the number of words of length 2 in B_N is given by

$$W_N(2) \;=\; 2 \sum_{k_1=1}^{N-1} \sum_{k_2=1}^{k_1+1} w_{k_1}(k_2),$$

where the equality holds because the remaining braid relations are longer than length 2. Moving to words of length ℓ, we have

$$W_N(\ell) \;\leq\; 2 \sum_{k_1=1}^{N-1} \sum_{k_2=1}^{k_1+1} w_{k_1}(k_2) \sum_{k_3=1}^{k_2+1} w_{k_2}(k_3) \cdots \sum_{k_\ell=1}^{k_{\ell-1}+1} w_{k_{\ell-1}}(k_\ell).$$

This is just an upper bound on the number of braids of length ℓ but it does represent what an attacker would have to do to be certain that all possibilities are checked. At present, the above method gives the best protocol known for generating braid words of length ℓ with the least over counting. There is no closed formula for the number of distinct braids of length ℓ; in fact the problem is NP hard [41].

Hence we are reduced to finding a lower bound for the right hand side above, which can be done as follows:

$$2 \sum_{k_1=1}^{N-1} \sum_{k_2=1}^{k_1+1} w_{k_1}(k_2) \sum_{k_3=1}^{k_2+1} w_{k_2}(k_3) \cdots \sum_{k_\ell=1}^{k_{\ell-1}+1} w_{k_{\ell-1}}(k_\ell) \;\geq\; 2^\ell \sum_{k_1=1}^{N-1} \sum_{\substack{k_2=1 \\ k_2\neq k_1}}^{k_1+1} \sum_{\substack{k_3=1 \\ k_3\neq k_2}}^{k_2+1} \cdots \sum_{\substack{k_\ell=1 \\ k_\ell\neq k_1}}^{k_{\ell-1}+1} 1$$

$$= \; 2^\ell \sum_{k_1=1}^{N-1} \sum_{k_2=1}^{k_1} \sum_{k_3=1}^{k_2} \cdots \sum_{k_\ell=1}^{k_{\ell-1}} 1 \;=\; \frac{2^\ell}{\ell} \cdot (N-1) \binom{\ell-2+N}{N-1},$$

where $\binom{\ell-2+N}{N-1}$ denotes the binomial symbol.

Thus, in order to defeat the brute force search at a security level = SL, the signer's private key must be a braid word of length ℓ which satisfies:

$$SL \;\geq\; \log_2\left(\frac{2^\ell}{\ell} \cdot (N-1) \binom{\ell-2+N}{N-1}\right).$$

Next, we may use Stirling's asymptotic formula for the Gamma function to obtain a lower bound for $\frac{2^\ell}{\ell} \cdot (N-1) \binom{\ell-2+N}{N-1}$. The final result is

$$SL \;>\; \log_2\left(\frac{(2^\ell/\ell)\cdot \ell^{(N-1)}}{(N-1)!}\right)$$

for fixed N as $\ell \to \infty$. To find the length ℓ associated to a given security level SL, one may apply Newton's method to solve the equation: $\ell + (N-2)\log_2(\ell) = SL + \log_2\Big((N-1)!\Big)$. For $N = 10$ this results in $\ell = 95$ for SL=128, and $\ell = 213$ for SL=256.

An alternative brute force attack would proceed by starting with the known permutation of a private key and look at the collection of inverse images of said permutation in the braid group. To prevent such an approach from being effective we must ensure that this search space of inverse images is sufficiently large. If two braids have the same associated permutation, they must differ by an element in the pure braid subgroup. Thus to ensure we are choosing private keys sufficiently long for our security level SL, each private key must be as long as the lift of a permutation times a sufficiently long expression in the pure braid generators.

The pure braid subgroup of B_N is generated [25] by the set of $N(N-1)/2$ braids given by

$$g_{i,j} = b_{j-1}b_{j-2}\cdots b_{i+1}\cdot b_i^2 \cdot b_{i+1}^{-1}\cdots b_{j-2}^{-1}b_{j-1}^{-1}, \qquad 1 \leq i < j \leq N. \tag{12}$$

The relations for the pure braid subgroups are given by

$$g_{r,s}^{-1}g_{i,j}g_{r,s} = \begin{cases} g_{i,j}, & \text{if } i<r<s<j \text{ or } r<s<i<j,\\ g_{r,j}g_{i,j}g_{r,j}^{-1}, & \text{if } r<i=s<j,\\ g_{r,j}g_{s,j}g_{i,j}g_{s,j}^{-1}g_{r,j}^{-1}, & \text{if } r=i<s<j,\\ g_{r,j}g_{s,j}g_{r,j}^{-1}g_{s,j}^{-1}g_{i,j}g_{s,j}g_{r,j}g_{s,j}^{-1}g_{r,j}^{-1}, & \text{if } r<i<s<j, \end{cases}$$

see [25] for details.

Given the nature of the above defining relations, a reasonable estimate for the number of words of length L in the pure braid generators is thus given by

$$(2\cdot N(N-1)/2)^L = (N(N-1))^L,$$

and hence the security level can be estimated to be

$$\log_2((N(N-1))^L) = L\cdot \log_2(N(N-1)).$$

In the case $N=10$, to obtain a security level of SL= 128, we would need $L=20$ and SL= 256 would require $L=40$. We experimentally determined that the average length of lifting a random permutation with $N=10$ results in a braid of length 40 (with a standard deviation of 12). Further, we experimentally determined that on average a word of length 20 in the purebraid generators results in an average 108 Artin generators (with a standard deviation of 24), which gives us a private key of length ≈ 148. Using 40 purebraid generators results in an average 215 (with standard deviation 25), which gives us a private key of length ≈ 255. These private key lengths, being slightly larger that those obtained from the first brute force attack, will suffice to prevent both of the brute force attacks on the private keys.

Search space of each Public Key Pub(S)**:** Recall that the signer's public key is given by the pair: Pub(S) $= \big(\mathcal{P}(w), \mathcal{P}(w')\big)$. When this is evaluated with the specified choices of B_N and $\mathbb{F}_q$ it results in two $N\times N$ matrices each with q possible elements for every entry. The last row will consist of zeros with the exception of the final entry on the bottom right. Thus an estimate for the number of possible matrices appearing in public keys is given by

$$q^{N(N-1)+1} = q^{N^2-N+1}.$$

The search space for all such matrices is again the square of this lower bound. At present, the only known way to determine Priv(S) from Pub(S) is a brute-force search.

Brute Force Removal of Cloaking Elements. If one knows the core of a cloaking element then one could attempt a brute-force attack to remove it from a braid. The simplistic attack proceeds as follows:

(1) Start at the first element in the braid
(2) Insert the inverse of the cloaking core at that point
(3) Run Dehornoy to reduce the braide
(4) Check if the overall length of the braid had a significant reduction
(5) If not, go to the next position and return to step 2.

Our testing has showed that different cloaking cores have significantly varying resistance to this type of attack. For example, the earlier, simplistic cores like $b_i^{\pm 4}$ can be removed by brute force 25% of the time[3] when running in B_{10}! Worst case, if one were guessing, one would only need to try all 18 possible cores at every position within the braid. The current set of cloaking cores proposed (see Appendix A), however, only have a 2^{-10} chance of being removable in B_{10}. This means if you have a sample braid and know the core being used, you only have a 2^{-10} chance of being able to remove the cloaking element. In other words, if you know the exact core being used then you will need to test over 1000 signatures before you will be able to remove it.

Note that this works only because the core is the center of a conjugate. This means that it can be made arbitrarily difficult by nesting cloaking elements. One can generate a cloaking element and then place another cloaking element in either the left or right side of the conjugating material. When this is done, one must remove both cloaking elements in order to proceed, which means there is a 2^{-20} chance of being able to remove both.

Moreover, all of this presupposes the attacker knows the exact cloaking core in use. However, when we add all the possible cores as shown in Equations 14, 15, and 16, there are at a minimum

$$\begin{aligned}
&\sum_{k=5}^{N} (k-2)\cdot 2^{(k-2)} \\
&+ \sum_{k=8}^{10}(k-2) \sum_{\substack{\ell=5 \\ \ell<k-3}}^{7} 2\cdot\Big((k-\ell-2)\cdot(\ell-2)\cdot 2^{(\ell-1)}\cdot 2^{(k-\ell-3)}\Big) \\
&+ \big(N-1\big)^{\frac{L_2-\theta}{5+k}} \cdot \frac{N\cdot 2^{L_1+2}}{L_1} \cdot \binom{L_1+3}{4} \cdot \prod_{j=1}^{k}(4+j)
\end{aligned}$$

possible cloaking cores to choose from, and an attacker would need to try all of them.

Note that the number of cores of a given length L grows exponentially in L. Leveraging both of these cases, by adjusting the various possible parameters, the signature generator can make removal of cloaking elements arbitrarily difficult for an attacker.

Quantum Resistance. We now quickly explore the quantum resistance of WalnutDSA. As shown in §8, the security of WalnutDSA is based on the hard problem of reversing E-Multiplication. The underlying math is intimately tied to the infinite non-abelian braid group that is not directly connected to any finite abelian group. We will show that this lends strong credibility for the choice of WalnutDSA as a viable post-quantum digital signature protocol.

[3]We also think this explains why the Kotov–Menshov–Ushakov attack was as successful as it was

The Hidden Subgroup Problem HSP on a group G asks to find an unknown subgroup H using calls to a known function on G which is constant on the cosets of G/H and takes different values on distinct cosets. Shor's [44] quantum attack breaking RSA and other public key protocols such as ECC are essentially equivalent to the fact that there is a successful quantum attack (the quantum Fourier transform QFT) on the HSP for finite cyclic and other finite abelian groups (see [34]).

There are at least two possible ways to try to use quantum methods for HSP to attack the underlying algebra: (i) one can try to use HSP in the braid group itself, for instance as an approach to CCSP, or (ii) one can try to use HSP in the general linear group $GL(N, \mathbb{F}_q)$, for instance to identify the image of B_N under E-Multiplication, or to identify the images of other subroups, such as the pure braids.

Both possibilities are far beyond what is currently known for HSP. First of all, the braid group is infinite, and no progress has been made for HSP for infinite groups. Moreover, every non-trivial element in B_N has infinite order, and in particular the braid group does not contain any non-trivial finite subgroups. Hence there does not seem to be any viable way at present to work with quantum solutions for HSP in B_N. Second, some progress has been made in quantum solutions to HSP for certain nonabelian finite groups, such as semidirect products of abelian groups, or groups with the property that all subgroups are normal. However progress for groups with large degree representations such as $GL(N, \mathbb{F}_q)$ and other finite groups of Lie type has been more limited. Currently the best one knows how to do is to construct subexponential circuits to compute the QFT on such groups [37]. This does not give an efficient algorithm to apply quantum attacks to such groups.

Given an element

$$\beta = b_{i_1}^{\epsilon_1}\, b_{i_2}^{\epsilon_2}\, \cdots\, b_{i_k}^{\epsilon_k} \in B_N, \tag{13}$$

where $i_j \in \{1, \ldots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$, we can define a function $f \colon B_N \to GL(N, \mathbb{F}_q)$ where $f(\beta)$ is given by the E-Multiplication $(1,1) \star (\beta, \sigma_\beta)$ and σ_β is the permutation associated to β. Now E-Multiplication is a highly non-linear operation. As the length k of the word β increases, the complexity of the Laurent polynomials occurring in the E-Multiplication defining $f(\beta)$ increases exponentially. It does not seem to be possible that the function f exhibits any type of simple periodicity, so it is very unlikely that inverting f can be achieved with a polynomial quantum algorithm.

Finally, we consider Grover's quantum search algorithm [23] which can find an element in an unordered N element set in time $\mathcal{O}(\sqrt{N})$. Grover's quantum search algorithm can be used to find the private key in a cryptosystem with a square root speed-up in running time. Basically, this cuts the security in half and can be defeated by doubling the key size. This is where E-Multiplication shines. When doubling the key size one only doubles the amount of work as opposed to RSA, ECC, etc. where the amount of work is quadrupled. Note that almost all of the running time of signature verification in WalnutDSA is taken by repeated E-Multiplications.

10. Conclusion

In this paper we presented an in-depth discussion of WalnutDSA, a quantum-resistant, group-theoretic public-key digital signature method with fast performance on verification. We show how to construct WalnutDSA keys and signatures, and how to validate the signature against the public key. We introduced cloaking elements and provide multiple means to generate them. Finally, we enumerated all known attacks against WalnutDSA and show how the current choices in parameters and cloaking elements defeat all known attacks.

References

[1] D. Atkins; D. Goldfeld, Addressing the algebraic eraser over the air protocol, https://eprint.iacr.org/2016/205.pdf (2016).

[2] I. Anshel; D. Atkins; D. Goldfeld; P. E. Gunnells, *Defeating the Ben-Zvi, Blackburn, and Tsaban Attack on the Algebraic Eraser*, arXiv:1601.04780v1 [cs.CR].

[3] I. Anshel; D. Atkins; D. Goldfeld; P. E. Gunnells, *A Class of Hash Functions Based on the Algebraic Eraser,* Groups Complex. Cryptol. 8 (2016), no. 1, 1–7.

[4] I. Anshel; D. Atkins; D. Goldfeld; P. E. Gunnells, *Hickory Hash^{TM}: Implementing an Instance of an Algebraic Eraser^{TM} Hash Function on an MSP430 Microcontroller,* 2016, https://eprint.iacr.org/2016/1052.

[5] I. Anshel; D. Atkins; D. Goldfeld; P. E. Gunnells, *Ironwood Meta Key Agreement and Authentication Protocol,* Advances in Mathematics of Communications, 2020, doi: 10.3934/amc.2020073.

[6] I. Anshel; M. Anshel; D. Goldfeld; S. Lemieux, *Key agreement, the Algebraic Eraser^{TM}, and Lightweight Cryptography,* Algebraic methods in cryptography, Contemp. Math., vol. 418, Amer. Math. Soc., Providence, RI, 2006, pp. 1–34.

[7] M. Bellare; G. Neven, *Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma*, Proceedings of the 13th Association for Computing Machinery (ACM) Conference on Computer and Communications Security (CCS), Alexandria, Virginia, (2006), pp. 390–399.

[8] A. Ben-Zvi; S. R. Blackburn; B. Tsaban, *A practical cryptanalysis of the Algebraic Eraser*, CRYPTO 2016, Lecture Notes in Computer Science 9814 (2016), 179–189.

[9] W. Beullens, *WalnutDSA Official Comment*, Computer Security Resource Center Post-Quantum Cryptography, National Institute of Standards and Technology, 15 January 2018. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/ WalnutDSA-official-comment.pdf. pp. 2 - 4. [Accessed 9 February 2018].

[10] W. Beullens, *WalnutDSA Official Comment*, Computer Security Resource Center Post-Quantum Cryptography, National Institute of Standards and Technology, 1 Feburary 2018. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/ WalnutDSA-official-comment.pdf. pp. 19 - 21. [Accessed 9 February 2018].

[11] W. Beullens; S. Blackburn, *Practical attacks against the Walnut digital signature scheme*, pre-print, May, 2018.

[12] J. Birman, *Braids, Links and Mapping Class Groups*, Annals of Mathematics Studies, Princeton University Press, 1974.

[13] J. Birman; K. H. Ko; S. J. Lee, *A new approach to the word and conjugacy problems in the braid groups,* Adv. Math. 139 (1998), no. 2, 322–353.

[14] S. R. Blackburn; M.J.B. Robshaw, *On the security of the Algebraic Eraser tag authentication protocol*, 14th International Conference on Applied Cryptography and Network Security (ACNS 2016), to appear. See http://eprint.iacr.org/2016/091.

[15] S. R. Blackburn, *WalnutDSA Official Comment*, Computer Security Resource Center Post-Quantum Cryptography, National Institute of Standards and Technology, 22 January 2018. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/ WalnutDSA-official-comment.pdf. pp. 8 - 12. [Accessed 9 February 2018].

[16] E. Brickell; D. Pointcheval; S. Vaudenay; M. Yung, *Design Validations for Discrete Logarithm Based Signature Schemes.* In Public Key Cryptography, Melbourne, Australia, Lectures Notes in Computer Science 1751, pp. 276–292, Springer- Verlag, (2000).

[17] P. Dehornoy, *A fast method for comparing braids,* Adv. Math. 125 (1997), no. 2, 200–235.

[18] M. Düll; B. Haase; G. Hinterwälder; M. Hutter; C. Paar; A. Sánchez; P. Schwab, *High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers,* https://eprint.iacr.org/2015/343.pdf (2015).

[19] D. Garber; S. Kaplan; M. Teicher; B. Tsaban; U. Vishne, *Length-based conjugacy search in the braid group,* Algebraic methods in cryptography, 75-87, Contemp. Math., 418, Amer. Math. Soc., Providence, RI, 2006.

[20] V. Gebhardt, *A new approach to the conjugacy problem in Garside groups,* J. Algebra 292(1) (2005), 282–302.

[21] D. Goldfeld and P. E. Gunnells, *Defeating the Kalka-Teicher-Tsaban linear algebra attack on the Algebraic Eraser*, Arxiv eprint 1202.0598, February 2012.

[22] A. Groch; D. Hofheinz; R. Steinwandt, *A Practical Attack on the Root Problem in Braid Groups,* Algebraic methods in cryptography, 121-131, Contemp. Math., 418, Amer. Math. Soc., Providence, RI, 2006.

[23] L.K. Grover, *A fast quantum mechanical algorithm for database search,* Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) p. 212.

[24] P. E. Gunnells, *On the cryptanalysis of the generalized simultaneous conjugacy search problem and the security of the Algebraic Eraser,* arXiv:1105.1141v1 [cs.CR] .

[25] V. Hansen, *Braids and coverings: selected topics,* With appendices by Lars Gæde and Hugh R. Morton, London Mathematical Society Student Texts, 18, Cambridge University Press, Cambridge, (1989).

[26] D. Hart; D. Kim; G. Micheli; G. Pascual Perez; C. Petit; Y. Quek, *A Practical Cryptanalysis of WalnutDSA*, preprint 2017. 1

[27] D. Hofheinz; R. Steinwandt, *A practical attack on some braid group based cryptographic primitives,* Public Key Cryptography, Proceedings of PKC 2003 (Yvo Desmedt, ed.), Lecture Notes in Computer Science, no. 2567, Springer-Verlag, 2002, pp. 187-198.

[28] J. Huang; H. Li; P. Sweany, *An FPGA Implementation of Elliptic Curve Cryptography for Future Secure Web Transaction*, Proceedings of the ISCA 20th International Conference on Parallel and Distributed Computing Systems, September 24-26, 2007.

[29] D. Kahrobaei; C, Koupparis, *Non-commutative digital signatures,* Groups Complexity Cryptography, Volume 4, Issue 2 (Dec 2012), 377-384.

[30] A. Kalka, M. Teicher and B. Tsaban, *Short expressions of permutations as products and cryptanalysis of the Algebraic Eraser*, Advances in Applied Mathematics 49 (2012), 57-76.

[31] K. Ko, D. Choi, M. Cho, and J. Lee, *New signature scheme using conjugacy problem,* Cryptology ePrint Archive: Report 2002/168 (2002).

[32] N. Koblitz; A. Menezes, *Another look at "provable security,"* J. Cryptol. 20, 3–37 (2007).

[33] M. Kotov; A. Menshov; A. Ushakov, *An attack on the Walnut digital signature algorithm*, Cryptology ePrint Archive: Report 2018/393 (2018).

[34] C. Lomont, *The hidden subgroup problem - review and open problems,* 2004, arXiv:0411037

[35] W. Magnus; A. Karrass; D. Solitar, *Combinatorial group theory: Presentations of groups in terms of generators and relations,* Interscience Publishers (John Wiley & Sons, Inc.), New York-London-Sydney (1966).

[36] Merz S.P., Petit C. (2018) *Factoring Products of Braids via Garside Normal Form,* https://eprint.iacr.org/2018/1142.

[37] C. Moore; D. Rockmore; A. Russell, *Generic Quantum Fourier Transforms*, ACM Transactions on Algorithms, 2 (4), pp. 707–723, 2006.

[38] H.R. Morton, *The multivariable Alexander polynomial for a closed braid, Low-dimensional topology,* (Funchal, 1998), 167–172, Contemp. Math., 233, Amer. Math. Soc., Providence, RI, 1999.

[39] C. Mulland; B. Tsaban; *SL2 homomorphic hash functions: Worst case to average case reduction and short collision search,* arXiv:1306.5646v3 [cs.CR] (2015).

[40] A. D. Myasnikov; A. Ushakov, *Cryptanalysis of the Anshel-Anshel-Goldfeld-Lemieux key agreement protocol*, Groups Complex. Cryptol. 1 (2009), no. 1, 63-75.

[41] M.S. Paterson; A.A. Razborov, *The Set of Minimal Braids is co-NP-Complete,* J. Algorithms,12, (1991), 393–408.

[42] D. Pointcheval; J. Stern, *Security arguments for digital signatures and blind signatures,* Journal of Cryptology, 13(3):361–396, (2000).

[43] G. Seroussi, *Table of low-weight binary irreducible polynomials*, Technical Report HP-98-135, Computer Systems Laboratory, Hewlett–Packard, 1998.

[44] P. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,* SIAM J. on Computing, (1997) 1484–1509.

[45] J. Stern; D. Pointcheval; J. Malone-Lee; N. P. Smart, *Flaws in Applying Proof Methodologies to Signature Schemes,* Advances in Cryptology - Proceedings of CRYPTO 2002 (18 - 22 August 2002, Santa Barbara, California, USA) M. Yung Ed. Springer-Verlag, LNCS 2442, pages 93-110.

[46] H. Tschofenig; M. Pégourié-Gonnard, *Crypto Performance on ARM Cortex-M Processors*, IETF-92, Dallas, TX, March, 2015.

[47] B.C. Wang; Y.P. Hu, *Signature scheme based on the root extraction problem over braid groups,* IET Information Security 3 (2009), 53-59.

[48] E. Wenger; T. Unterluggauer; M. Werner, *8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors.* Progress in Cryptology - INDOCRYPT 2013, volume 8250 of Lecture Notes in Computer Science, pages 244-261. Springer, 2013.

[49] *Atmel SAMD21 – ARM Cortex M0 – 48MHz*, Benchmarking wolfSSL and wolfCrypt. [Online]. Available: https://www.wolfssl.com/docs/benchmarks/. [Accessed 12 February 2018].

[50] G. Zémor; *Hash functions and graphs with large girths,* Eurocrypt '91, Lecture Notes in Computer Science 547 (1991), 508–511.

Appendix A. The Splicing Method

Recall from Proposition 4.3, that given $N \geq 10$, $1 \leq x_1 < x_2 < \cdots < x_\mu \leq N$, and exponents $\epsilon_{\ell_k} \in \{+1, -1\}$, the braid

$$\begin{aligned} w \;=\; & b_{x_2-1}^{\epsilon_{x_2-1}}\, b_{x_2-2}^{\epsilon_{x_2-2}} \cdots b_{x_1+1}^{\epsilon_{x_1+1}}\, b_{x_1}\, b_{x_1+1}^{-\epsilon_{x_1+1}} \cdots b_{x_2-1}^{-\epsilon_{x_2-1}} \\ & \cdot b_{x_3-1}^{\epsilon_{x_3-1}}\, b_{x_3-2}^{\epsilon_{x_3-2}} \cdots b_{x_2+1}^{\epsilon_{x_2+1}}\, b_{x_2}\, b_{x_2+1}^{-\epsilon_{x_2+1}} \cdots b_{x_3-1}^{-\epsilon_{x_3-1}} \\ & \cdot \cdots \cdot b_{x_\mu-1}^{\epsilon_{x_\mu-1}}\, b_{x_\mu-2}^{\epsilon_{x_\mu-2}} \cdots b_{x_{\mu-1}+1}^{\epsilon_{x_\mu+1}}\, b_{x_{\mu-1}}\, b_{x_{\mu-1}+1}^{-\epsilon_{x_\mu+1}} \cdots b_{x_\mu-1}^{-\epsilon_{x_\mu-1}}, \end{aligned}$$

is a core of a cloaking element for the identity $(1, 1)$ with exponent 2μ

$$(1, 1) \star w^{2\mu} = (1, 1),$$

provided the identity

$$t_{x_1}\, t_{x_2} \cdots t_{x_\mu} = -1,$$

holds. We will refer to the core w as a μ–core with $x_\mu - x_1 + 1$ strands.

In this Appendix we will work with the cases of $\mu = 3, 4$ and introduce a method of *splicing* one core into another to produce a large set of cloaking elements. Focusing first on the case of $\mu = 3$, we simplify the notation and assume $x_1 = 1, x_2 = x$, and $x_3 = k$ (the more general case is entirely similar); we have

$$1 < x < k.$$

Thus suppose that w is a 3–core with k strands which by definition takes the form,

$$w \;=\; b_{x-1}^{\epsilon_{x-1}}\, b_{x-2}^{\epsilon_{x-2}} \cdots b_2^{\epsilon_2}\, b_1\, b_2^{-\epsilon_2} \cdots b_{x-1}^{-\epsilon_{x-1}} \;\cdot\; b_{k-1}^{\epsilon_{k-1}}\, b_{k-2}^{\epsilon_{k-2}} \cdots b_{x+1}^{\epsilon_{x+1}}\, b_x\, b_{x+1}^{-\epsilon_{x+1}} \cdots b_{k-1}^{-\epsilon_{k-1}}.$$

For fixed x, k there are $2^{(k-3)}$ possible 3–cores in the above form. Since there are $(k-2)$ ways to choose x, the number of cores in the above form is given by $(k-2)\cdot 2^{(k-3)}$. Since the inverse of a core is again a core, this method produces

$$2(k-2)\cdot 2^{(k-3)} = (k-2)\cdot 2^{(k-2)}, \tag{14}$$

cores. Consider the strands emerging from the collection of points

$$P(w) \;=\; \{2, \ldots, x-1, x+1, \ldots, k-1\}.$$

Each of these strands begins and ends at the same point. A *ribbon* within the braid w is a collection of strands associated to a subset of consecutive elements $R = \{y_1, y_2, \ldots, y_\ell\} \subset P(w)$, where the initial exponents of all the b_{y_j} appearing in w are equal:

$$\epsilon_{y_1} = \epsilon_{y_2} = \cdots = \epsilon_{y_\ell}.$$

The strands emerging from $\{y_1, y_2, \ldots, y_\ell\}$ will all either lie behind or in front of the strands emerging from the three strands emerging from $1, x, k$. Further the set R can lie to the left of x, the right of x, or contain points from which lie both left and right of x.

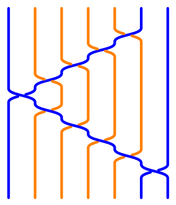


FIGURE 2.
$b_5^{-1}\, b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6$

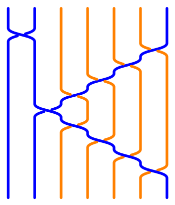


FIGURE 3.
$b_1\, b_6^{-1}\, b_5^{-1}\, b_4^{-1}\, b_3^{-1}\, b_2\, b_3\, b_4\, b_5\, b_6$

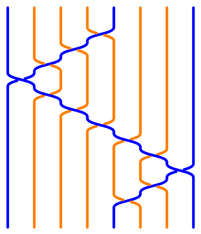


FIGURE 4.
$b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7\, b_6^{-1}\, b_5^{-1}$

We are now in a position to *splice* in a 3–core with ℓ strands into the 3–core with k strands, w, to obtain a new core with k strands and exponent 6. In general, if there is a section of a braid that takes the form of ℓ consecutive parallel strands, it is natural to insert an ℓ strand braid in that region; we term this operation *splicing* and begin with two concrete examples. First consider the core with T–identity $t_1\, t_5\, t_9 = -1$,

$$w = \left(b_4^{-1}\, b_3^{-1}\, b_2\, b_1\, b_2^{-1}\, b_3\, b_4\right)\cdot\left(b_8\, b_7^{-1}\, b_6^{-1}\, b_5\, b_6\, b_7\, b_8^{-1}\right).$$

Here $P(w) = \{2,3,4,6,7,8\}$ and then we can see that $R = \{3,4,6,7\}$ forms a ribbon since the initial exponent of b_3, b_4, b_6, b_7 are all -1. The ribbon that forms allows for a 4 strand 3–core to be spliced in which takes the form, for example,

$$v = b_5^{-1}\, b_6\, b_5\, b_7,$$

and the final core obtained is given by

$$x = \left(b_4^{-1}\, b_3^{-1}\, (b_5^{-1}\, b_4\, b_5\, b_6)\, b_2\, b_1\, b_2^{-1}\, b_3\, b_4\right)\cdot\left(b_8\, b_7^{-1}\, b_6^{-1}\, b_5\, b_6\, b_7\, b_8^{-1}\right).$$

The T–identity for the core x is given by

$$t_1\, t_5\, t_9 = -1, \quad t_3\, t_6\, t_7 = -1.$$

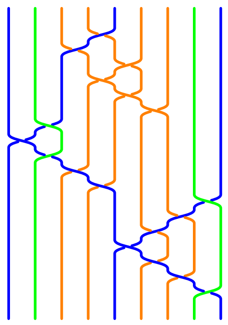


FIGURE 5. $b_4^{-1}\, b_3^{-1}\, b_5^{-1}\, b_4\, b_5\, b_6\, b_2\, b_1\, b_2^{-1}\, b_3\, b_4\, b_8\, b_7^{-1}\, b_6^{-1}\, b_5\, b_6\, b_7\, b_8^{-1}$

In the second example we have the 3–core, $b_3\, b_4$, spliced into a core where a three strand ribbon lies entirely to the left of x:

$$\mathit{Splice}(w, v) = \left(b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, (b_3\, b_4)\, b_1\, b_2\, b_3\, b_4\,\right) \cdot \left(b_8\, b_7\, b_6\, b_5\, b_6^{-1}\, b_7^{-1}\, b_8^{-1}\right).$$

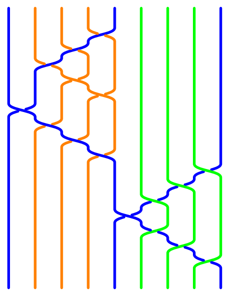


FIGURE 6. $b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_3\, b_4\, b_1\, b_2\, b_3\, b_4\, b_8\, b_7\, b_6\, b_5\, b_6^{-1}\, b_7^{-1}\, b_8^{-1}$

To write down the general case we begin by breaking R into the disjoint union

$$R \;=\; R_1 \cup R_2,$$

where $R_1 = \{y_1, \ldots, y_i\}, R_2 = \{y_j, \ldots, y_\ell\}$. Remark that either side of the above disjoint union could be empty making the ribbon appear on one side of x (if $R_2 = \emptyset$ then $y_i = y_\ell$ and likewise if $R_1 = \emptyset$ then $y_1 = y_j$). For ease of notation let

$$\delta = \epsilon_{y_1} = \epsilon_{y_2} = \cdots = \epsilon_{y_\ell}.$$

Choose a 3–core, v, involving the generators $\{b_{y_1+1}, \ldots, b_{y_\ell-1}\}$; we use the functional notation $v = v(b_{y_1+1}, \ldots, b_{y_\ell-1})$, and we note that T–identity is given by

$$t_{y_1}\, t_{x_0}\, t_{y_\ell} = -1,$$

for some $x_0 \in \{y_1+1, \ldots, y_\ell-1\}$. The output of the splicing method is given in the following proposition.

Proposition A.1. *With all the notation above in place, the core obtained by splicing the* 3*–core* v *into the* 3*–core*

$$w \;=\; b_{x-1}^{\epsilon_{x-1}}\, b_{x-2}^{\epsilon_{x-2}} \cdots b_2^{\epsilon_2}\, b_1\, b_2^{-\epsilon_2} \cdots b_{x-1}^{-\epsilon_{x-1}} \;\cdot\; b_{k-1}^{\epsilon_{k-1}}\, b_{k-2}^{\epsilon_{k-2}} \cdots b_{x+1}^{\epsilon_{x+1}}\, b_x\, b_{x+1}^{-\epsilon_{x+1}} \cdots b_{k-1}^{-\epsilon_{k-1}}$$

is given as follows:

(i) If $R_1, R_2 \neq \emptyset$*, we have* $y_i = x-1, y_j = x+1$ *and*

$$\begin{aligned} \mathit{Splice}(w,v) = {}& b_{y_i}^{\delta}\, b_{y_i-1}^{\delta} \cdots b_{y_1}^{\delta} v(b_{y_1+1}, \ldots, b_{y_\ell-1}) b_{y_1-1}^{\epsilon_{y_1-1}} \cdots b_2^{\epsilon_2}\, b_1\, b_2^{-\epsilon_2} \cdots b_{y_1-1}^{-\epsilon_{y_1-1}} b_{y_1}^{-\delta} \cdots b_{y_i}^{-\epsilon_{y_i}} \\ & \cdot b_{k-1}^{\epsilon_{k-1}} \cdots b_{y_\ell+1}^{\epsilon_{\ell+1}}\, b_{y_\ell}^{\delta} \cdots b_{y_j}^{\delta}\, b_x\, b_{y_j}^{-\delta} \cdots b_{y_\ell}^{-\delta}\, b_{y_\ell+1}^{-\epsilon_{\ell+1}} b_{k-1}^{-\epsilon_{k-1}} \end{aligned}$$

(ii) If $R_1 \neq \emptyset, R_2 = \emptyset$*, we have*

$$\begin{aligned} \mathit{Splice}(w,v) = {}& b_{x-1}^{\epsilon_{x-1}} \cdots b_{y_i+1}^{\epsilon_{y_i+1}}\, b_{y_i}^{\delta}\, b_{y_i-1}^{\delta} \cdots b_{y_1}^{\delta}\, v(b_{y_1+1}, \ldots, b_{y_i})\, b_{y_1-1}^{\epsilon_{y_1-1}} \cdots b_2^{\epsilon_2}\, b_1 \cdot \\ & b_2^{-\epsilon_2} \cdots b_{y_1-1}^{-\epsilon_{y_1-1}} b_{y_1}^{-\delta} \cdots b_{y_i}^{-\delta}\, b_{y_i+1}^{-\epsilon_{y_i+1}} \cdots b_{x-1}^{-\epsilon_{x-1}} \\ & \cdot b_{k-1}^{\epsilon_{k-1}}\, b_{k-2}^{\epsilon_{k-2}} \cdots b_{x+1}^{\epsilon_{x+1}}\, b_x\, b_{x+1}^{-\epsilon_{x+1}} \cdots b_{k-1}^{-\epsilon_{k-1}} \end{aligned}$$

(iii) If $R_1 = \emptyset, R_2 \neq \emptyset$*, we have*

$$\begin{aligned} \mathit{Splice}(w,v) = {}& b_{x-1}^{\epsilon_{x-1}}\, b_{x-2}^{\epsilon_{x-2}} \cdots b_2^{\epsilon_2}\, b_1\, b_2^{-\epsilon_2} \cdots b_{x-1}^{-\epsilon_{x-1}} \cdot \\ & \cdot b_{k-1}^{\epsilon_{k-1}} \cdots b_{y_\ell+1}^{\epsilon_{\ell+1}}\, v(b_{y_j}, \ldots, b_{y_\ell-1})\, b_{y_\ell}^{\delta} \cdots b_{y_j}^{\delta}\, b_{y_j-1}^{\epsilon_{y_j-1}} \cdots b_{x+1}^{\epsilon_{x+1}}\, b_x \\ & b_{x+1}^{-\epsilon_{x+1}} \cdots b_{y_j-1}^{-\epsilon_{y_j-1}} b_{y_j}^{-\delta} \cdots b_{y_\ell}^{-\delta}\, b_{y_\ell+1}^{-\epsilon_{\ell+1}} b_{k-1}^{-\epsilon_{k-1}}. \end{aligned}$$

A second method of splicing will involve splicing a 4–core into a 3–core. In the above examples a 3–core was spliced onto a ribbon that lay either behind or in front of the strands originating from the points $1, x, k$ in the 3–core being altered. It's natural to ask what happens when one if the strands in the ribbon originates, for example from x. Thus we consider, for example, the 3–core with $k = 6$ strands and $x = 5$:

$$w = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_4\, b_5,$$

whose T–identity is $t_1 t_5 t_6 = -1$. Next consider the 4–core

$$v = b_2\, b_3\, b_4,$$

whose T–identity is $t_2 t_3 t_4 t_5 = -1$. When we splice v^4 into w,

$$\mathit{Splice}(w, v^4) = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, (b_2\, b_3\, b_4)^4\, b_1\, b_2\, b_3\, b_4\, b_5,$$

we obtain a new core with exponent 6 whose T–identity takes the form

$$t_1 t_5 t_6 = 1, \qquad t_2 t_3 t_4 = -1.$$

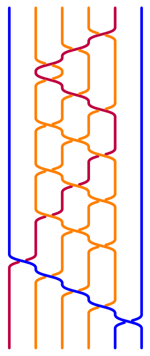


FIGURE 7. $b_4^{-1}b_3^{-1}b_2^{-1}(b_2b_3b_4)^4b_1b_2b_3b_4b_5$

Similarly, if we choose the 4–core $v = b_3\, b_4\, b_5$, whose T–identity is $t_3t_4t_5t_6 = -1$, splicing v^4 into w

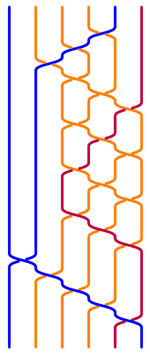


FIGURE 8. $b_4^{-1}b_3^{-1}b_2^{-1}(b_3b_4b_5)^4b_1b_2b_3b_4b_5$

$$Splice(w, v^4) = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, (b_3\, b_4\, b_5)^4\, b_1\, b_2\, b_3\, b_4\, b_5,$$

we obtain a new core with exponent 6 whose T–identity again takes the form

$$t_1t_5t_6 = 1, \qquad t_2t_3t_4 = -1.$$

A slightly different example begins with the 3–core $w = b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_5^{-1} b_4\, b_5$, and again $v = b_3\, b_4\, b_5$.

Here we obtain a new core with exponent 6,

$$w = b_3^{-1}\, b_2^{-1}\, b_5^{-1}\, (b_3\, b_4\, b_5)^4\, b_1\, b_2\, b_3\, b_4\, b_5,$$

where the T–identities are given by

$$t_1t_4t_6 = 1, \qquad t_2t_3t_5 = -1.$$

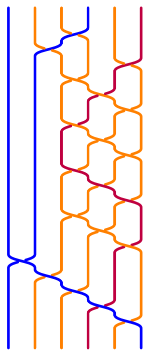


FIGURE 9. $b_3^{-1}b_2^{-1}b_5^{-1}(b_3b_4b_5)^4b_1b_2b_3b_4b_5$

The commonality in the above examples is the presence of one of the strands which begin at x (or k in the latter case) eliminates that T–value in the T–identity (said strand is purple in figures 8,9,10).

One approach to understanding these core is demonstrated as follows: returning to the first example

$$Splice(w, v^4) = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, (b_2\, b_3\, b_4)^4\, b_1\, b_2\, b_3\, b_4\, b_5,$$

where $w = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_4\, b_5$ and $v = b_2\, b_3\, b_4$, we begin by observing

$$Splice(w, v^4) = Splice(w, u^4) = b_4^{-1}\, b_3^{-1}\, b_2^{-1}\, b_1\, b_2\, b_3\, b_4\, b_5\, (\, b_1\, b_2\, b_3)^4,$$

where $u = b_1\, b_2\, b_3$.

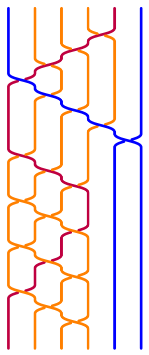


FIGURE 10. $b_4^{-1}b_3^{-1}b_2^{-1}b_1b_2b_3b_4b_5(b_1b_2b_3)^4$

Consider the following set of an alternative set of generators for the braid group B_N:

$$y_i = b_i b_{i+1} \cdots b_{N-1},$$

which satisfy the identity

$$y_j\, y_i = y_i\, y_{j-1}\, y_{N-1}^{-1}, \quad \text{if } \ i < j.$$

Specializing to our $N = 6$ case, we simplify $Splice(w, u^4)^3$ as follows:

$$Splice(w, u^4)^3 = y_1^6\, y_2^3\, y_4^6\, y_2^{-3}.$$

Observe that since y_1^6 generates the center of B_6, and hence for any power q,

$$Splice(w, u^4)^{3q} = y_1^{6q}\, y_2^3\, y_4^{6q}\, y_2^{-3}.$$

Now the T–identity which will make y_1^6 a core with exponent 2 is

$$t_1t_2t_3t_4t_5t_6 = -1.$$

Likewise T–identity which will make y_4^6 a core with exponent 2 is $t_4t_5t_6 = -1$. Letting the inverse of the permutation of y_2^3 act on $t_4t_5t_6 = -1$ we obtain $t_2t_3t_4 = -1$,

$$\sigma_{y_2^3}^{-1}\Big(t_4t_5t_6 = -1\Big) \text{ yields } \Big(t_2t_3t_4 = -1\Big).$$

and in combination with $t_1t_2t_3t_4t_5t_6 = -1$, we deduce the T–identity for $Splice(w, u^4)^3$: $Splice(w, u^4)^3$ is a core with exponent 2 and T–identities

$$t_1t_2t_3t_4t_5t_6 = -1, \qquad t_2t_3t_4 = -1,$$
$$\Longrightarrow\ t_1t_5t_6 = 1, \qquad t_2t_3t_4 = -1.$$

The above analysis serves as guide to generating even more cloaking elements. Again starting with $N = 6$ we can looks at braids of the form

$$y_1^6\, z\, y_4^6\, z^{-1} \;=\; (b_1\, b_2\, b_3\, b_4\, b_5)^6 \cdot z\, (b_4\, b_5)^6\, z^{-1}$$

where z is a braid that doesn't commute with $(b_4\, b_5)$. In practice we would choose z so that its associate permutation σ_z does not fix the set $\{4, 5, 6\}$.

To move from $k = 6$ to a larger k we can thread additional strands into a core of the type in such a way that the core itself is not altered. To accomplish this, suppose we would like to generate a core with $k = 8$ by threading additional strands in at points $3, 6$. Begin by choosing a permutation in $\sigma \in S_8$ which maps $3 \to 7, 6 \to 8$. Next choose a braid $\beta \in B_8$ which is a lift of σ. Viewing the braid $y_1^6\, z\, y_4^6\, z^{-1} \in B_8$, the conjugate

$$\beta\, y_1^6\, z\, y_4^6\, z^{-1}\, \beta^{-1},$$

will be a core with $k = 8$ strand of exponent 2 with T–identities

$$\sigma_{\beta\cdot z}^{-1}\Big(t_4t_5t_6 = -1\Big), \qquad \sigma_{\beta}^{-1}\Big(t_1t_2t_3t_4t_5t_6 = -1\Big).$$

Appendix B. Some combinatorics of the splicing method

Having generated two types of cores via a splicing technique, what remains is to obtain a lower bound on the size of the set of generated cores. With this goal in mind we begin with the cores described in Proposition A.1 of Appendix A. Let $N \geq 10$, and integers (points) x, k chosen so that $1 < x < k \leq N$. Thus, we begin again with w a 3–core with k strands) which by definition takes the form,

$$w \;=\; b_{x-1}^{\epsilon_{x-1}}\, b_{x-2}^{\epsilon_{x-2}} \cdots b_2^{\epsilon_2}\, b_1\, b_2^{-\epsilon_2} \cdots b_{x-1}^{-\epsilon_{x-1}} \;\cdot\; b_{k-1}^{\epsilon_{k-1}}\, b_{k-2}^{\epsilon_{k-2}} \cdots b_{x+1}^{\epsilon_{x+1}}\, b_x\, b_{x+1}^{-\epsilon_{x+1}} \cdots b_{k-1}^{-\epsilon_{k-1}},$$

and the set

$$P(w) \;=\; \{2, \ldots, x-1, x+1, \ldots, k-1\}.$$

Assuming we have an ℓ strand ribbon

$$R = \{y_1, y_2, \ldots, y_\ell\} \subset P(w),$$

on which to splice an ℓ strand 3–core v, the question becomes how many ways can the ribbon appear in $P(w)$, how many unconstrained strands will remain, and how many ℓ strand 3–cores v are there to splice in. It's clear that the number of ways to fit an ℓ strand ribbon is given by

$$k - \ell - 2,$$

and the number of strands that are left without constraint on their exponent is given by

$$k - \ell - 3.$$

As we saw in Appendix A, the number of ℓ strand 3–cores is given by $(\ell - 2) \cdot 2^{(\ell-1)}$, and in each such core there are $(N - 2)$ ways to choose the point $1 \leq x \leq N - 2$. Further, each such core can be spliced into a ribbon which lies either behind or in front of the original k strand 3–core. In summation, the number of cores generated with this first splicing method where we allow, for example, $8 \leq k \leq 10$ and $5 \leq \ell \leq 7$ is given by,

$$\sum_{k=8}^{10} (k-2) \sum_{\substack{\ell=5 \\ \ell < k-3}}^{7} 2 \cdot \Big((k - \ell - 2) \cdot (\ell - 2) \cdot 2^{(\ell-1)} \cdot 2^{(k-\ell-3)} \Big) \approx 2^{13.7}. \tag{15}$$

We move next to the cores where an ℓ strand 4–core is spliced into a k strand 3–core (where $k \leq 10$) which we saw in Appendix A may take the from

$$\beta\, y_1^6\, z\, y_4^6\, z^{-1}\, \beta^{-1}.$$

We recall $y_i = b_i \cdot b_{i+1} \cdots b_{N-1}$, z is an element in the subgroup generated by $\{b_2, b_3, b_4, b_5\}$ that does not commute with y_4^6. Further, β is a lift of a permutation that maps a set of $k - 6$ points to $\{7, \ldots, k\}$ and maps the remaining 6 points to $\{1, 2, 3, 4, 5, 6\}$. In order to get a lower bound on how many such cores we can generate we estimate the number of possible elements z and β that are bounded by a given length.

To that end, recall the estimate for the number of words of length L in the braid group B_N from §9 which is given by

$$\frac{2^L}{L} \cdot (N-1) \binom{L-2+N}{N-1}.$$

One collection of elements z that will not (in general) commute with y_4^6 are braids of the form $z_1 \cdot b_3$, where z_1 is again an element in the subgroup generated by $\{b_2, b_3, b_4, b_5\}$. We conclude that the collection of elements z that have expressions as a product of L_1 Artin generators is bounded below by

$$\frac{2^{L_1+2}}{L_1} \cdot \binom{L_1+3}{4}.$$

A lower bound for the number of choices for the element β can be obtained as follows. Let k be the number of strands being threaded into 6 strand braid $y_1^6\, z\, y_4^6\, z^{-1}$. Then, since $0 \leq k \leq 4$, we see that there are

$$\prod_{j=1}^{k} \Big((6-2) + j \Big)$$

possibilities. The permutation σ_β associated with β, when lifted to the braid group, can be modified by a pure braid of some fixed length making it possible to obtain a rough lower bound on the number of elements β. Assuming we are modifying by a pure braid which is expressed as a word of length q in the pure braid generators, the length as a word in the Artin generators will on average have length about $q(N-1)$. Hence, the number of ways to generate the requisite β (which we assume can be written as a product of L_2 Artin generators and is obtained by lifting a permutation $\sigma_\beta$which is written as a product of θ transpositions above) is given by

$$\prod_{j=1}^{k}(6+j-2)\cdot N(N-1)^{\frac{(L_2-\theta)}{6+k-1}}.$$

We observe that by restricting our attention to the case where $\theta = 5k$ then the collection of generated braids is on the order of 2^{32}.

To conclude this discussion we state the following proposition:

Proposition B.1. *Let $N \geq 6$. Consider cores of B_N that take the form*

$$\beta\, y_1^6\, z\, y_4^6\, z^{-1}\, \beta^{-1}$$

where

- *$\beta \in B_N$ is as discussed above and has an expression of length L_2 in Artin generators;*
- *θ is the minimal number of ways of writing σ_β as a product of transpositions;*
- *z is an element in the subgroup generated by $\{b_2, b_3, b_4, b_5\}$ as discussed above which has an expression of length L_1 in in Artin generators.*

Then a lower bound for the number of such cores is given by

$$(N-1)^{\frac{L_2-\theta}{5+k}}\cdot\frac{N\cdot 2^{L_1+2}}{L_1}\cdot\binom{L_1+3}{4}\cdot\prod_{j=1}^{k}(4+j). \tag{16}$$

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
Email address: ianshel@veridify.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
Email address: datkins@veridify.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
Email address: dgoldfeld@veridify.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
Email address: pgunnells@veridify.com