AN ATTACK ON THE WALNUT DIGITAL SIGNATURE ALGORITHM

MATVEI KOTOV, ANTON MENSHOV, AND ALEXANDER USHAKOV

Abstract. In this paper, we analyze security properties of the WalnutDSA, a digital signature algorithm recently proposed by I. Anshel, D. Atkins, D. Goldfeld, and P. Gunnels, that has been accepted by the National Institute of Standards and Technology for evaluation as a standard for quantum-resistant public-key cryptography. At the core of the algorithm is an action, named *E-multiplication*, of a braid group on some finite set. The protocol assigns a pair of braids to the signer as a private key. A signature of a message m is a specially constructed braid that is obtained as a product of private keys, the hash value of m encoded as a braid, and three specially designed cloaking elements.

We present a heuristic algorithm that allows a passive eavesdropper to recover a substitute for the signer's private key by removing cloaking elements and then solving a system of conjugacy equations in braids. Our attack has 100% success rate on randomly generated instances of the protocol. It works with braids only and its success rate is not affected by a choice of the base finite field. In particular, it has the same 100% success rate for recently suggested parameters values (including a new way to generate cloaking elements, see [1]). Implementation of our attack in C++, as well as our implementation of the WalnutDSA protocol, is available on GitHub [7].

Keywords. WalnutDSA, group-based cryptography, digital signature, algebraic eraser, braid group, colored Burau presentation, conjugacy problem.

2010 Mathematics Subject Classification. 94A60, 68W30.

1. Introduction

WalnutDSA is a digital signature algorithm recently proposed by I. Anshel, D. Atkins, D. Goldfeld, and P. Gunnels in [4]. It has been accepted by the National Institute of Standards and Technology for evaluation as a standard for quantum-resistant public-key cryptography. At the core of WalnutDSA is a highly specialized one-way function called E-multiplication, claimed to be quantum resistant and highly resistant to reverse engineering. The scheme is also claimed to be suitable for low-resource devices.

Date: April 30, 2018.

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

1.1. **Related work.** The protocol received significant attention; and very recently two algebraic attacks were proposed ([10] and [5]). Recall that the private key in WalnutDSA consists of a pair of braids (w, w'), see Section 5 for details. The first paper, [10], describes a factorization attack that breaks WalnutDSA under the assumption that $w = w'$. One of the drawbacks of that attack, besides the assumption $w = w'$, is that the forged signatures are much longer than the original. Imposing a limit on the length of signatures allows to defend against this attack.

The second paper, [5], describes three attacks on WalnutDSA:

- The authors show that the factorization attack proposed in [10] can be modified to work for general private keys (w, w'). Unfortunately, the modified attack has the same problem as the original one; it produces very long signatures.
- The authors describe a collision search attack. They make an observation that the sizes of orbits of E-multiplication are relatively small for the proposed parameter values and exploit that to create messages with the same signature.
- One problem underlying security of WalnutDSA is the so-called *reversing E-multiplication problem*, or REM-problem. [5] shows that REM-problem can be solved efficiently for the proposed parameter values. This attack directly constructs equivalent secret keys in less than a minute for the parameters suggested by the authors of WalnutDSA.

See Section 7 for more details, where we give a short overlook of these papers. Very recently, to counter [5], the authors of WalnutDSA suggested to increase certain parameter values and slightly changed the design of cloaking elements.

1.2. **Our contribution.** We describe a different method to break the protocol that ignores E-multiplication and operates purely with braids. In particular, its success does not depend on the size of the base field. It works for the original parameter values and for the new ones. It also successfully works with very recently proposed cloaking elements (see Proposition 2.3).

1.3. **Outline.** In Section 2 we give definitions that are necessary to define WalnutDSA: colored Burau representation, E-multiplication, and cloaking elements. Fast checks for identity and conjugacy problems in braids are presented in Section 3. In Section 4 we describe an encoding procedure used in WalnutDSA to convert a message into a braid word. In Section 5 WalnutDSA protocol is introduced. In Section 6 we define the parameters we used to test our attack. Section 7 is a short survey of the previous known attacks. In Section 8 we describe our attack. Section 9 contains the obtained results and further remarks on the performance of our algorithm. We conclude the paper in Section 10.

2. E-multiplication

Here we shortly review a non-faithful representation of a braid group B_n called the *colored Burau group* and an action of B_n on a certain finite set (called E-multiplication).

2.1. **Braid group.** The group B_n of braids on n strands has the following combinatorial presentation:

$$B_n \simeq \left\langle x_1, \dots, x_{n-1} \;\middle|\; \begin{array}{l} x_i x_j = x_j x_i \text{ for } |i-j| > 1, \\ x_i x_{i+1} x_i = x_{i+1} x_i x_{i+1} \end{array} \right\rangle.$$

A word $w = w(x_1, \dots, x_{n-1})$ in the generators of B_n and their inverses is called a *braid word*. Every braid word w defines an actual (physical) braid. Length $|w|$ of a braid word w is the number of letters in w. If $\overline{u} = (u_1, \dots, u_k)$ is a k-tuple of braid words, then the total length $|\overline{u}|$ of $\overline{u}$ is defined as $\sum_{i=1}^{k} |u_i|$.

Every braid w naturally defines a permutation σ_w, which is actually a permutation of the endpoints of the involved strands, and the corresponding map $w \mapsto \sigma_w$ is an epimorphism. If σ_w is trivial, then w is called a *pure* braid. The group of pure braids on n strands is denoted by P_n.

For a set of braids $u_1, \dots, u_k$ define a set

$$C(u_1, \dots, u_k) = \{c \in B_n \mid [c, u_i] = 1 \text{ for } 1 \le i \le k\},$$

called the *centralizer* of $u_1, \dots, u_k$. It is easy to check that a centralizer is a subgroup of B_n.

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

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

2.2. **Geodesic braid approximation.** Let w be a word in generators of B_n. The problem to find a shortest braid word representing the same element as w is known to be computationally hard (see [15]). It is known however (see e.g. [16, 11]) that many NP-complete problems have polynomial time generic- or average-case solutions or have good approximate solutions. In this paper, we use a geodesic-braid approximation method that was introduced in [13, 14] and proved to be very useful in many other attacks.

The following algorithm attempts to minimize the given braid word. It exploits the property of Dehornoy's form $D(w)$ (introduced in [8]) that for a "generic" braid word w one has $|D(w)| < |w|$.

Algorithm 1 (Braid Minimization)**.**
Input. *A word $w = w(x_1, \dots, x_{n-1})$ in the generators of the group B_n.*
Output. *A word w' such that $|w'| \le |w|$ and $w' = w$ in B_n.*
Initialization. *Put $w_0 = w$ and $i = 0$.*
Computations.

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

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

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

equipped with the operation

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

Define $n \times n$-matrices $C_1(t_1), \dots, C_{n-1}(t_{n-1})$ over polynomials in variables $\{t_1, \dots, t_n\}$:

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

for $2 \le i \le n-1$. It is easy to check that the map φ on the generators $x_1, \dots, x_{n-1}$ of B_n:

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

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

$$\langle (C_1(t_1), \pi_1), \dots, (C_{n-1}(t_{n-1}), \pi_{n-1}) \rangle$$

is called the *colored Burau representation* of B_n and is denoted by CB_n.

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

$$\epsilon \colon \mathbf{GL}_n(R_n) \to \mathbf{GL}_n(\mathbb{F}_q)$$

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

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

It is straightforward to check that the map $\star$ defines an action of CB_n on $\mathbf{GL}_n(\mathbb{F}_q) \times S_n$. By E-multiplication we understand the induced action of B_n on $\mathbf{GL}_n(\mathbb{F}_q) \times S_n$. For a braid w define a pair:

$$\mathcal{P}(w) = (I, \mathrm{id}) \star w. \tag{1}$$

For a pair (m, σ) define $\mathrm{Matrix}((m, \sigma)) = m$.

2.5. **Cloaking elements.** Let G be a group acting on a set X and $x \in X$. The *stabilizer* of x is the set

$$\mathbf{Stab}(x) = \{g \in G \mid x^g = x\}.$$

It is easy to check that $\mathbf{Stab}(x)$ is a subgroup of G. In general, it is a difficult problem to describe $\mathbf{Stab}(x)$ for a given x. The protocol [4] requires braids stabilizing some $(m, \sigma) \in \mathbf{GL}_n(\mathbb{F}_q) \times S_n$ through the right action of the braid group via E-multiplication. Such braids are called *cloaking elements* in [4, Definition 2.1]. Observe that these elements depend on t-values that are used to define E-multiplication.

The following way of constructing cloaking elements was proposed in [4]. Fix $(m, \sigma) \in \mathbf{GL}_n(\mathbb{F}_q) \times S_n$ and assume that $a, b, i \in \mathbb{N}$ and $w \in B_n$ satisfy the following conditions:

$$1 \le a < b \le n \text{ and } \tau_a = \tau_b = 1,$$
$$\sigma_w(i) = \sigma^{-1}(a) \text{ and } \sigma_w(i+1) = \sigma^{-1}(b).$$

Proposition 2.1 ([4, Proposition 2.2])**.** $wx_i^{\pm 2}w^{-1} \in \mathbf{Stab}((m, \sigma))$.

Proof. Denote by (C_w, σ_w) the image of w in CB_n. Step by step proof for $wx_i^2w^{-1}$:

- $(m, \sigma) \star w = (m \cdot \epsilon(C_w^\sigma), \sigma\sigma_w)$, where $\sigma\sigma_w(i) = a$ and $\sigma\sigma_w(i+1) = b$.
- $(m, \sigma) \star wx_i = (m \cdot \epsilon(C_w^\sigma) \cdot \epsilon(C_i(t_i)^{\sigma\sigma_w}), \sigma\sigma_w\sigma_i)$, where $C_i(t_i)^{\sigma\sigma_w} = C_i(t_a)$ and, hence, $\epsilon(C_i(t_i)^{\sigma\sigma_w}) = C_i(1)$. Thus, we get $(m \cdot \epsilon(C_w^\sigma) \cdot C_i(1), \sigma\sigma_w\sigma_i)$ and $\sigma\sigma_w\sigma_i(i) = b$ and $\sigma\sigma_w\sigma_i(i+1) = a$.
- $(m \cdot \epsilon(C_w^\sigma) \cdot C_i(1), \sigma\sigma_w\sigma_i) \star x_i = (m \cdot \epsilon(C_w^\sigma) \cdot C_i(1) \cdot \epsilon(C_i(t_i)^{\sigma\sigma_w\sigma_i}, \sigma\sigma_w)$, where $\epsilon(C_i(t_i)^{\sigma\sigma_w\sigma_i}) = \epsilon(C_i(t_b)) = C_i(1)$. Since $C_i(1) \cdot C_i(1) = I$, the result is $(m \cdot \epsilon(C_w^\sigma), \sigma\sigma_w)$.
- Finally, $(m \cdot \epsilon(C_w^\sigma), \sigma\sigma_w) \star w^{-1} = (m, \sigma) \star (w \cdot w^{-1}) = (m, \sigma)$. □

Observe that the property of a braid w to cloak (m, σ) depends on σ only. Hence, we can say that w cloaks σ.

Remark 2.2. Geometrically, the condition in Proposition 2.1 defines a braid that:

- twists strands to set strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$ next to each other using w,
- double twists strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$ using x_i^2,
- twists strands backwards using w^{-1}.

Schematically, $wx_i^2w^{-1}$ has the structure as shown in Figure 1.

Official WalnutDSA specification [2] contains a detailed description for generation of cloaking elements for a permutation σ:

(1) Pick a random integer $2 \le i \le n-1$.
(2) Compute a random permutation σ_w such that $\sigma_w(i) = \sigma^{-1}(a)$ and $\sigma_w(i+1) = \sigma^{-1}(b)$.

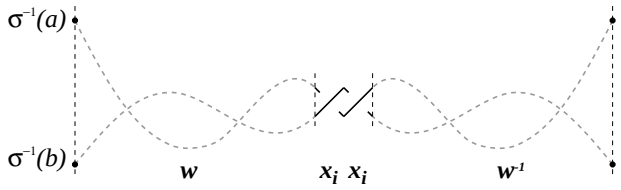

FIGURE 1. Cloaking element.

(3) Generate a random braid w with permutation σ_w.
(4) Extend w with L pure braids.
(5) Compute the cloaking element $v = wx_i^{\pm 2}w^{-1}$.

Generation of a random braid word w with a given permutation σ_w is also described in [2] and consists of decomposing σ_w into the product of transpositions

$$\sigma_w = (i_1, i_1 + 1) \cdot \ldots \cdot (i_k, i_k + 1),$$

where $i_j \in \{1, \ldots, n-1\}$, and taking $w = x_{i_1}^{\varepsilon_1} \ldots x_{i_k}^{\varepsilon_k}$, where $\varepsilon_j = \pm 1$. To meet security requirements, it is suggested to augment the resulting braid word w with L random pure braid generators. Recall that pure braid subgroup of B_n is generated by $\frac{n(n-1)}{2}$ generators

$$g_{i,j} = x_{j-1}x_{j-2} \ldots x_{i+1}x_i^2 x_{i+1}^{-1} \ldots x_{j-2}^{-1}x_{j-1}^{-1}, \qquad 1 \le i < j \le n. \tag{2}$$

Another way to generate cloaking elements was recently suggested by SecureRF in [1]. Fix $(m, \sigma) \in \mathbf{GL}_n(\mathbb{F}_q) \times S_n$ and assume that $a, b, i \in \mathbb{N}$ and $w \in B_n$ satisfy:

$$1 \le a < b \le n \text{ and } \tau_b = -\tau_a^{-1}$$
$$\sigma_w(i) = \sigma^{-1}(a) \text{ and } \sigma_w(i+1) = \sigma^{-1}(b).$$

Proposition 2.3. $wx_i^{\pm 4}w^{-1} \in \mathbf{Stab}((m, \sigma))$.

Proof. It follows from the identity $[C_i(\tau_a)C_i(-\tau_a^{-1})]^2 = I$. □

Geometrically, the braid defined in Proposition 2.3 is very similar to the braid defined in Proposition 2.1, but uses four copies of x_i in the middle instead of two.

3. Fast checks for identity and conjugacy problems in braids

Recall that a braid word $w(x_1, \ldots, x_{n-1})$ is called an *identity* of B_n if it represents the trivial braid. An algorithmic problem to determine if w is an identity, or not, is called the *identity problem* for B_n. Also, recall that braid words u and v are said to be *conjugate* in B_n if

$$v =_{B_n} x^{-1}ux,$$

for some braid word x. An algorithmic problem to determine if u and v are conjugate, or not, is called the *conjugacy problem* for B_n. Both problems are well studied for braids. The identity problem is known to have quadratic time complexity, and it is an open problem if the conjugacy problem can be solved by a polynomial time algorithm (exponential time solution is known).

In practice, the fastest way to solve the identity problem is to use the Dehornoy handle free form, which is conjectured to have linear time complexity, but even that requires some significant time to halt on braids of length 2000 and longer. To speed up computations, we designed the following heuristic that works very fast for nontrivial w.

- Fix a sufficiently large prime modulus p (199 in our experiments) and nontrivial t-values $\tau_1, \ldots, \tau_n \in \mathbb{F}_p$.
- If $(I, \mathrm{id}) \star w \neq (I, \mathrm{id})$, then return **No**. Otherwise return **DoNotKnow**.

Recall that abelianizations of conjugate elements are the same. Hence, a fast and easy way to check if u and v are not conjugate in B_n is to check their abelianizations, and report **No** if they are not the same. This is a very useful approach for practical computations, but it had limited success in our experiments. We designed the following heuristic to check if u and v are not conjugate.

- Fix a sufficiently large prime modulus p (199 in our experiments) and equal t-values $\tau_1 = \ldots = \tau_n \in \mathbb{F}_p \setminus \{0, 1\}$.
- Compute characteristic polynomials for

$$\mathrm{Matrix}((I, \mathrm{id}) \star u) \text{ and } \mathrm{Matrix}((I, \mathrm{id}) \star v)$$

  and return **No** if they are not the same. Otherwise return **DoNotKnow**.

Observe that the choice of all equal t-values gives a homomorphism from B_n into $\mathbf{GL}(n, \mathbb{F}_p)$. Hence, the images of conjugate braids must be conjugate in $\mathbf{GL}(n, \mathbb{F}_p)$. Hence, their characteristic polynomials must be the same.

4. Message encoding

In order to sign a message $m \in \{0, 1\}^*$, one needs to convert it into a braid word. In this section, we describe an encoding method proposed in [4, Section 7]. Let $H\colon \{0, 1\}^* \to \{0, 1\}^{4k}$ be a cryptographic hash function for some positive integer k. The method below defines an encoding function

$E\colon \{0,1\}^{4k} \to B_n$. Consider a collection of braids:

$$\begin{aligned} g_{n-1,n} &= x_{n-1}^2, \\ g_{n-2,n} &= x_{n-1}x_{n-2}^2x_{n-1}^{-1}, \\ g_{n-3,n} &= x_{n-1}x_{n-2}x_{n-3}^2x_{n-2}^{-1}x_{n-1}^{-1}, \\ &\vdots \\ g_{1,n} &= x_{n-1}x_{n-2}\dots x_2x_1^2x_2^{-1}\dots x_{n-2}^{-1}x_{n-1}^{-1}. \end{aligned}$$

It is a well-known fact that the elements $g_{n-1,n}, \dots, g_{1,n}$ form a basis of a free subgroup of B_n. Fix four elements of that form:

$$g_{k_1,n}, g_{k_2,n}, g_{k_3,n}, g_{k_4,n}.$$

This choice defines a function from 4-bit blocks to braids:

$$b_3b_2b_1b_0 \mapsto g_{k_\mu,n}^{\nu}, \tag{3}$$

where $\mu = 2b_1 + b_0 + 1$ and $\nu = 2b_3 + b_2 + 1$. Further, we can assign a braid $E(H(m))$ to any bit-sequence $H(m)$ by cutting $H(m)$ into a sequence of 4-bit blocks and taking the concatenation of the corresponding braid words. Below we mention some features of the encoding E that, in our opinion, make the encryption algorithm weaker.

Remark 4.1. It was noticed in [1] that encoding E defined above is not injective, for example, we have $E(0000\,0100) = (g_{k_1,n})^3 = E(0100\,0000)$. This is definitely a drawback of E. However, E can be easily fixed to guarantee an injective map of bit sequences into braids, and a few ways of achieving it were suggested in [1]. We note that a choice of E is not relevant for success of our attack and we work with the original E as it is defined in (3).

Remark 4.2. Each letter in $g_{i,n}$ crosses the nth strand with other strands and the whole $E(H(m))$ is a pure braid obtained by taking strands $1, \dots, n-1$ and pulling the nth strand through them (without moving strands $1, \dots, n-1$), see Figure 2. This situation (pulling a single strand through others) can

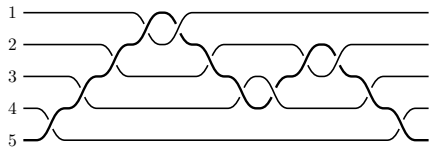

FIGURE 2. The element $g_{1,5}g_{3,5}^{-1}g_{4,5}$.

be easily recognized algorithmically. We believe this observation can lead to a successful attack that tries to find and isolate $E(H(m))$ inside a signature for m.

Remark 4.3. For any choice of $k_1, \ldots, k_4$ there exists a conjugator c such that $\langle g_{k_1,n}, g_{k_2,n}, g_{k_3,n}, g_{k_4,n}\rangle^c$ is a subgroup of $\langle x_1, x_2, x_3, x_4\rangle$. We believe that this property contributes to the efficacy of our attack. We enumerate conjugates of some tuples of braids in our attack. In all our experiments, our procedure was able to reduce the rank of the underlying braid group to 5 using an appropriate sequence of conjugations, see Section 8.2.1. This observation also explains the reason why the authors of [5] in Section 4.1 obtain dimension 13 in practical experiments.

5. The protocol

WalnutDSA allows a signer (Alice) 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 Alice's public-key and the verification protocol. We now describe the algorithms for private-/public-key generation.

Fix *initial public information* available to each interested party:

- The rank n of the braid group B_n ($n \geq 8$).
- A rewriting algorithm $\mathcal{R} : B_n \to B_n$, called an *obfuscation procedure*, that transforms a braid word w into an equivalent braid word $\mathcal{R}(w)$ and, in some sense, hides information in w.
- A finite field $\mathbb{F}_q$ ($q \geq 32$).
- Two integers $1 \leq a < b \leq n$.
- Non-zero values $\tau_1, \ldots, \tau_n \in \mathbb{F}_q$ such that $\tau_a = \tau_b = 1$ or $\tau_b = -\tau_a^{-1}$ depending on a choice of a cloaking element type.

By a *protocol instance* we mean a set of values for all the parameters above.

Alice's private key:

- A pair of braid words (w, w') representing elements in B_n.

Alice's public key:

- A pair $(\mathcal{P}(w), \mathcal{P}(w'))$, see (1) for the definition of $\mathcal{P}$.

To sign a message m, Alice does the following:

- computes $E(H(m))$;
- picks v that cloaks (I, id), v_1 that cloaks $\mathcal{P}(w)$, and v_2 that cloaks $\mathcal{P}(w')$;
- computes $S = \mathcal{R}(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2)$;
- finally, the signature for m is the pair $(H(m), S)$.

To validate the signature, one checks the equality:

$$\mathrm{Matrix}(\mathcal{P}(w) \star S) = \mathrm{Matrix}(\mathcal{P}(E(H(m)))) \cdot \mathrm{Matrix}(\mathcal{P}(w')).$$

It is easy to check that a valid signature passes this test.

We note that the official WalnutDSA specification [2] appeared later than paper [4], and the following changes to the protocol were made:

(1) the public key is of the form $(\mathcal{P}(w), \mathrm{Matrix}(\mathcal{P}(w'))$.
(2) $a = 1$ and $b = 2$ (with $\tau_1 = \tau_2 = 1$).

We note that the first change does not make any difference since the permutation $\sigma_{w'}$ can be restored from any signature. Also, a choice of a and b does not affect our attack and we used values from [4].

6. Key generation and parameter values

For 128-bit security level, the official specification [2] suggests the following parameters:

- $n = 8$.
- $q = 32$.
- $L = 15$ (see Section 2.5).
- $l = 132$ (the minimal length of braids in private keys).
- H is SHA2-256 hash function.

For 256-bit security level suggested parameters are:

- $n = 8$.
- $q = 256$.
- $L = 30$.
- $l = 287$.
- H is SHA2-512 hash function.

Recently the following changes to the original parameters were proposed in [1] to counter [5]:

(1) $n = 11$ instead of $n = 8$;
(2) $q = 2^{31} - 1$ and $q = 2^{61} - 1$ for 128-bit and 256-bit security levels respectively;
(3) $\tau_b = -\tau_a^{-1}$ and cloaking elements of the form $wx_i^{\pm 4}w^{-1}$, as defined in Proposition 2.3.

6.1. **Tested parameter values.** We tested originally proposed sets of parameters for 128-bit and 256-bit security levels and also 256-bit parameters with $n = 11$. For each set of parameters we tried 2 cases, one using cloaking elements of Proposition 2.1 and another using cloaking elements of Proposition 2.3. Note that:

- We did not test values $q = 2^{31} - 1$ or $q = 2^{61} - 1$ since the size of the base field is not relevant for the success of our attack as our algorithm works with braids only.
- In our experiments we did not work with actual hash values of messages $H(m)$. Instead, we generated random bit sequences of length 256 or 512.
- We used Garside normal forms [9] to obfuscate signatures instead of Birman-Ko-Lee forms [6] since they are the most tested normal forms in CRAG [7]. This technical preference does not give us any particular advantage.

7. Short survey of existing attacks

In this section, we give short explanations of the attacks presented in [10] and [5] to show that we use a completely different approach.

7.1. **Factorization attacks.** Recall the definition of existential *unforgeability under chosen message attacks*. An adversary can ask for polynomially many signatures of messages of its choice to a signing oracle. The attack is considered successful if the attacker is able to produce a valid pair (message, signature) for a message different from those queried to the oracle. In [10] it is shown that the earlier version of WalnutDSA, where the two secret braids w and w' are equal, is not resistant to this kind of attack. The method is modified in [5] to break the current version of WalnutDSA. It exploits the following property.

Theorem 7.1 ([5])**.** *Suppose m, m_1, m_2 are messages. Let h, h_1, h_2 be the matrix parts of $\mathcal{P}(E(H(m))), \mathcal{P}(E(H(m_1))), \mathcal{P}(E(H(m_2)))$ respectively. Let $w_1, w_2, w_3 \in B_n$ be braids. Then*

1. *If $m = m^{-1}$ and S_1 is a valid signature for m_1 under the public key $(\mathcal{P}(w_1), \mathcal{P}(w_2))$, then S_1^{-1} is a valid signature for m under the public key $(\mathcal{P}(w_2), \mathcal{P}(w_1))$.*
2. *If $m = m_1 \cdot m_2$ and S_1, S_2 are valid signatures for m_1 and m_2 under the public keys $(\mathcal{P}(w_1), \mathcal{P}(w_2))$ and $(\mathcal{P}(w_2), P(w_3))$ respectively, then $S_1 \cdot S_2$ is a valid signature for m under the public key $(\mathcal{P}(w_1), \mathcal{P}(w_3))$.*

Let O be an oracle that solves the factorization problem in the group A_n of invertible matrices of the form

$$\left\{ \begin{pmatrix} X & Y \\ 0 & 1 \end{pmatrix} \mid X \in \mathbf{GL}_{n-1}(\mathbb{F}_q), Y \in \mathbb{F}_q^{n-1} \right\}.$$

It is possible to break WalnutDSA using O as follows. Suppose we want to forge a signature for a message m under the public key $(\mathcal{P}(w), \mathcal{P}(w'))$. We start by collecting a number of pairs $(m_1, S_1), \ldots, (m_k, S_k)$ that are valid under the same public key. Now it suffices to find a factorization $h = h_{i_1} \cdot h_{i_2}^{-1} \cdot h_{i_3} \cdots h_{i_{m-1}}^{-1} \cdot h_{i_m}$, where h is the matrix part of $\mathcal{P}(E(H(m)))$, and h_i are the matrix parts of $\mathcal{P}(E(H(m_i)))$. To find such a factorization, one constructs the list of generators $\{h_i \cdot h_j^{-1} \mid i \neq j; 1 \leq i, j \leq k\}$ and calls the oracle O to obtain a factorization for $h \cdot h_1^{-1}$ with factors in this list. Appending the factor h_1 to the result, we get a factorization of h of the desired form. This reduces breaking WalnutDSA to the factorization problem in A_n.

[10] proposes an algorithm to solve the factorization problem. The algorithm has a time complexity $O(q^{\frac{n-1}{2}})$; and for the 128-bit security parameters it finds a factorization in minutes. However, the constructed factorizations contain roughly 2^{25} factors. Therefore, the factorization attacks can be blocked by imposing a length limit on valid signatures.

7.2. **A collision search attack.** The second attack in [5] constructs a pair of messages with the same signature. Recall that a signature S is validated using the equality

$$\mathrm{Matrix}(\mathcal{P}(w) \star S) = \mathrm{Matrix}(\mathcal{P}(E(H(m)))) \cdot \mathrm{Matrix}(\mathcal{P}(w')).$$

Therefore, if m_1 and m_2 satisfy $\mathcal{P}(E(H(m_1))) = \mathcal{P}(E(H(m_2)))$, then a signature S is valid for m_1 if and only if it is valid for m_2. A generic collision search requires roughly $|\mathcal{P}(E(H(\{0,1\}^*)))|^{1/2}$ evaluations of the composition function $\mathcal{P} \circ E \circ H$. [5] shows that this value is at most $q^{6.5}$ for the originally proposed parameter values. Therefore, finding a collision cannot take much more than $2^{32.5}$ evaluations of $\mathcal{P} \circ E \circ H$ for 128-bit security parameters and 2^{52} evaluations of $\mathcal{P} \circ E \circ H$ for 256-bit security parameters.

The authors of [5] implemented the generic collision finding algorithm of van Oorschot and Wiener and used it to find collisions for the function $g \circ \mathcal{P} \circ E \circ H$, where g is a function that takes the output of $\mathcal{P}$ and converts it to some plausible message m. It takes one hour on a standard desktop PC to find a pair of colliding messages.

7.3. **Reversing E-multiplication.** A fundamental problem underlying WalnutDSA is the "reversing E-multiplication" problem. Given a pair (m, σ) such that $(m, \sigma) = (I, \mathrm{id}) \star w$ for some braid $w \in B_n$, we need to find a braid $w' \in B_n$ such that $(I, \mathrm{id}) \star w' = (m, \sigma)$. The third attack proposed in [5] solves the REM problem for the proposed parameter values. The method exploits the fact that E-multiplication, when restricted to pure braids, is a group homomorphism and that this homomorphism maps the chain of subgroups $P_2 \subset P_3 \subset \cdots \subset P_n$ to a nice chain of subgroups of $\mathbf{GL}_n(\mathbb{F}_q)$. The attack directly constructs equivalent secret keys in under one second for 128-bit security parameters and in less than a minute for 256-bit security parameters.

All the attacks exploit various algebraic properties of the E-multiplication fundamental for WalnutDSA. In order to avoid these attacks, the parameters of WalnutDSA should be increased significantly.

8. The attack

We start out with a general outline of our attack. Details are presented in the following subsections. Initially, given an instance of the protocol, the attack collects a number of messages m_i signed by Alice with her private key (w, w'), i.e., pairs $(m_i, S_A(m_i))$, where:

$$S_A(m_i) = v_1^{(i)} \cdot w^{-1} \cdot v^{(i)} \cdot E(H(m_i)) \cdot w' \cdot v_2^{(i)},$$

for $i = 1, \ldots, k$. Then it removes cloaking elements from the signatures $S_A(m_i)$ and produces products $P_i = w^{-1} \cdot E(H(m_i)) \cdot w'$ for each i. Note

that the following equalities hold:

$$\begin{cases} P_1 P_2^{-1} = w^{-1} E(H(m_1)) E(H(m_2))^{-1} w, \\ \dots \\ P_{k-1} P_k^{-1} = w^{-1} E(H(m_{k-1})) E(H(m_k))^{-1} w, \end{cases} \tag{4}$$

with known elements $P_{i-1}P_i^{-1}$ and $E(H(m_{i-1}))E(H(m_i))^{-1}$ and the secret element w. Hence, on the next stage, we solve the system of conjugacy equations (4). Denote the obtained solution by x and compute x':

$$x' = E(H(m_i))^{-1} x P_i, \tag{5}$$

using some $i = 1, \dots, k$. We say that the attack is *successful* if the pair (x, x') is a valid substitute for the pair (w, w'). That property is checked by generating 10 random messages, signing them without cloaking elements using (x, x') as a private key, and validating the produced signatures.

We would like to point out that in our experiments we never obtained the original private key (w, w') and this is not an issue. Denote the centralizer

$$C = C(E(H(m_1))E(H(m_2))^{-1}, \dots, E(H(m_{k-1}))E(H(m_k))^{-1}).$$

Clearly, any solution x of (4) is of the form $x = hw$, where h belongs to the centralizer C, and $x' = E(H(m_i))^{-1} h w P_i$. Signing a random message m without cloaking elements and with (x, x') as a private key, we get

$$\begin{aligned} S_A(m) &= x^{-1} E(H(m)) x' \\ &= w^{-1} h^{-1} E(H(m)) E(H(m_i))^{-1} h w P_i \\ &= w^{-1} h^{-1} E(H(m)) E(H(m_i))^{-1} h E(H(m_i)) w'. \end{aligned}$$

If h centralizes $E(H(m))E(H(m_i))^{-1}$, then

$$\begin{aligned} S_A(m) &= w^{-1} h^{-1} E(H(m)) E(H(m_i))^{-1} h E(H(m_i)) w' \\ &= w^{-1} E(H(m)) E(H(m_i))^{-1} E(H(m_i)) w' \\ &= w^{-1} E(H(m)) w', \end{aligned}$$

so the signature of m obtained using (x, x') coincides as a braid word with the signature obtained using (w, w'). We note that this was the case in all our experiments; and all our forged signatures were equivalent as braids to the original signatures generated without cloaking elements.

In the rest of the section, we provide detailed description of the main steps in the attack.

8.1. **Uncloaking signatures.** Here we describe a heuristic procedure that removes cloaking elements from a given signature

$$S = \mathcal{R}(v_1 w^{-1} v E(H(m)) w' v_2)$$

and computes the product $w^{-1}E(H(m))w'$. Our algorithm uses the following rather informally stated observations:

- As mentioned in Remark 2.2, the letters x_i in the word $w x_i^{\pm 2} w^{-1}$ from Proposition 2.1 twist two particular strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$.

- Replacement $wx_i^{\varepsilon}x_i^{\varepsilon}w^{-1} \to wx_i^{-\varepsilon}x_i^{\varepsilon}w^{-1}$, where $\varepsilon = \pm 1$, produces the trivial braid.
- Replacement of a single letter $x_i^{\pm 1}$ that twists strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$ with $x_i^{\mp 1}$ in a braid word corresponds to multiplication of the word by a cloaking element.
- Multiplying a braid word with cloaking elements on the left or on the right increases the length of the braid.
- Even though obfuscation of a cloaked braid word changes the way the word looks, it preserves the isotopy type of the braid and the result of obfuscation typically twists strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$ at the crossing corresponding to the middle of $wx_i^{\pm 2}w^{-1}$. See Remark 9.1.
- By tracing strands in a given braid, we can algorithmically find all letters that twist strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$. We call those letters *critical letters*.

The uncloaking algorithm **operates on pairs of signatures** S_1, S_2 as follows:

(1) Trace the strands and find all critical letters in the signature S_1. For each critical letter, flip its power and braid-minimize the result:

$$S_1 = \dots x_i^{\pm 1} \dots \to \dots x_i^{\mp 1} \dots .$$

That produces a list of braid words $S_1^{(1)}, \dots, S_1^{(k)}$. Perform the same with S_2 and obtain a list of braid words $S_2^{(1)}, \dots, S_2^{(l)}$. Then consider all products of the type:

$$S_1^{(i)} \cdot \left(S_2^{(j)}\right)^{-1}, \quad \text{where } 1 \le i \le k,\ 1 \le j \le l. \tag{6}$$

Braid-minimize the length and find a pair $S_1^{(i)}, S_2^{(j)}$ that gives a product of minimal length.

Most of the time these manipulations produce a pair of signatures with cloaking elements v_2 removed, because a proper removal of v_2 (that comes from flipping the power of a critical letter) in S_1 and in S_2 results in full cancellation of w' in the product (6).

(2) We remove cloaking elements v_1 from S_1 and S_2 in a similar fashion. We find all critical letters, flip powers, consider all products of the type $\left(S_1^{(i)}\right)^{-1} \cdot S_2^{(j)}$, and choose the pair that gives the minimal length of the product (under assumption that a big cancellation in that product is a result of a proper removal of cloaking elements v_1).

(3) We remove the middle cloaking element from each signature S_1 and S_2 separately. As above, we find all critical letters in the signature, flip powers, minimize the result, and choose a word of minimal length.

The procedure outlined above works specifically for cloaking elements described in Proposition 2.1. To remove cloaking elements given in Proposition 2.3, we flip the powers of critical letters as follows: $x_i^{\pm 1} \to x_i^{\mp 3}$.

8.2. **Solving a system of conjugacy equations in B_n with errors.** As described above, a proper removal of cloaking elements leads to the system of conjugacy equations (4). Unfortunately, our uncloaking procedure occasionally fails to produce an element $w^{-1}E(H(m_i))w'$, which gives a system (4) that has no solutions. This situation can be countered by generating a new collection of signatures and running the process again hoping that the system (4) has no errors.

Instead, we devised a heuristic algorithm to solve a system of conjugacy equations in B_n:

$$\begin{cases} v_1 = x^{-1}u_1x, \\ \dots \\ v_k = x^{-1}u_kx, \end{cases}$$

with a few errors, i.e., a system where a few (one or two) equations do not hold.

The algorithm starts with two tuples of braids $\overline{v} = (v_1, \dots, v_k)$ and $\overline{u} = (u_1, \dots, u_k)$ and then enumerates conjugates of these tuples trying to minimize the total length. The algorithm runs until it finds an element conjugate to both $\overline{v}$ and $\overline{u}$. On each iteration the algorithm picks an unchecked tuple $\overline{v}' = (v_1', \dots, v_k')$ conjugate to $\overline{v}$ of the least total length and conjugates it by each generator x_i to produce a tuple

$$x_i^{-1}\overline{v}'x_i = (x_i^{-1}v_1'x_i, \dots, x_i^{-1}v_k'x_i),$$

with each entry being braid-minimized. If the tuple is new, we save it as a new unchecked tuple conjugate to $\overline{v}$. We process tuples conjugate to $\overline{u}$ in the same fashion. The algorithm was allowed to make sixty iterations to recover a pair (x, x').

8.2.1. *Reducing the rank.* If at any iteration we find out that a tuple $\overline{v}'$ (or $\overline{u}'$) does not involve one braid generator, then we appropriately conjugate $\overline{v}'$ to obtain a tuple from B_{n-1}. From that point on we conjugate tuples by generators $x_1, \dots, x_{n-2}$ only. This improves running time of the algorithm. In all our experiments, we observed that both tuples were conjugated to tuples of braid words on generators x_1, x_2, x_3, x_4 (a consequence of Remark 4.3), that improved the running time of the attack significantly (by about a factor of two).

8.2.2. *Fast checks to find errors in a system.* To identify errors in the system of the conjugacy equations, we used fast checks described in Section 3, namely abelianization and characteristic polynomials of non-colored Burau matrices. It turned out that this approach worked for all experiments we generated.

This requires some explanation. Our initial algorithm checked abelianization only, which missed about a half of bad conjugacy equations. That is the reason why we remove worst performing components (Section 8.2.3) and allow several attempts (Section 8.2.5). The reader can find these components in the implementation. Exploiting characteristic polynomials of non-colored Burau matrices solved the problem of erroneous equations (for randomly generated instances). But, of course, there exist cases for which our fast check will fail. We keep components described in Sections 8.2.3 and 8.2.5 for such cases.

8.2.3. *Removing worst performing components from the system.* On iterations $15, 20, 25, 30$ our algorithm finds the "worst performing" components and drops it from the system. We pick a tuple of the least total length $\overline{v}'$ and compute the values $|v_i| - |v_i'|$, which shows our progress (length decrease) in the ith component. The component with the least decrease is dropped from the system.

8.2.4. *Fast descend.* To improve running time of the heuristic descend, on each iteration, we conjugate the current tuple $\overline{v}' = (v_1', \ldots, v_k')$ with the inverted initial segment of v_1' of length 10. This gives a significant boost to performance and allows to significantly decrease the number of steps in the described heuristic descend.

Also, initially, we conjugate the tuple of elements

$$v_i = w^{-1}E(H(m_i))E(H(m_{i+1}))^{-1}w$$

with the inverted initial segment of v_1 of length 250. This trick dramatically improves performance of the descend.

8.2.5. *Attempts.* In rare cases when our algorithm makes too many errors removing cloaking elements, we generate new signatures and run the process from scratch. In our experiments, we allow 10 attempts before we accept failure.

9. Results and further remarks on performance of the attack

In this section, we describe two experiments we performed to demonstrate correctness of our heuristic claims. All experiments were performed on an Intel I7-4770 based personal computer (eight logical core CPU with each core clocked at 3.40GHz) with 16GB memory.

The first experiment shows that our heuristic indeed can identify (as described in Section 8.1) cloaking elements in a random signature. We tested 3 sets of parameters corresponding to 128-bit, 256-bit, and 256-bit with $n = 11$ security levels (see Section 6 for details). For each set of parameters we tested originally proposed (Proposition 2.1) as well as recently proposed (Proposition 2.3) cloaking elements. For each case we performed 100 tests described below:

(1) Generate a random protocol instance.

(2) Generate a random private key (w, w').
(3) Generate a signature $S = \mathcal{R}(v_1 w^{-1} v E(H(m)) w' v_2)$ for a random message m.
(4) Find all critical letters in S. For each critical letter, flip its power and braid-minimize the result to get a list of braids $S^{(1)}, \dots, S^{(k)}$.
(5) The test is called *successful* if for some i_1, i_2, i_3 we have

$$S^{(i_1)} = w^{-1} v E(H(m)) w' v_2,$$
$$S^{(i_2)} = v_1 w^{-1} E(H(m)) w' v_2,$$
$$S^{(i_3)} = v_1 w^{-1} v E(H(m)) w',$$

i.e. if the list $S^{(1)}, \dots, S^{(k)}$ contains S without each cloaking element.

Table 1 shows success rate for each set of parameters. Notice that it does not depend much on the rank of the braid group and on the length of private keys. It is about 80% for the cloaking elements defined in Proposition 2.1 and 100% for those defined in Proposition 2.3. That means that using cloaking elements from Proposition 2.3 makes the challenge easier.

Ultimately, the reason for failure in the first experiment is obfuscation of braid words. Note that we used normal forms to obfuscate signatures, which is the strongest possible way to obfuscate braids. Figure 3 gives an example of a product of two cloaking elements modified into a braid that has no critical letters.

Remark 9.1. It is not correct to claim that for any braid words u, v such that u is equal to v in B_n, u crosses strands $\sigma^{-1}(a)$ and $\sigma^{-1}(b)$ if and only if v does. Figure 3 shows that a product of two cloaking elements $x_1^{-1} x_2^2 x_1 \cdot x_1 x_2^{-2} x_1^{-1}$ (here $\tau_1 = \tau_3 = 1$) can be written as a braid $x_1^{-2} x_2^{-2} x_1^2 x_2^2$ that does not cross the first and the third strands, i.e., it does not contain critical letters. Notice that both words have length 8.

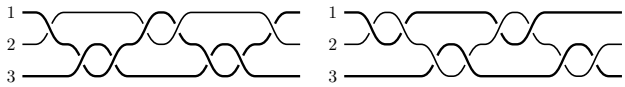

FIGURE 3. Hidden cloaking elements.

	128-bit	256-bit	256-bit, $n = 11$
Cloaking elements $w x_i^2 w^{-1}$	80%	77%	76%
Cloaking elements $w x_i^4 w^{-1}$	100%	100%	100%

TABLE 1. Percentage of properly identified cloaking elements v_1, v, v_2.

Cloaking elements given in Proposition 2.3 are much easier targets for our attack than those given in Proposition 2.1 as they contain more critical

letters and are easier to detect. Furthermore, it is easier to recognize bad critical pairs (those that do not come from cloaking elements). Indeed, if a critical letter x_i does not come from a cloaking element, then replacing x_i with x_i^{-3} results in length increase by two symbols.

Our second experiment tested success rate of the full attack. For each set of parameters and a way to generate cloaking elements, we generated one hundred random protocol instances with random private keys. As described in Section 8, we first uncloaked signatures and then solved the corresponding system of conjugacy equations. In all cases cloaking elements were correctly removed from signatures and, hence, systems of the conjugacy equations did not have errors. In particular, our attack used only one attempt to find private keys. Table 2 shows average running time of the attack.

	128-bit	256-bit	256-bit, $n = 11$
Cloaking elements $wx_i^2w^{-1}$	89.94	544.82	781.38
Cloaking elements $wx_i^4w^{-1}$	74.88	495.10	620.48

TABLE 2. Average running time (in seconds) for the full attack.

Increasing the rank of the braid group makes the algorithm slower as it has to perform more conjugations during heuristic descend. It also takes more time to compute Dehornoy form. On the other hand it makes the braids more sparse (without increasing lengths of the involved braid) and descends more straightforward.

10. Conclusion

The protocol described in [4] does not provide the claimed level of security. It suffers from poor choice of cloaking elements. By design, cloaking elements have very specific geometric type defined by a fixed pair of strands that can be algorithmically recognized (critical letters) and removed. Once the cloaking elements are removed, a heuristic length-minimization allows to solve simultaneous conjugacy search problem and find substitutes for the private key.

A potential way to defeat our attack is to artificially introduce many critical letters into private braids and use many (say 30) cloaking elements on each side instead of one. But, as we show in our analysis [12] of the Kayawood key agreement protocol proposed in [3], that might be fruitless especially with higher values of the rank. Also, we suggest to use short conjugators for cloaking elements. Long conjugators make cloaking elements more visible.

References

[1] NIST PQC forum. Available at `https://groups.google.com/a/list.nist.gov/forum/#!forum/pqc-forum`, accessed: April 4, 2018.

[2] I. Anshel, D. Atkins, and P. Goldfeld, D. Gunnels. The Walnut digital signature algorithm(TM) specification. Submitted to NIST PQC project (2017). Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`, accessed: April 4, 2018.

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

[4] I. Anshel, D. Atkins, and P. Goldfeld, D. Gunnels. WalnutDSA(TM): A Quantum-Resistant Digital Signature Algorithm. Preprint. Available at `https://eprint.iacr.org/2017/058` (version: 30-Nov-2017), 2017.

[5] W. Beullens and S. Blackburn. Practical attacks against the Walnut digital signature scheme. Preprint. Available at `https://eprint.iacr.org/2018/318/20180404:153741`, 2018.

[6] J. S. Birman, K. H. Ko, and S. J. Lee. A new approach to the word and conjugacy problems in the braid groups. *Adv. Math.*, 139:322–353, 1998.

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

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

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

[10] D. Hart, K. DoHoon, G. Micheli, G. Perez, C. Petit, and Y. Quek. A Practical Cryptanalysis of WalnutDSA. Preprint. Available at `https://eprint.iacr.org/2017/1160` (version: 30-Nov-2017), 2017.

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

[12] M. Kotov, A. Menshov, and A. Ushakov. Attack on Kayawood protocol: uncloaking private keys. In preparation.

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

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

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

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

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA

E-mail address: `mkotov@stevens.edu`

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA, Institute of Mathematics and Information Technologies, Dostoevskii Omsk State University, Omsk, Russia

E-mail address: `manton@stevens.edu`

Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ, USA

E-mail address: `aushakov@stevens.edu`