# WALNUTDSA$^{\text{TM}}$: A QUANTUM-RESISTANT DIGITAL SIGNATURE ALGORITHM

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

Abstract. In 2005 I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux introduced E-Multiplication$^{\text{TM}}$, a quantum-resistant, group-theoretic, one-way function which can be used as a basis for many different cryptographic applications. To date, all analysis and attacks on E-Multiplication have been exponential in their runtime and all have been readily addressed and defeated.

This paper introduces WalnutDSA, a new E-Multiplication-based public-key digital signature method that provides very efficient verification, allowing low-powered and constrained devices to quickly and inexpensively validate digital signatures (e.g., a certificate or authentication). This paper presents an in-depth discussion of the construction of the digital signature algorithm, analyzes the security of the scheme, provides a proof of security under EUF-CMA, and discusses the practical results from implementations on several constrained devices. With the implementation of parameters that defeat all known attacks, WalnutDSA is currently the fastest signature verification method in the NIST PQC standardization process and performs orders of magnitude faster than ECC, even on low-end embedded hardware. WalnutDSA delivers a 12-25x speed improvement over ECDSA on most platforms, and a 31x speed improvement on a 16-bit microcontroller, making it an ideal solution for low-resource processors found in the Internet of Things (IoT).

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

## 1. Introduction

Digital signatures provide a means for one party to create a document that can be sent through a second party and verified for integrity by a third party. This method ensures that the first party created the document and that it was not modified by the second party. Historically, digital signatures have been constructed using various number-theoretic, public-key methods like RSA, DSA, and ECDSA. However, these methods are not very efficient in tiny devices running on 16-bit or even 8-bit constrained processors (let alone some constrained 32-bit platforms), or systems with limited space or energy.

Digital signatures based on hard problems in group theory are relatively new. In 2002, Ko, Choi, Cho, and Lee [30] proposed a digital signature based on a variation of the conjugacy problem in non-commutative groups. In 2009, Wang and Hu [45] introduced a digital signature with security based upon the hardness of the root problem in braid groups. See also [28]. The attacks introduced in [18], [19], [21], and [26] suggest that the schemes by Ko et al. and Wang and Hu may not be practical over braid groups in low-resource environments.

**Previous Work.** E-Multiplication [5] is a group-theoretic, one-way function first introduced by I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux in 2005 [5]. E-Multiplication uses a combination of braids, matrices, and finite fields to translate the non-abelian, infinite group into a computable system. It has proven to be a very efficient, general-purpose, quantum-resistant one-way function; its use is broader than the original key-agreement construction. For example, using E-Multiplication as the basic building block, Anshel, Atkins, Goldfeld, and Gunnells recently introduced a cryptographic hash function, AEHash [3], which has been implemented using very little code space on a 16-bit platform [4].

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

**Our Contribution.** This paper introduces a new quantum-resistant digital signature algorithm, WalnutDSA$^{\text{TM}}$. Its security is based on the difficulty of *reversing E-Multiplication*. Details are given in §10. Reversing E-Multiplication is a hard problem in braid groups that is very different from the *Conjugacy Search Problem* (CSP), which formed the foundation of the earliest cryptographic systems based on the braid group. In fact, WalnutDSA appears immune to all the types of attacks related to the CSP given in [18], [19], [21], and [26], as well as the very recent work of [25], [8], [9], and [14] (for a fuller discussion see §9 below - linear algebraic, group theoretic, and probabilistic attacks). Likewise, attacks on the original 2005 key agreement construction noted in [7], [29], and [38], do not apply because the construction's structures do not map.

E-Multiplication is rapidly executable, even in the smallest of environments, and as a result, WalnutDSA provides very fast signature verification. We implemented WalnutDSA; its performance in various environments outperformed ECDSA by orders of magnitude in all cases tested using less code space and energy.

This paper proceeds as follows: First, it reviews the colored Burau representation of the Braid Group and E-Multiplication; Second, it introduces the concept of a cloaking element and shows the connection between braid groups, cloaking elements, and WalnutDSA; Third, it shows WalnutDSA key generation; Fourth, it presents a practical implementation via a message encoder algorithm as well as the signature generation and verification processes; Fifth, it discusses and analyzes the security implications associated with WalnutDSA; Sixth, it proposes a slightly modified version of WalnutDSA and presents a security proof under EUF-CMA that breaking this version will break the underlying hard problem; Seventh, it discusses brute-force security and quantum resistance; and Eighth, it tests WalnutDSA's size and performance characteristics on several constrained devices.

## 2. Colored Burau Representation of the Braid Group

For, $N \geq 2$, let $B_N$ denote the $N$-strand braid group with Artin generators $\{b_1, b_2, \ldots, b_{N-1}\}$, subject to the following relations:

$$b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1}, \qquad (i = 1, \ldots, N-2), \tag{1}$$

$$b_i b_j = b_j b_i, \qquad (|i-j| \geq 2). \tag{2}$$

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

$$\beta = b_{i_1}^{\epsilon_1} \, b_{i_2}^{\epsilon_2} \, \cdots \, b_{i_k}^{\epsilon_k}, \tag{3}$$

where $i_j \in \{1, \ldots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$. Note that $\beta$ is not uniquely represented by (3); any braid has infinitely many different expressions in terms of the Artin generators, thanks to the relations (1) and (2).

Let $S_N$ be the group of permutations on $N$ letters. Each braid $\beta \in B_N$ determines a permutation in $S_N$ as follows. For $1 \leq i \leq N-1$, let $\sigma_i \in S_N$ be the $i^{\text{th}}$ simple transposition, which maps

$i \to i+1,\ i+1 \to i$, and leaves $\{1, \dots, i-1, i+2, \dots, N\}$ fixed. Then the map $b_i \mapsto \sigma_i$ extends to a surjective homomorphism $B_N \to S_N$. A braid is called pure if its corresponding permutation is trivial (i.e., the identity permutation).

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

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

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

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

. For each Artin generator $b_i$ we define the $N \times N$ colored Burau matrix $CB(b_i))$ generator as follows [36]. If $i = 1$, we put

$$CB(b_1) = \begin{pmatrix} -t_1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \vdots \\ \vdots & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix}, \tag{4}$$

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

$$CB(b_i) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & t_i & -t_i & 1 & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}, \tag{5}$$

where the indicated variables appear in row $i$. We similarly define $CB(b_i^{-1})$ by modifying (5) slightly:

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

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

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

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

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

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

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

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

as in (3), we can define a colored Burau pair $(CB(\beta), \sigma_\beta)$ by

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

The colored Burau representation is then defined by

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

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

## 3. E-Multiplication

E-Multiplication was first introduced in [5] as a one-way function used as a building block to create multiple cryptographic constructions. We recall its definition here.

An ordered list of entries in the finite field (named T-values) is defined to be a collection of non-zero field elements:

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

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

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

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

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

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

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

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

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

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

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

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

$$(M, \sigma_0) \star (CB(\beta), \sigma_\beta) = (M, \sigma_0) \star (CB(b_{i_1}^{\epsilon_1}), \sigma_{b_{i_1}}) \star (CB(b_{i_2}^{\epsilon_2}), \sigma_{b_{i_2}}) \star \cdots \star (CB(b_{i_k}^{\epsilon_k}), \sigma_{b_{i_k}}), \tag{6}$$

where we interpret the right of (6) by associating left-to-right. One can check that this is independent of the expression of $\beta$ in the Artin generators.

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

## 4. Cloaking Elements

The security of WalnutDSA is based on the existence of certain braid words which we term cloaking elements. They are defined as follows.

**Definition 4.1. (Cloaking element)** *Let $M \in GL(N, \mathbb{F}_q)$ and $\sigma \in S_N$. An element $v$ in the pure braid subgroup of $B_N$ is termed a cloaking element of $(M, \sigma)$ if*

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

*Let $\mathrm{Cloak}_{(M,\sigma)}$ denote the set of all such cloaking elements.*

Thus a cloaking element is characterized by the property that it essentially disappears when performing E-Multiplication i.e., a cloaking element stabilizes the element $(M, \sigma)$ under the action of $B_N$ on $GL(N, \mathbb{F}_q) \times S_N$ which E-Multiplication defines. Thus the following proposition is immediate:

**Proposition 4.2.** *The set $\mathrm{Cloak}_{(M,\sigma)}$ forms a subgroup of $B_N$.*

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

**Proposition 4.3.** *Fix integers $N \geq 2$, and $1 \leq a < b \leq N$. Assume that the T-values $\tau_a$ and $\tau_b$ satisfy the property $\tau_a \cdot \tau_b = -1$. Let $M \in GL(N, \mathbb{F}_q)$ and $\sigma \in S_N$. Then a cloaking element $v$ of $(M, \sigma)$ is given by $v = w b_i^4 w^{-1}$ where $b_i$ is any Artin generator $(1 \leq i < N)$, and where the permutation corresponding to $w \in B_N$ satisfies*

$$i \longmapsto \sigma^{-1}(a), \quad i + 1 \longmapsto \sigma^{-1}(b).$$

*In addition, if the field $\mathbb{F}_q$ has characteristic 2, then by choosing $w$ as above the braid $w\, b_i^2 w^{-1}$ is also a cloaking element for $(M, \sigma)$.*

*Proof.* To facilitate a compact discussion, we consider the case $N = 4$, and $i = 2$. By definition we have that,

$$\mathrm{CB}(b_i^2) = \left( \begin{pmatrix} 1 & 0 & 0 & 0 \\ t_2 & -t_2 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{\sigma_2}\begin{pmatrix} 1 & 0 & 0 & 0 \\ t_2 & -t_2 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathrm{Id}_{S_N} \right)$$

$$= \left( \begin{pmatrix} 1 & 0 & 0 & 0 \\ (t_2 - t_2 t_3) & t_2 t_3 & (1 - t_2) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathrm{Id}_{S_N} \right),$$

where $\mathrm{Id}_{S_N}$ is the identity permutation. Thus

$$\mathrm{CB}(b_2^4) = \left( \begin{pmatrix} 1 & 0 & 0 & 0 \\ (t_2 - t_2t_3) & t_2t_3 & (1-t_2) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot {}^{\mathrm{Id}_{S_N}}\begin{pmatrix} 1 & 0 & 0 & 0 \\ (t_2 - t_2t_3) & t_2t_3 & (1-t_2) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathrm{Id}_{S_N} \right)$$

$$= \left( \begin{pmatrix} 1 & 0 & 0 & 0 \\ (t_2 - t_2t_3) + t_2t_3(t_2 - t_2t_3) & (t_2t_3)^2 & t_2t_3(1-t_2) + (1-t_2) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathrm{Id}_{S_N} \right),$$

and we see that by making the assumption $t_2t_3 = -1$, $\mathrm{CB}(b_2^4) = \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big)$. The argument is completed by conjugating this fourth power by a $w \in B_N$ where the permutation corresponding to $w$ satisfies

$$i \longmapsto \sigma^{-1}(a), \quad i+1 \longmapsto \sigma^{-1}(b).$$

□

As an aside, we note that the above computation can be iterated: for $k \geq 2$,

$$\text{Matrix Part}(\mathrm{CB}(b_i^{2^k})) =$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ (t_2 - t_2t_3)(1 + (t_2t_3)) \cdots (1 + (t_2t_3)^{2^{k-2}}) & (t_2t_3)^{2^{k-1}} & (1-t_2)(1 + (t_2t_3)) \cdots (1 + (t_2t_3)^{2^{k-2}}) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The above construction can be applied the following way. Fix a braid $\beta$, say

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

and choose some point $1 \leq k \leq \ell$. Clearly, $\beta = x_1 \cdot x_2$ where $x_1 = b_{i_1}^{\epsilon_1} \cdots b_{i_{k-1}}^{\epsilon_{k-1}}$ and $x_2 = b_{i_k}^{\epsilon_k} \cdots b_{i_\ell}^{\epsilon_\ell}$, and we hence for any for any matrix/permutation pair $(m_0, \sigma_0)$, we have that

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

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

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

Hence we have generated a new braid $\beta'$ which contains $v$,

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

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

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

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

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

## 5. Notation for cryptographic protocols

Let $S$ be a set.

$\langle S \rangle$ denotes a unique encoding of $S$ as a binary string.

$s \xleftarrow{\$} S$ denotes the operation of randomly choosing $s \in S$.

Let $A(\,*\,;\rho)$ be a randomized algorithm with randomness based on a coin $\rho$.

$A(y_1, \dots y_q; \rho)$ denotes the output of the algorithm $A$ on inputs $y_1, \dots y_q$ and coin $\rho$.

$z \xleftarrow{\$} A(y_1, \dots y_q)$ means choose $\rho$ at random and let $z = A(y_1, \dots y_q; \rho)$.

Let $\beta \in B_N$.

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

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

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

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

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

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

Support for the hardness of reversing E-Multiplication can be found in [37] which studies the security of Zémor's [48] hash function $h : \{0,1\}^* \to SL_2(\mathbb{F}_q)$, with the property that $h(u\,v) = h(u)h(v)$, where $h(0), h(1)$ are fixed matrices in $SL_2(\mathbb{F}_q)$ and $uv$ denotes concatenation of the bits $u$ and $v$. For example a bit string $\{0, 1, 1, 0, 1\}$ will hash to $h(0)h(1)h(1)h(0)h(1)$. Zémor's hash function has not been broken since its inception in 1991. In [37] it is shown that feasible cryptanalysis for bit strings of length 256 can only be applied for very special instances of $h$. Now E-Multiplication, though much more complex, is structurally similar to a Zémor type scheme involving a large finite number of fixed matrices in $SL_2(\mathbb{F}_q)$ instead of just two matrices $h(0), h(1)$. This serves as an additional basis for the assertion that E-Multiplication is a one-way function.

## 6. Key Generation for WalnutDSA

WalnutDSA allows a signer with a fixed private-/public-key pair to create a digital signature associated with a given message that can be validated by anyone who knows the public-key of the signer and the verification protocol. We now describe the algorithms for private-/public-key generation.

A central authority generates the system wide parameters denoted, *par*, via a parameter generation algorithm, denoted Pg, where $par \xleftarrow{\$}$ Pg. A signer S generates its own public and private key pair, denoted (Pub(S), Priv(S)), via a key generation algorithm denoted Kg. In other words, (Pub(S), Priv(S)) $\xleftarrow{\$}$ Kg(*par*).

**Public System Wide Parameters (*par*):**

- An integer $N \geq 10$ and associated braid group $B_N$.
- An integer $\kappa > 1$ which is chosen to meet the security level. The signature will utilize $\kappa$ concealed cloaking elements.
- A rewriting algorithm $\mathcal{R}\colon B_N \to B_N$ which uses the relations of the group to render a rewritten word unrecognizable. Example of such rewriting algorithms can be found in [12] or [16].
- A finite field $\mathbb{F}_q$.
- Two integers $1 < a < b < N$.
- T-values $= \{\tau_1, \tau_2, \ldots, \tau_N\}$, where each $\tau_i$ is an invertible element in $\mathbb{F}_q$, and $\tau_a \cdot \tau_b = -1$.

**Signer's Private Key:**

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

- Priv(S) $= (w, w') \in B_N \times B_N$.

Here the three braids $w$, $w'$ and $w' \cdot w$ are not in the pure braid group. We assume $w, w'$ are sufficiently long to provide the necessary resistance to brute-force searches for the desired security level (see §11).

**Signer's Public Key:**

The Signer's Public Key consists of two matrix and permutation pairs, each of which is generated from the Private Keys of the signer via E-Multiplication:

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

## 7. Message Encoder Algorithm

In order to generate a secure signature and prevent certain types of merging attacks, one must carefully convert the message to be signed into a braid word. Let $m \in \{0,1\}^*$ be a message. Let $H : \{0,1\}^* \to \{0,1\}^{2\eta}$ denote a cryptographically secure $2\eta$-bit hash function for $\eta \geq 1$. We now present an injective encoding function $E : \{0,1\}^{2\eta} \to H_N$, where $H_N$ is a free subgroup of the pure braid group generated by the $N-1$ braids defined below. We recall that a group is said to be freely generated by a subset of elements provided a reduced element (a word where the subwords $x \cdot x^{-1}$, and $x^{-1} \cdot x$ do not appear for any generator $x$) is never the identity.

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

$$\begin{aligned} g_{(N-1),N} &= b_{N-1}^2 \\ g_{(N-2),N} &= b_{N-1} \cdot b_{N-2}^2 \cdot b_{N-1}^{-1} \\ g_{(N-3),N} &= b_{N-1}b_{N-2} \cdot b_{N-3}^2 \cdot b_{N-2}^{-1}b_{N-1}^{-1} \\ g_{(N-4),N} &= b_{N-1}b_{N-2}b_{N-3} \cdot b_{N-4}^2 \cdot b_{N-3}^{-1}b_{N-2}^{-1}b_{N-1}^{-1} \\ &\vdots \\ g_{1,N} &= b_{N-1}b_{N-2}\cdots b_2 \cdot b_1^2 \cdot b_2^{-1}b_3^{-1}\cdots b_{N-1}^{-1}, \end{aligned} \tag{7}$$

generate a free subgroup $H_N \leq B_N$ [11]. For simplicity, we will write $g_i$ for $g_{i,N}$ when $N$ is clear from the context.

**Message Encoder Algorithm:** We determine a braid $E(H(m)) \in B_N$ as follows. The hashed message $H(m)$ consists of $\eta$ 2-bit blocks. Fix a collection $S$ of $\eta$ subsets $S_\eta$, where each $S_\eta$ consists of a four-tuple of distinct generators taken from (7):

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

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

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

[1] For a weak hash $H_1$ and a strong hash $H_2$, which has twice the output size of $H_1$, an attacker would need to find two messages $m$ and $m'$ that are preimages to the halves of $H_2$ of the desired forgery and then get the signer to use $H_1$ and sign both $m$ and $m'$. E.g. the attacker would need to take his or her desired forged message, hash it

also note that including a hash algorithm identifier in the message after it is hashed would prevent this attack.

## 8. Signature Generation and Verification

Fix a hash function $H$ as in §7. To sign a message $m \in \{0,1\}^*$ the Signer performs the following steps:

**Digital Signature Generation:**

**1.** Compute $H(m)$.

**2.** Generate cloaking elements $v$, $v_1$, and $v_2$ where

- $v$ cloaks $(\mathrm{Id}_N, \mathrm{Id}_{S_N})$,
- $v_1$ cloaks $\mathcal{P}(w)$.
- $v_2$ cloaks $\mathcal{P}(w')$.

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

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

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

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

**Signature Verification:** The signature $(m, \mathrm{Sig})$ is verified as follows:

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

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

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

**4.** Test the equality

$$\mathrm{Matrix}\Big(\mathcal{P}(w) \star \mathrm{Sig}\Big) \overset{?}{=} \mathrm{Matrix}\Big(\mathcal{P}\big(E(H(m))\big)\Big) \cdot \mathrm{Matrix}\Big(\mathcal{P}\big(w'\big)\Big), \tag{8}$$

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

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

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

using SHA2-256, find two preimages with MD5, get the signer to sign those MD5 preimages, and only then can he or she compose a message that would verify with SHA2-256.

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

## 9. Security Proof for WalnutDSA-I

In this section we will provide security proofs for a Schnorr/Brickell type model (see [31], [15]) of WalnutDSA, denoted WalnutDSA-I which is defined below. Specifically, we will prove that WalnutDSA-I is existentially unforgeable under adaptive chosen-message attacks (EUF-CMA-secure) in the random oracle model assuming a Forger has the ability to forge valid signatures of a specified type with non-negligible probability.

Keeping with the notation from §4, we define the set Cloak as follows:

$$\text{Cloak} := \Big\{(v, v_1, v_2) \;\Big|\; v, v_1, v_2 \in B_N,\ v \in \text{Cloak}_{\text{Id}},\ v_1 \in \text{Cloak}_{\mathcal{P}(w)},\ v_2 \in \text{Cloak}_{\mathcal{P}(w')}\Big\},$$

where Id$= (\text{Id}_N, \text{Id}_{S_N})$.

The system wide parameters and key generation algorithm for WalnutDSA-I is the same as for WalnutDSA and is given by

$$\mathit{par} \xleftarrow{\$} \text{Pg},$$

$$(\text{Pub}(S), \text{Priv(S)}) \;\xleftarrow{\$}\; \text{Kg}(\mathit{par}).$$

In WalnutDSA-I the signature of a message $m \in \{0,1\}^*$ for the public Pub(S) is based on two hash functions $H, G : \{0,1\}^* \to \{0,1\}^{2\kappa}$ and is generated by the following protocol.

**1.** $(v, v_1, v_2) \xleftarrow{\$} \text{Cloak}, \quad V = \big\langle(v, v_1, v_2)\big\rangle.$

**2.** Compute $E\left(H\big(m \,||\, G\,(V\,)\,\big)\,\right)$.

**3.** Compute Sig $= \mathcal{R}\big(\kappa(v_1\, w^{-1} v \cdot E\left(H\big(m \,||\, G\,(V\,)\,\big)\,\right) \cdot w'\, v_2)\big)$. The final signature is denoted $\big(m, H(m), G(V), \text{Sig}\big)$.

To validate the signature, one checks whether

$$\text{Matrix}\big(\mathcal{P}(w) \star \text{Sig}\big) \overset{?}{=} \text{Matrix}\Big(\mathcal{P}\left(E\left(H\big(m \,||\, G\,(V\,)\,\big)\,\right)\right)\Big) \cdot \text{Matrix}\Big(\mathcal{P}(w')\Big).$$

Note that all WalnutDSA-I signatures on a message $m$ created by an honest signer lie in the double coset

$$\text{(9)} \qquad \text{DC}_{m,V,H,G} := \Big\{\mathcal{R}\Big(X \cdot \left[\mathcal{P}\left(E\left(H\big(m \,||\, G\,(V\,)\,\big)\,\right)\right)\right] \cdot Y\Big) \;\Big|\; X, Y \in B_N\Big\},$$

where $X, Y$ depend only on the cloaking elements $V$ chosen by the honest signer and do not depend on the message $m$ or the hash function $H, G$. Not every valid signature needs to be of this form. This is due to the fact that the braid group $B_N$ is non-commutative and E-Multiplication is a highly randomized function.

**EUF-CMA Security Proof for WalnutDSA-I.** We now assume the existence of a forger, denoted $\mathcal{F}$, that on input Pub(S) and message $m$, can produce a valid WalnutDSA-I signature lying in the double coset $\text{DC}_{m,V,H,G}$ with non-negligible probability. The assumption that the Forger only can produce possible signatures lying in $\text{DC}_{m,V,H,G}$ is restrictive. As pointed out by Koblitz and Menezes [31], although it is a common approach in modern security proofs to restrict the capabilities of the adversary, it is important to show that certain classes of attacks can be ruled out.

More precisely, we define $\mathcal{F}$ to be a randomized algorithm which can make hash queries to a random oracle and signature queries to a simulator that does not know $\mathrm{Priv}(S)$ but can simulate an honest signer.

Hash Query: Let $\mathcal{O}_\rho$ denote a random oracle, depending on a coin $\rho$, which evaluates the hash of a string $str \in \{0,1\}^*$. A hash query is just a string $str$. The response to the query is the hash of $str$, provided by $\mathcal{O}_\rho$.

Signature Query: A signature query is the message and the public key of the signer. The response to the query is a valid signature.

The Forger $\mathcal{F}$: Consider WalnutDSA-I with system wide parameters and public/private key pair specified by

$$par \xleftarrow{\$} \mathrm{Pg}, \quad (\mathrm{Pub(S)}, \mathrm{Priv(S)}) \xleftarrow{\$} \mathrm{Kg}(par).$$

We assume the hash function $H$ is fixed and multi-collision-resistant while the hash function $G = G_\rho$ is given by the oracle $\mathcal{O}_\rho$ which depends on a coin $\rho$.

The Forger $\mathcal{F}$ is defined to be a randomized algorithm that on input a message $m \in \{0,1\}^*$, a signer's public key $\mathrm{Pub}(S)$, and a coin $\rho$, outputs a 4-tuple $\big(m, h, g_\rho, s\big)$, where $h = H(m)$ and $g_\rho = G_\rho(V)$ and $V \xleftarrow{\$} \mathrm{Cloak}$, $s \xleftarrow{\$} \mathrm{DC}_{m,V,H,G}$. It is assumed that the probability that $\big(m, h, g_\rho, s\big)$ is a valid WalnutDSA-I signature is non-negligible.

**Lemma 9.1. (Forking Lemma)** *Let $\mathcal{F}$ be run twice with inputs,*

$$(m, \mathrm{Pub(S)}, \rho), \qquad (m, \mathrm{Pub(S)}, \rho'),$$

*then with non-negligible probability, $\mathcal{F}$ will output two valid signatures*

$$\big(m, h, g_\rho, s\big), \qquad \big(m, h, g_{\rho'}, s'\big),$$

*such that $g_\rho \neq g_{\rho'}$.*

*Proof.* This follows from [40], [6]. □

The forking lemma 9.1 can be used to show that under an EUF-CMA attack it is possible for $\mathcal{F}$ to solve the REM problem (reversing E-multiplication is hard) with non-negligible probability provided there is a polynomial time solution to the conjugacy search problem CSP which is the problem of finding $X \in B_N$ assuming that $w \in B_N$ and $XwX^{-1} \in B_N$ are known. This is conjectured to be true by many people and it has been experimentally shown that if $X$ is chosen according to a standard uniform distribution then $X$ can be found with high probability in polynomial time [19], [21].

**Theorem 9.2.** *Assume that CSP can be solved in polynomial time. Further, assume that two WalnutDSA-I signatures*

$$\big(m, H(m), G_\rho(V), s\big), \qquad \big(m, H(m), G_{\rho'}(V), s'\big),$$

*with $G_\rho(V) \neq G_{\rho'}(V)$ are known to an adversary. Then it is possible for the adversary to solve the REM problem in polynomial time with non-negligible probability.*

*Proof.* Let

$$s = \mathcal{R}\Big(X \cdot \big(E\left(H\big(m \,||\, G_\rho\,(V)\,\big)\,\right)\big) \cdot Y\Big) = X \cdot \big(E\left(H\big(m \,||\, G_\rho\,(V)\,\big)\,\right)\big) \cdot Y,$$
$$s' = \mathcal{R}\Big(X \cdot \big(E\left(H\big(m \,||\, G_{\rho'}\,(V)\,\big)\,\right)\big) \cdot Y\Big) = X \cdot \big(E\left(H\big(m \,||\, G_{\rho'}\,(V)\,\big)\,\right)\big) \cdot Y,$$

be the two known signatures where "=" means equality in the braid group, and where $X, Y$ depend only on the choice of the cloaking elements $V$. It follows that

$$s \cdot (s')^{-1} = X \cdot \Big[\big(E\left(H\big(m \,||\, G_\rho\,(V)\,\big)\,\right)\big) \cdot \big(E\left(H\big(m \,||\, G_{\rho'}\,(V)\,\big)\,\right)\big)^{-1}\Big] \cdot X^{-1}.$$

By our assumptions, it is possible to solve for $X$, and then also solve for $Y$. Note that $X$ has the property that $\mathcal{P}(w) = (\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star X$, and, hence, E-Multiplication has been reversed in this case. □

**Strong existential forgery.** Strong existential forgery is the situation when an attacker is able to forge a second signature of a given message that is different from a previously obtained signature of the same message.

WalnutDSA as presented above is, a priori, subject to strong existential forgery. The signature of a message $\mathcal{M}$ is of the form

$$\mathrm{Sig} = \mathcal{R}\big(\kappa(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2)\big). \tag{10}$$

Clearly an attacker could augment the above signature by multiplying it (on the right) by an additional cloaking element, thus obtaining a second signature of the same message. This does not undermine WalnutDSA security if we require a forgery to be a message that was never signed previously because of the non-repudiation property discussed previously.

## 10. Security Discussion

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

**Definition 10.1. (Security Level):** *A secret is said to have security level $k$ over a finite field $\mathbb{F}$ if the best known attack for obtaining the secret involves running an algorithm that requires at least $2^k$ elementary operations (addition, subtraction, multiplication, division) in the finite field $\mathbb{F}$.*

**Linear Algebraic, Group Theoretic, and Probabilistic Attacks.** Neither the attack of Ben-Zvi–Blackburn–Tsaban [7], based on ideas in [29], or the invalid public key attack of Blackburn–Robshaw [13] (also [1]) target the underlying hard problems on which WalnutDSA is based. This is because the signature is a braid (a cloaked conjugate) and the public key is coming from E-Multiplication of the identity element with a braid that has very little algebraic structure.

The more recent work of Hart–Kim–Micheli–Perez–Petit–Quek [25] proposes a practical universal forgery attack on WalnutDSA in the special case where the two private braids $w$ and $w'$ are equal. The attack proceeds by taking a collection of signed messages $(M_i, s_i)$ indexed by a finite set $I$ and using them to produce a valid signature for a new message $M$. The main idea underlying the attack is finding a short expression in $GL(N, \mathbb{F}_q)$ for the element $h = \mathrm{Matrix}\Big(\mathcal{P}(E(M))\Big)$ in terms of elements $g_i := \text{Matrix}\Big(\mathcal{P}(E(M_i)\Big)$. Namely, one seeks an expression of the form

$$h = \prod_{j=1}^{l} g_{i_j}^{\epsilon_{i_j}}, \quad i_j \in I, \epsilon_{i_j} \in \{\pm 1\}. \tag{11}$$

Then the braid

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

will be a valid signature for $M$.

Thus the attack relies on both the equality of $w$ and $w'$ and on finding factorizations in nonabelian groups: the former implies that one can appropriately multiply the signatures $s_i$ together in the final step to produce a signature for $M$, and the latter implies that one can find the correct product of the $s_i$. This attack fails if $w \neq w'$, since one cannot multiply the $s_i$ together to produce a valid signature. It is observed in [8] that it is possible to modify the attack of [25] so that it reduces to the case $w = w'$ with forged signatures that are expected to be twice as long as forged signatures produced by the attack of [25]. The authors of [25] point out that the forged signatures produced by their method (in the case $w = w'$) are many orders of magnitude longer than the actual signatures produced by WalnutDSA, so the attack is easily thwarted by rejecting long signatures. Further, they also point out that their attack fails with moderate increases in the parameters $N, q$.

Four additional attacks have appeared recently. A Pollard-Rho type method taken by [14] uses the estimate for the number of braids of a given length in Artin generators (see §11), and assumes the output of E-multiplication is uniformly distributed, to give an exponential algorithm that recovers an equivalent private key of a signature from the corresponding public key. Specifically, [14] shows that to reach a $k$-bit security level:

$$q^{N(N-3)-1} > 2^{2k} \tag{12}$$

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

Further, the encoding method specified in §7 ensures that the vector space consisting of the matrix component of the signers' public keys has a sufficiently large dimension. It was observed in [9] that the encoding must ensure this property to maintain the specified security level, specifically, to reach a $k$-bit security level:

$$q^{\text{dimension}} > 2^{2k} \tag{13}$$

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

An alternate exponential factoring attack [10] found a more efficient way to find alternate private keys that produce short signatures. The attack was mounted against the WalnutDSA NIST submission, which uses an older version of WalnutDSA where $\tau_1 = \tau_2 = 1$, and suggested parameters $N = 8, q = 32$ for 128-bit security and $N = 8, q = 256$ for 256-bit security. Specifically, the attack in [10] showed that those parameters were too small. Against that older version of WalnutDSA using those parameters the attack runs in $q^{N-5/2}$ time although they claim it can be reduced to

$q^{(N/2)-1}$. While the former runtime was verified, the latter runtime was never observed using the attack code made available.

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

$$\sqrt{x}\; q^{(N-1)/2} \tag{14}$$

Lastly, a method for searching for cloaking elements of known permutations has been posited by Kotov–Menshov–Ushakov [32]. It is the presence of $\kappa$ concealed cloaking elements that blocks this attack. In general, knowing that $\kappa$ concealed cloaking elements have been placed in a known braid, it would require $(N!)^\kappa$ searches to find them and thus, taking the lack of possible birthday attacks into account, to insure $k$-bit security we would require

$$(N!)^\kappa > 2^k \tag{15}$$

and hence

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

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

## 11. Brute Force Attacks

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

$$W_N(\ell) \;\le\; \left(2(n-1)\right)^\ell .$$

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

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

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

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

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

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

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

This is just an upper bound on the number of braids of length $\ell$ but it does represent what an attacker would have to do to be certain that all possibilities are checked. At present, the above method gives the best protocol known for generating braid words of length $\ell$ with the least over counting. There is no closed formula for the number of distinct braids of length $\ell$; in fact the problem is NP hard [39].

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

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

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

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

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

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

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

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

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

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

The pure braid subgroup of $B_N$ is generated [24] by the set of $N(N-1)/2$ braids given by

$$g_{i,j} = b_{j-1}b_{j-2}\cdots b_{i+1}\cdot b_i^2\cdot b_{i+1}^{-1}\cdots b_{j-2}^{-1}b_{j-1}^{-1}, \qquad 1 \le i < j \le N. \tag{16}$$

The relations for the pure braid subgroups are given by

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

see [24] for details.

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

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

and hence the security level can be estimated to be

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

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

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

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

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

**Quantum Resistance.** We now quickly explore the quantum resistance of WalnutDSA. As shown in §10, the security of WalnutDSA is based on the hard problem of reversing E-Multiplication. The underlying math is intimately tied to the infinite non-abelian braid group that is not directly connected to any finite abelian group. We will show that this lends strong credibility for the choice of WalnutDSA as a viable post-quantum digital signature protocol.

The Hidden Subgroup Problem HSP on a group $G$ asks to find an unknown subgroup $H$ using calls to a known function on $G$ which is constant on the cosets of $G/H$ and takes different values on distinct cosets. Shor's [42] quantum attack breaking RSA and other public key protocols such as ECC are essentially equivalent to the fact that there is a successful quantum attack (the quantum Fourier transform QFT) on the HSP for finite cyclic and other finite abelian groups (see [33]).

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

Both possibilities are far beyond what is currently known for HSP. First of all, the braid group is infinite, and no progress has been made for HSP for infinite groups. Moreover, every non-trivial element in $B_N$ has infinite order, and in particular the braid group does not contain any non-trivial finite subgroups. Hence there does not seem to be any viable way at present to work with quantum solutions for HSP in $B_N$. Second, some progress has been made in quantum solutions to HSP for certain nonabelian finite groups, such as semidirect products of abelian groups, or groups with the property that all subgroups are normal. However progress for groups with large degree representations such as $GL(N, \mathbb{F}_q)$ and other finite groups of Lie type has been more limited. Currently the best one knows how to do is to construct subexponential circuits to compute the QFT on such groups [35]. This does not give an efficient algorithm to apply quantum attacks to such groups.

Given an element

$$\beta = b_{i_1}^{\epsilon_1} \, b_{i_2}^{\epsilon_2} \, \cdots \, b_{i_k}^{\epsilon_k} \in B_N, \tag{17}$$

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

Finally, we consider Grover's quantum search algorithm [22] which can find an element in an unordered $N$ element set in time $\mathcal{O}(\sqrt{N})$. Grover's quantum search algorithm can be used to find the private key in a cryptosystem with a square root speed-up in running time. Basically, this cuts the security in half and can be defeated by doubling the key size. This is where E-Multiplication shines. When doubling the key size one only doubles the amount of work as opposed to RSA, ECC, etc. where the amount of work is quadrupled. Note that almost all of the running time of signature verification in WalnutDSA is taken by repeated E-Multiplications.

## 12. Size and Performance Characteristics

To test WalnutDSA we wrote key and signature generation and validation software in C (and on some platforms implemented part of the verification engine in assembly). We ran the signature generation on a Thinkpad T470p laptop with an Intel i7-7820HQ CPU @ 2.90GHz running Fedora Linux to generate 500 keypairs, and for each key generated 100 random 256-bit messages and the resulting signatures. For the signature rewriting we used a combination of the Birman–Ko–Lee (BKL) [12] and Dehornoy [16] algorithms to obscure the braids and shorten them to reasonable lengths.

For our testing we settled on the parameters:

- $N = 10$
- $q = M31 = 2^{31} - 1$
- $\kappa = 6$
- $\ell = 124$

which yields a private key and signature security level of at least $2^{128}$ against all known attacks, with a public keyspace of approximately $2^{2821}$ possible public keys. These parameters also assure sufficient security based on Equations 12, 13, 15, and 14, yielding $2^{2139}$, $2^{2542} > 2^{256}$, $2^{130} > 2^{128}$, and at least $2^{142.4}$ (assuming that the attack's parameter $x$ remains at 60).

Each of the public keys are always a fixed size. They need to include the T-Values, both Matrices, and Permutation which requires

$$N \log_2(q) + 2(N(N-1)+1)\log_2(q) + N\log_2(N) = 310 + 2 * 2821 + 50 = 6002 \text{ bits.}$$

Private keys and signatures, however, are variable length. Recall that each private key has two braids. In the 500 private keys (1000 braids), the braids varied in length from 90 generators to 124 generators, with a mean of 110.24 and a standard deviation of 5.03. With our encoding, this results in a private key storage of 900 to 1240 bits; the maximum storage is 1240 bits.

Using those 500 keys we generated 50,000 signatures using random input messages of 256 bits (simulating SHA256 hash output), and then used BKL and Dehornoy as the rewriting methods. Of these 50,000 signatures, their lengths varied from 1158 to 3306 generators, with a mean of 2037.21 and a standard deviation of 286.85. These signatures also require 5 bits per generator, which results in signatures of length of 5790 to 16530 bits (with an average of 10186.05 bits).

For a 256-bit security level we settled on $N = 10, q = M61 = 2^{61} - 1, \kappa = 12$, and $\ell = 275$ which results in private keys that range from 217 to 267 generators, with a mean of 244.52, and signatures that range from 2420 to 5704 generators, with a mean of 3856.16.

**Signature Validation.** Where WalnutDSA shines is in signature validation, because E-Multiplication is rapidly computable even in the tiniest of environments. To prove its viability we implemented the WalnutDSA signature verification routines on several platforms: an Intel x86_64 (the NIST PQC target platform), an ARM Cortex R5, an ARM Cortex M3, an ARM Cortex M4, an ARM Cortex M0, a RISC-V FE10, and a Renesas RL78.

To provide a common testing platform, we chose a single message with an average-length signature of 2038 generators, which encodes into 1274 bytes. Then we built our code on the various platforms and measured the time to validate the signature. On some platforms we also tested 256-bit signatures where we tested an average length of 3856 generators (2410 bytes).

The Intel x86_64 platform is one of the most widely used in the world for laptops and servers. The family has dozens of different processors to choose from with varying sets of extensions. For our testing we chose an Intel Xeon X5355 at 2.66GHz. On that platform a 128-bit signature verifies in approximately 200,000 cycles, and a 256-bit signature verifies in approximately 500,000 cycles. On the same platform we ran "openssl speed ecdsap256" to compare the speeds, which resulted in 2283.1 verifications per second, or 0.438ms each, using a highly-optimized assembly routine. The simple WalnutDSA implementation is still 5.8x faster. On the same platform, ECDSA p521 performs 598.4 verifies per second (1.671ms each), which makes WalnutDSA 9x faster at a 256-bit security level.

The ARM Cortex-R5 is a middle-of-the-road embedded processor used for larger embedded systems. We tested on a TI TMS570LC4357ZWT running at 200MHz with plenty of RAM and ROM. On this platform we implemented a combination of C and Assembly to leverage the capabilities of the platform, and compiled using the IAR ANSI C/C++ Compiler, version 8.20.1.14183/W32 for ARM. The verification code compiled down to 2024 bytes and required 1060 bytes of RAM. Signature verification of an average length signature required 483,210 cycles, which works out to 2.4ms. On this same platform, a worst-case signature of 3350 generators required only 706,842 cycles (3.534ms) to verify.

The ARM Cortex-M3 is a much smaller 32-bit embedded microcontroller. We utilitized an STM32F103 (NUCLEO-F103RB board) running at 64MHz using the built-in Atollic TrueSTUDIO 9.0.1 with GCC version: (GNU Tools for ARM Embedded Processors (Build 17.03)) 6.3.1 20170215 (release) [ARM/embedded-6-branch revision 245512]. On this platform we implemented WalnutDSA purely in C without any assembly language optimizations. The verification engine compiled down to 4568 bytes of ROM and executed in only 1116 bytes of RAM. At 128-bit security, an average signature required 1,001,829 cycles to verify (15.67ms). Compare this result to ECC, where [46] showed a full assembly language implementation that required 7168 bytes of ROM and 540 bytes of RAM, but still required 233ms at 48MHz to perform a point multiplication (recall that ECDSA verification requires two). ARM itself produced a report [44] where they measured an ECDSA verification on an LPC1768 in 458ms. With these results, WalnutDSA in C is more than 11x faster than the assembly implementation, and 21.9x faster than ARM's speed reports (when normalized to the same clock speed).

Further, on the M3 we compiled two different ECDSA implementations, micro-ecc and libECC. The libECC implementation required 255,508,762 cycles to verify a SECP256 signature, and 1,155,388,938 cycles to verify SECP521! On the other hand, micro-ecc required only 17,208,994 cycles to verify P256 (358ms at 48MHz), however it does not have an implementation of P521 that could be used to compare against WalnutDSA.

The ARM Cortex-M4 is a step up from the M3. We built a C-only implementation of WalnutDSA and ECDSA and measured the signature validation time on an STMicroelectronics STM32F4-DISCOVERY evaluation board running at 168MGz. When we compiled WalnutDSA with -O3 optimization, validating a WalnutDSA signature at 128-bit security required 817,545 cycles (4.866ms) using 4512 bytes of ROM and 1116 bytes of RAM. Using optimization level -O2 increased the time to 1,092,750 cycles (6.504ms) but decreased the code size to 1692 bytes of ROM (RAM was unaffected). ECDSA at the same security level required 22,184,817 cycles (132ms, leveraging the M4 floating-point features). This yields a 27.1x speed improvement of WalnutDSA over ECDSA. At a

256-bit security level, WalnutDSA required 11,230,464 cycles at -O3 (16473/3764 ROM/RAM) and 10,326,739 cycles and 2592/2280 ROM/RAM at -O2.

The ARM Cortex-M0 is the smallest of the ARM processors. We tested on an Infineon XMC1100 running at 32MHz. WalnutDSA (C-only) compiled down to 4522 bytes of ROM and ran in 1156 bytes of RAM. A 128-bit signature validated in 3,262,152 cycles (101.6ms). On the same platform, microECC curve secp256r1 in only C compiled at -O3 required 12592 bytes of ROM, required 1228 bytes of RAM, and validated a signature in 56,559,519 cycles (1760ms). This is an 17.3x speed improvement of WalnutDSA over ECC. One reason for the lower improvement is the lack of a 32-bit multiplier.

RISC-V is an open-architecture ISA platform with a growing cadre of support from chip manufacturers. For our tests we picked a SiFive FE310 32-bit platform running at 256MHz. On this platform we implemented purely in C an optimized for speed. An average 128-bit signature required 2,207,010 cycles to verify (8.62ms) and an average 256-bit signature required 8,389,555 cycles (32.81ms). On this same platform an ECC P256 signature required 43,559,147 cycles (179ms) for a gain of 19.7x; P521 was not supported.

The Renesas RL78 is a 16-bit microcontroller that runs at 32MHz. On this platform we implemented WalnutDSA using C and also using assembly language. We tuned the compiler to optimize the code for speed, which compiled WalnutDSA into 3,890 bytes; runtime required 1,116 bytes of RAM. The C implementation of WalnutDSA verified a signature in 5,744,917 cycles, and the assembly optiomizations reduced the verification time to only 4,215,741 cycles (131.7ms). On the same platform, ECC P-256 required 13,544 bytes of ROM, 1,230 bytes of RAM, and took 125,565,670 cycles to verify a signature, which shows a 29.8x speedup of WalnutDSA over ECDSA.

## 13. Conclusion

This paper introduced WalnutDSA, a quantum-resistant Group Theoretic public-key signature scheme based on the E-Multiplication one-way function. Key generation is accomplished by producing random T-values and a random braid of a specific form, and then using E-Multiplication to compute the public key. Signature generation involves creating the cloaking elements, building the signature braid, and then running one of the many known braid rewriting algorithms to obscure the form and hide the private key.

At a 128-bit security level the public key is 1438 bits and the private key length ranges from 900 to 1240 bits long (with a maximum theoretical length of 1240 bits). The signatures, after using BKL and Dehornoy braid rewriting techniques, range from 5790 to 16530 bits in length.

In addition, WalnutDSA signature verification proves to be extremely fast, even on small, embedded systems. Verification requires two sets of E-Multiplications, a matrix multiplication, and then a matrix compare. Implementations of WalnutDSA outperformed ECDSA on all platforms tested. Whereas on most platforms WalnutDSA showed an average speed increase of about 23x over an equivalent ECDSA verification. Even on the 16-bit Renesas RL78 this performance improvement was almost 30x.

## References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Appendix A. Performance Matrix

Table 1. Raw WalnutDSA Performance Data (128-bit security level)

| **Platform** | **Clock** | WalnutDSA | | | | ECDSA | | | | **Gain** |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **ROM** | **RAM** | **Cycles** | **Time** | **ROM** | **RAM** | **Cycles** | **Time** | **vs ECDSA** |
| Intel x86_64 | 2.66G | n/a | n/a | 200,000 | 0.075 | n/a | n/a | n/a | 0.438 | 5.8x |
| ARM Cortex R5 | 200 | 2024 | 1060 | 483,210 | 2.4 | n/a | n/a | n/a | n/a | n/a |
| ARM Cortex M4 | 168 | 4512 | 1116 | 817,545 | 4.866 | n/a | n/a | 22,184,817 | 132.05 | 27.1x |
| RISC-V FE310 | 256 | n/a | n/a | 2,207,010 | 8.62 | n/a | n/a | 43,559,147 | 179 | 19.7x |
| ARM Cortex M3 | 48 | 4568 | 1116 | 1,001,829 | 20.89 | 7168 | 540 | n/a | 233 | 11.2x |
| ARM Cortex M0 | 32 | 4524 | 1156 | 3,262,152 | 101.6 | 12592 | 1228 | 56,559,519 | 1760 | 17.3x |
| RL78 (16b) | 32 | 3830 | 1116 | 4,215,741 | 131.7 | 13544 | 1230 | 125,565,670 | 3,924 | 29.8x |

Note that a 'n/a' in Table 1 implies that this data was not available. Also note that "Time" is in milliseconds (ms).

## Appendix B. Example Data

The following sections detail an example of an actual WalnutDSA transaction using low-security parameters. The example uses $N = 10$, $q = 31$, $\kappa = 6$, and $\ell = 124$, which results in a security level of approximately $2^{25}$ against [10] and at least $2^{128}$ against all other known attacks.

For ease of encoding here we represent each Artin generator as a positive or negative integer. For example $b_1$ is represented as 1, and $b_4^{-1}$ is represented as $-4$.

**Private/Public Key Pair.** The private data:

- $a = 1$
- $b = 2$
- Priv(S): `4 9 -4 2 8 -9 2 -6 8 2 -5 -6 3 -2 -8 4 7 5 5 -8 9 -2 -9 -7 -2 -2 7 -3 2 4 1 5 5 5 4 6 8 6 -5 -7 -3 -6 -6 3 2 -6 -5 3 4 -2 -4 -7 -4 -3 6 9 -5 -3 9 -3 -4 -1 7 -2 -3 -2 7 5 -3 -1 -2 8 7 -9 4 9 -3 -9 1 2 -1 4 -9 -8 -1 -9 8 -1 -1 -6 5 -7 8 9 5 7 8 -4 2 -4 8 -4 9 -4 2 6 -5 7 -1 3`

- Priv(S’): `5 9 -8 7 2 2 6 4 2 6 3 -4 -5 4 7 4 5 8 5 9 -4 6 -7 2 5 -4 -2 -2 -4 -3 6 8 -3 -5 -3 9 4 -5 9 9 9 7 -2 9 -4 3 5 4 3 4 3 9 -5 3 -1 -6 2 -7 -8 -4 -8 -3 7 1 4 -3 7 3 8 4 1 -9 3 -5 6 -2 -3 6 -8 -5 7 1 -7 -1 3 2 -6 -2 -9 -5 -4 -4 6 7 -5 -5 -5 9 2 2 -4 7 -2 -3 4 -6 4 3 -4 2`

The public data:

- T-values: `14 11 16 9 12 20 30 8 11 3`
- Pub(S):
  - Matrix:

$$\begin{pmatrix}
4 & 22 & 11 & 8 & 21 & 12 & 11 & 6 & 21 & 21 \\
2 & 2 & 21 & 19 & 24 & 9 & 9 & 13 & 6 & 5 \\
10 & 4 & 19 & 28 & 17 & 10 & 8 & 23 & 11 & 19 \\
22 & 14 & 23 & 3 & 7 & 3 & 15 & 20 & 28 & 18 \\
26 & 16 & 5 & 4 & 23 & 7 & 21 & 2 & 0 & 29 \\
15 & 24 & 26 & 21 & 23 & 4 & 29 & 12 & 28 & 20 \\
17 & 14 & 19 & 5 & 6 & 4 & 6 & 22 & 25 & 11 \\
12 & 19 & 28 & 9 & 17 & 15 & 3 & 20 & 15 & 19 \\
8 & 18 & 3 & 15 & 11 & 14 & 21 & 8 & 2 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

  - Perm: `1 7 5 10 2 6 8 9 3 4`
- Pub(S’):
  - Matrix:

$$\begin{pmatrix}
26 & 27 & 0 & 21 & 0 & 30 & 14 & 15 & 0 & 0 \\
3 & 11 & 21 & 16 & 9 & 10 & 15 & 21 & 5 & 26 \\
11 & 15 & 12 & 11 & 6 & 13 & 11 & 20 & 16 & 30 \\
0 & 11 & 29 & 20 & 0 & 19 & 14 & 29 & 18 & 7 \\
17 & 11 & 19 & 13 & 23 & 20 & 10 & 20 & 24 & 22 \\
5 & 25 & 15 & 22 & 8 & 15 & 23 & 28 & 15 & 6 \\
4 & 26 & 7 & 13 & 20 & 12 & 1 & 10 & 15 & 5 \\
25 & 12 & 28 & 11 & 13 & 3 & 20 & 27 & 10 & 19 \\
28 & 7 & 9 & 21 & 13 & 25 & 23 & 3 & 25 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

  - Perm: `5 1 3 8 9 6 4 10 2 7`

An astute observer will notice the last row is all zeros except for the last element. This is expected and always happens with $E$-Multiplication.

**Example Message.** For the following signature and verfication examples we chose the following random 256-bit string which we treat as the output of a 256-bit hash:

```
a3 c6 1b 0a b3 e4 62 ac 43 d3 4d 6b b3 af 5a b3
1e ee 6d 58 01 75 26 7b 0c f2 e6 2b d7 ea 9a a2
```

**Example Signature and Verification.** For this example we parameterize the encoder using the set of generators specified in §9 from (13). After free reduction, we find that the message becomes the following braid $E(\mathcal{M})$:

```
9 8 7 7 7 6 6 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8 9 9 8 7 6 5 4 3 2 2 3 -4 -5 6 6 5 4
3 3 4 5 -6 -7 8 8 7 6 5 4 3 3 3 2 2 -3 -4 5 6 7 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6
```

```
-7 8 9 9 8 7 6 6 6 5 4 3 3 3 2 2 -3 -4 5 6 6 5 4 3 2 1 1 -2 -3 -4 -5 6 7 7 6 6 7 7
6 5 4 3 2 2 -3 -4 5 5 4 3 2 2 2 1 1 -2 -3 -4 -5 -6 -7 8 9 9 8 7 6 5 4 4 4 3 2 1 1 -2
-3 -4 -5 -6 -7 8 8 7 6 5 5 5 4 3 2 2 -3 -4 -5 -6 7 7 6 5 4 4 5 6 6 5 5 -6 -7 8 8 7
7 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8 8 7 7 7 6 6 7 8 8 7 6 5 5 5 4 4 5 6 7 8 8
7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8 8 7 6 5 4 3 3 4 -5 -6 7 7 6 6 -7 -8 9 9 8 7 6 6
7 7 6 6 6 5 5 6 7 7 6 5 4 4 5 5 4 4 5 5 4 3 2 2 3 3 2 2 2 1 1 -2 -3 4 5 -6 -7 8 8 7
6 5 4 3 3 4 4 3 3 -4 -5 6 6 5 4 3 3 -4 -5 6 6 5 5 -6 -7 8 8 7 6 5 5 -6 -7 8 8 7 6 5
4 3 3 3 2 2 -3 -4 -5 -6 7 7 6 5 4 3 2 2 -3 -4 -5 -6 -7 -8 9 9 8 8 8 7 6 5 4 3 2 1 1
-2 -3 -4 -5 6 -7 -8 9 9 8 7 6 6 6 5 4 3 3 -4 -5 6 6 5 4 3 3 -4 -5 6 6 5 5 -6 -7 8 9
9 8 7 6 5 4 4 4 3 3 -4 -5 -6 -7 8 9 9 8 7 6 6 6 5 5 6 7 7 6 5 4 4 5 6 7 7 6 6 6 5 4
3 2 1 1 -2 -3 -4 -5 6 -7 -8 -9
```

Notice the runs of various generators (i.e., 9 9 8 7 6 6 6 5 4 3 3 3 2 2). These occur because we select neighbor generators to be close together, and because cancellations occur upon performing the free reduction.

After generating cloaking elements, we formed the raw signature $(v_1\mathrm{Priv(S)}^{-1}vE(\mathcal{M})\mathrm{Priv(S')}v_2)$:

```
-4 -5 6 7 -8 -7 -6 -5 -4 -6 7 -8 -7 6 -5 6 7 8 -7 -6 5 8 -2 -3 4 -5 -6 -7 8 7 -8 -7
-2 3 -4 -5 -4 -3 2 -3 -4 -5 -4 3 1 2 3 -2 1 -4 5 6 -7 -6 -5 4 4 5 6 7 -8 9 -8 7 -6
5 8 8 8 8 -5 6 -7 8 -9 8 -7 -6 -5 -4 -4 5 6 7 -6 -5 4 -1 2 -3 -2 -1 -3 4 5 4 3 -2 3
4 5 4 -3 2 7 8 -7 -9 8 7 6 5 4 -3 2 -1 -2 1 3 4 -5 6 5 4 -3 7 7 7 7 3 -4 5 -6 7 6 5
4 6 -7 6 3 -4 -5 6 -5 4 3 -1 -2 -3 4 5 6 -7 8 -7 6 -5 -4 3 3 4 5 6 -7 -8 9 8 -7 6 -5
4 -3 -2 -2 -3 -4 3 -2 3 3 3 3 2 -3 4 3 2 2 3 -4 5 -6 7 -8 -9 8 7 -6 -5 -4 -3 -3 4 5
-6 7 -8 7 -6 -5 -4 3 2 1 -3 -4 5 -6 5 4 -3 -6 7 -6 -4 -5 -6 -7 6 -5 -5 -6 5 -4 -3 -1
2 1 -2 3 -4 -5 -6 -7 -8 9 -8 7 6 5 -4 3 2 -8 -5 6 7 -8 -7 -6 5 -6 7 8 -7 6 4 5 6 7
8 -7 -6 5 4 -3 1 -7 5 -6 -2 4 -9 4 -8 4 -2 4 -8 -7 -5 -9 -8 7 -5 6 1 1 -8 9 1 8 9 -4
1 -2 -1 9 3 -9 -4 9 -7 -8 2 1 3 -5 -7 2 3 2 -7 1 4 3 -9 3 5 -9 -6 3 4 7 4 2 -4 -3 5
6 -2 -3 6 6 3 7 5 -6 -8 -6 -4 -5 -5 -5 -1 -4 -2 3 -7 2 2 7 9 2 -9 8 -5 -5 -7 -4 8 2
-3 6 5 -2 -8 6 -2 9 -8 -2 4 -9 -4 3 4 -5 -4 -3 -6 4 5 6 -5 2 -3 4 5 -4 3 2 6 -7 8 -7
-6 3 -4 5 4 3 -6 7 -8 9 -8 7 6 -1 2 -3 -4 5 6 7 6 -5 4 3 -2 -1 8 8 8 8 1 2 -3 -4 5
-6 -7 -6 -5 4 3 -2 1 -6 -7 8 -9 8 -7 6 -3 -4 -5 4 -3 6 7 -8 7 -6 -2 -3 4 -5 -4 3 -2
5 -6 -5 -4 2 -3 -4 5 -6 7 -8 -7 6 5 -4 -3 -2 -1 -2 3 4 -3 -2 1 5 -6 7 6 -5 -5 -6 -7
-8 9 -8 7 -6 -5 -4 8 8 8 8 4 5 6 -7 8 -9 8 7 6 5 5 -6 -7 -6 -7 8 -7 -2 3 -2 4 -5 6
7 -6 5 4 3 -4 1 2 3 -4 -5 6 -7 -8 9 8 -7 6 5 -4 -3 -2 -1 6 6 6 6 1 2 3 4 -5 -6 7 -8
-9 8 7 -6 5 4 -3 -2 -1 4 -3 -4 -5 6 -7 -6 5 -4 2 -3 2 7 -8 7 6 6 -5 -1 2 3 -4 -3 2
1 2 3 4 -5 -6 7 8 -7 6 -5 4 3 -2 6 3 4 5 -4 -3 9 8 7 7 7 6 6 6 5 4 3 2 1 1 -2 -3 -4
-5 -6 -7 8 9 9 8 7 6 5 4 3 2 2 3 -4 -5 6 6 5 4 3 3 4 5 -6 -7 8 8 7 6 5 4 3 3 3 2 2
-3 -4 5 6 7 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8 9 9 8 7 6 6 6 5 4 3 3 3 2 2 -3
-4 5 6 6 5 4 3 2 1 1 -2 -3 -4 -5 6 7 7 6 6 7 7 6 5 4 3 2 2 -3 -4 5 5 4 3 2 2 2 1 1
-2 -3 -4 -5 -6 -7 8 9 9 8 7 6 5 4 4 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8 8 7 6 5 5 5 4 3 2
2 -3 -4 -5 -6 7 7 6 5 4 4 5 6 6 5 5 -6 -7 8 8 7 7 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6
-7 8 8 7 7 7 6 6 7 8 8 7 6 5 5 5 4 4 5 6 7 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 8
8 7 6 5 4 3 3 4 -5 -6 7 7 6 6 -7 -8 9 9 8 7 6 6 7 7 6 6 6 5 5 6 7 7 6 5 4 4 5 5 4 4
5 5 4 3 2 2 3 3 2 2 2 1 1 -2 -3 4 5 -6 -7 8 8 7 6 5 4 3 3 4 4 3 3 -4 -5 6 6 5 4 3 3
-4 -5 6 6 5 5 -6 -7 8 8 7 6 5 5 -6 -7 8 8 7 6 5 4 3 3 3 2 2 -3 -4 -5 -6 7 7 6 5 4 3
2 2 -3 -4 -5 -6 -7 -8 9 9 8 8 8 7 6 5 4 3 2 1 1 -2 -3 -4 -5 6 -7 -8 9 9 8 7 6 6 6 5
4 3 3 -4 -5 6 6 5 4 3 3 -4 -5 6 6 5 5 -6 -7 8 9 9 8 7 6 5 4 4 4 3 3 -4 -5 -6 -7 8 9
9 8 7 6 6 6 5 5 6 7 7 6 5 4 4 5 6 7 7 6 6 6 5 4 3 2 1 1 -2 -3 -4 -5 6 -7 -8 -9 5 9
```

```
-8 7 2 2 6 4 2 6 3 -4 -5 4 7 4 5 8 5 9 -4 6 -7 2 5 -4 -2 -2 -4 -3 6 8 -3 -5 -3 9 4
-5 9 9 9 7 -2 9 -4 3 5 4 3 4 3 9 -5 3 -1 -6 2 -7 -8 -4 -8 -3 7 1 4 -3 7 3 8 4 1 -9
3 -5 6 -2 -3 6 -8 -5 7 1 -7 -1 3 2 -6 -2 -9 -5 -4 -4 6 7 -5 -5 -5 9 2 2 -4 7 -2 -3
4 -6 4 3 -4 2 -3 -4 5 2 3 -2 4 -5 -6 7 -8 7 6 5 -4 -4 5 -6 -7 8 9 -8 -7 6 6 -5 4 -1
2 -3 -4 5 -6 -7 6 -5 -4 3 2 -1 3 4 5 -6 7 6 -5 4 -3 9 9 9 9 3 -4 5 -6 -7 6 -5 -4 -3
1 -2 -3 4 5 -6 7 6 -5 4 3 -2 1 -4 5 -6 -6 7 8 -9 -8 7 6 -5 4 4 -5 -6 -7 8 -7 6 5 -4
2 -3 -2 -5 -5 6 7 8 -7 -6 5 4 -3 -6 -7 -8 -7 6 5 -6 7 8 -7 6 5 1 -2 3 -4 -5 6 -5 4
-3 2 -1 -7 4 5 -6 5 -4 7 -8 -9 -8 7 2 2 2 2 -7 8 9 8 -7 4 -5 6 -5 -4 7 1 -2 3 -4 5
-3 -4 5 -6 -7 6 5 4 -3 7 2 3 -4 5 -4 3 -2 -1 2 -3 -4 -5 -6 -7 -8 -7 6 5 -4 3 -2 -1
-5 6 7 8 -7 6 5 -4 5 -6 -7 -8 7 -6 5 4 -9 5 5 5 5 9 -4 -5 6 -7 8 7 6 -5 4 -5 -6 7 -8
-7 -6 5 1 2 -3 4 -5 -6 7 8 7 6 5 4 3 -2 1 2 -3 4 -5 4 -3 -2 -7 3 -4 -5 -6 7 6 -5 4
3 -6 5 4 -3 2 -1 -5 -6 7 -8 -7 6 -5 -6 7 8 7 6 3 -4 -5 6 7 -8 -7 -6 5 4 3
```

After running the raw signature through both BKL Normal Form and then Dehornoy reduction we obtain the following 2038-generator braid:

```
5 5 -4 -5 6 6 -4 -5 6 -4 -5 -8 -7 8 8 -6 -5 -3 -4 -6 7 -3 -2 -4 -3 -6 5 -4 -7 -6 -5
6 -4 -5 -6 -8 7 -5 6 6 -7 -3 -2 -9 8 -7 -7 6 -5 -4 -3 -2 -4 7 -6 -5 8 -7 -6 1 1 1 -2
-3 -4 -5 -6 -7 -8 -6 -7 -3 -4 -5 -6 -2 -3 -4 -5 1 1 -2 1 1 -9 -8 -7 -6 2 -7 3 -4 -4
2 -3 1 -2 -2 1 2 5 4 3 -4 -8 -7 6 5 3 4 -9 -8 7 6 8 7 -8 9 5 6 -7 8 2 3 1 2 1 4 5 -6
7 3 4 -5 2 3 -4 2 -3 1 -2 -2 -3 1 -2 3 6 5 4 -5 -6 1 2 3 -4 -5 1 2 -3 -3 -4 1 -2 -2
1 8 -7 -6 -6 -5 -6 7 -8 2 -3 1 -2 1 1 1 -2 -3 -3 -4 -3 -4 -5 -2 -3 -4 -3 -2 -3 -4 -5
1 2 3 4 5 -6 3 4 -5 2 -3 2 3 -4 2 -3 1 -2 -2 -3 -4 -5 -6 -4 -5 -3 -4 -2 -3 1 2 3 4
5 6 -7 -9 -8 4 5 3 4 2 1 1 2 3 4 5 -6 3 4 -5 2 3 1 2 2 3 1 1 2 1 3 -4 2 -3 2 -3 -7
-6 5 -4 1 -2 -3 -4 -4 -2 1 -3 2 1 -7 6 -5 -6 -4 3 4 -5 2 3 -4 -4 2 -3 1 -2 -2 -3 1
-2 1 1 3 -2 1 2 -3 -4 6 5 1 -2 -2 -3 4 5 4 1 -2 3 4 5 -2 1 -3 2 -4 3 2 1 -5 4 5 -6
7 3 2 1 4 -5 3 -4 -4 -5 -5 2 -3 -4 1 -2 -3 -2 6 5 4 3 -4 8 7 6 -5 -6 -7 -2 -3 1 -2
1 -4 -5 -6 -3 -2 3 -4 -5 -7 6 1 2 -3 -4 -4 1 -2 -3 -2 -3 -2 1 -2 1 -2 9 -8 -7 6 5 4
3 -2 1 8 -9 7 6 5 4 3 -4 -5 -6 -7 -8 2 -3 -4 -5 -6 -7 8 9 -5 -6 7 1 -2 -3 -4 -3 -2
-3 -2 1 2 1 3 5 -4 2 -3 -3 6 5 4 -5 2 2 3 1 2 1 3 -4 -4 2 -3 1 2 -3 -3 5 4 -3 2 3 2
1 7 6 5 -6 4 -5 -6 3 -4 2 -3 -3 -4 -5 -3 -4 1 -2 -3 -2 8 7 -6 5 5 5 -4 3 -6 -5 4 -6
5 3 9 8 -9 -7 -7 6 7 5 4 7 7 7 7 6 7 -8 -9 6 -7 -8 5 1 2 1 3 2 5 -6 -7 -8 4 -5 3 -4
5 -4 3 1 3 -4 6 -5 7 -6 9 8 7 9 -8 7 -6 8 -7 -8 2 -3 -4 1 -2 3 -2 1 -2 5 4 -3 6 5 -4
-5 7 -6 1 2 1 3 -4 -5 2 -3 -3 4 -3 2 3 3 3 3 6 5 4 6 -5 3 -4 2 -3 1 -2 -4 -5 3 -2 -2
4 -3 5 -4 -3 4 4 -5 -3 -4 -5 -2 -3 -4 1 -3 -3 -3 -3 2 3 -4 3 5 8 -9 7 -8 -9 -6 -7 -8
-9 -4 -5 -6 -7 -8 2 1 1 3 -4 -5 -6 -7 2 1 3 -4 -4 -5 -6 2 -3 -4 -4 -5 1 -2 -3 4 -3
-2 -2 1 1 1 -2 1 -3 -2 5 -4 -3 -2 1 -3 2 -4 3 6 -5 4 1 8 7 -6 -7 5 -6 4 -5 3 -4 -8
7 8 6 7 5 6 -7 2 -3 4 -3 2 2 5 4 -5 -6 9 8 8 -9 -7 -8 -9 3 -4 -5 2 -3 -4 1 -2 -2 -3
1 -2 3 -4 -2 -3 -4 -5 1 -2 -3 4 -3 -2 1 5 4 3 6 5 -6 -7 -8 -8 -9 -4 -5 -6 2 1 -3 -4
-5 2 -3 -3 -4 1 -2 3 -2 4 3 -2 1 1 1 -2 -3 -4 6 -7 -8 -8 -5 -6 -7 -8 -9 1 -2 -3 -7
-8 -9 -6 -4 1 -2 3 -2 -2 4 3 5 -4 7 6 -5 8 7 -6 8 -7 9 -8 -2 -2 -2 -3 -4 -5 -6 -7 1
-2 3 -2 4 -5 3 -2 1 1 1 -2 1 -3 -4 -5 -6 7 7 7 -8 -2 -3 -4 7 -8 -9 1 -2 -3 5 4 -3 -4
6 6 -5 -2 -3 1 1 -2 3 -2 4 -3 -2 7 6 5 7 -6 -7 -8 -9 5 -6 -6 1 4 3 2 5 4 4 3 5 4 6
-7 5 -6 4 -5 3 -4 2 -3 1 -2 1 6 5 4 3 -2 -3 1 1 -2 8 7 6 5 4 3 -2 1 9 8 7 6 5 -6 -7
-8 -9 4 -5 -6 -7 -3 -4 -5 -6 -7 2 1 2 -3 -4 -5 -6 -8 -9 1 -2 -3 -4 -5 -5 -6 -7 -6 -5
-4 -3 -2 -3 -4 -4 -3 -2 7 6 5 4 -3 5 -4 -5 7 -8 -6 -2 1 2 2 -3 -4 -5 1 -2 -3 4 -3 5
-4 6 -7 9 -8 -5 -6 -7 -8 -9 1 2 3 3 -4 -5 -6 -7 -8 3 3 -4 -5 -6 -7 2 -3 -4 -5 1 -2
-3 -4 -2 -3 -4 -5 -6 8 -7 -8 -4 -5 -3 -4 1 2 2 -3 -4 -5 -6 -7 -8 1 -2 3 -2 4 -3 5 -4
```

```
6 -5 -6 -7 -8 1 1 -2 1 2 3 3 -4 -5 3 3 -4 2 -3 1 -2 3 -2 1 6 5 4 -3 -4 -5 2 2 1 1 2
2 -3 -4 1 -2 -3 -2 -2 1 2 2 5 4 -3 2 2 7 6 5 -4 -3 2 8 7 9 -8 6 -7 -5 -6 -4 -5 3 3
-4 2 -3 1 -2 5 4 3 -2 4 -3 -4 1 1 -2 -3 1 1 -2 6 8 -7 -8 -9 -5 4 3 -2 5 4 3 1 2 3 1
2 3 -4 -5 -6 -7 -8 -9 2 -5 -4 3 2 2 1 1 -6 -5 -4 -3 2 -4 5 3 3 3 4 4 -5 -6 -5 4 5 6
2 1 6 6 6 5 5 5 4 3 3 5 4 6 5 5 -6 4 3 2 1 1 1 1 2 3 4 5 6 6 6 5 6 -7 -8 5 -6 4 -5
3 -4 2 -3 1 -2 -2 -3 5 -4 -5 7 -6 9 8 -7 -7 -2 3 -2 4 -3 5 -4 -5 1 1 -2 3 -2 1 2 2
-3 1 -2 1 3 -4 2 2 2 2 -3 5 -4 1 -2 -3 -2 -3 -4 -4 6 -5 7 -6 9 8 -9 -7 -8 -3 -4 -5
-6 -7 -8 -2 1 2 2 1 1 2 2 -3 -4 -5 1 -2 -3 -4 1 2 2 2 2 -3 1 -2 1 3 4 3 2 5 -6 4 -5
3 3 -4 6 -5 2 -3 -4 5 -4 1 -2 3 -2 4 -3 7 6 5 -4 6 -5 1 2 2 -3 2 3 3 -4 2 2 -5 2 -3
2 4 3 3 2 2 3 5 7 -6 4 4 4 3 4 -5 8 -9 7 -8 -9 -6 -7 3 -4 2 -3 1 -2 5 -6 4 -5 3 -4
-2 1 2 2 2 1 2 -3 1 -2 7 -8 6 -7 5 -6 4 -5 9 8 7 -6 9 8 -7 9 -8 -9 3 -4 5 -4 -2 -3
-4 1 1 -2 -3 4 6 -5 7 -6 -3 -4 -5 -6 8 -7 -8 -9 1 2 2 1 1 1 1 -3 -4 2 2 1 -2 -2 -3
1 1 -2 1 -5 -4 3 2 -5 -6 4 -5 3 3 -4 6 -7 5 -6 2 -3 4 -5 1 -2 1 3 -4 2 2 2 -3 5 4 6
5 1 -2 1 5 7 -6 -7 -8 4 -5 -6 -7 9 4 4 -5 -6 -6 -5 4 5 6 6 5 4 3 3 5 4 6 -7 5 -6 7
7 7 8 8 7 -6 5 4 4 5 3 3 4 3 -2 1 7 6 5 4 -3 2 3 1 2 8 7 6 5 4 -5 6 8 9 3 -5 -4 -4
2 -7 8 9 1 1 5 -4 -5 3 -4 3 -6 5 4 -5 6 3 -4 5 7 8 6 7 3 -4 5 6 9 8 7 2 3 1 2 5 4 2
6 5 7 2 3 4 1 2 3 5 4 2 2 7 8 9 7 6 5 8 7 6 4 5 3 2 1 3 3 3 -4 3 2 7 6 7 8 6 -7 6 9
-8 7 8 2 4 3 7 6 5 4 8 7 6 5 7 -6 -6 8 -7 -6 -7 -6 3 4 3 2 1 9 8 7 4 1 5 6 4 5 3 4
7 -8 -9 6 5 7 -8 6 4 3 3 -2 1 2 7 7 6 4 3 2 1 4 5 7 8 9 7 8 6 7 -3 8 -2 -2 1 2 -3 5
4 6 5 -6 -3 4 2 3 4 -7 -6 5 -7 6 7 4 -5 4 3 -4 3 8 3 9 3 -4 5 6 3 2 1 8 -4 5 3 3 4
6 5 4 2 1 4 -7 6 7 5 6 3 4 5 2 3 4 7 6 8 1 -5 4 5 2 1 2 3 4 2 1 1 6 5 4 3 -4 6 7 2
5 -4 6 -5 -4 -3 8 7 -6 -5 4 5 6 2 1 9 8 7 3 4 5 6 -2 -2 3 1 2 3 4 2 9 8 -9 7 6 5 4
3 2 1 6 8 7 9 8 -5 -4 -3 -2 -6 -5 -4 -3 -2 -7 -6 -5 6 -4 -3 -2 1 -3 2 -4 3 -5 4 -6
5 6 7 4 5 6 3 3 4 5 6 5 6 2 1 -2 3 1 2 -4 5 3 2 3 4 5 8 7 9 5 4 3 2 1 6 7 -5 4 3 2
-6 5 4 5 -8 -7 -7 6 4 -5 -5 6 6 6 -5 6 6 7 3 4 5 4 7 8 6 7 5 6 4 5 3 4 2 -3 2 7 6 5
-6 -4 3 2 1 -5 4 -3 2 -6 5 6 7 8 4 5 6 2 3 4 5 6 7 1 -2 3 4 5 6 -2 1 -3 2 -4 3 4 5
6 1 2 3 4 1 2 3 2 2 2 7 6 5 8 7 1 5 4 3 2 1 6 5 4 3 2 7 6 5 4 3 8 8 -7 6 7 5 5 4 3
7 8
```

Notice that one sees runs of generators after this process. This again reflects the structure of the message encoding algorithm. In particular, the Dehornoy reduction algorithm works by replacing certain subwords of the form $\pm\mathtt{i}, \ldots, \mp\mathtt{i}$ with new words, and that ultimately words of the form $\pm\mathtt{j}, \ldots, \pm\mathtt{j}$ with $\mathtt{j} < \mathtt{i}$ tend to survive to the end. This explains the appearance of these generators in the obscured signature. We remark that even though these runs resemble those seen in the encoded message $E(\mathcal{M})$, they are not part of $E(\mathcal{M})$, and thus no hidden information from the raw signature is revealed.

To validate this signature, one first needs to compute the E-Multiplication $(Id_N, Id_{S_N}) \star E(\mathcal{M})$ which results in the following matrix:

$$\begin{pmatrix}
29 & 28 & 26 & 6 & 23 & 26 & 8 & 4 & 28 & 0 \\
19 & 28 & 6 & 18 & 18 & 2 & 11 & 30 & 14 & 23 \\
25 & 10 & 27 & 22 & 6 & 8 & 7 & 16 & 15 & 28 \\
0 & 8 & 12 & 3 & 4 & 12 & 25 & 5 & 16 & 26 \\
16 & 3 & 3 & 20 & 5 & 4 & 7 & 11 & 11 & 12 \\
27 & 1 & 30 & 2 & 29 & 24 & 26 & 29 & 19 & 7 \\
13 & 24 & 17 & 14 & 25 & 2 & 21 & 7 & 24 & 4 \\
21 & 0 & 7 & 5 & 10 & 18 & 15 & 21 & 26 & 26 \\
24 & 16 & 3 & 27 & 8 & 15 & 2 & 22 & 7 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Note the zeros in the last row. This, too, is expected because of the way $E$-Multiplication works.

Next, one multiplies that matrix by the matrix part of Pub(S'), which results in the following matrix:

$$\begin{pmatrix}
19 & 19 & 27 & 1 & 16 & 5 & 28 & 15 & 17 & 4 \\
7 & 24 & 24 & 24 & 22 & 0 & 24 & 10 & 8 & 29 \\
14 & 7 & 23 & 26 & 5 & 10 & 6 & 19 & 15 & 30 \\
27 & 10 & 29 & 19 & 20 & 28 & 18 & 11 & 23 & 8 \\
27 & 5 & 19 & 18 & 29 & 24 & 26 & 5 & 8 & 2 \\
2 & 2 & 4 & 0 & 22 & 24 & 1 & 26 & 4 & 27 \\
10 & 8 & 8 & 22 & 27 & 9 & 12 & 23 & 27 & 13 \\
26 & 3 & 4 & 8 & 25 & 9 & 4 & 22 & 30 & 0 \\
27 & 10 & 24 & 12 & 15 & 5 & 0 & 26 & 30 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Finally, one computes the E-Multiplication Pub(S) $\star$ Sig, which results in the following matrix:

$$\begin{pmatrix}
19 & 19 & 27 & 1 & 16 & 5 & 28 & 15 & 17 & 4 \\
7 & 24 & 24 & 24 & 22 & 0 & 24 & 10 & 8 & 29 \\
14 & 7 & 23 & 26 & 5 & 10 & 6 & 19 & 15 & 30 \\
27 & 10 & 29 & 19 & 20 & 28 & 18 & 11 & 23 & 8 \\
27 & 5 & 19 & 18 & 29 & 24 & 26 & 5 & 8 & 2 \\
2 & 2 & 4 & 0 & 22 & 24 & 1 & 26 & 4 & 27 \\
10 & 8 & 8 & 22 & 27 & 9 & 12 & 23 & 27 & 13 \\
26 & 3 & 4 & 8 & 25 & 9 & 4 & 22 & 30 & 0 \\
27 & 10 & 24 & 12 & 15 & 5 & 0 & 26 & 30 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

which is obviously equal to the previous matrix by inspection. We expect there to be $q^{N(N-1)+1} = 2^{455}$ possible matrices.

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
*E-mail address*: ianshel@securerf.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
*E-mail address*: datkins@securerf.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
*E-mail address*: dgoldfeld@securerf.com

SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484
*E-mail address*: pgunnells@securerf.com