# WalnutDSA™: A Quantum Resistant Group Theoretic Digital Signature Algorithm

Iris Anshel, Derek Atkins, Dorian Goldfeld, and Paul Gunnells

SecureRF Corporation
100 Beard Sawmill Rd #350, Shelton, CT 06484
ianshel@securerf.com, datkins@securerf.com, dgoldfeld@securerf.com, pgunnells@securerf.com

**Abstract.** In 2005 I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux introduced E-Multiplication, a quantum-resistant group-theoretic one-way function which can be used as a basis for many different cryptographic applications. For example, Anshel and Goldfeld recently introduced AEHash, a cryptographic hash function constructed from E-Multiplication and later defined an instance thereof called Hickory Hash.

This paper introduces a new public key method based on E-Multiplication, called WalnutDSA. WalnutDSA provides efficient verification, allowing low-power and constrained devices to more quickly and inexpensively validate digital signatures (e.g. a certificate or authentication). This paper introduces the construction of the digital signature algorithm, analyzes the security of the scheme, and discusses the practical results from an implementation.

**Keywords:** Algebraic Eraser, Group Theoretic Cryptography, Digital Signature, E-Multiplication, Braids

## 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.

Digital signatures based on hard problems in group theory are relatively new. In 2002, Ko, Choi, Cho, and Lee [22] proposed a digital signature based on a variation of the conjugacy problem in non-commutative groups. In 2009, Wang and Hu [31] introduced a digital signature with security based upon the hardness of the root problem in braid groups. See also [20]. The attacks introduced in [11], [12], [14], and [19] suggest that these schemes may not be practical over braid groups in low resource environments.

### Previous Work

E-Multiplication [4] is a group-theoretic, one-way function first introduced by I. Anshel, M. Anshel, D. Goldfeld, and S. Lemieux in 2005 [4]. 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 general-purpose, quantum-resistant one-way function; its use is broader than the 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].

Implementations of E-Multiplication in various instances have shown that 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 E-Multiplication and a new hard problem in braid groups (see the cloaked conjugacy search problem in §8). WalnutDSA appears immune to all the types of attacks related to the conjugacy search problem given in [11], [12], [14], and [19], as well as the recent attacks on the original 2005 key agreement construction noted in [5], [21], and [26].

The underlying hard problems on which the security of the digital signature rests are reversing E-Multiplication and solving a novel equation over the braid group.

E-Multiplication is rapidly executable, even in the smallest of environments, and as a result, WalnutDSA provides very fast signature verification. We have implemented and shown WalnutDSA's performance in various environments, and it outperformed ECDSA by orders of magnitude in all cases we tried.

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; and Sixth, it tests WalnutDSA's size and performance characteristics.

## 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 $\epsilon_j \in \{\pm 1\}$.

Each braid $\beta \in B_N$ determines a permutation in $S_N$ (group of permutations of $N$ letters) 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 $\{1, \ldots, i-1, i+2, \ldots, N\}$ fixed. Then $\sigma_i$ is associated to the Artin generator $b_i$. Further, if $\beta \in B_N$ is written as in (3), we take $\beta$ to be associated to the permutation $\sigma_\beta = \sigma_{i_1} \cdots \sigma_{i_k}$. A braid is called pure if its underlying permutation is trivial.

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

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

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

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

First, we define the colored Burau matrix (denoted $CB$) of each Artin generator as follows.

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

For $2 \leq i \leq N-1$, the matrix $CB(b_i)$ is defined 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 $i$, and if $i = 1$ the leftmost $t_1$ is omitted.

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 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 $f(t_1, \ldots, t_N)$ in $N$ variables, a permutation in $\sigma \in S_N$ can act (on the left) by permuting the indices of the variables. We denote this action by $f \mapsto {}^{\sigma}f$:

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

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

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

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

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

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

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

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 [4] as a one-way function used as a building block to create multiple cryptographic constructions. We recall its definition here.

A set of T-values is defined to be a collection of non-zero field elements:

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

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

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

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

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

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

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

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

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

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

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

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

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

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*

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

*The cloaking element is defined by the property that it essentially disappears when performing E-Multiplication.*

It is not immediately obvious how to construct cloaking elements of braid words. The following proposition addresses this issue.

**Proposition 4.2** *Fix integers $N \geq 2$, and $1 < a < b < N$. Let $m \in GL(N, \mathbb{F}_q)$. Let $\sigma_\beta \in S_N$ have the property that it fixes $a, b$. Then a cloaking element $v$ of $(m, \sigma)$ is given by $v = wb_i^2w^{-1}$ where $b_i$ is any Artin generator $(1 \leq i < N)$ and $w \in B_N$ has associated permutation which move $i \to a,\ i+1 \to b$.*

Since cloaking elements of an ordered pair $(m, \sigma) \in GL(N, \mathbb{F}_q) \times S_N$ act as stabilizing elements of $(m, \sigma)$, when viewing E-Multiplication as a right action, the following proposition is immediate:

**Proposition 4.3** *The set of braids that cloak a specific ordered pair $(m, \sigma)$ form a subgroup of $B_N$.*

## 5 Key Generation

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.

**Public Information:**

- An integer $N \geq 8$ and associated braid group $B_N$.
- A rewriting algorithm $\mathcal{R}\colon B_N \to B_N$ such as [7] or [9].
- A finite field $\mathbb{F}_q$ of $q \geq 32$ elements.
- 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 $\tau_a = \tau_b = 1$.
- An integer $L$ that determines the minimal length of certain random braid words ($L$ is derived from the desired security level).

**Signer's Private Key:**

The Signer's Private Key is a braid of the form:

- $\text{Priv(S)} = w_1 \cdot w(a,b) \cdot w_2$.

Here $w_1, w_2$ are random products of at least $L$ generators of the pure braid subgroup of $B_N$, and where $w(a,b) \in B_N$ is a random braid whose associated permutation fixes $a, b$.

The pure braid subgroup of $B_N$ is generated [17] by the set of $(N-1)(N-2)/2$ braids given by:

$$g_{i,j} = b_{j-1}b_{j-2}\cdots b_{i+1} \cdot b_i^2 \cdot b_{i+1}^{-1}\cdots b_{j-2}^{-1}b_{j-1}^{-1}, \qquad 1 \le i < j \le N. \tag{7}$$

To create the private key of the signer, one first chooses $w_1, w_2$ as random products of $L$ of the generators given in (7). One can compute $L$ from the desired security level (in bits) by computing $L = \lceil (\text{SecurityLevel})/(2\log_2((N-1)(N-2)))\rceil$. For example, suppose 128-bit security is desired, and the braid group is $B_8$. Then $L = \lceil 128/(2\log_2(7\cdot 6))\rceil = 12$. Next one constructs $w(a,b)$ as described earlier, and the private key of the signer is computed to be $w_1 \cdot w(a,b) \cdot w_2$.

**Signer's Public Key:**

The Signer's Public Key is a matrix and permutation generated from the Private Key via E-Multiplication:

- $\text{Pub(S)} = \left(\text{Id}_N, \text{Id}_{S_N}\right) \star \text{ Priv(S)}$,

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

## 6 Message Encoder Algorithm

In order to generate a secure signature and prevent certain types of merging attacks, one must carefully convert the message to be signed into a braid word. The following encoding method handles messages of fixed length which are digests of a cryptographic hash function applied to a message of arbitrary length.

One requirement of any encoding algorithm is that distinct messages should be encoded as distinct braid words. Another important requirement is that $E(\mathcal{M})E(\mathcal{M}')$ should be different from $E(\mathcal{M}\mathcal{M}')$: the result of encoding the concatenation of two messages should not be the concatenation of the encodings of the individual messages.

To satisfy these requirements, we shall realize the message as an element of a free subgroup of $B_N$. A free subgroup is where a reduced element (a word where the subwords $x \cdot x^{-1}$, and $x^{-1} \cdot x$ do not appear) is never the identity. In the case of the braid group, there are subsets of pure braids that generate free subgroups. 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 nontrivial, reduced elements in a free subgroup of the pure braid group. This requirement ensures that unique messages will result in unique encodings.

The encoding algorithm we present is based on the following classical observation: the collection of pure braids given by

$$\begin{aligned} x_1 &= g_{(N-1),N} = b_{N-1}^2 \\ x_2 &= g_{(N-2),N} = b_{N-1} \cdot b_{N-2}^2 \cdot b_{N-1}^{-1} \\ x_3 &= g_{(N-3),N} = b_{N-1}b_{N-2} \cdot b_{N-3}^2 \cdot b_{N-2}^{-1}b_{N-1}^{-1} \\ x_4 &= g_{(N-4),N} = b_{N-1}b_{N-2}b_{N-3} \cdot b_{N-4}^2 \cdot b_{N-3}^{-1}b_{N-2}^{-1}b_{N-1}^{-1} \\ &\vdots \\ x_{N-1} &= g_{1,N} = b_{N-1}b_{N-2} \cdots b_2 \cdot b_1^2 \cdot b_2^{-1}b_3^{-1} \cdots b_{N-1}^{-1}, \end{aligned} \tag{8}$$

generate a free subgroup $B_N$ [6]. Since any subset of the above free generators will itself freely generate a subgroup we can leverage the pure braids above and create an encoding mechanism that maps an input message to a unique braid word.

**Message Encoder Algorithm:** We assume that the message $\mathcal{M}$ is a hash digest of $4\ell$ bits. Choose and fix a subset of four generators

$$\{g_{k_1,N}, g_{k_2,N}, g_{k_3,N}, g_{k_4,N}\} \subset \{g_{1,N}, g_{2,N}, \ \cdots \ , g_{(N-1),N}\}.$$

Each 4-bit block of $\mathcal{M}$ then determines a unique power of one of these specifies generators $g_{k_\mu,N}^i$ with $1 \leq i \leq 4$; the two lowest bits determine the generator $g_{k_\mu,N}$ to use, and the two high bits determine the power $1 \leq i \leq 4$ to raise the generator to. The output $E(\mathcal{M})$ of the message encoder is then the freely reduced product of these $\ell$ generators.

An astute reader will note that on its own this encoding method fails our second requirement above. In other words, in this encoding method we have $E(\mathcal{M})E(\mathcal{M}') = E(\mathcal{M}\mathcal{M}')$ for all messages $\mathcal{M}, \mathcal{M}'$.

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 $H(\mathcal{M})H(\mathcal{M}')$ will never equal $H(\mathcal{M}\mathcal{M}')$. We also know it is unlikely to find two classes of hash functions $H1$, $H2$ such that the output size of $H1$ is half the output size of $H2$, and then to further find three messages $\mathcal{M}$, $\mathcal{M}'$, and $\mathcal{M}''$ such that $H1(\mathcal{M})$ $H1(\mathcal{M}')$ results in the same output[1] as $H2(\mathcal{M}'')$, and also get a signer to sign both messages $\mathcal{M}$ and $\mathcal{M}'$ using $H1$. We also note that including a hash algorithm identifier in the message after it is hashed would prevent this attack.

## 7 Signature Generation and Verification

To sign a message $\mathcal{M}$ the Signer performs the following steps:

[1] For a weak hash $H1$ and a strong hash $H2$, which has twice the output size of $H1$, an attacker would need to find two messages $\mathcal{M}$ and $\mathcal{M}'$ that are preimages to the halves of $H2$ of the desired forgery and then get the signer to use $H1$ and sign both $\mathcal{M}$ and $\mathcal{M}'$. E.g. the attacker would need to take his or her desired forged message, hash it using SHA2-256, find two preimages with MD5, get the signer to sign those MD5 preimages, and only then can he or she compose a message that would verify with SHA2-256.

**Digital Signature Generation:**

**1.** Choose an integer $1 < i < n$.

**2.** Generate the cloaking elements $v$ and $v_1$ as defined in §4.

**3.** Generate the encoded message $E(\mathcal{M})$.

**4.** Compute $\mathrm{Sig} = \mathcal{R}\big(\mathrm{Priv(S)}^{-1} \cdot v \cdot E(\mathcal{M}) \cdot \mathrm{Priv(S)} \cdot v_1\big)$, which is a rewritten braid.

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

*As addressed earlier, the cloaking elements $v, v_1 \in B_n$ contain a random product of at least $2L$ pure braid generators, and disappear when the signature is E-Multiplied by the public key, Pub(S). Also note that the signature Sig could be sent separately from the message $\mathcal{M}$.*

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

**1.** Generate the encoded message $E(\mathcal{M})$.

**2.** Define $\mathrm{Pub}(E(\mathcal{M}))$ by

$$\mathrm{Pub(E}(\mathcal{M})) \;=\; \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big) \star E(\mathcal{M}).$$

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

**3.** Evaluate the E-Multiplication $(\mathrm{M}_{PubSig}, \sigma_{PubSig}) = \mathrm{Pub(S)} \star \mathrm{Sig}$.

**4.** Verify the equality

$$\mathrm{MatrixPart}\big(\mathrm{Pub(S)} \star \mathrm{Sig}\big) = \mathrm{MatrixPart}\big(\mathrm{Pub(E}(\mathcal{M}))\big) \cdot \mathrm{MatrixPart}\big(\mathrm{Pub(S)}\big),$$

where the matrix multiplication on the right is performed over the finite field. The signature is valid if this equality holds. If the results are not equal then the signature validation has failed.

## 8 Security Discussion

The security of WalnutDSA is based on the following two highly non-linear problems which we believe are computationally infeasible for sufficiently large key sizes.

**Problem (1) (Reversing E-Multiplication is hard)** *Consider the braid group $B_N$ and symmetric group $S_N$ with $N \geq 8$. Let $\mathbb{F}_q$ be a finite field of $q$ elements with $q \geq 32$, and fix a set of non-zero $T$-values $\{\tau_1, \tau_2, \ldots, t_N\}$ in $\mathbb{F}_q$. Let $\beta \in B_N$ and set $(m, \sigma) \in (GL(N, \mathbb{F}_q),\ S_N)$ where $(m, \sigma) = (Id_N, Id_{S_N}) \star \beta$. It is infeasible to determine $\beta$ from $(m, \sigma)$ if the expression for the rewritten form of $\beta$ has length at least 2000 Artin generators.*

If we consider $\beta$ varying over $B_N$, the entries of $CB(\beta)$ are Laurent polynomials in $N$ variables of arbitrarily high degree. Thus computing $CB(\beta)$ for long braids $\beta$ becomes very inefficient, even though the colored Burau matrices themselves are very simple. An attempt to reverse E-Multiplication by evaluating products of CB matrices and then trying to solve the multivariable equations that would emerge would rapidly become unmanageable. It is, in

fact, the rapid growth of these Laurent polynomial entries combined with the permutation of their variables that leads us to the conjecture that E-Multiplication is hard to reverse.

Further strong support for the hardness of reversing E-Multiplication can be found in [25] which studies the security of Zémor's [33] hash function $h : \{0,1\}^* \to SL_2(\mathbb{F}_q)$, with the property that $h(uv) = h(u)h(v)$, 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 [25] 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.

**Problem (2) (Cloaked Conjugacy Search Problem (CCSP))** *Consider the braid group $B_N$ and symmetric group $S_N$ with $N \geq 8$. Let $Y, v, v_1 \in B_N$ be unknowns where $v$ cloaks $(Id_N, Id_{S_N})$ and $v_1$ cloaks $(Id_N, Id_{S_N}) \star Y$. Assume $A \in B_N$ and $\mathcal{R}(Y^{-1}\, v\, A\, Y\, v_1)$ are known. Then it is infeasible to determine $Y$ if the expression for the rewritten form of $Y$ has length at least 2000 Artin generators.*

Note that if the cloaking elements $v, v_1$ are trivial then CCSP reduces to the ordinary conjugacy search problem (CSP). If an attacker can determine the cloaking elements $v, v_1$ then it is easy to see that CCSP again reduces to CSP and fast methods for solving for $Y$ were obtained in [12] provided the super summit set of the conjugate $Y$ was not too large.

It does not seem possible to mount a length attack of the type proposed in [26] to solve CCSP. First of all, as pointed out in [16], the length attack only works effectively for short conjugates. Secondly, the placement of the unknown cloaking element $v, v_1$ in the braid word $Y^{-1}\, v\, A\, Y\, v_1$ completely thwarts any type of length attack.

Note that definitionally a rewriting method will obscure the cloaked conjugate, making it impossible to read off $Y^{-1}$ from the rewritten result. However Y, like any braid, is not uniquely determined; there are infinite forms that any equivalent braid can take. Rewriting uses this to its advantage because $Y$ and $Y^{-1}$ can effectively be rewritten into different forms with different pieces get mixed together and making separation intractable.

In fact, at present there does not seem to be any tractable method to solve CCSP in sub exponential time.

### Attacks on the underlying math

The recent attack of Ben-Zvi–Blackburn–Tsaban [5] based on ideas in [21] does not seem to apply to WalnutDSA because the signature is a braid and the public key is coming from E-Multiplication of the identity element with a braid that has very little algebraic structure. As a result it does not seem possible to apply a linear algebraic attack as in [5] to solve the hard problems (1) and (2) above, or to forge a signature. See also [2], which provides methods to defeat the attack in [5], and [13] which shows how to defeat the attack in [21].

The more recent attack of Blackburn–Robshaw [8] seems completely irrelevant to WalnutDSA. Their paper does not even break the original algebraic eraser key agreement protocol. See [1] which provides a simple way to defeat the attack by simply adding a hash or MAC challenge/response to the authentication protocol. What Blackburn and Robshaw have found is an invalid public key attack similar to the invalid elliptic curve attacks on ECC.

**Attacks on WalnutDSA**

We now discuss various possible attacks on WalnutDSA following the terminology in [29] and why we perceive them to be ineffective due to their direct connection with the hard problems CCSP and reversing E-Multiplication.

- Total Break – Obtaining the private key of the signer

  The most straightforward attack is to reverse E-Multiplication and obtain the private key of the signer from the signer's public key.

  The signer public key is, by definition, $\mathrm{Pub(S)} = \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big) \star \mathrm{Priv(S)}$. If the attacker can reverse E-Multiplication and obtain another braid group element S′ such that $\mathrm{Pub(S)} = \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big) \star \mathrm{Priv(S')}$, then the attacker can use S′ to easily forge signatures.

  To date, no attacks have been found that can reverse E-Multiplication that are better than exponential.

  The attacker may also try to solve for Priv(S) using the signature, i.e., solving the equation below in the braid group.

  $$\mathrm{Sig} = \mathcal{R}\big(\mathrm{Priv(S)}^{-1} \cdot v \cdot E(\mathcal{M}) \cdot \mathrm{Priv(S)} \cdot v_1\big). \tag{9}$$

  This is precisely the presumed hard problem CCSP.

- Universal forgery or (weak) existential forgery

  Constructing an efficient algorithm that is able to sign any message with significant probability of success is called universal forgery. Providing a single message/signature pair is called existential forgery.

  The attacker may try to forge a signature of the form $Y^{-1}AY$ with $Y, A \in B_N$ where the permutation of $Y$ is the same as the permutation of the public key and $A$ is a pure braid. The verification equation then takes the form

  $$\mathrm{Pub}(S) \star Y^{-1}AY = \mathrm{Pub}(E(\mathcal{M})) \star \mathrm{Priv}(S)$$

  which implies

  $$\big(\mathrm{Pub}(S) \star Y^{-1}\big) \star A = \Big(\mathrm{Pub}(E(\mathcal{M})) \star \mathrm{Priv}(S)\Big) \star Y^{-1}.$$

  Now, the permutation associated to the braid $\mathrm{Pub}(S) \star Y^{-1}$ is trivial so that E-Multiplication of $\mathrm{Pub}(S) \star Y^{-1}$ and $A$ reduces to ordinary matrix multiplication of the matrix entries. The attacker may choose various $Y$ and then try to solve for $A \in B_N$

since the attacker knows both $\mathrm{Pub}(S) \star Y^{-1}$ and $\Big(\mathrm{Pub}(E(\mathcal{M})) \star \mathrm{Priv}(S)\Big) \star Y^{-1}$ which implies that the matrix component of $(\mathrm{Id}_N, \mathrm{Id}_{S_N}) \star A$ is given by the ratio of the matrix components of these known elements. But this reduces to the hard problem of trying to reverse E-Multiplication for every choice of $Y$.

- Non-repudiation

  Non-repudiation is defined to be the property of a digital signature scheme which asserts that the signer of a message cannot later claim not to have signed the message. This is tantamount to saying that it should be infeasible to produce two distinct messages with the same signature.

  We now present an argument to show that it is infeasible for the signer of a message (who is in possession of the public/private WalnutDSA key) to produce two distinct messages with the same signature. The signer must produce messages $\mathcal{M},\ \mathcal{M}'$ and cloaking elements $v, v', v_1, v_1'$ such that

$$v \cdot E(\mathcal{M}) \cdot \mathrm{Priv(S)} \cdot v_1 \ = \ v' \cdot E(\mathcal{M}') \cdot \mathrm{Priv(S)} \cdot v_1'. \tag{10}$$

  By proposition 4.3 the elements $v, v'$ and $v_1, v_1'$ lie in two respective subgroups of $B_N$, while the elements $E(\mathcal{M}),\ E(\mathcal{M}')$ lie in a free subgroup of the pure braid group and appear never to be cloaking elements. In the simpler case when $v_1 = v_1'$ this implies there is no solution to equation (10). More generally it seems infeasible to solve (10) for the key sizes suggested in this paper.

- 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

$$\mathrm{Sig} = \mathcal{R}\big(\mathrm{Priv(S)}^{-1} \cdot v \cdot E(\mathcal{M}) \cdot \mathrm{Priv(S)} \cdot v_1\big). \tag{11}$$

  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.

- Using brute-force to solve for the various secret elements.

We now discuss the security level of the individual secret components which are used to create the digital signature of a message $\mathcal{M}$. For accuracy we give the following definition of *security level*:

**Definition 8.1 (Security Level):** *A secret is said to have security level $2^k$ 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 $\mathbb{F}$.*

**Brute force security level of the cloaking elements, $v, v_1$:** The pure braid subgroup of $B_N$ is generated by the set of $(N-1)(N-2)/2$ braids given by

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

The braid elements $v, v_1$ are defined to be conjugates of $b_i^2$ by lifts of permutations that move $i\to a$, $i+1\to b$ times a random word in the pure braid subgroup of length at least $L$. The number of words of length $L$ in the above generators (7) of the pure braid subgroup is given by

$$\left(2\cdot\frac{(N-1)(N-2)}{2}\right)^L \;=\; \big((N-1)(N-2)\big)^L.$$

Hence, a lower bound for the security level of the pair $v, v_1$ of the cloaking elements, is given by

$$\big((N-1)(N-2)\big)^{2L},$$

assuming an attacker does a brute force search of the set of all possible pairs of such cloaking elements.

**Remarks:** To date there is no known method to efficiently enumerate all distinct braid elements of length $L$ in the generators $g_{ij}$ given in (7). Consequently, to perform the above attack, an attacker must execute a brute force search of all possible words in the generators as described above. It is possible to speed up the running time of the brute force search for the cloaking elements by use of a Pollard 'lambda or rho' -style algorithm based on the birthday paradox. Since these methods are very well known we do not give them here.

**Brute force security level of Priv(S):** Recall that the signer private key is of the form $w_1\cdot w(a,b)\cdot w_2$ where $w_1, w_2$ are words of length at least $L$ in the generators of the pure braid subgroup of $B_N$. Assuming an attacker does an ordinary brute force search of all possible private keys, it follows that the security level is at least the number of all possible pairs $w_1, w_2$, which is

$$\big((N-1)(N-2)\big)^{2L},$$

as above. Because the braid-words from (7) cannot be enumerated efficiently, the only way to determine Priv(S) is by a brute-force search.

**Search space of the Public Key Pub(S):** Recall that the signer public key is computed by $\mathrm{Pub(S)} = \big(\mathrm{Id}_N, \mathrm{Id}_{S_N}\big)\star\ \mathrm{Priv(S)}$. When this is evaluated over $B_N, \mathbb{F}_q$ it results in an $N\times N$ matrix with $q$ possible elements in each entry. The last row, however, is all zeros (except for the final element)., which implies there are

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

possible public keys available. 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 §8, the security of WalnutDSA is based on the hard problems of reversing E-Multiplication and solving the cloaked conjugacy search problem (CCSP). The math behind these hard problems 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 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 [28] 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 on the Hidden Subgroup Problem for finite cyclic and other finite abelian groups (see [23]). Since the braid group does not contain any non-trivial finite subgroups at all, there does not seem to be any viable way to connect to connect CCSP with HSP.

Given an element

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

where $i_j \in \{1, \ldots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$, we can define a function $f\colon B_N \to GL(N, \mathbb{F}_q)$ where $f(\beta)$ is given by the E-Multiplication $(1,1)\star(\beta, \sigma_\beta)$ and $\sigma_\beta$ 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 [15] which can find an element in an unordered $N$ element set in time $\mathcal{O}\big(\sqrt{N}\big)$. 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.

## 9 Size and Performance Characteristics

To test WalnutDSA we wrote key and signature generation and validation software in C (and on one platform implemented part of the verification engine in assembly). We ran the signature generation on a Thinkpad T540p laptop running Fedora Linux to generate 100 keypairs, and for each key generated 50 random 256-bit messages and the resulting signatures. For the signature rewriting we used a combination of the Birman–Ko–Lee (BKL) [7] and Dehornoy [9] algorithms to obscure the braids and shorten them to reasonable lengths.

For our testing we settled on the parameters:

- $N = 8$
- $q = 32$
- $L = 18$

which yields a private key security level of at least $2^{128}$ against brute force attacks,[2] with a public keyspace of $2^{285}$ possible public keys.

The public keys are always a fixed size. They need to include the T-Values, Matrix, and Permutation which requires

$$N \log_2(q) + (N(N-1)+1)\log_2(q) + N\log_2(N) = 40 + 285 + 24 = 359 \text{ bits.}$$

Private keys and signatures, however, are variable length. The 100 private keys varied in length from 58 generators to 114 generators, with a mean of 85.8 and a standard deviation of 9.6.

Using those 100 keys we generated 5000 signatures using input (hash output) messages of 256 bits. Of these 5000 signatures, their lengths varied from 604 to 1612 generators, with a mean of 1072.9 and a standard deviation of 138.1. These signatures also require 4 bits per generator, which results in signatures of length of 2416 to 6448 bits (with an average of 4291.6 bits).

### 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: a Silicon Industries 8051 8-bit microcontroller, a Texas Instruments (TI) MSP430F5172 16-bit microcontroller, an ARM Cortex M3 (NXP LPC1768), and as a hardware accelerator for an Altera Cyclone V and a Microsemi Smartfusion 2). The implementation on the MSP430 and ARM is fully in C but has not been optimized in any way; on the 8051 we implemented the underlying E-Multiplication engine in assembly.

To provide a common testing platform we chose a single message with an average-length signature of 1074 generators, which encodes into 537 bytes. Then we built our code on the various platforms and measured the time to validate the signature.

On the MSP430 we built with TI's GCC compiler version 4.9.1 (20140707) using the -O3 compiler option. The compiled code took up only 3244 bytes of ROM and required only 236 bytes of RAM to process the signature. The signature verification required 370944 cycles. At a clock speed of 8MHz this equates to 46ms. Compare this to ECC Curve25519, which requires two seconds to compute an ECDSA validation (extrapolated from a one second ECDH calculation in [10]), a 43x speed improvement. WalnutDSA does not require a 32-bit hardware multiplier.

[2] Technically we only need $L = 12$ for a $2^{128}$ security level; using $L = 18$ results in a theoretical security level of $2^{194}$, but since the majority of the signature length is the encoded message, we increased $L$ by 50% for safe measure.

On the ARM Cortex M3 we compiled WalnutDSA using GCC version 4.9.3 (20150303) also using the -O3 level of optimization. The code compiled down to only 2952 bytes of ROM and ran in 272 bytes of RAM. The signature verification executed in 275563 cycles, which at 48MHz tool only 5.7ms. Compare this result to ECC, where [32] showed a full assembly language implementation that required 7168 bytes of ROM and 540 bytes of RAM, but still required 233ms to perform a point multiplication (recall that ECDSA verification requires two). ARM itself produced a report [30] where they measured an ECDSA verification on the same platform (and LPC1768) in 458ms. With these results WalnutDSA in C is more than 40x faster than the assembly implementation (and requires less than half the ROM and RAM), and 80x faster than ARM's speed reports.

On the 8051 we used the Keil V9.54 compiler to build WalnutDSA, with the small memory module and optimization set to OPTIMIZE(11,SPEED). We specifically chose to use assembly due to the poor mapping of the E-Multiplication C implementation to the 8051 platform. The code compiled into 3370 bytes of ROM. The 8051 platform we chose is unique in the way it handles RAM. Specifically it includes a "relocatable" section. When we ran WalnutDSA it required a total of 312 bytes of RAM (split into 251 bytes of "xdata," 3 bytes of "data," and 58 bytes of "relocatable data.") Verifying the signature required 864101 cycles; running at 24.5 MHz this equates to 35.3ms.

Finally, we implemented WalnutDSA as a hardware coprocessor to tie into a CPU core running on a Field Programmable Gate Array (FPGA). The devices we tested run the fabric at a speed of 50 MHz, and devices can vary significantly in size and capabilities. In our case we included not just the raw processing time but also the time required to transfer the data (public keys, message, and signature) from the processor into the fabric. Specifically, we need to pass 161 words into the fabric; the time required varied and was dependent on the actual platform.

The majority of the execution time was, indeed, the data transfer time. In total we performed a signature validation in under 2500 cycles (depending on the platform). This implies, at 50 Mhz, an execution time of under 50$\mu$s!

Compare this to an ECDSA implementation, such as that in [18]. They implemented ECDSA on a Xilinx Virtex 4 platform and computed a point multiplication would take 304$\mu$s at 171.247MHz. When you normalize to a 50Mhz fabric speed this equates to 1041$\mu$s for a point multiplication. Considering ECDSA verification requires two we can estimate a verification at approximates 2.08ms, yielding a 41x improvement of WalnutDSA over ECDSA.

## 10 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 359 bits and the private key length ranges from 232 to 456 bits long. The signatures, after using BKL and Dehornoy braid rewriting techniques, range from 2416 to 6448 bits in length.

In addition, WalnutDSA signature verification proves to be extremely fast. It is two E-Multiplications, a matrix multiplication, and then a matrix compare. An initial, non-optimized implementation on a 16-bit MSP430 verifies a 4296-bit length (128-bit strength) signature 43-times faster than an ECC Curve25519 signature verification. Similar speed improvement is seen on an 8051, ARM Cortex M, and within FPGA environments.

## References


1. D. Atkins; D. Goldfeld, Addressing the algebraic eraser over the air protocol, https://eprint.iacr.org/2016/205.pdf
2. Anshel, Iris; Atkins, Derek; Goldfeld, Dorian; Gunnells, Paul, *Defeating the Ben-Zvi, Blackburn, and Tsaban Attack on the Algebraic Eraser*, arXiv:1601.04780v1 [cs.CR].
3. Anshel, Iris; Atkins, Derek; Goldfeld, Dorian; Gunnells, Paul E.; *A Class of Hash Functions Based on the Algebraic Eraser,* 2015.
4. Anshel, Iris; Anshel, Michael; Goldfeld, Dorian; Lemieux, Stephane, *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.
5. Adi Ben-Zvi, Simon R. Blackburn and Boaz Tsaban, *A practical cryptanalysis of the Algebraic Eraser*, CRYPTO 2016, Lecture Notes in Computer Science 9814 (2016), 179–189.
6. Birman, Joan, *Braids, Links and Mapping Class Groups*, Annals of Mathematics Studies, Princeton University Press, 1974.
7. Birman, Joan; Ko, Ki Hyoung; Lee, Sang Jin; *A new approach to the word and conjugacy problems in the braid groups,* Adv. Math. 139 (1998), no. 2, 322–353.
8. Simon R. Blackburn and 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.
9. P. Dehornoy, *A fast method for comparing braids,* Adv. Math. 125 (1997), no. 2, 200–235.
10. Düll, Michael; Haase, Björn; Hinterwälder, Gesine; Hutter, Michael; Paar, Christof; Sánchez, Ana Helena; Schwab, Peter; *High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers,* https://eprint.iacr.org/2015/343.pdf (2015).
11. Garber, David; Kaplan, Shmuel; Teicher, Mina; Tsaban, Boaz; Vishne, Uzi, *Length-based conjugacy search in the braid group,* Algebraic methods in cryptography, 75-87, Contemp. Math., 418, Amer. Math. Soc., Providence, RI, 2006.
12. V. Gebhardt, *A new approach to the conjugacy problem in Garside groups,*, J. Algebra 292(1) (2005), 282–302.
13. D. Goldfeld and P. Gunnells, *Defeating the Kalka-Teicher-Tsaban linear algebra attack on the Algebraic Eraser*, Arxiv eprint 1202.0598, February 2012.
14. Anja Groch; Dennis Hofheinz; and Rainer 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.
15. Grover L.K.; *A fast quantum mechanical algorithm for database search,* Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) p. 212.
16. P. Gunnells, *On the cryptanalysis of the generalized simultaneous conjugacy search problem and the security of the Algebraic Eraser,* arXiv:1105.1141v1 [cs.CR] .
17. Hansen, Vagn Lundsgaard, *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).
18. Jian Huang, Hao Li, and Phil Sweany, *An FPGA Implementation of Elliptic Curve Cryptography for Future Secure Web Transaction*

19. Dennis Hofheinz; Rainer 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.

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

21. 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.

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

23. Lomont, C.; *The hidden subgroup problem - review and open problems,* 2004, arXiv:0411037

24. Magnus, Wilhelm; Karrass, Abraham; Solitar, Donald; *Combinatorial group theory: Presentations of groups in terms of generators and relations,* Interscience Publishers (John Wiley & Sons, Inc.), New York-London-Sydney (1966).

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

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

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

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

29. Jacques Stern; David Pointcheval; John Malone-Lee; Nigel 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.

30. Tschofenig, Hannes; Pégourié-Gonnard, Manuel, *Crypto Performance on ARM Cortex-M Processors*, IETF-92, Dallas, TX, March, 2015

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

32. Wenger, Unterluggauer, 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

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

# A Performance Matrix

**Table 1.** Raw WalnutDSA Performance Data

| **Platform** | **Clock** | WalnutDSA | | | | ECDSA | | | | **Improvement** |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **ROM** | **RAM** | **Cycles** | **Time (ms)** | **ROM** | **RAM** | **Cycles** | **Time (ms)** | **over ECDSA** |
| MSP430 | 8 | 3244 | 236 | 370944 | 46 | ? | ? | ? | 2000 | 43x |
| ARM Cortex M3 | 48 | 2952 | 272 | 275563 | 5.7 | 7168 | 540 | ? | 233 | 40x |
| 8051 | 24.5 | 3370 | 312 | 864101 | 35.3 | ? | ? | ? | ? | ? |
| FPGA | 50 | | | 2500 | 0.05 | ? | ? | ? | 2.08 | 41x |

Note that a '?' in Table 1 implies that this data was not made available.

# B Example Data

The following sections detail an example of an actual WalnutDSA transaction. This is all based on $N = 8$, $q = 2^5 = 32$, and $L = 18$. We construct the finite field $\mathbb{F}_{32}$ as $\mathbb{F}_2[x]/(f)$, where $f$ is the irreducible polynomial $x^5 + x^2 + 1$ (cf. [27]). Elements of $\mathbb{F}_{32}$ are then represented as 5-bit numbers: the finite field element $a_4x^4 + a_3x^3 + \cdots + a_0 \bmod f$ is converted to the bitstring $a_4a_3 \cdots a_0$ (note that the coefficients of high degree monomials become the high-order bits in the bitstring).

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:

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

The public data:

- T-values: `28 24 1 9 26 1 18 18`
- Pub(S):
  - Matrix:
$$\begin{pmatrix} 15 & 30 & 7 & 18 & 13 & 20 & 15 & 31 \\ 10 & 19 & 13 & 19 & 6 & 17 & 11 & 21 \\ 10 & 14 & 16 & 19 & 6 & 17 & 11 & 21 \\ 17 & 31 & 21 & 28 & 15 & 23 & 2 & 16 \\ 9 & 16 & 10 & 13 & 12 & 7 & 31 & 20 \\ 9 & 16 & 10 & 13 & 21 & 30 & 31 & 20 \\ 0 & 0 & 2 & 4 & 23 & 0 & 8 & 24 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
  - Permutation: `2 4 3 8 1 6 7 5`

### Example Message

For the following signature and verfication examples we chose the following random 256-bit string:

```
2d 18 06 35 27 39 19 91 16 7b 12 81 85 de be 56
7b 3d d0 1b 01 e2 af 03 ed ce 2d ef 04 8e 06 cc
```

We then treat this as the output of a 256-bit hash.

**Example Signature and Verification**

For this example chose $x_1, x_3, x_5, x_7$ as the specific generators $g^i$. In this case the example message encodes into the following freely reduced braid $E(\mathcal{M})$:

7 6 5 5 5 4 3 3 3 3 3 3 3 3 3 3 3 2 1 1 1 1 1 1 1 1 -2 -3 -4 5 5 5 -6 7 7 6 5 4
3 3 3 3 -4 5 -6 7 7 7 7 7 7 6 5 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 -4 5 5 5 -6 7 7 7 7 7 7 7 7 7 7 6 5 4 3 3 -4 5 5 4 3 2 1 1 1 1 1 1 -2 3 3 2 1
1 1 1 1 1 -2 3 3 3 3 3 3 3 3 3 3 3 -4 5 5 5 5 5 5 5 -6 7 7 7 7 7 7 6 5 5 5 5 5
5 5 5 5 4 3 3 3 3 -4 5 5 5 -6 7 7 7 7 7 7 7 7 7 7 7 7 6 5 4 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 2 1 1 -2 3 -4 -5 -6 7 7 7 7 7 7 6 5 4 3 2 1 1 -2 3 -4 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 -6 7 7 7 7 7 7 7 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 7 7 6 5 5 5 5
5 5 5 5 5 4 3 3 3 3 3 3 3 3 3 2 1 1 1 1 1 1 1 -2 -3 -4 5 5 5 5 5 5 5 5 5 5 4
3 3 3 3 3 3 3 3 -4 5 5 5 5 5 5 5 -6 7 7 7 7 7 7 7 7 6 5 4 3 2 1 1 1 1 1 1 1 1
1 1 1 -2 -3 -4 5 5 5 5 5 5 5 5 4 3 2 1 1 -2 -3 -4 5 5 5 5 4 3 2 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 -2 -3 -4 -5 -6 -7

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

6 7 -6 1 -2 4 -5 -6 -5 4 -3 -2 3 -4 -3 -2 -4 -6 1 -5 -2 5 -2 3 4 7 -3 1 1 7 6 -4
2 5 -1 -5 3 -6 3 -4 1 5 2 1 4 -6 -2 -1 5 7 -1 -2 -4 7 -5 -6 3 7 4 -3 -4 -5 -4 -4
1 -5 -5 1 3 2 2 1 -2 5 4 -1 3 4 -1 6 -3 4 2 -3 4 5 4 3 -1 -2 3 -4 -5 -6 7 6 -5
4 3 2 1 -7 -5 -6 7 6 5 -4 -5 6 7 -6 5 4 2 -1 -1 -2 -7 6 5 4 3 2 -1 -1 -2 -3 -4
-5 -6 -7 -4 -4 -5 -5 -7 -7 6 5 -4 -4 -5 -6 3 -2 -2 -3 5 -4 -4 -5 -2 -2 5 4 -3 -3
-4 -5 7 6 5 4 -3 -3 -4 -5 -6 -7 6 -5 -5 -6 5 4 -3 -3 -4 -5 7 -6 -6 -7 -7 -7 -4
-4 -7 -7 2 2 7 7 4 4 7 7 7 6 6 -7 5 4 3 3 -4 -5 6 5 5 -6 7 6 5 4 3 3 -4 -5 -6 -7
5 4 3 3 -4 -5 2 2 5 4 4 -5 3 2 2 -3 6 5 4 4 -5 -6 7 7 5 5 4 4 7 6 5 4 3 2 1 1 -2
-3 -4 -5 -6 7 2 1 1 -2 -4 -5 6 -7 -6 5 4 -5 -6 -7 6 5 7 -1 -2 -3 -4 5 -6 -7 6 5
4 -3 2 1 -3 -4 -5 -4 3 -2 7 6 5 5 5 4 3 3 3 3 3 3 3 3 3 3 3 2 1 1 1 1 1 1 1 -2
-3 -4 5 5 5 -6 7 7 6 5 4 3 3 3 3 -4 5 -6 7 7 7 7 7 7 6 5 4 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 -4 5 5 5 -6 7 7 7 7 7 7 7 7 7 7 6 5 4 3 3 -4 5 5 4
3 2 1 1 1 1 1 1 -2 3 3 2 1 1 1 1 1 1 -2 3 3 3 3 3 3 3 3 3 3 -4 5 5 5 5 5 5 5
-6 7 7 7 7 7 7 7 6 5 5 5 5 5 5 5 5 5 4 3 3 3 3 -4 5 5 5 -6 7 7 7 7 7 7 7 7 7 7 7
7 6 5 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 1 1 -2 3 -4 -5 -6 7 7 7 7 7 7 6 5 4
3 2 1 1 -2 3 -4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 -6 7 7 7 7 7 7 7 7 6 5 4 3 2 1 1
-2 -3 -4 -5 -6 7 7 6 5 5 5 5 5 5 5 5 5 4 3 3 3 3 3 3 3 3 3 2 1 1 1 1 1 1 1 -2
-3 -4 5 5 5 5 5 5 5 5 5 5 4 3 3 3 3 3 3 3 3 -4 5 5 5 5 5 5 5 -6 7 7 7 7 7 7 7 7
6 5 4 3 2 1 1 1 1 1 1 1 1 1 1 1 -2 -3 -4 5 5 5 5 5 5 5 5 4 3 2 1 1 -2 -3 -4 5
5 5 5 4 3 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -2 -3 -4 -5 -6 -7 -4 3 -6 1 -4 -3 1
-4 -5 2 -1 -2 -2 -3 -1 5 5 -1 4 4 5 4 3 -4 -7 -3 6 5 -7 4 2 1 -7 -5 1 2 6 -4 -1
-2 -5 -1 4 -3 6 -3 5 1 -5 -2 4 -6 -7 -1 -1 3 -7 -4 -3 2 -5 2 5 -1 6 4 2 3 4 -3
2 3 -4 5 6 5 -4 2 -1 6 -7 -6 4 5 -6 -7 6 -5 4 5 -6 -7 6 5 1 2 -3 -4 -5 -6 -5 -4
-3 2 2 -3 4 5 -4 3 -2 -2 7 6 -5 -5 -6 -7 2 -1 -1 -2 6 5 4 -3 -3 -4 -5 -6 7 6 5
4 3 -2 -2 -3 -4 -5 -6 -7 6 5 4 3 2 -1 -1 -2 -3 -4 -5 -6 -7 -7 -3 -3 -6 -6 3 2 -1
-1 -2 -3 5 4 3 -2 -2 -3 -4 -5 -4 -4 6 -5 -5 -6 7 6 -5 -5 -6 -7 -7 -7 -6 5 4 3 -2
-2 -3 -4 -5 -6 7 6 5 4 3 2 -1 -1 -2 -3 -4 -5 -6 -7 5 4 3 2 -1 -1 -2 -3 -4 -5 2
2 5 4 3 2 1 1 -2 -3 -4 -5 7 6 5 4 3 2 1 1 -2 -3 -4 -5 -6 -7 6 5 4 3 2 2 -3 -4 -5
6 7 7 7 6 5 5 -6 -7 6 5 5 -6 4 4 5 4 3 2 2 -3 -4 -5 3 2 1 1 -2 -3 6 6 3 3 7 7 6

5 4 3 2 1 1 -2 -3 -4 -5 -6 7 6 5 4 3 2 2 -3 -4 -5 -6 -7 6 5 4 3 3 -4 -5 -6 2 1
1 -2 7 6 5 5 -6 -7 2 2 -3 4 -5 -4 3 -2 -2 3 4 5 6 5 4 3 -2 -1 -5 -6 7 6 -5 -4 5
-6 7 6 -5 -4

And after running the raw signature through both the BKL Normal Form and then Dehornoy we resulted in the following 1170-generator braid:

4 -7 6 -5 -5 7 -6 -5 1 -2 -2 -3 -4 -2 -2 1 2 -3 -3 7 7 7 -6 5 1 -2 6 1 -2 -2 -3
-4 -2 -2 1 2 -3 -4 1 -2 -3 -3 -4 -5 1 1 2 3 4 4 1 2 3 4 5 -6 -7 -7 4 -5 3 -4 2
-3 1 -2 -2 -3 -4 -5 1 2 3 4 1 1 5 -6 -6 4 -5 3 -4 2 -3 -3 -3 -4 -5 -6 1 -2 -3 -3
-4 -5 1 -2 -3 -4 -5 1 2 3 4 5 6 -7 5 -6 4 -5 3 -4 2 -3 1 -2 -2 -3 -4 -5 -6 1 -2
-3 1 -2 3 1 2 3 4 5 6 -7 5 -6 4 -5 3 -4 2 -3 1 -2 1 -2 -3 -4 -5 -6 1 2 3 4 5 6
-7 3 4 5 -6 4 -5 3 -4 2 -3 1 -2 -2 -2 -3 -3 -4 -5 1 2 3 4 1 3 2 5 -6 -6 4 -5 3
-4 -5 2 -3 -4 -4 1 -2 -2 -2 -3 -4 -5 -6 1 2 3 -4 -5 -5 -4 6 5 -6 -7 1 2 3 1 2 4
-5 -5 3 -4 -4 2 -3 1 -2 -2 -3 -4 -5 -6 1 2 3 3 4 -5 6 1 3 -4 2 -3 2 5 4 3 -4 -5
-5 2 -3 -4 -4 1 -2 -2 -3 -3 -2 1 -2 -3 -4 -5 -2 -2 1 2 -3 -4 7 -6 -5 -6 1 -2 -2
1 2 -3 1 -2 -2 1 2 -3 1 -2 -2 -3 -4 -5 1 2 3 4 5 6 -7 3 4 5 -6 2 3 4 -5 1 2 3 -4
-5 2 -3 -4 -5 -6 -7 1 1 -2 -2 1 2 -3 -4 -5 -6 7 1 -2 -3 -4 1 -2 -3 -4 1 1 2 2 3
3 4 -5 1 1 2 3 -4 2 -3 1 -2 -2 -3 1 1 2 1 2 3 -4 2 -3 1 -2 -2 -3 1 2 3 5 4 6 5
-6 2 3 4 -5 3 -4 2 -3 1 -2 -2 -3 -4 1 -2 -3 -4 -5 -4 7 -6 -5 -7 -6 1 1 2 -3 1 -2
-2 4 3 1 2 3 5 4 6 5 -6 4 -5 3 -4 2 -3 1 -2 -2 -3 -4 -5 7 6 1 -2 -3 1 1 1 1 1 1
1 1 1 1 -2 1 1 1 1 1 1 1 -2 3 3 3 1 2 3 -4 2 -3 1 -2 -2 -3 -3 1 1 -2 -3 1 1 1 1
1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 1 1 1 1 1 1 1 1 1 2
3 3 2 2 3 2 3 -4 1 2 -3 4 1 -2 1 1 1 1 -2 -2 1 1 1 1 -2 1 1 1 1 1 1 1 1 1 1 2
1 2 -3 -4 1 -2 -3 4 1 1 1 1 1 1 -2 1 1 1 1 -2 -3 1 1 1 1 1 1 1 1 -2 3 1 1 1 2 3
3 3 3 -4 2 2 2 2 2 2 2 2 2 2 2 -3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -2 -2 1 2 3
3 3 3 3 3 2 1 1 2 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 2 2 3 3 3 4 4
4 4 4 4 4 4 4 1 2 3 3 3 3 3 3 3 3 3 4 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4
4 4 4 4 4 4 4 1 2 3 1 2 1 1 1 1 1 1 1 1 2 3 3 3 3 3 3 3 3 4 -5 2 2 2 2 2 2 2 2 2
2 2 1 1 1 1 1 1 1 1 2 2 3 1 1 1 1 2 3 1 2 3 -4 5 -6 -6 -6 -7 1 2 -3 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 -3 -2 1 4 3 2 3 4 -5 1 2 3 -4 3 -4 3 -5 -4 -4 3 2 1 4 5 -6
3 2 7 4 -5 3 -4 -4 2 -3 1 -2 -2 -3 1 2 2 3 6 5 4 -5 -6 -7 2 3 -4 -5 -6 1 2 -3 -4
-5 -5 -6 -7 1 1 -2 -3 -4 -5 -6 -2 1 3 2 4 3 5 4 -5 -5 3 -4 2 -3 1 -2 1 6 5 4 3
2 -3 7 6 5 4 -5 -6 -5 -6 -7 1 -2 3 -4 -5 -5 -6 3 4 -5 3 -4 5 5 -7 -7 6 1 2 -3 -3
2 2 1 2 1 4 -3 2 3 2 1 5 4 -5 3 -4 2 -3 -3 -4 1 -2 3 4 6 5 -6 1 2 3 4 -5 2 3 -4
1 2 -3 1 -2 7 6 -7 5 -6 7 -6 7 -6 -7 4 3 -2 1 1 2 -3 1 -2 -2 -3 5 -4 -4 6 -5 -3
-4 7 6 -7 5 -2 1 -3 2 -4 -5 -5 -6 -4 -5 3 -4 2 -3 1 -2 -7 -6 -7 -6 5 -7 4 3 -2
4 -7 -6 -6 5 4 3 6 7 -4 5 6 -7 3 -2 1 4 -3 2 1 5 -4 3 -5 -4 2 -3 -3 1 -2 -2 1 3
2 -3 -6 5 4 -3 -4 -5 2 -3 1 -2 -2 4 3 -2 1 -2 -3 -4 1 -2 -3 1 5 6 4 -3 -2 1 -2
1 1 -3 -4 5 7 -6 -7 -2 -3 4 -5 -6 1 -2 3 -2 1 4 -3 -2 -4 -3 -5 -4 -6 5 1 5 -7 6
-4 -3 -2 6 7 5 4 -5 3 -4 6 5 7 -6 -7 3 4 5 -6 -2 1 4 4 -3 2 1 5 7 6 -7 5 -4 3 4
2 6 7 5 2 3 -6 7 1 -4 3 4 -2 3 1 2 3 4 -6 -5 4 -6 5 6

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:

$$\begin{pmatrix} 31 & 7 & 7 & 4 & 4 & 26 & 27 & 1 \\ 29 & 3 & 2 & 15 & 15 & 18 & 10 & 24 \\ 29 & 28 & 29 & 15 & 15 & 18 & 10 & 24 \\ 14 & 7 & 7 & 25 & 24 & 22 & 16 & 6 \\ 7 & 1 & 1 & 27 & 26 & 12 & 18 & 30 \\ 7 & 1 & 1 & 27 & 27 & 13 & 18 & 30 \\ 29 & 5 & 5 & 6 & 6 & 23 & 8 & 30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

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

$$\begin{pmatrix} 15 & 17 & 17 & 3 & 5 & 27 & 23 & 18 \\ 10 & 15 & 14 & 18 & 24 & 13 & 12 & 28 \\ 10 & 6 & 7 & 18 & 24 & 13 & 12 & 28 \\ 17 & 29 & 11 & 21 & 26 & 16 & 24 & 27 \\ 8 & 14 & 21 & 31 & 28 & 20 & 17 & 25 \\ 8 & 14 & 21 & 31 & 5 & 13 & 17 & 25 \\ 14 & 7 & 5 & 24 & 5 & 17 & 21 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

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

$$\begin{pmatrix} 15 & 17 & 17 & 3 & 5 & 27 & 23 & 18 \\ 10 & 15 & 14 & 18 & 24 & 13 & 12 & 28 \\ 10 & 6 & 7 & 18 & 24 & 13 & 12 & 28 \\ 17 & 29 & 11 & 21 & 26 & 16 & 24 & 27 \\ 8 & 14 & 21 & 31 & 28 & 20 & 17 & 25 \\ 8 & 14 & 21 & 31 & 5 & 13 & 17 & 25 \\ 14 & 7 & 5 & 24 & 5 & 17 & 21 & 21 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

which is obviously equal to the previous matrix by inspection.