A Practical Cryptanalysis of WalnutDSA™

Daniel Hart[1], DoHoon Kim[1], Giacomo Micheli[1], Guillermo Pascual Perez[1], Christophe Petit[2], and Yuxuan Quek[1]

[1] University of Oxford.
[2] University of Birmingham.

Abstract. We present a practical cryptanalysis of WalnutDSA, a digital signature algorithm trademarked by SecureRF. WalnutDSA uses techniques from permutation groups, matrix groups, and braid groups, and is designed to provide post-quantum security in lightweight IoT device contexts. The attack given in this paper bypasses the E-Multiplication™ and cloaked conjugacy search problems at the heart of the algorithm and forges signatures for arbitrary messages in approximately two minutes. We also discuss potential countermeasures to the attack.

1 Introduction

Most of the cryptosystems in use today are based on two difficult problems: the integer factorization problem and the Discrete Logarithm Problem (DLP). Both of these problems can be solved efficiently by running Shor's algorithm [1] on a sufficiently large quantum computer. As of now, such quantum computers do not exist, but organisations such as NIST and the NSA are striving for cryptosystems resilient to quantum attacks to prepare for the time when they become a reality [2,3,4].

The problem at the heart of Shor's algorithms, the so-called hidden subgroup problem, can be solved in polynomial time on a quantum computer for any finite abelian group, but has so far appeared much harder in the case for non-abelian groups. Cryptography based on non-abelian groups is therefore considered an appealing direction for post-quantum cryptography. Braid groups have traditionally been used in non-abelian based cryptography: for example, the Anshel-Anshel-Goldfeld (AAG) key-exchange protocol and the Diffie-Hellman-type key-exchange protocol are both based on the conjugacy search problem (or at least one of its variants) in a braid group [5, Section 1.6]. Today, more advanced protocols have evolved from these schemes.

SecureRF [6] is a corporation founded in 2004 specializing in security for the Internet of Things (IoT), i.e. devices with low processing power that require ultra-low energy consumption, whose partners include the US Air Force. WalnutDSA [7] is a digital signature algorithm developed by SecureRF that was presented at the NIST Lightweight Cryptography Workshop 2016. SecureRF has collaborated with Intel [8] to develop an implementation of WalnutDSA

on secure field-programmable gate arrays (FPGAs). Hence, the importance of WalnutDSA as a digital signature algorithm, in a world where corporations and government agencies are consistenly pushing for post-quantum cryptosystems, cannot be understated, due to its potential for widespread use.

1.1 Our Contribution

We provide a universal forgery attack on WalnutDSA. Our attack does not require a signing oracle: in fact, having access to a small set of random message-signature pairs suffice. In principle, the security of WalnutDSA is based on the difficulty of reversing E-Multiplication and the cloaked conjugacy search problem [7, Problems 1, 2], but we go around this by reducing the problem of forging a WalnutDSA signature to an instance of the factorization problem in a non-abelian group (given a group element $g \in G$ and a generating set Γ for G, find a word w over Γ such that $w = g$). While this problem is hard in general, we give an efficient algorithm for solving it in this context. Given a couple of valid signatures on random messages, our attack can produce a new signature on an arbitrary message in approximately two minutes. We also discuss countermeasures to prevent this attack.

Responsible Disclosure Process. Since WalnutDSA is advertised as a security product by SecureRF, we notified its authors of our findings before making them available to the public. We informed them by email on October 17th with full details of our attack. They acknowledged the effectiveness of our attack on October 19th, and we agreed to postpone our publication until November 26th.

Two countermeasures are discussed in our paper, namely checking the signature length and increasing the parameters. SecureRF have communicated to us that they have always had a limit on signature lengths in their product offerings, and that the increase in parameter sizes we suggest may still allow for many applications in devices with limited computing power. These two countermeasures can prevent our attack for now. As we briefly argue in Section 5 below, improved versions of our attack might be able to defeat them, but we leave these to further work.

In reaction to our attack, SecureRF developed a modification to WalnutDSA using two private keys (instead of conjugation), such that Proposition 4 of this paper fails to apply.

1.2 Related Work

Ben-Zvi, Blackburn and Tsaban [9] provide a complete attack on a version of SecureRF's Algebraic Eraser scheme, a public key encryption protocol also based on E-Multiplication. Other attacks on Algebraic Eraser include those by Myasnikov and Ushakov [10], which is a length based attack on SecureRF's specific

realisation of the general scheme, and by Kalka, Teicher and Tsaban [11], which is a cryptanalysis for arbitrary parameter sizes.

Other important work includes Garside's papers [12,13] on solving the conjugacy search problem in braid groups using Summit Sets and papers on Garside normal form [12] and Dehornoy Handle Reduction [14].

Other instances of factorization problems in non-Abelian groups have been solved previously, in both cryptographic contexts [15,16,17] and in the mathematics literature [18]. The algorithms we develop in this paper for factorization in $\mathrm{GL}_N(\mathbb{F}_q)$ belongs to the family of subgroup attacks [19].

1.3 Outline

In Section 2, we provide the definition of security for signature schemes, and introduce the factorization problem as well as some preliminary results about braid groups. In Section 3, we introduce the WalnutDSA protocol. In Section 4, we provide a cryptanalysis of WalnutDSA by first reducing the problem to a factorization problem in $\mathrm{GL}_N(\mathbb{F}_q)$ (Section 4.1) and then solving it (Section 4.2). In Section 5, we describe possible countermeasures to prevent the attack. We conclude the paper in Section 6.

2 Preliminaries

2.1 Security Definition

The standard security definition for signatures is *existential unforgeability under chosen message attacks* [20, Introduction] (we sometimes abbreviate this to *secure*). An adversary can ask for polynomially many signatures of messages of its choice to a signing oracle. Then, the attack is considered successful if the attacker is able to produce a valid pair of message and signature for a message different from those queried to the oracle. We will show that the version of WalnutDSA proposed in [7] is not resistant to this kind of attack and propose a modification to the scheme which fixes this weakness.

Definition 1. *A signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be* existentially unforgeable under adaptive chosen-message attacks *(or* secure*, for short) if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *with access to* $\mathsf{Sign}_{\mathrm{SK}}(\cdot)$*,*

$$\left|\Pr\left[\begin{array}{l}(\mathrm{PK},\mathrm{SK}) \leftarrow \mathsf{Gen}(1^\lambda); s_i \leftarrow \mathsf{Sign}_{\mathrm{SK}}(m_i) \text{ for } 1 \le i \le k; \\ (m,s) \leftarrow \mathcal{A}\big(\mathrm{PK}, (m_i)_{i=1}^k, (s_i)_{i=1}^k\big) : \\ \mathsf{Verify}_{\mathrm{PK}}(m,s) = 1 \text{ and } m \notin \mathcal{M}\end{array}\right]\right| \le \mathrm{negl}(\lambda) \ .$$

where $\mathcal{M} = \{m_1, \ldots, m_k\}$ *is the set of messages queried by* $\mathcal{A}$ *to the oracle, and* $k = \#\mathcal{M}$ *is polynomial in* λ*.*

For our cryptanalysis, the m_i can actually be random messages, leading to a stronger attack.

2.2 Braid Groups

For $N \geq 2$, the braid group [5] on N strands, denoted B_N, is a group with presentation

$$B_N = \left\langle b_1, \ldots, b_{N-1} \;\middle|\; \begin{array}{c} b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1} \\ b_i b_j = b_j b_i \text{ for } |i-j| \geq 2 \end{array} \right\rangle \ , \tag{1}$$

where the b_i are called *Artin generators*. There are other presentations for the braid group, but unless otherwise stated, we will use the definition provided in (1) and "generators" will refer to the Artin generators. Geometrically, the elements of a braid group are the equivalence classes of N strands under ambient isotopy, and the group operation is concatenation of the N strands. More precisely, the generator b_i corresponds to the $(i+1)$-th strand crossing over the i-th strand. Note that there is a natural homomorphism from B_N onto the symmetric group S_N: if $\beta = b_{i_1} \cdots b_{i_k}$, then the permutation induced by β is precisely

$$\prod_{j=1}^{k} (i_j,\ i_j + 1) \ ,$$

where $(i_j,\ i_j + 1)$ is the standard transposition in S_N.

Notation. Let $\mathfrak{p}\colon B_N \to S_N$ be the above map sending a braid to its induced permutation.

Braids that induce trivial permutations are called *pure braids*. The set of pure braids is exactly the kernel of the homomorphism $\mathfrak{p}$, and so forms a normal subgroup of B_N.

Garside Normal Form A normal form of an element in a group is a canonical way to represent the element. One known normal form for braid groups is the *Garside normal form*. The details can be found in Appendix A. We can compute the Garside normal form of a braid with complexity $O(|W|^2 N \log N)$ where $|W|$ is the length of the word in Artin generators [21]. Such a normal form is important for WalnutDSA, but the cryptanalysis we provide in Section 4 is independent of the choice of it.

The Colored Burau Representation. Let q be an arbitrary prime power, and let $\mathbb{F}_q$ be the finite field with q elements. Let $\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]$ be the ring of Laurent polynomials with coefficients in $\mathbb{F}_q$. Note that there is a natural

action of S_N on $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1},\dots,t_N^{\pm 1}]\right)$, where a permutation acts on a Laurent polynomial by permuting its variables. In other words, we have an action $f \mapsto {}^{\sigma}f$ where $f(t_1,\dots,t_N)$ is mapped to $f(t_{\sigma(1)},\dots,t_{\sigma(N)})$. Similarly, a permutation may act on a matrix M in $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1},\dots,t_N^{\pm 1}]\right)$ entrywise, and we will denote the image of M under this action as ${}^{\sigma}M$.

Proposition 1. *There exists a group homomorphism, called the* colored Burau representation [7]*,*

$$\Phi\colon B_N \to \mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1},\dots,t_N^{\pm 1}]\right) \rtimes S_N \ ,$$

where $\rtimes$ denotes the semidirect product.
Let $\mathfrak{m}$ be the projection of $\mathfrak{m}$ on $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1},\dots,t_N^{\pm 1}]\right)$. Then Φ is defined as follows:

- *For the generator $b_1 \in B_N$, define*

$$\mathfrak{m}(b_1) = \begin{pmatrix} -t_1 & 1 & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \ ,$$

  and

$$\mathfrak{m}(b_1^{-1}) = \begin{pmatrix} -\frac{1}{t_2} & \frac{1}{t_2} & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \ .$$

- *For $2 \le i < N$, define*

$$\mathfrak{m}(b_i) = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & t_i & -t_i & 1 & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix} \ ,$$

  where the $-t_i$ occurs in the i-th row. Also define

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

– *Define*

$$\Phi(b_i) \coloneqq \big(\mathfrak{m}(b_i), \mathfrak{p}(b_i)\big) \ .$$

– *Given generators* $b_i^{\pm 1}, b_j^{\pm 1}$, *we define* $\Phi(b_i^{\pm 1} b_j^{\pm 1})$ *to be*

$$\big(\mathfrak{m}(b_i^{\pm 1}), \mathfrak{p}(b_i)\big) \cdot \big(\mathfrak{m}(b_j^{\pm 1}), \mathfrak{p}(b_j)\big) = \Big(\mathfrak{m}(b_i^{\pm 1}) \cdot \big({}^{\mathfrak{p}(b_i)}\mathfrak{m}(b_j^{\pm 1})\big), \mathfrak{p}(b_i)\mathfrak{p}(b_j)\Big) \ .$$

For a general braid β, *we extend this definition inductively to define* $\Phi(\beta)$.

Note that Φ and $\mathfrak{p}$ are homomorphisms, but $\mathfrak{m}$ is not a homomorphism in general. However, the following lemma shows that its restriction to pure braids is a homomorphism.

Lemma 1. *Let* $\phi : PB_N \to \mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]\right)$ *be the restriction map of* $\mathfrak{m}$ *to* PB_N. *This map is a group homomorphism.*

Proof. Let β_1, β_2 be pure braids. Then

$$\begin{aligned} \phi(\beta_1\beta_2) &= \mathfrak{m}(\beta_1\beta_2) \\ &= \mathfrak{m}(\beta_1) \cdot \big({}^{e_{\mathfrak{P}}}\mathfrak{m}(\beta_2)\big) \\ &= \mathfrak{m}(\beta_1)\mathfrak{m}(\beta_2) = \phi(\beta_1)\phi(\beta_2) \ , \end{aligned}$$

and so ϕ is indeed a homomorphism. □

Previous Cryptosystems Based on Braid Groups A problem that is generally difficult to solve in non-abelian groups is the conjugacy search problem (CSP), i.e. given conjugate elements $u, w \in B_N$, find $v \in B_N$ such that $w = v^{-1}uv$. This motivated the development of several cryptosystems based on the CSP in braid groups, some of which are given in [5]. Techniques such as summit sets [13,22,23], length-based attacks [24,25,26], and linear representations [27, 28,29], have been developed to attack the CSP in braid groups however, and so those cryptosystems have been rendered impractical. The design of WalnutDSA uses a variant of the CSP, the cloaked conjugacy search problem, to avoid these attacks.

2.3 Factorization Problem in Non Abelian Groups

Factorization Problem in Groups. Let G be a group, let $\Gamma = \{g_1, \ldots, g_\gamma\}$ be a generating set for G, and let $h \in G$. Find a "small" integer L and sequences $(m_1, \ldots, m_L) \in \{1, \ldots, \gamma\}^L$ and $(\epsilon_1, \ldots, \epsilon_L) \in \{\pm 1\}^L$ such that

$$h = \prod_{i=1}^{L} g_{m_i}^{\epsilon_i} \ .$$

Depending on the context, "small" may refer to a concrete practical size, or it may mean polynomial in $\log |G|$. The existence of products of size polynomial in $\log |G|$ for any finite simple non-abelian group, any generating set, and any element h was conjectured by Babai. This conjecture has attracted considerable attention from the mathematics community in the last 15 years, and has now been proven for many important groups [30,31].

The potential hardness of the factorization problem for non-abelian groups underlies the security of Cayley hash functions. The problem was solved in the particular cases of the Zémor [32,33], Tillich-Zémor [15,17,34], and Charles-Goren-Lauter [16,35,36] hash functions, and to a large extent in the case of symmetric and alternating groups [18], but it is still considered as a potentially hard problem in general. Over cyclic groups, this problem is known to be equivalent to the discrete logarithm problem when removing the constraint on L [37]. We refer to [19] for a more extensive discussion of the factorization problem and its connection with Babai's conjecture [38].

The instance of the factorization problem that appears in our attack is over $GL_N(\mathbb{F}_q)$, the general linear group of rank N over the finite field $\mathbb{F}_q$. Our solution for it exploits the particular subgroup structure of this group.

3 WalnutDSA

WalnutDSA$^{\text{TM}}$ is a digital signature scheme proposed by I. Anshel, D. Atkins, D. Goldfeld, P.E. Gunnells in [7], based on braid groups, E-Multiplication$^{\text{TM}}$ and cloaked conjugacy.

3.1 E-Multiplication

Let B_N be the braid group on N braids, let q be a prime power and let $\mathbb{F}_q^\times$ denote the non-zero elements of the finite field $\mathbb{F}_q$. Define a sequence of "T-values":

$$\tau = (\tau_1, \tau_2, \ldots, \tau_N) \in (\mathbb{F}_q^\times)^N \ .$$

Given the T-values, we can evaluate any Laurent polynomial $f \in \mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]$ to produce an element of $\mathbb{F}_q$:

$$f\big\downarrow_\tau \coloneqq f(\tau_1, \ldots, \tau_N) \ .$$

We can similarly evaluate any matrix M in $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]\right)$ by evaluating entrywise to produce a matrix $M\big\downarrow_\tau$ in $\mathrm{GL}_N(\mathbb{F}_q)$.

E-Multiplication [39] is a right action, denoted by $\star$, of the colored burau group $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]\right) \rtimes S_N$ on $\mathrm{GL}_N(\mathbb{F}_q) \times S_N$. In other words, it takes two ordered pairs

$$(M, \sigma_0) \in \mathrm{GL}_N(\mathbb{F}_q) \times S_N \ ,$$
$$\left(\mathfrak{m}(\beta), \mathfrak{p}(\beta)\right) \in \mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]\right) \rtimes S_N \ ,$$

and produces another ordered pair

$$(M', \sigma') = (M, \sigma_0) \star \big(\mathfrak{m}(\beta), \mathfrak{p}(\beta)\big)$$

in $\mathrm{GL}_N(\mathbb{F}_q) \times S_N$.

E-Multiplication is defined inductively. For a single generator b_i,

$$(M, \sigma_0) \star \big(\mathfrak{m}(b_i), \mathfrak{p}(b_i)\big) \coloneqq \Big(M \cdot {}^{\sigma_0}\big(\mathfrak{m}(b_i)\big)\big\downarrow_\tau,\ \ \sigma_0 \cdot \mathfrak{p}(b_i)\Big) \ .$$

For a general braid $\beta = b_{i_1}^{\epsilon_1} \cdots b_{i_k}^{\epsilon_k}$,

$$(M, \sigma_0) \star \big(\mathfrak{m}(\beta), \mathfrak{p}(\beta)\big) = (M, \sigma_0) \star \big(\mathfrak{m}(b_{i_1}^{\epsilon_1}), \mathfrak{p}(b_{i_1}^{\epsilon_1})\big) \star \cdots \star \big(\mathfrak{m}(b_{i_k}^{\epsilon_k}), \mathfrak{p}(b_{i_k}^{\epsilon_k})\big) \ \ ,$$

where $\star$ associates to the left. This is well-defined, as it is independent of how we write β in terms of the generators [7, Section 3].

Notation. We will follow the notation in [7] and write

$$(M, \sigma_0) \star \beta$$

instead of $(M, \sigma_0) \star \big(\mathfrak{m}(\beta), \mathfrak{p}(\beta)\big)$ for a braid $\beta \in B_N$.

Notation. For $\xi = (M, \sigma)$ in $\mathrm{GL}_N(\mathbb{F}_q) \times S_n$, let $\mathfrak{m}(\xi)$ denote the matrix part of ξ, i.e. $\mathfrak{m}(\xi) = M$.

3.2 Key Generation

Before the signer generates the private-/public-key pair, some public parameters are fixed:

- An integer N and the associated braid group B_N;
- A rewriting algorithm $\mathcal{R} : B_N \to B_N$, such as the Garside normal form;
- A prime power q defining a finite field $\mathbb{F}_q$ of q elements;
- Two integers $1 < a < b < N$;
- T-values $\tau = (\tau_1, \tau_2, \ldots, \tau_N) \in (\mathbb{F}_q^\times)^N$ with $\tau_a = \tau_b = 1$;
- An encoding function $\mathcal{E} : \{0,1\}^* \to B_N$ taking messages to braids.

The signer then chooses a random freely-reduced braid $\mathrm{SK} \in B_N$ (of the desired length to prevent brute force attacks from being effective) to be the private-key, and then calculates the public-key as

$$\mathrm{PK} = (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK} \ .$$

We will also use the following convention:

Notation. We follow the notaion in [7] and write $\mathrm{Pub}(\beta) \coloneqq (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \beta$ for a braid $\beta \in B_N$.

In [7], it is recommended to use $N \geq 8$ and $q \geq 32$ for the public parameters.

3.3 Message Encoding

To sign a message $m \in \{0,1\}^*$ using WalnutDSA, it must first be encoded as a braid $\mathcal{E}(m) \in B_N$. WalnutDSA achieves this by encoding messages as pure braids: given a message m, it is first hashed using a cryptographically secure hash function $H : \{0,1\}^* \to \{0,1\}^{4\kappa}$, where $\kappa \geq 1$. The paper [7] does not provide a formal definition of "cryptographically secure", but we believe that the intended meaning is that of a "random oracle" [40], and in this paper we will treat the hash function as such. The bitstring $H(m)$ is then encoded as a pure braid by noting that the $N-1$ braids

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

are pure braids that freely generate a subgroup of B_N [41]. Fix four of these generators, say $g_{k_1,N}, g_{k_2,N}, g_{k_3,N}, g_{k_4,N}$ for $1 \leq k_i \leq N-1$, and define

$$C = \langle g_{k_1,N}, g_{k_2,N}, g_{k_3,N}, g_{k_4,N} \rangle \subset PB_N \ .$$

Each 4-bit block of $H(m)$ can then be mapped to a unique power of one of these generators: the first two bits determine the generator $g_{k_\mu,N}$ to use, while the last two bits determine the power $1 \leq i \leq 4$ to raise the generator to. The encoded message $\mathcal{E}(m) \in C$ is then defined to be the freely reduced product of the κ powers of the $g_{k_i,N}$ obtained via the above process.

3.4 Cloaking Elements

WalnutDSA defines and uses "cloaking elements" to avoid being reduced to CSP, reducing instead to the cloaked conjugacy search problem. A braid $\beta \in B_N$ is said to be a cloaking element of $(M,\sigma) \in \mathrm{GL}_N(\mathbb{F}_q) \times S_N$ if $(M,\sigma) \star \beta = (M,\sigma)$. The set of cloaking elements of (M,σ) is then the stabilizer of (M,σ) under the E-Multiplication action, and so forms a subgroup of B_N.

Lemma 2. *Any cloaking element is a pure braid.*

Proof. Let $\beta \in B_N$ be a cloaking element of $(M,\sigma) \in \mathrm{GL}_N(\mathbb{F}_q) \times S_N$. Then

$$(M,\sigma) = (M,\sigma) \star \beta = \Big(M \cdot {}^{\sigma}\big(\mathfrak{m}(\beta)\big) \ , \ \sigma \cdot \mathfrak{p}(\beta) \Big) \ ,$$

which implies that $\mathfrak{p}(\beta) = \mathrm{Id}_{S_N}$. □

The authors of WalnutDSA provide a method of generating cloaking elements [7, Proposition 4.2], which we recap here for the reader's convenience.

Proposition 2. *Fix integers $N \geq 2$ and $1 < a < b < N$. Assume that $\tau_a = \tau_b = 1$. Let $M \in \mathrm{GL}_N(\mathbb{F}_q)$ and $\sigma \in S_N$. Then a cloaking element β of (M, σ) is given by $\beta = wb_i^2w^{-1}$ where b_i is any Artin generator and $w \in B_n$ is any braid such that the associated permutation $\mathfrak{p}(w)$ satisfies*

$$\mathfrak{p}(w)(i) = \sigma^{-1}(a)\ ,\ \mathfrak{p}(w)(i+1) = \sigma^{-1}(b)\ .$$

Remark 1. A detailed algorithm for constructing cloaking elements is not provided. In particular, no algorithm to generate w is given. Hence, in our implementation, we generate it in the following way:

Algorithm 1 Generating w

repeat
  Pick a random integer l such that $30 \leq l \leq 80$.
  Pick a random freely-reduced word w in the generators $\{b_1, \ldots, b_7\}$ of length l.
until $\mathfrak{p}(w)$ satisfies the condition in Proposition 2.

We stress that our attack works independently of the way cloaking elements β are generated.

3.5 Signing

Signing. To sign a message m, the signer does as follows:

1. Compute $\mathcal{E}(m)$ as in Section 3.3;
2. Generate cloaking elements v for $(\mathrm{Id}_N, \mathrm{Id}_{S_N})$ and v_1, v_2 for $(\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK}$;
3. Compute $s = \mathcal{R}(v_2 \cdot \mathrm{SK}^{-1} \cdot v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1)$;
4. Output the final signature for the message, (m, s).

The cloaking elements are necessary to preclude the possiblity of recovering for SK by solving the CSP (any solution to the CSP is sufficient), since both s and $\mathcal{E}(m)$ are publicly available (the latter after some computation).

Proposition 3. *For any message m, its signature*

$$s = \mathcal{R}\left(v_2 \cdot \mathrm{SK}^{-1} \cdot v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right)$$

is a pure braid.

Proof. Recall that $\mathcal{E}(m)$ is a product of pure braids and is, therefore, a pure braid. Moreover, by Lemma 2, v, v_1 and v_2 are pure braids. Hence, the induced

permutation $\mathfrak{p}(s)$ of s is:

$$\begin{aligned}\mathfrak{p}(s) &= \mathfrak{p}\left(v_2 \cdot \mathrm{SK}^{-1} \cdot v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right) \\ &= \mathrm{Id}_{S_N} \cdot \mathfrak{p}(\mathrm{SK}^{-1}) \cdot \mathrm{Id}_{S_N} \cdot \mathrm{Id}_{S_N} \cdot \mathfrak{p}(\mathrm{SK}) \cdot \mathrm{Id}_{S_N} \\ &= \mathrm{Id}_{S_N} \ .\end{aligned}$$

□

3.6 Verifying

Verifying. To verify a signature (m, s), the verifier does as follows:

1. Compute $\mathcal{E}(m)$;
2. Compute $\mathrm{Pub}\left(\mathcal{E}(m)\right) = (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathcal{E}(m)$.

The signature is then valid if and only if the verification equation

$$\mathfrak{m}(\mathrm{PK} \star s) = \mathfrak{m}\Big(\mathrm{Pub}\left(\mathcal{E}(m)\right)\Big) \cdot \mathfrak{m}(\mathrm{PK})$$

holds.

Lemma 3. *A message-signature pair (m, s), generated as in Section 3.5 satisfies the verification process.*

Proof. We have that

$$\begin{aligned}\mathrm{PK} \star s &= (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK} \star s \\ &= (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK} \star \left(v_2 \cdot \mathrm{SK}^{-1} \cdot v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right) \\ &\overset{(1)}{=} (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK} \star \left(\mathrm{SK}^{-1} \cdot v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right) \\ &= (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \left(v \cdot \mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right) \\ &\overset{(2)}{=} (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \left(\mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right) \ ,\end{aligned}$$

where

- (1) holds since v_2 cloaks $\mathrm{PK} = (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK}$;
- (2) holds since v cloaks $(\mathrm{Id}_N, \mathrm{Id}_{S_N})$.

Looking at the matrix parts of the above equality, we see that

$$\begin{aligned}\mathfrak{m}(\mathrm{PK} \star s) &= \mathfrak{m}\Big((\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \left(\mathcal{E}(m) \cdot \mathrm{SK} \cdot v_1\right)\Big) \\ &\overset{(3)}{=} \mathfrak{m}\big((\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathcal{E}(m)\big) \cdot \mathfrak{m}\big((\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star (\mathrm{SK} \cdot v_1)\big) \\ &\overset{(4)}{=} \mathfrak{m}\big((\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathcal{E}(m)\big) \cdot \mathfrak{m}\big((\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star \mathrm{SK}\big) \\ &= \mathfrak{m}\Big(\mathrm{Pub}\left(\mathcal{E}(m)\right)\Big) \cdot \mathfrak{m}(\mathrm{PK}) \ ,\end{aligned}$$

where

– (3) holds since $\mathcal{E}(m)$ is a pure braid
– (4) holds since v_1 cloaks $\text{PK} = (\text{Id}_N, \text{Id}_{S_N}) \star \text{SK}$.

□

4 Practical Cryptanalysis of WalnutDSA

In this section we present a universal forgery attack on WalnutDSA. The structure of the section is as follows: in Section 4.1, we show that an attacker can produce a signature for a new message if they are able to solve a factorization problem over $\text{GL}_N(\mathbb{F}_q)$. In Section 4.2, we present an algorithm solving this factorization problem by exploiting the subgroup structure of $\text{GL}_N(\mathbb{F}_q)$, and in Section 4.3, we describe a meet-in-the-middle approach which reduces the complexity of this attack. In Section 4.4, we analyze the complexity of our attack and provide some experimental results. Finally, we discuss further improvements to our attack in Section 4.5.

4.1 Reduction to the Factorization Problem

Let I be a finite indexing set. For each $i \in I$, let m_i be a message and s_i be its signature generated as in Section 3.5. Define the set $\mathcal{M} = \{(m_i, s_i) : i \in I\}$. Recall that for a braid β, we define

$$\text{Pub}(\beta) = (\text{Id}_N, \text{Id}_{S_N}) \star \beta \ ,$$

where Id_N is the identity matrix and Id_{S_N} is the identity permutation.

Proposition 4. *Let m be an arbitrary message. Let $g_i = \mathfrak{m}\big(\text{Pub}(\mathcal{E}(m_i))\big)$ for each $i \in I$ and let $h = \mathfrak{m}\big(\text{Pub}(\mathcal{E}(m))\big)$. Suppose*

$$h = \prod_{j=1}^{L} g_{i_j}^{\epsilon_{i_j}} \qquad \textit{where } i_j \in I,\ \epsilon_{i_j} \in \{\pm 1\} \textit{ and } L \in \mathbb{N} \ .$$

Then $s = \prod_{j=1}^{L} s_{i_j}^{\epsilon_{i_j}}$, the concatenation of the corresponding braids $s_{i_j}^{\epsilon_{i_j}}$, is a valid signature for m.

Proof. Each pair in $\mathcal{M}$ satisfies the verification equation:

$$\mathfrak{m}(\text{PK} \star s_i) = \mathfrak{m}\Big(\text{Pub}\left(\mathcal{E}(m_i)\right)\Big) \cdot \mathfrak{m}(\text{PK}) \ .$$

Writing σ as $\mathfrak{p}(\text{PK})$ and M as $\mathfrak{m}(\text{PK})$, the above equation is equivalent to

$$\big({}^{\sigma}\mathfrak{m}(s_i)\big)\big\downarrow_{\tau} = M^{-1} \cdot g_i \cdot M \ , \tag{2}$$

where $\tau = (\tau_1, \ldots, \tau_N)$ is the sequence of T-values. Also, by Proposition 3, each $s_i^{\epsilon_i}$ is a pure braid, so that

$$(M, \sigma) = (M, \sigma) \star (s_i \cdot s_i^{-1}) = \left(M \cdot^{\sigma} \mathfrak{m}(s_i)\big\downarrow_{\tau} \cdot^{\sigma} \mathfrak{m}(s_i^{-1})\big\downarrow_{\tau}, \sigma\right) \ ,$$

which implies

$$\left(\left({}^{\sigma}\mathfrak{m}(s_i)\right)\big\downarrow_{\tau}\right)^{-1} = \left({}^{\sigma}\mathfrak{m}(s_i^{-1})\right)\big\downarrow_{\tau} \ .$$

By taking the inverse of (2), we obtain

$$\left({}^{\sigma}\mathfrak{m}(s_i^{-1})\right)\big\downarrow_{\tau} = \left(\left({}^{\sigma}\mathfrak{m}(s_i)\right)\big\downarrow_{\tau}\right)^{-1} = M^{-1} \cdot g_i^{-1} \cdot M \ .$$

and so

$$\left({}^{\sigma}\mathfrak{m}(s_i^{\epsilon_i})\right)\big\downarrow_{\tau} = M^{-1} \cdot g_i^{\epsilon_i} \cdot M \tag{3}$$

By Lemma 1,

$$\mathfrak{m}(s) = \mathfrak{m}\Big(\prod_{j=1}^{L} s_{i_j}^{\epsilon_{i_j}}\Big) = \prod_{j=1}^{L} \mathfrak{m}(s_{i_j}^{\epsilon_{i_j}}) \ ,$$

and hence,

$$\begin{aligned}
\left({}^{\sigma}\mathfrak{m}(s)\right)\big\downarrow_{\tau} &= \left({}^{\sigma}\Big(\prod_{j=1}^{L} \mathfrak{m}(s_{i_j}^{\epsilon_{i_j}})\Big)\right)\Big\downarrow_{\tau} = \Big(\prod_{j=1}^{L} {}^{\sigma}\mathfrak{m}\big(s_{i_j}^{\epsilon_{i_j}}\big)\Big)\Big\downarrow_{\tau} \\
&= \prod_{j=1}^{L} \left({}^{\sigma}\mathfrak{m}(s_{i_j}^{\epsilon_{i_j}})\right)\big\downarrow_{\tau} = \prod_{j=1}^{L} \left(M^{-1} \cdot g_{i_j}^{\epsilon_{i_j}} \cdot M\right) \\
&= M^{-1} \cdot \Big(\prod_{j=1}^{L} g_{i_j}^{\epsilon_{i_j}}\Big) \cdot M = M^{-1} \cdot h \cdot M \ .
\end{aligned}$$

Therefore s is a valid signature for m, as the above equation is equivalent to

$$\mathfrak{m}(\mathrm{PK} \star s) = \mathfrak{m}\Big(\mathrm{Pub}\left(\mathcal{E}(m)\right)\Big) \cdot \mathfrak{m}(\mathrm{PK}) \ ,$$

the verification equation for (m, s). □

4.2 Solution to the Factorization Problem

Let $\Gamma = \{g_i \mid i \in I\}$. Following our discussion in Section 4.1, we want express h as a short word over Γ. We first define the following chain of subgroups:

Definition 2. *For $k \in \{1, \ldots, 2N-2\}$, let*

$$G_k = \left\{M \in \mathrm{GL}_N(\mathbb{F}_q) \;\middle|\; M_{N,N} = 1 \ \textit{and} \ M_{i,j} = 0 \ \textit{for} \ (i,j) \in A_1 \cup A_2\right\} \ ,$$

where

$$A_1 = \left\{(i,j) \mid \left(N - \left\lceil \frac{k}{2} \right\rceil\right) \leq i \leq N \text{ , } i \neq j\right\} \text{ ,}$$
$$A_2 = \left\{(i,j) \mid \left(N - \left\lfloor \frac{k}{2} \right\rfloor\right) \leq j \leq N \text{ , } i \neq j\right\} \text{ .}$$

That is, for even k,

$$G_k = \left\{ \left(\begin{array}{c|ccccc} A & 0 & \cdots & & 0 \\ \hline 0 & \lambda_{\frac{k}{2}-1} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & & \vdots \\ & \vdots & & \lambda_1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{array} \right) \middle| A \in \mathrm{Mat}_{N-\frac{k}{2}, N-\frac{k}{2}}(\mathbb{F}_q) \right\} \cap \mathrm{GL}_N(\mathbb{F}_q) \text{ ,}$$

and for odd k,

$$G_k = \left\{ \left(\begin{array}{c|ccccc} A & * & 0 & \cdots & 0 \\ \hline 0 & \lambda_{\frac{k-1}{2}} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & & \vdots \\ & \vdots & & \lambda_1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{array} \right) \middle| A \in \mathrm{Mat}_{N-\frac{k+1}{2}, N-\frac{k+1}{2}}(\mathbb{F}_q) \right\} \cap \mathrm{GL}_N(\mathbb{F}_q) \text{ ,}$$

where $*$ is a column of length $N - \frac{k+1}{2}$ and $\lambda_i \in \mathbb{F}_q^\times$ for $i \in \{1, \ldots, \lfloor \frac{k-1}{2} \rfloor\}$.

Remark 2. Checking whether $g \in \mathrm{GL}_N(\mathbb{F}_q)$ is in G_k for any k is straightforward given the characteristic shape of the matrices in each group.

Lemma 4. *For any braid $\beta \in B_N$, $\mathfrak{m}\big(\mathrm{Pub}(\beta)\big) \in G_1$.*

Proof. Let G_1' be the subgroup of $\mathrm{GL}_N\left(\mathbb{F}_q[t_1^{\pm 1}, \ldots, t_N^{\pm 1}]\right)$ consisting of matrices with their last row all zeroes except for the last entry, which is equal to 1. For each $i \in \{1, \ldots, N-1\}$, $\mathfrak{m}(b_i) \in G_1'$. Therefore, $\mathfrak{m}(\beta) \in G_1'$ and hence, $\mathfrak{m}\big(\mathrm{Pub}(\beta)\big) = \mathfrak{m}(\beta)\big\downarrow_\tau \in G_1$. □

We also make use of the following assumption:

Assumption 1. *For any k, a small set of random elements of G_k generates G_k with high probability.*

This assumption is supported by [42] and our experiments.

Our algorithm aims to solve an instance of a factorization problem over G_1. This is done in $2N-2$ stages. The first $2N-3$ stages are inductive: in stage k, we reduce the problem in G_k to an instance of the problem over the next subgroup G_{k+1}. At the end of stage $2N-3$, we have reduced the original problem to factorization problem over G_{2N-2}, the diagonal subgroup. In the last stage of the algorithm, we reduce the factorization problem in G_{2N-2} to an easy case of the discrete logarithm problem over $\mathbb{F}_q$ and a system of linear equations.

Let $\gamma_1 \coloneqq |\mathcal{M}|$, let $\Gamma_1 \coloneqq \Gamma = \left\{g_i^{(1)} : 1 \leq i \leq \gamma_1\right\}$, and let $h_1 \coloneqq h$. Further, for $2 \leq k \leq 2N-2$, let γ_k be a positive integer and $L_k \coloneqq \left\lceil \log_{\gamma_k} \frac{\gamma_{k+1}|G_k|}{|G_{k+1}|} \right\rceil$. We will aim to produce γ_{k+1} elements of G_{k+1} in stage k, and we hope that these elements will generate G_{k+1}, which we will need for us to reduce the factorization problem into the next subgroup. L_k captures some information about the number of elements we need to consider from G_k before we find γ_{k+1} elements of G_{k+1}: in our algorithm, the elements from G_k that we will consider will be words of some fixed length $\mathcal{L}_k$ over some generating set of size γ_k; by considering the relative sizes of G_k and G_{k+1}, it then follows that $\mathcal{L}_k$ should be L_k.

Inductive Stages. In stage k $\left(\text{for } k \in \{1, \ldots, 2N-3\}\right)$, we will find a set $\Gamma_{k+1} \coloneqq \left\{g_i^{(k+1)} : 1 \leq i \leq \gamma_{k+1}\right\} \subset G_{k+1}$ and an element $h_{k+1} \in G_{k+1}$, where $g_i^{(k+1)}$ are words over Γ_k and h_{k+1} is a product of h_k with a word over Γ_k.

Algorithm 2 From Γ_k and h_k, we find Γ_{k+1} and h_{k+1}

repeat
  Generate products of the form:

$$p = \prod_{j=1}^{L_k} g_{i_j}^{(k)} \text{ where } 1 \leq i_j \leq \gamma_k$$

  if $p \in G_{k+1}$ **then**
    Add p to Γ_{k+1}
  else if h_{k+1} has not yet been defined **then**
    if $p \in h_k G_{k+1}$ **then**
      Define $h_{k+1} = p^{-1} \cdot h_k$
    end if
  end if
until $|\Gamma_{k+1}| = \gamma_{k+1}$ and h_{k+1} is defined

Following Assumption 1, we expect that for large enough γ_k, Γ_k will be a generating set for G_k. We therefore expect to be able to find γ_{k+1} elements in $G_{k+1} \subset G_k$ given enough iterations of the loop. Moreover, $h_k G_{k+1} \subset G_k$, and so we expect to be able to find h_{k+1} as well.

Remark 3. We see from the above algorithm that for all $k \in \{1, \ldots, 2N-3\}$, we can write h_{k+1} as

$$h_{k+1} = \prod_j \left(g_{i_j}^{(k)}\right)^{-1} \cdot h_k \quad \text{for } 1 \leq i_j \leq \gamma_k \ .$$

Moreover, we can write any element in Γ_{k+1} as a product of elements in Γ_k. Hence, we can recursively write h_{k+1} as a product of a word over $\Gamma_1 = \Gamma$ with $h_1 = h$, i.e. we can express h_{k+1} as

$$h_{k+1} = \Big(\prod_j g_{i_j}^{\epsilon_{i_j}}\Big) \cdot h \quad \text{for } 1 \leq i_j \leq \gamma_1 \ . \tag{4}$$

Similarly, we can express each element $g_i^{(2N-2)} \in \Gamma_{2N-2}$ as a word over Γ

$$g_i^{(2N-2)} = \Big(\prod_j g_{i_j}^{\epsilon_{i_j}}\Big) \quad \text{for } 1 \leq i_j \leq \gamma_1 \ . \tag{5}$$

Final Stage. At the end of stage $2N-3$, we will have a set

$$\Gamma_{2N-2} = \left\{g_i^{(2N-2)} \ : \ 1 \leq i \leq \gamma_{2N-2}\right\} \subset G_{2N-2}$$

and an element $h_{2N-2} \in G_{2N-2}$. Note that G_{2N-2} is the subgroup of diagonal matrices, and so all of the above elements are diagonal matrices as well.

We want to express h_{2N-2} as a word over Γ_{2N-2}. Since G_{2N-2} is abelian, this is equivalent to finding exponents $v_1, \ldots, v_{\gamma_{2N-2}} \in \mathbb{Z}$ such that

$$h_{2N-2} = \prod_{i=1}^{\gamma_{2N-2}} \left(g_i^{(2N-2)}\right)^{v_i} \ . \tag{6}$$

(4) and (5) then allows us to rewrite the above equation as

$$h = \prod_j g_{i_j}^{\epsilon_{i_j}} \ ,$$

an expression for h as a word over Γ, given that we can find the exponents v_i. We describe how to find these exponents next.

Note that all the matrices on both sides of (6) are diagonal matrices. For each $i \in \{0, \ldots, \gamma_{2N-2}\}$, let $c_i = (\lambda_{i_1}, \ldots, \lambda_{i_{N-1}}, 1)$ be the sequence of diagonal entries in $g_i^{(2N-2)}$, and let $c \coloneqq (\mu_1, \ldots, \mu_{N-1}, 1)$ be the diagonal entries in h_{2N-2}. Further, let δ be a generator of $\mathbb{F}_q^\times$. By solving the discrete logarithm problem over $\mathbb{F}_q^\times$ (which is straighforward for small q), for each $i \in \{1, \ldots, \gamma_{2N-2}\}$, and each $j \in \{1, \ldots, N-1\}$, we can find e_{i_j} and u_j such that:

$$\delta^{e_{i_j}} = \lambda_{i_j} \ ,$$
$$\delta^{u_j} = \mu_j \ ,$$

i.e., we are able to write all non-zero entries of the matrices in (6) as powers of δ. Finding the exponents v_i is then reduced to solving a system of linear equations over $\mathbb{Z}_{q-1}$. More explicitly, for each $i \in \{1, \ldots, \gamma_{2N-2}\}$, define $c'_i = (e_{i_1}, \ldots, e_{i_{N-1}}, 1)$. Also, let $c' = (u_1, \ldots, u_{N-1}, 1)$ and let $D = (c'_1, \ldots, c'_{\gamma_{2N-2}})$, i.e., the matrix with i^{th} column equal to c'_i. So (6) above is equivalent to the system of linear equations

$$D \cdot v = c' \tag{7}$$

which can be solved with standard linear algebra techniques.

4.3 Meet-in-the-Middle Approach

We can improve the recursive step of our attack as follows. Instead of computing products of length L_k until we hit an element of G_{k+1}, we compute pairs of products each of length $\left\lfloor \frac{L_k}{2} \right\rfloor$ and then check for pairs which lie in the same coset of G_{k+1}. This meet-in-the-middle approach will lead to a square root improvement of the complexity. In order to use this approach, we need an efficient method to check whether two elements are in the same coset of G_{k+1}. The following lemma provides such a method.

Lemma 5. *Let G_k for $k \in \{1, \ldots, 2N-2\}$ be the subgroups in Definition 2, and let p, $p' \in G_k$. Then*

- *For odd k, $p' \in pG_{k+1}$ if and only if the $(N - \frac{k+1}{2} + 1)^{th}$ columns of p and p' are multiples of each other.*
- *For even k, $p' \in G_{k+1}p'$ if and only if the $(N - \frac{k}{2})^{th}$ rows of p and p' are multiples of each other.*

Proof. Let k be odd, let $g \in G_{k+1}$, and let $r = N - \frac{k+1}{2} + 1$. Note that the r^{th} column of g is zero except for the entry $\lambda_r \coloneqq g_{r,r} \in \mathbb{F}_q^\times$. Finally, let $p, p' \in G_k$.

Assume that $p' \in pG_{k+1}$, and so $\exists g : p' = pg$. Let $p_{i,j}$ be the $(i,j)^{th}$ entry of p. Then the entries of the r^{th} column of p' are:

$$p'_{i,r} = \sum_{j=1}^{N} p_{i,j} g_{j,r} = p_{i,r} \cdot \lambda_r \quad \text{for } 1 \le i \le N$$

and hence r^{th} columns of p and p' are multiples of each other.

Conversely, let c_r be the r^{th} column of p and c'_r be the r^{th} column of p', and assume $c'_r = \lambda \cdot c_r$ for some $\lambda \in \mathbb{F}_q^\times$. Let $\pi = p^{-1} \cdot p'$. Then the entries of the r^{th}

column of π are

$$\begin{aligned}\pi_{i,r} &= \sum_{j=1}^{N}(p^{-1})_{i,j}\cdot p'_{j,r} = \sum_{j=1}^{N}(p^{-1})_{i,j}\cdot \lambda p_{j,r}\\ &= \lambda\sum_{j=1}^{N}(p^{-1})_{i,j}\cdot p_{j,r} = \lambda\cdot(\mathrm{Id}_N)_{i,r}\\ &= \lambda\cdot\delta_{ir}\end{aligned}$$

But this implies that the r^{th} column of π is zero everywhere except at the $(r,r)^{th}$ entry. Since $\pi\in G_k$, this implies $\pi\in G_{k+1}$ and hence $p'\in pG_{k+1}$.

The case for even k is similar. □

Using the above lemma, we are able to construct an improved version of Algorithm 2:

Algorithm 3 From Γ_k and h_k, we find Γ_{k+1} and h_{k+1}

Generate all products of the form:

$$p_1 = \prod_{j=1}^{L_k/2} g_{i_j}^{(k)} \text{ where } 1\le i_j\le\gamma_k$$

repeat
  Pick a product p_2 of the form above
  if k is odd **then**
    if for some p_1, we have $p_2\in p_1^{-1}G_{k+1}$ **then**
      Add $p=p_1p_2$ to Γ_{k+1}
    else if h_{k+1} has not been defined yet **then**
      if for some p_1, we have $p_2\in p_1^{-1}h_kG_{k+1}$ **then**
        Define $h_{k+1}=p_2^{-1}p_1^{-1}\cdot h_k = p^{-1}\cdot h_k$
      end if
    end if
  else if k is even **then**
    if for some p_1, we have $p_1\in G_{k+1}p_2^{-1}$ **then**
      Add $p=p_1p_2$ to Γ_{k+1}
    else if h_{k+1} has not been defined yet **then**
      if for some p_1, we have $h_k^{-1}p_1\in G_{k+1}p_2^{-1}$ **then**
        Define $h_{k+1}=p_2^{-1}p_1^{-1}\cdot h_k = p^{-1}\cdot h_k$
      end if
    end if
  end if
until $|\Gamma_{k+1}|=\gamma_{k+1}$ and h_{k+1} is defined

4.4 Complexity Analysis and Experiments

Time Complexity. We first analyse the time complexity of the attack.

We observe that the complexity of the algorithm is dominated by the complexity of finding each Γ_{k+1}: the last step involves solving a discrete logarithm problem over a small field and a small linear system modulo $q-1$. Moreover, finding an element h_{k+1} is essentially the same as finding one element of Γ_{k+1}.

Lemma 6. *The size of G_k is as follows:*

- *For k even,* $|G_k| = (q-1)^{(\frac{k}{2}-1)} \cdot |\operatorname{GL}_{N-\frac{k}{2}}(\mathbb{F}_q)|$.
- *For k odd,* $|G_k| = (q-1)^{\lfloor \frac{k}{2} \rfloor} \cdot q^{N-\lfloor \frac{k}{2} \rfloor - 1} \cdot |\operatorname{GL}_{N-\lfloor \frac{k}{2} \rfloor - 1}(\mathbb{F}_q)|$.

Proof. For k even, the block diagonal structure of G_k consists of an invertible matrix of size $N - \frac{k}{2}$ and $\frac{k}{2}$ entries on the diagonal. The bottommost such entry is 1, and the other diagonal entries can be any of the nonzero elements in $\mathbb{F}_q$, and so we obtain the formula above. For k odd, the block diagonal structure of G_k consists of an invertible matrix of size $N - \lfloor \frac{k}{2} \rfloor$ with a zero bottom row except for the last entry, and $\lfloor \frac{k}{2} \rfloor$ other entries on the diagonal. Note that $\lfloor \frac{k}{2} \rfloor - 1$ of the diagonal entries can be any nonzero element in $\mathbb{F}_q$ while the bottommost entry is 1. The invertible matrix of size $N - \lfloor \frac{k}{2} \rfloor$ consists of any element in $\operatorname{GL}_{N-\lfloor \frac{k}{2} \rfloor - 1}(\mathbb{F}_q)$ on the upper diagonal, any nonzero entry from $\mathbb{F}_q$ for the bottom right entry, and a value in $\mathbb{F}_q$ for the rest of the entries in the last column. From this we obtain the formula above. □

Lemma 7. $\frac{|G_k|}{|G_{k+1}|} \approx q^{N-1-\lfloor \frac{k}{2} \rfloor}$

Proof. This follows immediately from the previous lemma. □

If we pick a random element of G_k, the probability that it will also be in G_{k+1} is therefore approximately $1/q^{N-1-\lfloor \frac{k}{2} \rfloor}$. In our algorithm, we make the assumption that random products of elements in Γ_k produces random elements in G_{k+1}, and so we expect that we will be able to obtain one element of Γ_{k+1} after considering $q^{N-1-\lfloor \frac{k}{2} \rfloor}$ random products. By using the meet-in-the-middle approach described earlier, we reduce the expected number of products we need to consider by $q^{(N-1-\lfloor \frac{k}{2} \rfloor)/2}$. Since we need to generate $|\Gamma_{k+1} \cup \{h_{k+1}\}| \approx \gamma_{k+1}$ new elements, the expected number of products we need to consider is bounded by $\gamma_{k+1} \cdot q^{(N-1-\lfloor \frac{k}{2} \rfloor)/2}$. The total number of products our algorithm needs to consider is therefore

$$\sum_{k=1}^{2N-3} \gamma_{k+1} \cdot q^{(N-1-\lfloor \frac{k}{2} \rfloor)/2} \ .$$

If we further assume that $\gamma_k = \gamma$ is constant, the above simplifies to

$$\gamma \cdot \sum_{k=1}^{2N-3} q^{(N-1-\lfloor \frac{k}{2} \rfloor)/2} = 2 \cdot \gamma \cdot \sum_{l=0}^{N-2} q^{\frac{N-1-l}{2}} \approx 2 \cdot \gamma \cdot q^{\frac{N-1}{2}} \ .$$

Thus, the complexity of the attack is exponential in N.

Memory Complexity. The final stage of the algorithm requires a negligible amount of memory. For the inductive stages, in stage k of the algorithm, we need to store up to $q^{\frac{1}{2}\left(N-1-\lfloor \frac{k}{2} \rfloor\right)}$ square matrices of size $N \times N$, each entry being in $\mathbb{F}_q$, so we will need $\log_2(q) \cdot N^2 q^{\frac{1}{2}\left(N-1-\lfloor \frac{k}{2} \rfloor\right)}$ bits of memory for each stage. However, we do not need to keep the matrices from stage k when proceeding to stage $k+1$ (except to store the relatively small number of matrices of Γ_{k+1} and h_{k+1}), and so the total amount of memory required for the entire algorithm is the maximum amount of memory required by each stage, which is $\log_2(q) \cdot N^2 q^{\frac{N-1}{2}}$.

Length Complexity. We now analyze the length of the forged signature that we obtain.

Note that the length of any element in Γ_{k+1}, as a word over elements of Γ_k, is given by L_k. Also, our algorithm expresses h_{k+1} as the product of h_k with L_K elements of Γ_k. Unfolding this recurrence, we see that h_{2_N-2} is the product of h with α elements of Γ, where

$$\begin{aligned}
\alpha = \sum_{k=1}^{2N-3} \prod_{j=1}^{k} L_j &= \sum_{k=1}^{2N-3} \prod_{j=1}^{k} \log_{\gamma_j} \left(\frac{|G_j|}{|G_{j+1}|} \gamma_{j+1} \right) \\
&= \sum_{k=1}^{2N-3} \prod_{j=1}^{k} \log_{\gamma_j} \left(q^{N-1-\lfloor \frac{j}{2} \rfloor} \cdot \gamma_{j+1} \right) \\
&\approx \prod_{j=1}^{2N-3} \log_{\gamma_j} \left(q^{N-1-\lfloor \frac{j}{2} \rfloor} \cdot \gamma_{j+1} \right) \ ,
\end{aligned}$$

since the last summand dominates the sum. Similarly, we see that each $g_i^{(2N-2)}$ is a product of $\approx \alpha$ elements of Γ.

If we further assume that $\gamma_k = \gamma$ is constant, the above formula simplifies to

$$\begin{aligned}
\alpha &\approx \prod_{j=1}^{2N-3} \left(1 + \left(N - 1 - \left\lfloor \frac{j}{2} \right\rfloor \right) \log_\gamma q \right) \\
&= \left(\prod_{k=1}^{N-2} 1 + k \log_\gamma q \right) \left(\prod_{k=1}^{N-1} 1 + k \log_\gamma q \right) \ .
\end{aligned}$$

In the final step of the algorithm, we find a relation (6)

$$h_{2N-2} = \prod_{i=1}^{\gamma_{2N-2}} \left(g_i^{(2N-2)}\right)^{v_i} \quad .$$

Since the v_i come from the solution to a system of linear equations over $\mathbb{Z}_{q-1}$, we know that $v_i < q-1$. Also, since the space we are working over in our system of linear equations (7) has dimension $N-1$, it follows then that we need at most $N-1$ terms in the product above. Putting this all together, we see that h is then the product of $\big(1+(n-1)(q-1)\big)\alpha \approx nq\alpha$ elements of Γ, and so our forged signature is of length $\approx lnq\alpha$, where l is the length of the WalnutDSA signatures in $\mathcal{M}$.

Experimental Results We have implemented our factorization algorithm in Magma [43] and tested it experimentally (the code is provided as a supplementary material of this submission). The only parameters of our algorithm are the values of L_k, which we can control via γ_k. Note that increasing γ_k decreases the length of our forged signature but increases the running time of our algorithm. In our experiments, we first assumed that we are able to obtain ten legitimate message-signature pairs. We then chose γ_k such that L_k is large enough for us to find the relations for all h_k. This allowed us to obtain a signature of length 2^{35} in approximately two minutes. To reduce the length of the forged signature, we increased γ_k such that $\gamma_k \approx 200000$ for $k > 3$. This allowed us to obtain signatures of length 2^{25} in five minutes.

4.5 Practical Improvements

In this section we present two improvements on our attack.

Shorter Subgroup Chain. The subgroup chain we used above was chosen to have small subgroup indices $[G_k : G_{k+1}]$ in order to minimize computation time at each step. However, the first few stages of the algorithm contribute to the majority of the running time, whereas all stages contribute significantly to the total length of the signatures we produce.

To reduce signature lengths without affecting the computation time significantly, one can replace the above subgroup chain by another chain. An example of such a chain could have the same first five subgroups (at a cost roughly $q^{3.5}$, q^3, q^3, $q^{2.5}$ and $q^{2.5}$ respectively), but then instead of considering a subgroup where the lower diagonal entries in the last four rows are zeroes (at a cost q^2), consider a subgroup where the lower diagonal entries in the last five rows are zeroes (at a cost $q^{3.5}$), then a subgroup where the upper diagonal entries in the last five rows

are also zeroes (at a cost $q^{3.5}$), and finally considering the diagonal subgroup (at a cost q^3). In that case, the factorization length can be approximated by

$$nq \prod_k c_k \log_{\gamma_k} q$$

where $(c_1, c_2, \ldots, c_8) = (7, 6, 6, 5, 5, 7, 7, 6)$, which for $\gamma_k = 256$ gives a signature size approximately 2^{11} times that of a normal signature size, while retaining the time complexity of roughly $q^{3.5}$.

Dealing with Non-Generating Sets. We have not been able to prove that the elements we construct in our recursive step are indeed generators for the next subgroup. We expect this is the case with a high probability on the initial matrix choices when choosing product lengths as above, and in our experimental tests this was verified for all recursive steps.

The diagonal matrices generated for the last stage, however, may not generate the whole diagonal group when the number of generators constructed at each step is very small. We observed this experimentally when using $\gamma_k = 2$ in all but the last inductive stage, and can explain it intuitively as follows. Let $\Gamma_k \coloneqq \{A^{(k)}, B^{(k)}\}$. At each stage, the diagonal entries in the diagonal part (in block diagonal form) of $A^{(k)}$ and $B^{(k)}$ can be approximated as random elements in $\mathbb{F}_q^\times$. Consider any pair of indices $\big((i_1, i_1), (i_2, i_2)\big)$ in the diagonal part of the matrix, and consider the 2-dimensional vectors $\big(A^{(k)}_{i_1,i_1}, A^{(k)}_{i_2,i_2}\big)$ and $\big(B^{(k)}_{i_1,i_1}, B^{(k)}_{i_2,i_2}\big)$. It is a necessary condition for these two matrices to generate the whole subgroup, that there is no linear dependence between the two vectors obtained by taking entrywise logarithms of the above vectors. For a fixed pair of indices (i_1, i_1) and (i_2, i_2), this happens with probability $\frac{q-2}{q-1}$. In the later inductive stages, the diagonal part of the matrices are larger, and hence the probability that all pairs of the logarithm vectors are linearly independent decreases. Moreover, any linear dependence occurring in one stage will be preserved in subsequent stages. It is therefore intuitively plausible that Γ_{2N-2} may not generate G_{2N-2} when γ_k is very small.

In our experiments, it was easy to choose γ_k large enough such that all stages would produce a sufficient number of generators for the following subgroup, including that of diagonal subgroup G_{2N-2}. We note also that in the event that Γ_{2N-2} does not generate G_{2N-2}, one can simply set $h_1 = \mathrm{Id}_n$ and relaunch the whole factorization algorithm: this will produce a new set of diagonal matrices Γ'_{2N-2} that, together with Γ_{2N-2}, is likely to generate the G_{2N-2}. This therefore allows our attack to succeed with high probability even when we only have access to two WalnutDSA message-signature pairs.

5 Discussion and Further Work

Due to its algebraic structure, WalnutDSA is inherently vulnerable to malleability attacks. The use of a cryptographic hash function in the message encoding process is intended to remove this inherent malleability, in the same way as Full Domain Hash removes the inherent malleability in the RSA signature algorithm. Our attack, however, goes around this protection mechanism by reducing the cryptanalysis of WalnutDSA to an instance of a factorization problem in the group $\mathrm{GL}_N(\mathbb{F}_q)$.

We briefly discuss two countermeasures against this attack, namely increasing the parameter sizes and checking the signature lengths.

5.1 Increasing the Parameters

In order to defeat our attack, one can choose to increase the parameters of WalnutDSA such that the complexity of our attack is increased to $\sim 2^{100}$. As shown in Section 4.4, the complexity of our attack can be estimated by $\gamma \cdot q^{\frac{N-1}{2}} \approx q^{\frac{N-1}{2}}$. One can therefore choose to increase the value of q and N such that $q^{\frac{N-1}{2}} \approx 2^{100}$, by choosing $q = 2^{16}$ and $N = 14$ for example.

5.2 Checking Signature Length

Recall that our forged signature s is obtained from concatenating existing signatures. The length of s depends primarily on the length of the products L_k considered in algorithm 3. As discussed in Sections 4.4 and 4.5, larger values for $\gamma_k = |\Gamma_k|$ and a different choice of subgroup chain can achieve shorter forged signature lengths at the cost of higher time and memory complexity. Our best attempt produced a forged signature 2^{25} times larger than the original WalnutDSA signatures.

Observe that the length of a legitimate signature (one produced according to WalnutDSA) depends on the length of $\textsc{sk}$, $\mathcal{E}(m)$, and the cloaking elements. Even though these lengths are not fixed, we expect them to be within certain bounds, which will depend on the implementation of the protocol. However, in principle, the length of s should greatly exceed these bounds. Therefore, we suggest that the length of both cloaking elements and private keys be bounded above, so that the length of a WalnutDSA signature is always less than some constant $\mathcal{L}$. Then, any signature of length greater than $\mathcal{L}$ should then be rejected.

5.3 Limitations of the Countermeasures

We do not know, however, whether s could be shortened to fit the new imposed bounds. Methods such as Dehornoy's handle reduction [14] could potentially

reduce the length of our forged signatures sufficiently in a non-negligible fraction of instances.

We stress that more efficient algorithms for solving the factorization problem in $\mathrm{GL}_N(\mathbb{F}_q)$ may also exist. One may expect factorizations as small as $\log_{|M|} |\mathrm{GL}_N(\mathbb{F}_q)| = \log_{|M|} q^{n^2-n-1}$ to exist, where M is the set of WalnutDSA message-signature pairs one has access to. If an efficient algorithm to compute short factorizations exists, the increase in parameters q and N needed to achieve a sufficient level of security would then make WalnutDSA unsuitable for embedded devices. Moreover, with $|\mathcal{M}|$ large enough, the forged signatures will only be a small constant factor bigger than legitimate signatures, and hence determining a suitable bound $\mathcal{L}$ to apply our second countermeasure may be challenging.

Finally, we observe that our work has not considered the hard problems underlying the WalnutDSA protocol, that of reversing E-Multiplication and the cloaked conjugacy search problem. The study of these problems, along with the effectiveness of the above countermeasures, will be of interest for further work.

6 Conclusion

In this paper we provided a practical cryptanalysis of WalnutDSA. Given a couple of random valid message-signature pairs, our attack is able to produce new signatures on arbitrary messages in approximately two minutes. We also discuss countermeasures to our attack, including a simple modification of the verification algorithm.

References

1. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on, Ieee (1994) 124–134
2. McEliece, R.J.: A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report **44** (January 1978) 114–116
3. Micciancio, D., Regev, O.: Lattice-based Cryptography. In: Post-Quantum Cryptography. Springer (2009) 147–191
4. Ding, J., Yang, B.Y.: Multivariate Public Key Cryptography. In: Post-Quantum Cryptography. Springer (2009) 193–241
5. Garber, D.: Braid Group Cryptography. CoRR **abs/0711.3941** (2007)
6. : SecureRF. `https://www.securerf.com/`.
7. Anshel, I., Atkins, D., Goldfeld, D., Gunnells, P.E.: WalnutDSA$^{\mathrm{TM}}$: A Quantum-Resistant Digital Signature Algorithm. Cryptology ePrint Archive, Report 2017/058 (2017) `http://eprint.iacr.org/2017/058`.
8. : SECURERF AND INTEL COLLABORATION DELIVERS FUTURE-PROOF FPGA SECURITY SOLUTIONS. `https://www.iot-now.com/2017/09/28/67603-securerf-intel-collaboration-delivers-future-proof-fpga-security-solutions/`.

9. Ben-Zvi, A., Blackburn, S.R., Tsaban, B.: A practical cryptanalysis of the algebraic eraser. Cryptology ePrint Archive, Report 2015/1102 (2015) `http://eprint.iacr.org/2015/1102`.
10. Myasnikov, A.D., Ushakov, A.: Cryptanalysis of anshel-anshel-goldfeld-lemieux key agreement protocol. Groups Complexity Cryptology **1**(1) (2009) 63–75
11. Kalka, A., Teicher, M., Tsaban, B.: Short expressions of permutations as products and cryptanalysis of the algebraic eraser. Advances in Applied Mathematics **49**(1) (2012) 57 – 76
12. GARSIDE, F.A.: The braid group and other groups. The Quarterly Journal of Mathematics **20**(1) (1969) 235–254
13. Birman, J., Gebhardt, V., González-Meneses, J.: Conjugacy in Garside groups I: cyclings, powers and rigidity. Groups, Geometry, and Dynamics (2007) 221–279
14. Dehornoy, P.: A Fast Method for Comparing Braids. Advances in Mathematics **125**(2) (February 1997) 200–235
15. Tillich, J.P., Zémor, G.: Hashing with SL_2. In: CRYPTO. (1994) 40–49
16. Petit, C., Lauter, K., Quisquater, J.J.: Full cryptanalysis of LPS and Morgenstern hash functions. In: SCN. (2008) 263–277
17. Petit, C., Quisquater, J.J.: Preimages for the Tillich-Zémor hash function. In: Selected Areas in Cryptography. (2010) 282–301
18. Babai, L., Hayes, T.: The probability of generating the symmetric group when one of the generators is random. Publ Math Debrecen **69/3** (2006) 271–280
19. Petit, C., Quisquater, J.J.: Rubik's for cryptographers. Notices of the American Mathematical Society **60** (2013) 733–739
20. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. second edn. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, Taylor & Francis Group (2014)
21. Epstein, D.B.A., Paterson, M.S., Cannon, J.W., Holt, D.F., Levy, S.V., Thurston, W.P.: Word Processing in Groups. A. K. Peters, Ltd., Natick, MA, USA (1992)
22. Elrifai, E.A., Morton, H.: Algorithms for Positive Braids. The Quarterly Journal of Mathematics **45**(4) (1994) 479–497
23. Gebhardt, V.: A new approach to the conjugacy problem in Garside groups. Journal of Algebra **292**(1) (2005) 282 – 302 Computational Algebra.
24. Hughes, J., Tannenbaum, A.: Length-Based Attacks for Certain Group Based Encryption Rewriting Systems. CoRR **cs.CR/0306032** (2003)
25. Garber, D., Kaplan, S., Teicher, M., Tsaban, B., Vishne, U.: Probabilistic solutions of equations in the braid group. Advances in Applied Mathematics **35**(3) (2005) 323–334
26. Myasnikov, A.D., Ushakov, A.: Length Based Attack and Braid Groups: Cryptanalysis of Anshel-Anshel-Goldfeld Key Exchange Protocol. In Okamoto, T., Wang, X., eds.: Public Key Cryptography – PKC 2007. Volume 4450 of Lecture Notes in Computer Science., PKC 2007, Springer, Berlin, Heidelberg (2007)
27. Hughes, J.: A Linear Algebraic Attack on the AAFG1 Braid Group Cryptosystem. In: Information Security and Privacy. Volume 2384 of Lecture Notes in Computer Science., ACISP 2002, Springer, Berlin, Heidelberg (2002)
28. Lee, S.J., Lee, E.: Potential Weaknesses of the Commutator Key Agreement Protocol Based on Braid Groups. In: Advances in Cryptology — EUROCRYPT 2002. Volume 2332 of Lecture Notes in Computer Science., EUROCRYPT 2002, Springer, Berlin, Heidelberg (2002)
29. Cheon, J.H., Jun, B.: A Polynomial Time Algorithm for the Braid Diffie-Hellman Conjugacy Problem. In: Advances in Cryptology - CRYPTO 2003. Volume 2729 of

Lecture Notes in Computer Science., CRYPTO 2003, Springer, Berlin, Heidelberg (2003)
30. Helfgott, H.A.: Growth and generation in $SL_2(Z/pZ)$. Ann. of Math. (2) **167 (2)** (2008) 601–623
31. Pyber, L., Szabó, E.: Growth in finite simple groups of Lie type. 29 **29**(1) (2016) 95–146
32. Zémor, G.: Hash functions and Cayley graphs. Des. Codes Cryptog. **4**(4) (1994) 381–394
33. Tillich, J.P., Zémor, G.: Group-theoretic hash functions. In: Proceedings of the First French-Israeli Workshop on Algebraic Coding, London, UK, Springer-Verlag (1993) 90–110
34. Grassl, M., Ilic, I., Magliveras, S.S., Steinwandt, R.: Cryptanalysis of the Tillich-Zémor hash function. J. Cryptology **24**(1) (2011) 148–156
35. Charles, D.X., Lauter, K.E., Goren, E.Z.: Cryptographic hash functions from expander graphs. J. Cryptology **22**(1) (2009) 93–113
36. Tillich, J.P., Zémor, G.: Collisions for the LPS expander graph hash function. In: EUROCRYPT. (2008) 254–269
37. Bellare, M., Micciancio, D.: A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In: EUROCRYPT. (1997) 163–192
38. Babai, L., Ákos Seress: On the diameter of permutation groups. European Journal of Combinatorics **13**(4) (1992) 231 – 243
39. Anshel, I., Anshel, M., Goldfeld, D., Lemieux, S.: Key Agreement, the Algebraic Eraser$^{\mathrm{TM}}$, and Lightweight Cryptography. In: Algebraic Methods in Cryptography. Volume 418 of Contemporary Mathematics. American Math. Soc., Providence, RI (2006) 1–34
40. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols, ACM Press (1993) 62–73
41. Birman, J.S.: Braids, Links, and Mapping Class Groups. Volume 82 of Annals of Mathematics Studies. Princeton University Press (1975)
42. Waterhouse, W.C.: Two generators for the general linear groups over finite fields. Linear and Multilinear Algebra **24**(4) (1989) 227–230
43. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. **24**(3-4) (1997) 235–265 Computational algebra and number theory (London, 1993).

A The Garside Normal Form

We follow the presentation in [5]. Define a *positive braid*, which is an element of B_N that can be written as a product of positive powers of the generators. Let B_N^+ denote the set of positive braids. One example of a positive braid is the *fundamental braid* $\Delta_N \in B_N$:

$$\Delta_N = (b_1 \cdots b_{N-1})(b_1 \cdots b_{N-2}) \cdots (b_1 b_2)(b_1) \ .$$

Geometrically, Δ_N is the braid in which any two strands cross positively *exactly* once.

We now define a partial order on B_N: for $A, B \in B_N$, write $A \preceq B$ if there exists $C \in B_N^+$ such that $B = AC$. With this definition, we say that $P \in B_N$ is

a *permutation braid* if $\varepsilon \preceq P \preceq \Delta_N$, where ε is the empty braid. Geometrically, a permutation braid is a braid in which any two strands cross positively *at most* once.

Let P be a permutation braid. Then the *starting set* of P is

$$S(P) = \{i \mid P = b_i P' \text{ for some } P' \in B_N^+\}$$

and the *finishing set* of P is

$$F(P) = \{j \mid P = P' b_j \text{ for some } P' \in B_N^+\} \ .$$

Furthermore, if A is any positive braid, its *left-weighted decomposition* into permutation braids is

$$A = P_1 \cdots P_m$$

where $S(P_{i+1}) \subset F(P_i)$ for any i.