Practical attacks against the Walnut digital signature scheme

Ward Beullens[1] and Simon R. Blackburn[2]

[1] imec-COSIC KU Leuven,
Kasteelpark Arenberg 10 - bus 2452, 3001 Heverlee, Belgium
Ward.Beullens@esat.kuleuven.be
[2] Department of Mathematics, Royal Holloway, University of London,
Egham, Surrey TW20 0EX, Royal Holloway, UK
S.Blackburn@rhul.ac.uk

Abstract. Recently, NIST started the process of standardizing quantum-resistant public-key cryptographic algorithms. WalnutDSA, the subject of this paper, is one of the 20 proposed signature schemes that are being considered for standardization. Walnut relies on a one-way function called E-Multiplication, which has a rich algebraic structure. This paper shows that this structure can be exploited to launch several practical attacks against the Walnut cryptosystem. The attacks work very well in practice; it is possible to forge signatures and compute equivalent secret keys for the 128-bit and 256-bit security parameters submitted to NIST in less than a second and in less than a minute respectively.

Keywords: WalnutDSA, NIST PQC, post-quantum digital signatures, cryptanalysis, group based cryptography

1 Introduction

As more and more progress is being made towards building large scale quantum computers, the need for cryptography that can withstand cryptanalysis from these machines has become increasingly urgent. In recognition of this fact, NIST has started the Post-Quantum Cryptography standardization project [20] and made a call for quantum-resistant public-key cryptographic algorithms for standardization. The community has answered this call by submitting 20 proposals for signature schemes and 49 proposals for encryption schemes. One of the submitted signature schemes is the Walnut digital signature algorithm [5, 8], submitted by D. Atkins and owned by SecureRF. SecureRF is a corporation founded in 2004 that develops and licenses public-key security tools for the low-resource processors powering the Internet of Things (IoT) [1]. SecureRF received the ARM Techcon 2017 "Best contribution to IoT security" award for the Walnut signature scheme and their "Key Agreement Protocol". SecureRF wants to

achieve widespread usage of the Walnut signature scheme in the booming IoT market through standardization, partnerships with manufactures like Intel and STMicroelectronics and by providing free toolkits for popular low end platforms. Because of this potential for widespread use, it is crucial to analyze the Walnut scheme for potential weaknesses.

Related work. For its security, Walnut relies on problems taken from the theory of infinite non-commutative groups (more precisely, problems based on an action of a braid group on a finite set via the coloured Burau representation). The idea of using infinite groups in cryptography goes back at least as far as Wagner and Magyarik [26] in 1985; see González Vasco and Steinwandt [25] for an attack on this proposal. Problems in braid groups have been proposed as hard problems for cryptographic primitives for about 20 years now: key agreement protocols due to Ko et al. [18] and Anshel, Anshel and Goldfeld [3] (which is in a more general setting) are the best known examples. The Algebraic Eraser [4] is a more recent proposal, also promoted by SecureRF, which uses many of the same algebraic techniques as Walnut. Early cryptanalyses of these schemes used length-based attacks [15, 16], but the most convincing attacks [11, 10, 12, 17, 23] have generally been based on representation theory (where 'linearisation' techniques reduce the underlying security to a problem in linear algebra). Walnut is interesting because these linearisation techniques do not seem to apply.

The first attack on (an earlier version of) Walnut [6] is due to Hart et al. [14]. The attack forges signatures in minutes for the suggested parameters, but the resulting signatures are significantly longer than legitimately produced signatures. So the Hart et al. attack can be blocked by imposing a length limit on valid signatures. In their submission to NIST, the designers of Walnut impose such a length limit in order to block the Hart et al. attack, but also modify the scheme in a significant way (in particular changing the form of the public and private keys) in an attempt to block the attack altogether.

Contributions. In this paper we present three independent practical attacks on the Walnut signature scheme. The first attack is a modification of the attack of [14] that applies to the adapted version of Walnut that was supposed to resist this attack. This first attack is practical, but has the same limitation as the original attack by Hart et al: the forged signatures are very long. This attack demonstrates that the modifications intended to completely block the Hart et al. attack are not effective, but the attack can be blocked (as before) by imposing a length limit on signatures. The other two attacks presented in this paper produce forgeries whose lengths are the same or even shorter than those of legitimate signatures. The second attack in this paper constructs pairs of messages with the same signature; the attacker can choose a large amount of the structure of these messages. Our third attack directly constructs equivalent secret keys. We are able to forge signatures and compute equivalent secret

keys in under one second for 128-bit security parameters, and in less than a minute for 256-bit security parameters. This shows that the parameter sets submitted to the NIST PQC standardization project are totally insecure, and that the corresponding implementation (which is freely available on the SecureRF website) should not be used. Our attacks exploit various algebraic properties of the one-way function called E-Multiplication, which is fundamental for the Walnut scheme (and other SecureRF methods). In fact, we give a practical algorithm to break the one-wayness of this function for the parameters submitted to NIST. In order to avoid the attacks given here, the parameters of Walnut need to be increased significantly (see the conclusion at the end of the paper for details). However, with these increased parameter sizes, it seems that Walnut loses its performance advantage over other post-quantum signature schemes such as lattice-based, code-based, multivariate and hash-based signatures.

Outline. In Sect. 2 we explain some necessary preliminaries such as distinguished point collision finding, a very short introduction to braid groups, and an explanation of E-Multiplication and the workings of the Walnut signature scheme. The following sections, Sections 3, 4 and 5, each introduce a practical attack against the Walnut scheme and discusses the feasibility of countermeasures. Sect. 3 contains an adaptation of the factorization attack of [14] that applies to the updated version of Walnut that was submitted to NIST. Sect. 4 describes an attack where we use a generic distinguished point collision finding method to find two documents d_1 and d_2 such that a signature that is valid for d_1 is automatically valid for d_2 and vice versa. In Sect. 5 we give an algorithm that breaks the one-wayness of the E-Multiplication map. This algorithm can be used to forge signatures and compute equivalent secret keys, even for the 256 bits of security parameters. The last section presents the conclusions of the paper.

2 Preliminaries

2.1 Distinguished point collision finding

The attacks introduced in this paper rely on a collision finding algorithm that is able to find a collision in any function $f : D \to D$ which maps a domain D to itself. Our algorithm of choice is the distinguished point method of van Oorschot and Wiener [24]. Finding a single collision with this method has the same $O(\sqrt{|D|})$ time complexity as Pollard's rho method with cycle finding [21, 22], but it can be parallelized more efficiently. Moreover, the method of van Oorschot and Wiener is much more efficient for finding multiple collisions; the number of collisions found grows quadratically with the time spent.

The algorithm repeatedly chooses a random starting point $x_1 \in D$ and iteratively applies the function f to obtain a chain of values $x_1, x_2, \cdots$, where $x_i = f(x_{i-1})$ for all $i > 1$. This process continues until a *distinguished* value x_k is reached. This is a value which satisfies some easily verified property, such as having a fixed number of leading zero bits. This property is chosen such that it is satisfied by a fraction ϑ of the elements of D. When the distinguished point is reached the starting point x_1, the distinguished point x_k and the length k of the chain is stored in a table. Assuming f behaves like a random function, after an expected number of $O(\sqrt{|D|})$ function calls the current chain will collide with one of the previously computed chains. From this point on we will follow the same chain and we will end up at the same distinguished point. We read the starting poins x_1, x_1' and the corresponding chain lengths k, k' from the table. Without loss of generality, we assume that $k \geq k'$. We then know that for some $i < k'$

$$x_{k-k'+i} \neq x_i' \qquad \text{and} \qquad f(x_{k-k'+i}) = f(x_i')\,,$$

unless the starting point x_0' appears in the chain starting at x_0 (which only happens with a very small probability). This collision can be extracted with $k - k' + 2i$ function calls. If we require more than one collision we can continue the process, maintaining the contents of the table. Since over time the table will contain more and more chains, the rate at which collisions are found will also increase.

2.2 Braid groups

Informally, the braid group on N strands is a group whose elements are represented by a configuration of N non-intersecting vertical strands in three dimensional space, where 2 configurations are considered equal if one can be transformed continuously into the other configuration without intersecting the strands. The group multiplication is defined as the concatenation of the strands. E. Artin [9] showed that there is an equivalent definition of braid groups, given by the presentation

$$\left\langle b_1, \cdots, b_{N-1} \,\middle|\, \begin{array}{cc} b_i b_j = b_j b_i & \text{for } 1 \leq i < j < N \text{ and } j - i \geq 2 \\ b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1} & \text{for } 1 \leq i < N - 1 \end{array} \right\rangle .$$

Here, the *Artin generator* b_i represents the braid where the i-th strand crosses over the $(i+1)$-th strand. The relations $b_i b_j = b_j b_i$ for $|i - j| \geq 2$ correspond to the fact that crossings that involve different strands are free to move past each other. The relations $b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1}$ correspond to moving one strand over the crossing of two other strands. The Artin generators and their relations are graphically represented in Fig. 1, 2 and 3.

There is a natural homomorphism $\sigma : B_N \to S_N$ from the braid group on N strands to the symmetric group of order N that maps each braid to the permutation obtained by following the strands. This map sends an Artin generator b_i

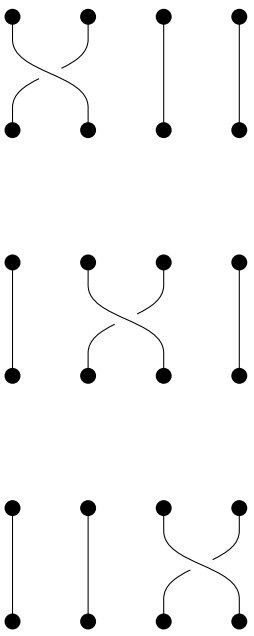


Fig. 1. The three Artin generators b_1, b_2 and b_3 that generate B_4.

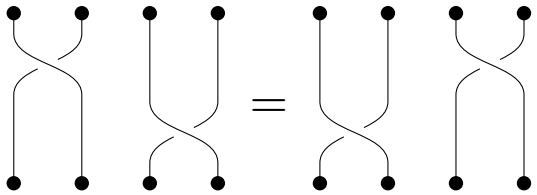


Fig. 2. Crossings that do not share strands commute, i.e. $b_1 b_3 = b_3 b_1$

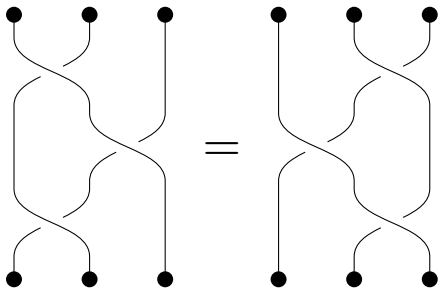


Fig. 3. The first strand moves over the crossing of strand 2 and 3, i.e. $b_1 b_2 b_1 = b_2 b_1 b_2$.

to the transposition $\sigma(b_i) = (i \quad i+1)$. Elements in the kernel of this homomorphism are called *pure braids*, the kernel itself is called the *pure braid group on N strands* and is denoted by P_N.

The braid group B_2 on two strands is the infinite cyclic group, so this group is its own center. For $N > 2$ the center of the braid group on N strands is generated by the full-twist braid which is obtained by grabbing the ends of the strands of the identity braid and rotating them by 360 degrees [13]. This braid is commonly denoted by Δ^2 and is depicted in Fig. 4.

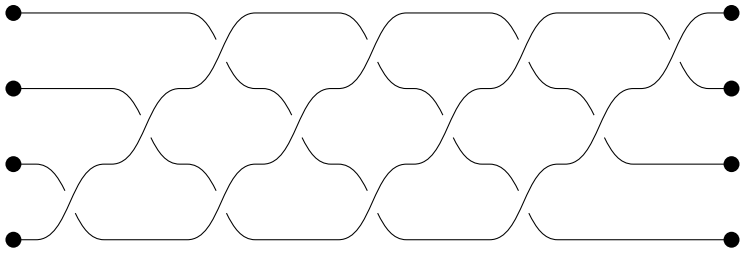


Fig. 4. The full-twist braid Δ^2 in the braid group on 4 strands.

2.3 The colored Burau representation and E-multiplication

The Walnut digital signature algorithm relies heavily on a group action called *E-Multiplication*. To define this group action we need the colored Burau Representation (see, for example, Anshel et al. [2]) which is a homomorphism from the braid group B_N to the *colored Burau group* $GL_N(\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}]) \rtimes S_N$. This group is defined as a semidirect product, by letting the symmetric group S_N act on $GL_N(\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}])$ by permuting the variables t_i. More concretely, the elements of the colored Burau group are pairs $(\mathbf{A}(t_1, \cdots, t_N), \pi)$ where $\pi \in S_N$ is a permutation and where $\mathbf{A}(t_1, \cdots, t_N)$ is an invertible $N \times N$ matrix whose entries lie in $\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}]$. Multiplication in the colored Burau group is defined by

$$\begin{aligned}(\mathbf{A}(t_1, \cdots, t_N), \pi) \cdot (\mathbf{B}(t_1, \cdots, t_N), \tau) &:= (\mathbf{A}(t_1, \cdots, t_N) \cdot \pi(\mathbf{B}(t_1, \cdots, t_N)), \pi\tau) \\ &= (\mathbf{A}(t_1, \cdots, t_N) \cdot \mathbf{B}(t_{\pi(1)}, \cdots, t_{\pi(N)}), \pi\tau).\end{aligned}$$

The *colored Burau representation* $CB : B_N \to GL_N(\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}]) \rtimes S_N$ is defined at each Artin generator as $CB(b_i) = (CBM(i), \sigma(b_i))$, where $CBM(i)$ is a matrix and $\sigma(b_i)$ is a permutation, defined as follows. The permutation $\sigma(b_i)$ is the transposition $(i\ i+1)$. We define $CBM(b_1)$, the colored Burau matrix of b_1, as

$$CBM(b_1) = \left(\begin{array}{cc|c} -t_1 & 1 & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & \mathbb{1}_{N-2} \end{array}\right),$$

where $\mathbb{1}_{N-2}$ is the $(N-2)\times(N-2)$ identity matrix. For $i>1$ the colored Burau matrix of b_i is defined as

$$CBM(b_i) = \left(\begin{array}{c|ccc|c} \mathbb{1}_{i-2} & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 \\ 0 & t_i & -t_i & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & \mathbb{1}_{N-i-1} \end{array}\right).$$

This definition of $CB(b_i)$ is compatible with the relations of the braid group, so it can be extended to define a homomorphism on the entire group B_N. For a braid b, the matrix component of $CB(b)$ is called the *colored Burau matrix* of b and is denoted by $CBM(b)$, the permutation component of $CB(b)$ is simply equal to $\sigma(b)$. This implies that pure braids are mapped into the subgroup $GL_N(\mathbb{Z}[t_1^{\pm 1},\cdots,t_N^{\pm 1}]) \subset GL_N(\mathbb{Z}[t_1^{\pm 1},\cdots,t_N^{\pm 1}]) \rtimes S_N$.

Now we fix a finite field $\mathbb{F}_q$, and for any integer k with $1<k\leq N$ we define A_k to be the group of invertible N-by-N matrices of the form

$$A_k = \left\{ \begin{pmatrix} X & Y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \mathbb{1}_{N-k} \end{pmatrix} | X \in GL_{k-1}(\mathbb{F}_q), Y \in \mathbb{F}_q^{k-1} \right\}.$$

Given a list $T=[\tau_1,\cdots,\tau_N]$ of N values in a finite field we can define an entry-wise evaluation map $GL_N(\mathbb{Z}[t_1^{\pm 1},\cdots,t_N^{\pm 1}]) \to GL_N(\mathbb{F}_q)$. The evaluation of a matrix $\mathbf{M}(t_1,\cdots,t_n)$ at T is denoted by

$$M\downarrow_T := M(\tau_1,\cdots,\tau_N).$$

For a list T containing N non-zero finite field elements, we can now define a right group action, called E-Multiplication and denoted by $\star$, of the braid group B_N on the set $A_N\times S_N$. A braid b acts on the first component of the pair (M,π) by multiplying from the right with a matrix obtained from the colored Burau matrix of b by permuting the variables t_i using π and then evaluating at T. The second component of the acvtion is obtained by multiplying on the right by $\sigma(b)$. Written out symbolically, this is

$$(M,\pi)\star b := (M\cdot\pi(CBM(b))\downarrow_T, \pi\sigma(b)).$$

The fact that this defines a group action follows from the fact that the colored Burau representation is a homomorphism of groups. In practice, when calculating $(M,\pi)\star b$, the action is calculated one Artin generator at a time (see Alg. 1). Given the sparsity of the colored Burau matrices $CBM(b_i)$, acting with an Artin generator requires only a few column operations on M and one swap on π, so this is very efficient. This action was first introduced in [4], where it was used to build a key agreement protocol. More recently, E-Multiplication has been used as the basic building block for a cryptographic hash function [7] and the Walnut digital signature scheme [5].

Algorithm E-Multiplication

input: (M, π) — a pair in $A_N \times S_N$ to act on
s — a braid to act with
$T = \{\tau_1, \cdots, \tau_N\}$ — a list of T-values
output: (M, π) — the resulting pair

while $|s| > 0$ **do**
  $b_i^{\pm 1} || s \leftarrow s$ ▷ split the first generator $b_i^{\pm 1}$ from the rest of s
  $N \leftarrow CBM(b_i)^{\pm 1}$ ▷ The CB Matrix of b_i, inverted if necessary.
  $N \leftarrow N(\tau_{\pi(i)})$ ▷ Evaluate in $\tau_{\pi(i)}$
  $M \leftarrow M \cdot N$
  $\pi \leftarrow \pi \circ \sigma(b_i)$
end while
return (M, π)

Alg. 1. The algorithm for computing the E-Multiplication action.

By letting B_N act on $(\mathbb{1}_N, e) \in A_N \times S_N$ we define a map $\mathcal{P}$

$$\mathcal{P} : B_N \to A_N \times S_N : s \mapsto (\mathbb{1}_N, e) \star s\,.$$

When restricted to the subgroup of pure braids P_N, the second component of $\mathcal{P}$ always maps to the identity permutation, so we can think of it as a map $\mathcal{P}|_{P_N} : P_N \to A_N$. The map $\mathcal{P}|_{P_N}$ is actually a homomorphism because it is the composition of the colored Burau representation $CB : P_N \to GL_N(\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}])$ and the evaluation homomorphism $|_T : GL_N(\mathbb{Z}[t_1^{\pm 1}, \cdots, t_N^{\pm 1}]) \to A_N$. Moreover, if we further restrict $\mathcal{P}$ to the subgroup P_k of pure braids where only the first k strands cross over each other, i.e. the intersection of P_N with the subgroup generated by $b_1, \cdots, b_{k-1}$, the homomorphism $\mathcal{P}|_{P_k} : P_k \to A_k$ maps into the subgroup A_k. This fact will be exploited in the attack of Sect. 5.

2.4 The Walnut signature scheme

We now introduce the Walnut signature scheme, which is the subject of our cryptanalysis. Before we describe the key generation, signing and verification algorithms (Alg. 2, 4 and 5) in detail we will summarize the scheme very briefly: the secret key consists of two braids s_1, s_2 and the public key is $(M_1, \pi_1) = \mathcal{P}(s_1)$ and $M_2 = \mathsf{mat}(\mathcal{P}(s_2))$, the matrix component of $\mathcal{P}(s_2)$. To sign or verify a document d it is hashed and encoded as a pure braid $E(d)$ with an encoding mechanism E. The Walnut design [5] defines a braid sig to be a valid signature for the document d if and only if the verification equation

$$\mathsf{mat}(\mathcal{P}(s_1) \star \mathsf{sig}) = \mathsf{mat}(\mathcal{P}(E(d))) \cdot M_2 \tag{1}$$

is satisfied. However, this equation is equal to the matrix component of

$$\mathcal{P}(s_1) \star \mathsf{sig} = \mathcal{P}(E(d)) \star s_2 \,, \tag{2}$$

and the permutation component of equation (2) is also satisfied by all the legitimately produced signatures. In this document we define a valid signature as a braid sig that satisfies the stronger verification equation (2). It is clear that $\mathsf{sig} = s_1^{-1} E(d) s_2$ would be a valid signature. In order to prevent length-based attacks [16, 19] *cloaking elements*, namely braids that do not affect E-Multiplication, are inserted into the signature and the braids are put through a rewriting algorithm so that s_1 and s_2 cannot easily be extracted from the signature.

Parameters. The scheme is parametrized by:

- The order of the braid group that is being used N.
- The size q of a finite field $\mathbb{F}_q$.
- A rewriting algorithm $\mathcal{R} : B_N \to B_N$.
- L and l, the length of certain random braid words.
- A hash function $\mathcal{H}$.

Table 1. The Walnut parameter sets submitted to the NIST Post Quantum Cryptography project, and the corresponding public key and signature sizes.

claimed security level	128-bit	256-bit
N	8	8
q	2^5	2^8
L	15	30
l	132	287
$\mathcal{H}$	SHA2-256	SHA2-512
$\lvert\mathsf{pk}\rvert$	83 Bytes	128 Bytes
$\lvert\mathsf{sig}\rvert$[3]	±646 Bytes	±1248 Bytes

Key generation. The private key consists of two randomly chosen braids $s_1, s_2 \in B_N$ of length l. The braids are chosen such that their underlying permutations $\sigma(s_1)$ and $\sigma(s_2)$ are distinct and not equal to the identity permutation e. The public key contains a list $T = \{\tau_1 = 1, \tau_2 = 1, \tau_3, \cdots, \tau_n\} \in \mathbb{F}_q^N$ of N elements of the finite field $\mathbb{F}_q$ such that the first two elements are equal to 1, and such that the remaining values are non-zero and different from 1. The public key also contains $\mathcal{P}(s_1)$ and the matrix component of $\mathcal{P}(s_2)$.

[3] Signatures have variable length. The reported signature size is an average, using the BKL + Dehornoy rewriting method.

Algorithm GenerateKeys

input: random bits to generate s_1, s_2 and τ_i
output: pk — a public key
sk — a corresponding secret key

1: $s_1, s_2 \leftarrow$ a randomly chosen braid words of length l.
2: $\tau_1, \tau_2 \leftarrow 1$
3: **for** i from 3 to N **do**
4: $\quad \tau_i \leftarrow$ a randomly chosen field element, not equal to 0 or 1
5: **end for**
6: $T \leftarrow \{\tau_1, \cdots, \tau_N\}$
7: $(M_1, \pi_1) \leftarrow \mathcal{P}(s_1)$
8: $(M_2, \pi_2) \leftarrow \mathcal{P}(s_2)$
9: **return** $\mathsf{pk} = (T, M_1, M_2, \pi_1)$ and $\mathsf{sk} = (s_1, s_2)$

Alg. 2. The Walnut key pair generation algorithm

Encoding a document. In order to sign a document d or verify a signature the document is converted to a pure braid $E(d) \in P_N$. This conversion consists of two stages. First, a hash digest of d is computed with a standard hash function (SHA2-256 or SHA2-512), then this hash is converted to a braid. To make the second conversion 4 pure braids g_1, g_2, g_3, g_4 are fixed such that they generate a free subgroup of P_N. The Walnut specification document [8] defines

$$\begin{aligned} g_1 &= b_N b_{N-1} \cdots b_2 \cdot b_1^2 \cdot b_2^{-1} \cdots b_{N-1}^{-1} b_N^{-1} \\ g_2 &= b_N b_{N-1} \cdots b_4 \cdot b_3^2 \cdot b_4^{-1} \cdots b_{N-1}^{-1} b_N^{-1} \\ g_3 &= b_N b_{N-1} \cdots b_6 \cdot b_5^2 \cdot b_6^{-1} \cdots b_{N-1}^{-1} b_N^{-1} \\ g_4 &= b_N b_{N-1} \cdots b_8 \cdot b_7^2 \cdot b_8^{-1} \cdots b_{N-1}^{-1} b_N^{-1} \,. \end{aligned}$$

The encoding process starts from the trivial braid. Two bits are taken from the hash digest to choose one g_i of the 4 generators, and the next two bits of the digest define an exponent $e \in \{1, 2, 3, 4\}$. Then g_i^e is appended to the braid, and four bits are removed from the digest. This is repeated until the entire hash output is consumed.

Signing algorithm. The signing algorithm produces a signature which is a braid word of the form

$$\mathsf{sig}' = v_1 \cdot s_1^{-1} \cdot v \cdot E(d) \cdot s_2 \cdot v_2 \,,$$

Algorithm EncodeDocument

input: A document d
output: b — a pure braid

$\mathsf{h} \leftarrow \mathcal{H}(d)$
$b \leftarrow e$
for a from 0 to $|\mathsf{h}|/4 - 1$ **do**
  $\mathsf{i} \leftarrow \mathsf{h}[4a : 4a + 1]$ ▷ Select index
  $\mathsf{e} \leftarrow \mathsf{h}[4a + 2 : 4a + 3]+1$ ▷ Select exponent
  $b \leftarrow b \cdot g_{\mathsf{i}}^{\mathsf{e}}$
end for
return b

Alg. 3. The document encoding mechanism.

where v_1, v and v_2 are so called cloaking elements, which are braids in the stabilizer of $\mathcal{P}(s_1), (\mathbb{1}_N, e)$ and $\mathcal{P}(E(d)s_2)$ respectively. Therefore we have

$$\begin{aligned}(\mathbb{1}_N, e) \star s_1 \cdot \mathsf{sig}' &= \mathcal{P}(s_1) \star s_1^{-1} \cdot v \cdot E(d) \cdot s_2 \cdot v_2 \\ &= (\mathbb{1}_N, e) \star v \cdot E(d) \cdot s_2 \cdot v_2 \\ &= \mathcal{P}(E(d)s_2) \cdot v_2 \\ &= (\mathbb{1}_N, e) \star E(d) \cdot s_2\,,\end{aligned}$$

so sig' is a valid signature. To hide the secret key s_1 and s_2 which are substrings of sig' one of three proposed rewriting algorithms (BKL + Dehornoy, Stochastic + Dehornoy or Stochastic) is used to produce a different braid word sig which represents the same braid as sig'. The various rewriting algorithms differ in performance and in the length of the signatures that are produced.

The cloaking elements are generated using the following lemma.

Lemma 1. *Suppose that $\tau_1 = \tau_2 = 1$. Take any pair $(M, \pi) \in A_N \times S_N$, an Artin generator b_i, and any braid w such that*

$$\pi \circ \sigma(w)(i) = 1 \qquad \textit{and} \qquad \pi \circ \sigma(w)(i + 1) = 2\,.$$

Then the braid $v = w \cdot b_i^2 \cdot w^{-1}$ is in the stabilizer of (M, π).

To produce a cloaking element for $\mathcal{P}(s_1), (\mathbb{1}_N, e)$ or $\mathcal{P}(E(d)s_2)$ we first pick a random integer i such that $1 < i < N$, then we choose a random braid w satisfying the conditions of Lemma 1 and we set $v = wb_i^2w^{-1}$. For the details of how w is chosen (which depends on the parameter L) and the details on how the various rewriting algorithms work we refer to the WalnutDSA NIST submission [8].

Algorithm Sign

input: d — a document to sign
$\mathsf{sk} = (s_1, s_2)$ — a secret key
output: sig — a signature for document d

1: $v_1 \leftarrow \text{GetCloakingElement}(\sigma(s_1))$
2: $v \leftarrow \text{GetCloakingElement}(e)$
3: $v_2 \leftarrow \text{GetCloakingElement}(\sigma(s_2))$
4: $E_d \leftarrow \text{EncodeDocument}(d)$
5: $\mathsf{sig}' \leftarrow v_1 \cdot s_1^{-1} \cdot v \cdot E_d \cdot s_2 \cdot v_2$
6: $\mathsf{sig} \leftarrow \mathcal{R}(\mathsf{sig}')$
7: **return** sig

Alg. 4. The Walnut signature generation algorithm

Verification Algorithm. Given a document d, a public key $\mathsf{pk} = (T, M_1, M_2, \pi_1)$ and a signature sig. The verification algorithm simply calculates the encoding of the message $E(d)$ and the matrix components of $(M_1, \pi_1) \star \mathsf{sig}$ and $\mathcal{P}(E(d))$. It then accepts the signature if the computed matrices satisfy the equation

$$\mathsf{mat}((M_1, \pi_1) \star \mathsf{sig}) = \mathsf{mat}(\mathcal{P}(E(d))) \cdot M_2 \,.$$

Algorithm Verify

input: d — a document
$\mathsf{pk} = (T, M_1, M_2, \pi_1)$ — a secret key
sig — a signature
output: **True** if sig is a valid signature for d, **False** otherwise

1: $E_d \leftarrow \text{EncodeDocument}(d)$
2: LHS $\leftarrow \mathsf{mat}(\text{ E-Multiplication}((M_1, \pi_1), \mathsf{sig}, T))$
3: RHS $\leftarrow \mathsf{mat}(\text{ E-Multiplication}((\mathbb{1}_N, e), E_d, T)) \cdot M_2$
4: **if** LHS equals RHS **then**
5: | **return True**
6: **end if**
7: **return False**

Alg. 5. The Walnut signature verification algorithm

3 A factorization attack

This section describes an adaptation of the factorization attack of Hart et al. [14] on an earlier version of Walnut [6]. This earlier version is a special case of the

newer construction where the two secret braids s_1 and s_2 are equal. This means that the secret key essentially consists of only a single braid s, and that the public key is a single matrix-permutation pair $(M, \pi) = \mathcal{P}(s)$. The signing and verification algorithms of the earlier version are the same as the algorithms described in the previous section after substituting s for s_1 and s_2, and substituting M for M_1 and M_2. The attack of Hart et al. exploits the following malleability property:

Theorem 1. (for the earlier version of Walnut with $s_1 = s_2$) *Suppose d, d_1, d_2 are three documents. Let h, h_1, h_2 be the matrix part of $\mathcal{P}(E(d)), \mathcal{P}(E(d_1))$ and $\mathcal{P}(E(d_2))$ respectively. Then we have*

1. *If $h = h_1^{-1}$ and sig_1 is a valid signature for d_1, then sig_1^{-1} is a valid signature for d.*
2. *If $h = h_1 \cdot h_2$ and $\mathsf{sig}_1, \mathsf{sig}_2$ are valid signatures for d_1 and d_2 respectively, then $\mathsf{sig}_1\mathsf{sig}_2$ is a valid signature for d.*

This opens up the following strategy to attack the signature scheme. First we collect a set of valid document-signature pairs (d_i, sig_i) and we let $h_i = \mathsf{mat}(\mathcal{P}(E(d_i)))$. Then, if we want to forge a signature for a document d with $h = \mathsf{mat}(\mathcal{P}(E(d)))$ it suffices to write h as a product $\prod_{j=1}^{k} h_{i_j}^{e_j}$ of the h_i. Once we have this, a valid signature for d is given by $\prod_{j=1}^{k} \mathsf{sig}_{i_j}^{e_j}$. This reduces breaking the signature scheme to breaking the factorization problem in A_N:

Factorization problem in a group G. Given a list of elements $g_1, \cdots, g_k$ that generate the group G and a target element g, write the target g as a (preferably short) product of the g_i and their inverses.

The paper of Hart et al [14] proposes an algorithm to solve the factorization problem in A_N, exploiting a chain of subgroups. This allows them to forge signatures in minutes, but the factorizations that are found by the algorithm are very long, so this results in very long signatures. The forged signatures are many orders of magnitude longer than legitimate signatures, so the Walnut scheme can be saved by imposing an upper limit to the length of the signatures.

The Walnut signature scheme was adapted to destroy the malleability property of Theorem 1. In the remainder of this section we prove that an adapted version of the maleability property still holds for the new WalnutDSA scheme and we show how the property can be used to reduce breaking Walnut to solving the factorization problem in A_N, which can be solved with the techniques of [14].

3.1 Signature malleability of Walnut

Walnut has the following malleability property, which is a generalization of the property discovered by Hart et al. (Theorem 1).

Theorem 2. *Suppose d, d_1, d_2 are three documents. Let h, h_1, h_2 be the matrix part of $\mathcal{P}(E(d)), \mathcal{P}(E(d_1))$ and $\mathcal{P}(E(d_2))$ respectively. Let $s_1, s_2, s_3 \in B_N$ be three braids. Then*

1. *If $h = h_1^{-1}$ and sig_1 is a valid signature for d_1 under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_2))$, then sig_1^{-1} is a valid signature for d under the public key $(\mathcal{P}(s_2), \mathcal{P}(s_1))$.*
2. *If $h = h_1 \cdot h_2$ and $\mathsf{sig}_1, \mathsf{sig}_2$ are valid signatures for d_1 and d_2 under the public keys $(\mathcal{P}(s_1), \mathcal{P}(s_2))$ and $(\mathcal{P}(s_2), \mathcal{P}(s_3))$ respectively, then $\mathsf{sig}_1 \cdot \mathsf{sig}_2$ is a valid signature for d under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_3))$.*

Proof. We start by proving 1. Since sig_1 is a valid signature for d_1 we have

$$\mathcal{P}(s_1) \star \mathsf{sig}_1 = \mathcal{P}(E(d_1)) \star s_2.$$

Acting on this by sig_1^{-1} and using the definition of $\mathcal{P}$ we get

$$(\mathbb{1}_N, e) \star s_1 = (h_1, e) \star s_2 \cdot \mathsf{sig}_1^{-1},$$

where we have used the fact that $E(d_1)$ is a pure braid. Multiplying the matrix part of this equality by h_1^{-1} from the left (multiplying on the left by a matrix commutes with $\star$), we get

$$(h_1^{-1}, e) \star s_1 = (\mathbb{1}_N, e) \star s_2 \cdot \mathsf{sig}_1^{-1},$$

or equivalently

$$\mathcal{P}(E(d)) \star s_1 = \mathcal{P}(s_2) \star \mathsf{sig}_1^{-1},$$

which shows that sig_1^{-1} is a valid signature for d for the public key $(\mathcal{P}(s_2), \mathcal{P}(s_1))$.

To prove 2 we start by acting with sig_2 on the verification equation for sig_1 to get

$$\begin{aligned}\mathcal{P}(s_1) \star \mathsf{sig}_1 \cdot \mathsf{sig}_2 &= \mathcal{P}(E(d_1)) \star s_2 \cdot \mathsf{sig}_2 \\ &= (h_1 \cdot CBM(s_2)_{\downarrow T} \cdot^{\sigma(s_2)} (CBM(\mathsf{sig}_2))_{\downarrow T}, \sigma(s_2) \circ \sigma(\mathsf{sig}_2)).\end{aligned}$$

Using the fact that sig_2 is a valid signature for d_2 under the public key $(\mathcal{P}(s_2), \mathcal{P}(s_3))$, we see that

$$\begin{aligned}\mathcal{P}(s_1) \star \mathsf{sig}_1 \cdot \mathsf{sig}_2 &= (h_1 \cdot h_2 \cdot CBM(s_3)_{\downarrow T}, \sigma(s_3)) \\ &= (h_1 \cdot h_2, e) \star s_3 \\ &= \mathcal{P}(E(d)) \star s_3,\end{aligned}$$

which shows that $\mathsf{sig}_1 \cdot \mathsf{sig}_2$ is a valid signature for d under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_3))$.

3.2 The factorization attack

Given an oracle $\mathcal{O}_f$ (which can be instantiated by the algorithm of [14]) that solves the factorization for the group A_N, we can now break Walnut as follows. Suppose we want to forge a signature for a document d under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_2))$. Let h be the matrix part of $\mathcal{P}(E(d))$. We start by collecting a number of document-signature pairs $(d_1, \mathsf{sig}_1), \cdots, (d_k, \mathsf{sig}_k)$ that are valid under the same public key, and we compute the matrix part h_i of each pair $\mathcal{P}(E(d_i))$. Now it suffices to find a factorization $h = h_{i_1} \cdot h_{i_2}^{-1} \cdot h_{i_3} \cdots h_{i_{m-1}}^{-1} \cdot h_{i_m}$ whose factors have powers that alternate between 1 and -1. Indeed, combining properties of Theorem 2 we see that $\mathsf{sig}_{i_1} \cdot \mathsf{sig}_{i_2}^{-1}$ is a valid signature for any document d' such that $\mathsf{mat}(\mathcal{P}(E(d'))) = h_{i_1} \cdot h_{i_2}^{-1}$ under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_1))$. Adding an extra factor, we get that $\mathsf{sig}_{i_1} \cdot \mathsf{sig}_{i_2}^{-1} \cdot \mathsf{sig}_{i_3}$ is a valid signature for an appropriate document under the public key $(\mathcal{P}(s_1), \mathcal{P}(s_2))$. Continuing the same argument for the odd number m of factors of the product we get that $\mathsf{sig}_{i_1} \cdot \mathsf{sig}_{i_2}^{-1} \cdot \mathsf{sig}_{i_3} \cdots \mathsf{sig}_{i_{m-1}}^{-1} \cdot \mathsf{sig}_{i_m}$ is a valid document for d under the desired public key $(\mathcal{P}(s_1), \mathcal{P}(s_2))$.

We can use the oracle $\mathcal{O}_f$ to find the factorization $h = h_{i_1} \cdot h_{i_2}^{-1} \cdot h_{i_3} \cdots h_{i_{m-1}}^{-1} \cdot h_{i_m}$. We construct the list of generators

$$\mathsf{gens} = \{h_i \cdot h_j^{-1} \quad | \quad i \neq j \in \{1, \cdots, k\}\}$$

and call the oracle $\mathcal{O}_f$ to obtain a factorization for $h \cdot h_1^{-1}$ with factors in this set of generators. Appending the factor h_1 to the resulting factorization we then get a factorization of h of the desired form.

3.3 Implications and countermeasures

The factorization algorithm of [14] has a time complexity of $O\left(q^{\frac{N-1}{2}}\right)$ and for the 128 bit security parameters of Walnut (i.e. $N = 8, q = 2^5$) the algorithm finds a factorization in minutes. However, these factorizations contain roughly 2^{25} factors, so the forged signatures are the concatenation of roughly 2^{25} legitimate signatures. This implies that the forged signatures are many orders of magnitude longer than legitimate signatures and so they can be detected easily by the verifier. To protect against this attack it suffices to impose a limit on the length of signatures. Interestingly, when the WalnutDSA scheme was updated to counter the attack of [14], no such upper limit was included in the design. Our adaptation of the attack shows that this limit is necessary for the security of the scheme, because long forgeries can be produced in a matter of minutes.

The implementation submitted to the NIST PQC standardization project implicitly imposes such an upper limit by specifying that the length of the signature (measured by the number of Artin generators) be encoded by two bytes.

This effectively limits the signature braids to be at most 2^{16} Artin generators long. Therefore the attack cannot be used to break the NIST implementation of WalnutDSA.

4 A collision search attack

From the verification equation

$$\mathcal{P}(s_1) \star \mathsf{sig} = \mathcal{P}(E(d)) \star s_2$$

it is clear that the only dependence on the document d is through the encoding mechanism E and the mapping $\mathcal{P}$. This implies that if d_1 and d_2 are two documents such that $\mathcal{P}(E(d_1) = \mathcal{P}(E(d_2))$, then any signature that is valid for d_1 is automatically valid for d_2 and vice versa. Therefore breaking EUF-CMA security reduces to finding such a pair of documents. Once an attacker has found two such documents he can ask the signing oracle to produce a signature sig for d_1, and return (sig, d_2) to win the EUF-CMA game. Since the first step of the encoding function E is the application of a cryptographically secure hash function to the document d we cannot reasonably expect to have a more efficient way of finding collisions than with a generic collision search. A generic collision search requires roughly $|\mathcal{P}(E(\{0,1\}^*))|^{1/2}$ evaluations of $\mathcal{P} \circ E$. In the rest of this section we give an upper bound for this quantity and we demonstrate with computer experiments that a collision attack is practical.

4.1 Sizes of orbits of E-multiplication

To estimate the time complexity of the collision search attack we need to find the size of $\mathcal{P}(E(\{0,1\}^*))$. Without much motivation the designers of WalnutDSA claim that $q^{N(N-3)}N!$ is a conservative lower bound on the number values that $\mathcal{P}$ can take [5]. For 128-bit and 256-bit security parameters this number is roughly 2^{216} and 2^{336} respectively, which means that finding a collision should require roughly 2^{108} and 2^{168} evaluations of $\mathcal{P} \circ E$. Note that this is already significantly less than the claimed security levels. Moreover, an elementary analysis will reveal that this "conservative lower bound" is actually much larger than the true value of $|\mathcal{P}(B_N)|$. Even worse, when $\mathcal{P}$ is restricted to the set of braids that can be produced by the encoding mechanism E, the number of values that can be reached is much smaller still.

We know that $\mathcal{P}$, when restricted to the subgroup of pure braids, is a homomorphism from P_N to A_N. This implies that the full twist braid Δ^2 (see Sect. 2.2) which generates the center of P_N is mapped to a matrix in the center of $\mathcal{P}(P_N)$. It can be verified that the only matrix in the center of A_N is the

identity matrix, but for a randomly chosen set of T-values $\mathcal{P}(\Delta^2)$ is typically not the identity matrix. This means that $\mathcal{P}(A_N)$ sits inside the centralizer of $\mathcal{P}(\Delta^2)$, which is typically a proper subspace of $\langle A_N \rangle$. This begs the question of what the dimension of $\langle \mathcal{P}(P_N) \rangle$ is. From computer experiments we can conclude that for randomly chosen T-values this is equal to the dimension of the centralizer of $\mathcal{P}(\Delta^2)$, which is equal to $(N-1)^2+1$ (since $\mathcal{P}(\Delta^2)$ has one eigenspace of dimension $N-1$ and one of dimension 1). However, if we impose the extra condition that the first two T-values are equal to one, $\mathcal{P}(P_N)$ is contained in an affine subspace of dimension $(N-2)^2+1$, so $|\mathcal{P}(P_N)|$ is at most $q^{(N-2)^2+1}$. Our computer experiments suggest that this upper bound is reasonably tight, and so we estimate $|\mathcal{P}(P_N)| \approx q^{(N-2)^2+1}$. Since P_N is a subgroup of B_N of index $N!$ we have $|\mathcal{P}(B_N)| < q^{(N-2)^2+1} N!$. Note that this upper bound is strictly lower than the lower bound which was claimed by the designers of Walnut.

Any braid output by the encoding mechanism E is a product of the generators g_1, g_2, g_3, g_4. From computer experiments we conclude that when applying $\mathcal{P}$ to braids of this form we end up with matrices in an affine subspace of surprisingly low dimension. We found that they live in a subspace of dimension 13, independent of the values of q or N (provided that $N^2 > 13$). This means that $|\mathcal{P}(E(\{0,1\}^*))|$ is at most q^{13}, and that finding a collision cannot take much more than $q^{13/2}$ evaluations of $\mathcal{P} \circ E$. For 128-bit security parameters this number is as low as $2^{32.5}$, and for 256-bit security parameters this is 2^{52}.

4.2 Implementation

We implemented the generic collision finding algorithm of van Oorschot and Wiener [24] (briefly explained in Sect. 2.1) and used it to find collisions for the function $g \circ \mathcal{P} \circ E$, where g is a function that takes the ouput of $\mathcal{P}$, and converts it to some plausible document d. Even though the method is completely generic, it is still efficient enough to find colliding documents in practice. It took approximately $2^{32.2}$ evaluations of f (which agrees very well with the expected value of $2^{32.5}$) or one hour on a standard desktop PC to find the following pair of colliding documents.

d_1 =`"I would like to receive 7181666883746416503 free`
`samples of delicious cookies."`

d_2 =`"I pledge to donate 35195330520899884 69 USD to Ward Beullens."`

The documents are cunningly crafted such that a victim would be eager to sign the first document with his/her secret key. However, by producing a signature for this document, the victim would unknowingly also sign the second document.

4.3 Implications and countermeasures

This practical attack shows that the Walnut signature scheme should not be used with the parameters that are submitted to the NIST PQC project.

Increasing q to raise $q^{13/2}$ to the required security level would lead to $q = 2^{20}$ and $q = 2^{40}$ for 128-bit and 256-bit security parameters respectively. For 256-bit security parameters this would increase the size of the public key by a factor of 5 and we estimate that this would slow down the verification algorithm by a factor of 25. A better approach would be to change the encoding algorithm to output pure braids that are not restricted to the subgroup generated by g_1, g_2, g_3, g_4 (or any other proper subgroup). Since $\mathcal{P}(P_N)$ is contained in an affine subspace of dimension $(N-2)^2+1$, this would lead to an upper bound on the complexity of the attack of $\sqrt{q^{(N-2)^2+1}}$ evaluations of $\mathcal{P} \circ E$. We would then only need a slight increase in the parameters. For example, 256 bits of security would be achieved (against this attack) by the parameters $q = 2^8$ and $N = 10$, leading to an increase of the key size of roughly 50%, the signature size by at least 25% and an expected slowdown of the verification algorithm by a factor of 2.

5 Reversing E-multiplication

A fundamental hard problem underlying the Walnut signature scheme is the "Reversing E-Multiplication" (REM) problem. This problem asks, given a pair $(M, \sigma) \in A_N \times S_N$, such that $(M, \sigma) = (\mathbb{1}_N, e) \star s$ for some braid $s \in B_N$, to find a braid $s' \in B_N$ such that $(\mathbb{1}_N, e) \star s' = (M, \sigma)$. In other words, the problem is to break the one-wayness of the function

$$\mathcal{P} : B_N \to A_N \times S_N : s \mapsto (\mathbb{1}_N, e) \star s.$$

The secret key in Walnut consists of two braids $s_1, s_2 \in B_N$. The corresponding public key is $\mathcal{P}(s_1)$ and the matrix part of $\mathcal{P}(s_2)$. The fact that the permutation part of $\mathcal{P}(s_2)$ is not available to the attacker is not a problem, because given a single signature sig which is valid for any message (which might be unknown to the attacker), the attacker can deduce the permutation of s_2 from the permutation component of the verification equation (2)

$$\sigma(s_1) \circ \sigma(\mathsf{sig}) = \sigma(s_2).$$

After solving the REM problem to get s_1', s_2' such that $\mathcal{P}(s_1) = \mathcal{P}(s_1')$ and $\mathcal{P}(s_2) = \mathcal{P}(s_2')$, an attacker can use the pair (s_1', s_2') as a secret key to sign any message. Alternatively, instead of solving two instances of the REM problem to obtain an equivalent secret key, it is also possible to solve a single instance of the REM problem to obtain a signature for a document which can be chosen freely.

In this section we give an algorithm that solves the REM problem in practice for the parameters that are proposed for Walnut. First, we describe a generic birthday attack that can reverse any group action. Then, we introduce an algorithm that exploits the subgroup structure of B_N and is much more efficient.

5.1 Birthday attack

A brute force attack would repeatedly pick a random $s \in B_N$, compute $(\mathbb{1}_N, e) \star s$ and check if this is equal to the target (M, σ). This attack would take $O(|\mathcal{P}(B_N)|)$ attempts, where $|\mathcal{P}(B_N)|$ is the size of the orbit of $(\mathbb{1}_N, e)$. A more efficient approach is to look for $s_1, s_2 \in B_N$ such that

$$(M, \sigma) \star s_1 = (\mathbb{1}_N, e) \star s_2 \,.$$

If such s_1 and s_2 are found, the solution to the REM problem is given by $s_2 s_1^{-1}$. A naive way of finding s_1 and s_2 is to compute a large table containing $\sqrt{|\mathcal{P}(B_N)|}$ values of s_1 and the corresponding values of $(M, \sigma) \star s_1$ and check for random values of s_2 whether $(\mathbb{1}_N, e) \star s_2$ lies in this table. This method takes $O(\sqrt{|\mathcal{P}(B_N)|})$ E-Multiplications, but requires a lot of memory. The problem can be reduced to collision finding for a function $f : \mathcal{P}(B_N) \to \mathcal{P}(B_N)$. Then, distinguished point methods (see Sect. 2.1) can solve the REM problem with the same time complexity as the naive approach but with constant memory complexity. Concretely, suppose $\mathsf{b} : \mathcal{P}(B_N) \to \{0, 1\}$ and $\mathsf{s} : \mathcal{P}(B_N) \to B_N$ are hash functions that take a matrix and a permutation from the orbit of $(\mathbb{1}_N, e)$ as input, and output a bit or a braid respectively. Then we can define

$$f(x) = \begin{cases} (\mathbb{1}_N, e) \star \mathsf{s}(x) \text{ if } \mathsf{b}(x) = 0, \\ (M, \sigma) \star \mathsf{s}(x) \text{ if } \mathsf{b}(x) = 1. \end{cases}$$

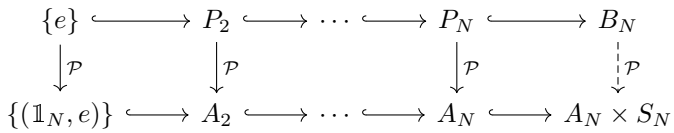
If s outputs sufficiently long braids such that $\mathcal{P}(\mathsf{s}(x))$ is distributed uniformly in the orbit of $(\mathbb{1}_N, e)$, then the distinguished point method will yield collisions $f(x_1) = f(x_2)$ such that $\mathsf{b}(x_1) \neq \mathsf{b}(x_2)$ with probability $1/2$. Once such a collision is found, a solution to the REM problem is given by $\mathsf{s}(x_1)\mathsf{s}(x_2)^{-1}$ or $\mathsf{s}(x_2)\mathsf{s}(x_1)^{-1}$ when $\mathsf{b}(x_1)$ is 0 or 1 respectively. For the security parameters aiming for 128 bits of security, the size of the orbit $\mathcal{P}(B_N)$ is bounded by 2^{200} (see Sect. 4.1), so the number of E-multiplications required to solve REM is not much more than 2^{100}, considerably less than 2^{128} but still far from practical. For the 256 bit security parameters the number of E-multiplications is not much more than 2^{157}.

5.2 Subgroup chain attack

We next propose a practical method for solving the REM problem that improves the attack above by exploiting the following chain of subgroups of B_N:

$$\{e\} = P_1 \subset P_2 \subset \cdots \subset P_N \subset B_N.$$

The map $\mathcal{P}$ sends a braid to an element of $A_N \times S_N$ and, when restricted to P_i it is a homomorphism to A_i (see Sect. 2.3). Therefore we have the following commuting diagram:



The meet-in-the-middle attacks in the previous subsection attempt to find a braid s such that $(M,\sigma)\star s = (\mathbb{1}_N, e)$ in one step. Given this subgroup structure, it is more efficient to solve REM in several steps. The first step is to find a braid $s' \in B_N$ such that $(M,\sigma)\star s' = (M', e) \in A_N$. This is trivial because any $s' \in B_N$ whose underlying permutation is σ^{-1} will do the job. The next step is to find a pure braid $s_N \in P_N$ such that $(M', e)\star s_N \in A_{N-1}$. Then, one continues iteratively to find $s_i \in P_i$ such that $(M,\sigma)\star s' s_N \cdots s_i \in A_{i-1}$. After the last step we have found $s' s_N \cdots s_2$ such that $(M,\sigma)\star s' s_N \cdots s_2 = (\mathbb{1}_N, e)$, so $(s' s_n \cdots s_2)^{-1}$ is a solution to the REM problem.

One caveat when using this method is that, a priori, it is possible to get stuck. After each step, we get a new target $(M,\sigma)\star s' s_N \cdots s_i$ which is sampled randomly from $\mathcal{P}(P_i) \cap A_{i-1}$. However, from that point on, we will only act on this taget with pure braids from P_{i-1}. This means that if the new target is not in $\mathcal{P}(P_{i-1})$ we will not be able to complete the attack. If we assume for each i that

$$\mathcal{P}(P_i) \cap A_{i-1} = \mathcal{P}(P_{i-1})\,,$$

then the attack is guaranteed to work. In practice, this assumption seems to hold with large probability for the parameter sets that are proposed, because the algorithm works without having to backtrack. We encounter this problem when instantiating the Walnut scheme with a smaller finite field such as $\mathbb{F}_5$. Then, it occurs for a small but noticeable fraction of the choices of T-values that for some small i all the generators of $\mathcal{P}(P_{i-1})$ have determinant 1 or -1, while the subgroup $\mathcal{P}(P_i) \cap A_{i-1}$ contains matrices with any determinant. This problem is unlikely to occur in large finite fields and with large i, because then there are many generators of $\mathcal{P}(P_{i-1})$ that all have to map to a matrix with determinant ± 1.

Each step can be solved with a collision search in the space $A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1}$ of cosets of A_{i-1} in $A_{i-1}\mathcal{P}(P_i)$. Let $\mathsf{b} : A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1} \to \{0,1\}$ and $\mathsf{s} : A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1} \to P_i$ be hash functions that take a right coset and output a bit or a pure braid respectively. Then we can define $f : A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1} \to A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1}$ as

$$f(x) = \begin{cases} A_{i-1}\mathcal{P}(\mathsf{s}(x)) & \text{if } \mathsf{b}(x) = 0, \\ A_{i-1}M'\mathcal{P}(s_N \cdots s_{i+1}\mathsf{s}(x)) & \text{if } \mathsf{b}(x) = 1. \end{cases}$$

The distinguished point method can find collisions $f(x_1) = f(x_2)$ at a cost of roughly $\sqrt{|A_{i-1}\mathcal{P}(P_i)\backslash A_{i-1}|}$ E-Multiplications. Under the assumption we made

earlier that $\mathcal{P}(P_i) \cap A_{i-1} = \mathcal{P}(P_{i-1})$ this is equal to $\sqrt{|\mathcal{P}(P_i)|/|\mathcal{P}(P_{i-1})|}$ E-Multiplications.

If we plug the estimate of $|\mathcal{P}(P_i)| \approx q^{(i-2)^2+1}$ from Sect. 4.1 into this formula, we get an estimate of $\sqrt{\frac{q^{(i-2)^2+1}}{q^{(i-3)^2+1}}} = q^{i-5/2}$ E-Multiplications to find s_i. The runtime of the algorithm is dominated by the step that searches for s_N, which is estimated to require $q^{N-5/2}$ E-Multiplications. For 128-bit security parameters this number is $2^{27.5}$ and this agrees very well with our computer experiments. For 256-bit security parameters, the required number of E-Multiplications is estimated to be 2^{44}.

5.3 Representing and manipulating cosets of A_k.

In order to implement the hash functions b and s we need to be able to uniquely represent right cosets with respect to A_k. We give a method to do this efficiently in this subsection. Suppose, $\mathbf{X}, \mathbf{Y}$ are two matrices in A_N, that are in the same right coset of A_{N-1}. That is, there exists a matrix $\mathbf{A} \in A_{N-1}$ such that $\mathbf{AX} = \mathbf{Y}$. If we split up the matrices to make their structure visible we get:

$$\begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{0} \\ \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 & \mathbf{X}_2 \\ \mathbf{X}_3 & \mathbf{X}_4 \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{Y}_1 & \mathbf{Y}_2 \\ \mathbf{Y}_3 & \mathbf{Y}_4 \\ \mathbf{0} & 1 \end{pmatrix}.$$

From this it is obvious that the $(N-1)$-th row of $\mathbf{X}$ and $\mathbf{Y}$ are identical, and that the first $(N-1)$ rows of $\mathbf{X}$ and $\mathbf{Y}$ span the same $(N-1)$-dimensional subspace. It is easily checked that the converse also holds, which implies that the right coset of A_{N-1} that contains a matrix $\mathbf{X} \in A_N$ is completely determined by the $(N-1)$-th row of $\mathbf{X}$ and the subspace spanned by the first $N-1$ rows of $\mathbf{X}$. In turn, this subspace is uniquely represented by the row reduced echelon form of the upper $(N-1)$-by-N submatrix of $\mathbf{X}$, which will be of the form

$$\begin{pmatrix} \mathbb{I}_{N-1} & \mathbf{v} \end{pmatrix}$$

for some $\mathbf{v} \in \mathbb{F}^{N-1}$. Therefore, the coset containing $\mathbf{X}$ is completely determined by the $(N-1)$-th row of $\mathbf{X}$, and the last column of the first $N-1$ rows of X after putting it in row reduced echelon form. More generally, we have the following lemma.

Lemma 2. *A right coset of $A_k \backslash A_{k-1}$ with representative $\mathbf{X} \in A_k$ is completely determined by the pair of vectors $(\mathbf{v}_1, \mathbf{v}_2) \in \mathbb{F}_q^N \times \mathbb{F}_q^{k-1}$, where $\mathbf{v}_1$ is the $(k-1)$-th row of $\mathbf{X}$ and $\mathbf{v}_2$ is the k-th column of the matrix $\mathbf{X}'$, which is obtained from $\mathbf{X}$ by taking the first $k-1$ rows and putting them in row reduced echelon form.*

This lemma gives a method for deciding whether two matrices $\mathbf{X}$ and $\mathbf{Y}$ are in the same coset. One simply computes the pair of vectors for both matrices $\mathbf{X}$ and $\mathbf{Y}$ and checks whether they are equal. To run the algorithm we also need a way to act on cosets by multiplying on the right by matrices. One way to do this is to work with a representative from the coset and carry out a matrix multiplication to get a representative from the next coset. It is more efficient to compute directly with the two-vector representation of the coset. The following lemma gives a way to do this.

Lemma 3. *Suppose* $\mathbf{M}$ *is a matrix in* A_k *for some* k *with* $1 < k \leq N$. *Let* $\mathbf{A} \in GL_{k-1}(\mathbb{F}_q)$ *and* $\mathbf{b} \in \mathbb{F}_q^{k-1}$ *be submatrices of* $\mathbf{M}$ *such that*

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{b} & \mathbf{0} \\ \mathbf{0} & 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbb{1}_{N-k} \end{pmatrix} .$$

If $(\mathbf{v}_1, \mathbf{v_2})$ *is the representation of a coset* S *as in Lemma 2, then the representation of the coset* $S\mathbf{M}$ *is given by* $(\mathbf{v}_1\mathbf{M}, \mathbf{A}^{-1}(\mathbf{b} + \mathbf{v}_2))$.

Proof. It is clear that if $\mathbf{v}_1$ is the $(k-1)$-th row of a representative of S, then $\mathbf{v}_1\mathbf{M}$ is the $(k-1)$-th row of a representative $S\mathbf{M}$. For the second vector, suppose that the subspace spanned by the first $k-1$ rows of a representative of S is the row subspace of

$$\begin{pmatrix} \mathbb{1} & \mathbf{v}_2 & \mathbf{0} \end{pmatrix} .$$

Then there is a representative of $S\mathbf{M}$ whose first $k-1$ rows span the rowspace of

$$\begin{pmatrix} \mathbb{1} & \mathbf{v}_2 & \mathbf{0} \end{pmatrix} \mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{b} + \mathbf{v}_2 & \mathbf{0} \end{pmatrix} .$$

Putting this in row reduced echelon form we get

$$\begin{pmatrix} \mathbb{1} & \mathbf{A}^{-1}(\mathbf{b} + \mathbf{v}_2) & \mathbf{0} \end{pmatrix} ,$$

which shows that the second vector in the representation of $S\mathbf{M}$ is equal to $\mathbf{A}^{-1}(\mathbf{b} + \mathbf{v}_2)$.

5.4 Permuting T-values to improve the attack

From Sect. 4.1 we know that the size of $\mathcal{P}(P_N)$ is influenced by the fact that the first two T-values are chosen to be equal to 1. This also impacts the performance of the subgroup chain attack, since at each step we carry out a search in the space of cosets $\mathcal{P}(P_k)\backslash\mathcal{P}(P_k + 1)$. In the first column of Tab. 2 we see that if the T-Values would have been chosen randomly, the most expensive step would have

been the first step, where we would have to perform a collision search in a set of at most q^{13} elements. However, Walnut fixes the two first T-values to be 1, so the most expensive step consists of a collision search in a space of at most q^{11} elements. In the last column of Tab. 2 we see that if the designers had chosen to fix the last two T-values to one instead, the complexity of the subgroup chain attack would be reduced: the most expensive step would have been a collision search in a space with only at most q^9 elements. It turns out that we can first apply a transformation to the REM instance to reduce it to an instance of the REM problem where the final two T-values are set to one. Solving this REM instance then only takes $\sqrt{q^9}$ E-Multiplications, so this approach reduces the amount of work by a factor of q. For general values of N, the new method requires approximately $q^{N-7/2}$ E-Multiplications. The reduction relies on the following lemma.

Lemma 4. *Let s_1, s_2 be braids, let (M, π) be a matrix-permutation pair and let T be a set of T-values. Then $s_1 s_2$ is a solution for the REM problem for the pair (M, π) with respect to the list of T-values T if and only if s_2 is a solution for the REM problem for the pair $((CBM(s_1) \downarrow_T)^{-1} M, \sigma(s_1)^{-1}\pi)$ with respect to the permuted list of T-values $\sigma(s_1)(T)$.*

Proof. By applying the definition of E-Multiplication we find that

$$(\mathbb{1}_N, e) \star_T s_1 s_2 = (CBM(s_1) \downarrow_T \cdot \sigma(s_1)(CBM(s_2) \downarrow_T), \sigma(s_1 s_2)) \,.$$

By multiplying from the left by $CBM(s_1) \downarrow_T^{-1}$ and $\sigma(s_1)^{-1}$ we see that the value above is equal to (M, π) if and only if

$$(\sigma(s_1)(CBM(s_2) \downarrow_T), \sigma(b_2)) = ((CBM(s_1) \downarrow_T)^{-1} M, \sigma(s_1)^{-1}\pi) \,.$$

The main insight is that permuting the variables $t_i \mapsto t_{\sigma(b_1)(i)}$ and then evaluating at the values of T leads to the same result as evaluating at the set of permuted values $\sigma(s_1)(T)$. Therefore the left hand side is equal to

$$(CBM(s_2) \downarrow_{\sigma(b_1)(T)}, \sigma(s_2)) = (\mathbb{1}_N, e) \star_{\sigma(s_1)(T)} s_2 \,.$$

Given this lemma, the reduction is straightforward. In order to solve the REM problem for (M, π) we fix a "transport braid" $s_1 = b_2 b_3 \cdots b_{N-1} b_1 b_2 \cdots b_{N-2}$ whose underlying permutation transports the first two entries to the back of the list. Then we calculate the pair $((CBM(s_1) \downarrow_T)^{-1} M, \sigma(s_1)^{-1}\pi)$ and use our REM solving algorithm with respect to the permuted T-values $\sigma(s_1)(T)$ on this pair to find s_2. This is now faster by a factor q because the last two T-values are equal to one. Then $s_1 s_2$ is a solution to the original REM problem.

Table 2. The dimension of the subspaces containing various subgroups, depending on the T-Values

	generic T-values		First two T-values are equal to 1		Last two T-values are equal to 1	
	dim	Δ	dim	Δ	dim	Δ
$\mathcal{P}(P_2)$	1	1	0	0	1	1
$\mathcal{P}(P_3)$	4	3	2	2	4	3
$\mathcal{P}(P_4)$	9	5	5	3	9	5
$\mathcal{P}(P_5)$	16	7	10	5	16	7
$\mathcal{P}(P_6)$	25	9	17	7	25	$\underline{9}$
$\mathcal{P}(P_7)$	36	11	26	9	31	6
$\mathcal{P}(P_8)$	49	$\underline{13}$	37	$\underline{11}$	37	6

5.5 Using a finer chain of subgroups

With a complexity of $O(q^{N/2})$, the factorization algorithm of Hart et al. is more efficient (asymptotically) than the REM solving algorithm that we have described so far. This is due to the fact that Hart et al. use a finer chain of subgroups, which leads to smaller spaces of cosets to search in. In the next paragraph we describe a faster variant of our REM solving algorithm that uses a finer chain of subgroups, similar to the chain used by Hart et al. This variant is much faster than the previous REM solver, but yields solution braids that are longer.

In each step of our REM solving algorithm we have a matrix $M \in A_i$ and we are looking for a braid $s_i \in P_i$ such that $\mathsf{mat}((M, e) \star s_i)$ lies in A_{i-1}. To speed this process up, we can split each step in two substeps. Let C_{i-1} be the subgroup of invertible N-by-N matrices that only differ from the identity matrix in the upper left $(i-1)$-by-$(i-1)$ submatrix. This is a proper subgroup of A_i, which itself contains A_{i-1} as a proper subgroup. To solve the step of the REM solving algorithm we can first search for an $s'_i \in P_i$ such that $\mathsf{mat}((M, e) \star s'_i)$ lies in the intermediate group C_{i-1}, then we search for a braid s''_i such that $\mathsf{mat}((M, e) \star s'_i s''_i)$ lies in A_{i-1}. The first substep of finding s'_i can be carried out with a meet in the middle search. In order to be able to complete the second substep we start by searching for a list of braids $c_1, c_2, \cdots, c_k$ such that $\mathsf{mat}(\mathcal{P}(c_i)) \in C_1$. Then, to solve the second substep, we search for a braid s''_i in the subgroup generated by the braids c_i such that $\mathsf{mat}((M, e) \star s'_i s''_i)$ lies in A_{i-1}.

5.6 Implications and countermeasures

With this method we split each step into two much easier substeps, which greatly improves the efficiency of the algorithm. The downside is that the solutions to

the REM problem that are produced are longer than those produced by the original algorithm. This is because the solution now contains braids s_i'' which are themselves a concatenation of several slightly longer braids c_i. To avoid inflating the size of the output signature needlessly, it is best to only use this technique for solving the most expensive steps. For 128-bit security parameters the signatures output are longer than legitimately produced signatures, but still small enough to be accepted by the NIST implementation. For 256-bit security parameters, the forged signatures are smaller than some legitimately produced signatures, depending on which variant of the signing algorithm is used. Hence, we cannot defend Walnut against this attack by imposing an upper limit on the length of the signatures. Note that it is trivial to convert a short signature into a longer signature, so imposing a lower bound does not help either.

With this method the most expensive step of the algorithm requires only $q^{N/2-1}$ E-Multiplications. The attack is very efficient in practice. We can produce a forgery for 128-bit security parameters in less than one second. Even for 256-bit security parameters we can forge signatures for any document in less than a minute.

There does not seem to be a better way to block the attack other than just increasing the parameters to ensure that $q^{N/2-1}$ is higher than the desired security level. One way to do this is to take $N = 10, q = 2^{32}$ to achieve 128 bits of security, and $N = 10, q = 2^{64}$ for 256 bits of security. For 128-bit security parameters, this leads to a public key of 762 Bytes (a $\times$ 9 increase), signatures of about one kB (an increase of roughly 50%) and an expected slowdown of the signing and verification algorithms by a factor 2 and a factor 30 respectively.

6 Conclusion

In this paper we presented three different practical methods to break the Walnut digital signature scheme (See Table 3). All three attacks are made possible because of the rich algebraic structure of the E-Multiplication map, which is central to the Walnut scheme (and other protocols developped by SecureRF). The first method exploits a signature malleability property of Walnut, and expands on the work of [14] which attacks an earlier version of the Walnut scheme. The second attack is purely generic. It is much more efficient that expected because E-Multiplication maps a certain subgroup of P_N into a subspace of very low dimension. The last attack exploits the fact that E-Multiplication, when restricted to pure braids, is a homomorphism of groups 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 $GL_N(\mathbf{F}_q)$. Some poor design choices such as adopting an encoding mechanism that produces matrices in a low dimensional subspace and a failed attempt to block the attack of Hart et al. [14] seem to be symptomatic of a lack of understanding of the algebraic structure of E-Multiplication. It is the

opinion of the authors that E-Multiplication can not be credibly used as a basis for cryptography until this structure and its implications for cryptography are better understood.

Table 3. An overview of the attacks introduced in this paper, compared with the legitimate signing algorithms.

	Complexity (in number of E-Mults or Mat mults)	128 bits of security		256 bits of security	
		Time	Length of signature (Artin generators)	Time	Length of signature (Artin generators)
Legitimate signing:					
BKL		< 1 sec	± 1480	< 1 sec	± 2661
Stoch. w/o Dehornoy		< 1 sec	± 2788	< 1 sec	± 5260
Attacks:					
factorization	$q^{(N-1)/2}$	5 min	$> 2^{32}$	—	—
collision [4]	$q^{13/2}$	68 min	± 1480	—	—
subgroup chain	$q^{N-7/2}$	4 sec	899	58 hours	1374
fine subgroup chain	$q^{N/2-1}$	< 1 sec	4534	39 sec	4525

The security of the parameter sets submitted to the NIST PQC project is completely broken by the attacks. We show that it is possible to forge signatures or compute equivalent secret keys in under a second for 128-bit security parameters. Even for 256-bit security parameters this takes less than a minute. Updating the parameters to resist the best known attack (see Sect. 5) would significantly increase the public key and signature sizes. This would make the scheme more difficult to implement on the low-resource processors that SecureRF is targeting and destroy the size advantages of Walnut over other post-quantum signature schemes such as lattice-based, hash-based and multivariate signature schemes. We note also that these latter schemes have been subject to much more scrutiny, which improves our confidence in their security.

Acknowledgements This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the European Commission through the Horizon 2020 research and innovation programme under grant agreement No.H2020-ICT-2014-645622 PQCRYPTO. This project was partially supported by the FWO through the WOG Coding Theory and Cryptography. Ward Beullens is funded by an FWO fellowship.

References

1. About SecureRF. https://www.securerf.com/about-us/, accessed: 2018-03-08

[4] Has exactly the same length distribution as legitimately produced signatures

2. Anshel, I., Anshel, M., Fisher, B., Goldfeld, D.: New key agreement protocols in braid group cryptography. In: Cryptographers' Track at the RSA Conference. pp. 13–27. Springer (2001)
3. Anshel, I., Anshel, M., Goldfeld, D.: An algebraic method for public-key cryptography. Mathematical Research Letters 6, 287–292 (1999)
4. Anshel, I., Anshel, M., Goldfeld, D., Lemieux, S.: Key agreement, the algebraic eraser^{TM}, and lightweight cryptography. Contemporary Mathematics 418, 1–34 (2007)
5. Anshel, I., Atkins, D., Goldfeld, D., Gunnells, P.E.: WalnutDSA^{TM}: A quantum-resistant digital signature algorithm. IACR eprint 2017/058 (version: 30-Nov-2017)
6. Anshel, I., Atkins, D., Goldfeld, D., Gunnells, P.E.: WalnutDSA^{TM}: A quantum-resistant digital signature algorithm. IACR eprint 2017/058 (version: 18-Sept-2017)
7. Anshel, I., Atkins, D., Goldfeld, D., Gunnells, P.E.: A class of hash functions based on the algebraic eraser^{TM}. Groups Complexity Cryptology 8(1), 1–7 (2016)
8. Anshel, I., Atkins, D., Goldfeld, D., Gunnells, P.E.: The Walnut digital signature algorithm^{TM} specifcation. Submitted to NIST PQC project (2017)
9. Artin, E.: Theory of braids. Annals of Mathematics pp. 101–126 (1947)
10. Ben-Zvi, A., Blackburn, S.R., Tsaban, B.: A practical cryptanalysis of the algebraic eraser. In: Annual Cryptology Conference. pp. 179–189. Springer (2016)
11. Ben-Zvi, A., Kalka, A., Tsaban, B.: Cryptanalysis via algebraic spans. IACR eprint 41 (2014)
12. Cheon, J.H., Jun, B.: A polynomial time algorithm for the braid diffie-hellman conjugacy problem. In: Annual International Cryptology Conference. pp. 212–225. Springer (2003)
13. Garside, F.A.: The braid group and other groups. The Quarterly Journal of Mathematics 20(1), 235–254 (1969)
14. Hart, D., Kim, D., Micheli, G., Perez, G.P., Petit, C., Quek, Y.: A practical cryptanalysis of WalnutDSA^{TM} (2018)
15. Hughes, J.: A linear algebraic attack on the aafg1 braid group cryptosystem. In: Australasian Conference on Information Security and Privacy. pp. 176–189. Springer (2002)
16. Hughes, J., Tannenbaum, A.: Length-based attacks for certain group based encryption rewriting systems. arXiv preprint cs/0306032 (2003)
17. Kalka, A., Teicher, M., Tsaban, B.: Cryptanalysis of the algebraic eraser and short expressions of permutations as products. Arxiv preprint (2008)
18. Ko, K.H., Lee, S.J., Cheon, J.H., Han, J.W., Kang, J.s., Park, C.: New public-key cryptosystem using braid groups. In: Annual International Cryptology Conference. pp. 166–183. Springer (2000)
19. Myasnikov, A.D., Ushakov, A.: Length based attack and braid groups: cryptanalysis of anshel-anshel-goldfeld key exchange protocol. In: International Workshop on Public Key Cryptography. pp. 76–88. Springer (2007)
20. National Institute for Standards and Technology (NIST): Post-quantum crypto standardization (2016), http://csrc.nist.gov/groups/ST/post-quantum-crypto/
21. Pollard, J.M.: A monte carlo method for factorization. BIT Numerical Mathematics 15(3), 331–334 (1975)
22. Sedgewick, R., Szymanski, T.G., Yao, A.C.: The complexity of finding cycles in periodic functions. SIAM Journal on Computing 11(2), 376–390 (1982)
23. Tsaban, B.: Polynomial-time solutions of computational problems in noncommutative-algebraic cryptography. Journal of Cryptology 28(3), 601–622 (2015)

24. Van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of cryptology 12(1), 1–28 (1999)
25. Vasco, M.I.G., Steinwandt, R.: A reaction attack on a public key cryptosystem based on the word problem. Applicable Algebra in Engineering, Communication and Computing 14(5), 335–340 (2004)
26. Wagner, N.R., Magyarik, M.R.: A public-key cryptosystem based on the word problem. In: Workshop on the Theory and Application of Cryptographic Techniques. pp. 19–36. Springer (1984)