WalnutDSA$^{\text{TM}}$: A Quantum-Resistant Digital Signature Algorithm

Iris Anshel, Derek Atkins, Dorian Goldfeld, and Paul E. 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$^{\text{TM}}$, a quantum-resistant, group-theoretic, one-way function which can be used as a basis for many different cryptographic applications. This one-way function was specifically designed for constrained devices, running extremely quickly and requiring very little code.

This paper introduces WalnutDSA, a new E-Multiplication-based public-key method which provides efficient verification, allowing low-power and constrained devices to quickly and inexpensively validate digital signatures (e.g., a certificate or authentication). It 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.

Keywords: 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. However these methods are not very efficient in tiny devices like 16- or even 8-bit constrained devices (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 [26] proposed a digital signature based on a variation of the conjugacy problem in non-commutative groups. In 2009, Wang and Hu [39] introduced a digital signature with security based upon the hardness of the root problem in braid groups. See also [24]. The attacks introduced in [14], [15], [17], and [22] suggest that these schemes 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 §9. The latter 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 [14], [15], [17], and [22], as well as the very recent work of of [21] (for a fuller discussion see §9 below - Attacks on the underlying math). Likewise, attacks on the original 2005 key agreement construction noted in [7], [25], and [32], do not apply.

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, 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 $\{b_1, b_2, \dots, b_{N-1}\}$, subject to the following relations:

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

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

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 $i_j \in \{1, \dots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$. Note that β is not unique; there are an infinite number of equivalent expressions as you apply (1) and (2).

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^{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 σ_i is associated to the Artin generator b_i. Further, if $\beta \in B_N$ is written as in (3), we take β 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 (i.e., the identity permutation).

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

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

First, we define the $N \times N$ colored Burau matrix (denoted CB) of each Artin generator as follows [30].

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

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 σ_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

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

The colored Burau representation is then defined by

$$ \mathit{\Pi}_{CB}(\beta) := (CB(\beta), \sigma_\beta). $$

One checks that $\mathit{\Pi}_{CB}$ 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

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

where we interpret the right of (6) by associating left-to-right. One can check that this is independent of the expression of β 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, σ) if*

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

Let $\text{Cloak}_{(M,\sigma)}$ *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. We remark that this notion depends on the T-values, which are used in defining the operation $\star$.

It is not immediately obvious how to construct cloaking elements. The following proposition provides one technique to build them:

Proposition 4.2 *Fix integers $N \geq 2$, and $1 < a < b < N$. Assume that the T-values τ_a and τ_b both equal 1. Let $M \in GL(N, \mathbb{F}_q)$ and $\sigma \in S_N$. Then a cloaking element v of (M, σ) is given by $v = w b_i^2 w^{-1}$ where b_i is any Artin generator $(1 \leq i < N)$, and where the permutation corresponding to $w \in B_N$ satisfies*

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

By definition, any cloaking element of an ordered pair $(M, \sigma) \in GL(N, \mathbb{F}_q) \times S_N$ stabilizes (M, σ) through the right action of the braid group via E-multiplication. Thus the following proposition is immediate:

Proposition 4.3 *The set* $\text{Cloak}_{(M,\sigma)}$ *forms a subgroup of B_N.*

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 ρ.

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

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

Let $\beta \in B_N$.

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

where Id_N is the $N \times N$ identity matrix and Id_{S_N} is the identity permutation in S_N.

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 $N \geq 8$ and associated braid group B_N.
- A rewriting algorithm $\mathcal{R}\colon B_N \to B_N$ such as [9] or [12].
- A finite field $\mathbb{F}_q$ of $q \;\geq\; 32$ elements.
- Two integers $1 < a < b < N$.
- T-values $= \{\tau_1, \tau_2, \dots, \tau_N\}$, where each τ_i is an invertible element in $\mathbb{F}_q$, and $\tau_a = \tau_b = 1$.

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:

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

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 $H : \{0,1\}^* \to \{0,1\}^{4\kappa}$ denote a cryptographically secure hash function for $\kappa \geq 1$. We now present an encoding function $E : \{0,1\}^{4\kappa} \to C_{N,4}$, where $C_{N,4}$ is a free 4 generator subgroup of B_N defined below. 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

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

generate a free subgroup B_N [8]. 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: Choose and fix a subset of four generators

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

and define $C_{N,4}$ to be the subgroup generated by these 4 generators. Each 4-bit block of $H(m)$ then specifies a unique power of one of these 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 $E(H(m))$ of the message encoder is then the freely reduced product of these κ powers of generators.

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 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 (Definition 4.1) such that

– 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 $\mathrm{Sig} = \mathcal{R}\big(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2\big)$, 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 $v, v_1, v_2 \in B_n$ contain a random product of pure braid generators, and disappear when the signature is E-Multiplied by the public key Pub(S).

Signature Verification: The signature (m, 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. The signature is valid if and only if (8) holds.

[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 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.

9 Preliminary Security Discussion

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

The REM Problem (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 $\{\tau_1, \tau_2, \ldots, \tau_N\}$ in $\mathbb{F}_q$. Suppose we are given a generic $\beta \in B_N$ and the pair $(M, \sigma) \in (GL(N, \mathbb{F}_q),\ S_N)$ where $(M, \sigma) = \mathcal{P}(\beta)$. Then it is infeasible to determine a braid β' such that*

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

if the expression for β in Birman-Ko-Lee canonical form [9] is sufficiently long as a word in Artin generators.

If we consider β 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 β 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 [31] which studies the security of Zémor's [41] 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 [31] 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.

Attacks on the underlying math

The attack of Ben-Zvi–Blackburn–Tsaban [7], based on ideas in [25], 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 [7] 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 [7], and [16] which shows how to defeat the attack in [25].

The more recent attack of Blackburn–Robshaw [10] 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.

Finally, very recent work of Hart–Kim–Micheli–Perez–Petit–Quek [21] 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 $h = \text{Matrix}\Big(\mathcal{P}(E(M))\Big)$ in terms of elements $g_i := \text{Matrix}\Big(\mathcal{P}(E(M_i)\Big)$.. Namely, one seeks an expression of the form

$$h = \prod_{j=1}^{l} g_{i_j}^{\epsilon_{i_j}}, \quad i_j \in I, \epsilon_{i_j} \in \{\pm 1\}. \tag{9}$$

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 completely if $w \neq w'$, since one cannot multiply the s_i together to produce a valid signature. We remark that even if $w = w'$ (as pointed out in [21]), the attack fails if the parameters N, q are moderately larger, since then it becomes infeasible to produce the expression (9). The authors of [21] also 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.

10 Security Proof for WalnutDSA-I

We will now provide security proofs for a Schnorr/Brickell type model (see [27], [11]) 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

$$par \overset{\$}{\longleftarrow} \text{Pg},$$

$$(\text{Pub}(S), \text{Priv(S)}) \overset{\$}{\longleftarrow} \text{Kg}(par).$$

In WalnutDSA-I the signature of a message $m \in \{0,1\}^*$ for the public Pub(S) is based on two hash functions $H, G : \{0,1\}^* \to \{0,1\}^{4\kappa}$ and is generated by the following protocol.

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

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

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

$$\text{DC}_{m,V,H,G} := \Big\{\mathcal{R}\Big(X \cdot \left[\mathcal{P}\left(E\left(H\big(m \,||\, G\,(V\,)\,\big)\right)\right)\right] \cdot Y\Big) \;\Big|\; X, Y \in B_N\Big\}, \tag{10}$$

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 [27], 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 $\text{Priv}(S)$ but can simulate an honest signer.

Hash Query: Let $\mathcal{O}_\rho$ denote a random oracle, depending on a coin ρ, 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

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

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 ρ.

The Forger $\mathcal{F}$ is defined to be a randomized algorithm that on input a message $m \in \{0,1\}^*$, a signers public key $\text{Pub}(S)$, and a coin ρ, outputs a 4-tuple $\big(m, h, g_\rho, s\big)$, where $h = H(m)$

and $g_\rho = G_\rho(V)$ and $V \xleftarrow{\$} \text{Cloak}$, $s \xleftarrow{\$} \text{DC}_{m,V,H,G}$. It is assumed that the probability that $\big(m, h, g_\rho, s\big)$ is a valid WalnutDSA-I signature is non-negligible.

Lemma 10.1 (Forking Lemma) *Let $\mathcal{F}$ be run twice with inputs,*

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

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 [34], [6].

The forking lemma 10.1 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 [15], [17].

Theorem 10.2 *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

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

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 $\mathcal{P}(w) = (\text{Id}_N, \text{Id}_{S_N}) \star X$, 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

$$\text{Sig} = \mathcal{R}\big(v_1 \cdot w^{-1} \cdot v \cdot E(H(m)) \cdot w' \cdot v_2\big). \tag{11}$$

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.

11 Brute Force Attacks

We now discuss the brute force security levels 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 11.1 (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 $\mathbb{F}$.*

Brute force security level for each Private Key:

In order to choose private keys of security level = SL that defeat a brute force attack, we need to analyze the set of braids in B_N of a given length ℓ 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 ℓ in B_N, the most basic estimate for $W_N(\ell)$ is given by

$$W_N(\ell) \;\leq\; (2(n-1))^{\ell}.$$

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 $b_i\, b_j \;=\; b_j\, b_i \quad |i-j| \geq 2$, allow us to write products of generators far enough apart in weighted form, i.e., given $b_i\, b_j$ where $|i-j| \geq 2$, 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 $44 < (2(5-1))^2 = 64$. 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 ℓ, 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 ℓ 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 ℓ with the least over counting. There is no closed formula for the number of distinct braids of length ℓ; in fact the problem is NP hard [33].

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 ℓ 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 ℓ associated to a given security level SL, one may apply Newton's method to solve the equation: $\ell + (N-2)\log_2(\ell) = SL + \log_2\Big((N-1)!\Big)$.

Brute force security level of the Cloaking Elements, $\boldsymbol{v, v_1, v_2}$:

The pure braid subgroup of B_N is generated [20] by the set of $N(N-1)/2$ braids given by

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

The cloaking element v is defined to be a conjugate of some b_i^2 by a lift of a permutation that moves $i \to a$, $i+1 \to b$ times a random word in the pure braid subgroup of length at least L. The cloaking element v_1, is defined to be the conjugate of some b_i^2 by a lift of a permutation that moves $i \to \sigma_w^{-1}(a)$, $i+1 \to \sigma_w^{-1}(b)$ (where σ_w is the permutation associated to w) times a random word in the pure braid subgroup of length at least L. Likewise, v_2, is defined to be the conjugate of some b_j^2 by a lift of a permutation that moves $j \to \sigma_{w'}^{-1}(a)$, $j+1 \to \sigma_{w'}^{-1}(b)$ 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 (12) of the pure braid subgroup is bounded by

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

Hence, a lower bound for the security level of the triple v, v_1, v_2 of the cloaking elements is given by

$$3 \cdot L \cdot \log_2\big(N(N-1)\big),$$

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

One can compute L from the desired security level SL (in bits) by computing:

$$L = \lceil \mathrm{SL}/(3\log_2(N(N-1)))\rceil. \tag{13}$$

For example, suppose 128-bit security is desired, and the braid group is B_8, then

$$L = \lceil 128/(3\log_2(8\cdot 7))\rceil = 8.$$

Remarks: To date there is no good method known to efficiently enumerate all distinct pure braid elements of length L in the generators g_{ij} given in (12). Consequently, to perform the above attack, an attacker must execute a brute force search of all possible words in the generators as described above.

Search space of each Public Key Pub($\boldsymbol{S}$):

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 $N \times N$ matries each with q possible elements for every entry. The last row, however, is all zeros (except for the final element). Moreover, due to the fact that two T-values are set to 1, in practice there is more duplication within the matrix which further reduces the number of potential states. A conservative estimate is that there are

$$q^{N(N-3)} = q^{N^2-3N}$$

possible choices for each of the matrices appearing in public keys. 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 §9, the security of WalnutDSA is based on the hard problems of reversing E-Multiplication. 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 [36] 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 [28]). 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

$$\beta = b_{i_1}^{\epsilon_1}\, b_{i_2}^{\epsilon_2}\, \cdots\, b_{i_k}^{\epsilon_k} \in B_N, \tag{14}$$

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 σ_β is the permutation associated to β. Now E-Multiplication is a highly non-linear operation. As the length k of the word β 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 [18] 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.

12 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 T470p laptop 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) [9] and Dehornoy [12] algorithms to obscure the braids and shorten them to reasonable lengths.

For our testing we settled on the parameters:

- $N = 8$
- $q = 32$
- $L = 15$
- $\ell = 132$

which yields a private key security level of at least 2^{128} against brute force attacks,[2] with a public keyspace of 2^{200} possible public keys.

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

$$N \log_2(q) + 2(N(N-1)+1)\log_2(q) + N\log_2(N) = 40 + 2 * 285 + 24 = 634 \text{ bits}.$$

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 94 generators to 130 generators, with a mean of 113.37 and a standard deviation of 5.84. With our encoding, this results in a private key storage of 752 to 1040 bits, and a theoretical maximum storage of 1056 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 770 to 1926 generators, with a mean of 1298.12 and a standard deviation of 159.67. These signatures also require 4 bits per generator, which results in signatures of length of 3080 to 7704 bits (with an average of 5192.48 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 above-average-length signature of 1400 generators, which encodes into 700 bytes. Then we built our code on the various platforms and measured the time to validate the signature.

[2] Technically we only need $L = 12$ and $\ell = 105$ for a 2^{128} security level; using $L = 15$ results in a theoretical security level of 2^{161}, but since the majority of the signature length is the encoded message, we increased L by 25% for safe measure. Similarly, we increased ℓ due to braid generator cancellation.

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 [13]), a 43x speed improvement. WalnutDSA does not require a 32-bit hardware multiplier.

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 took only 5.7ms. Compare this result to ECC, where [40] 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 [38] 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) using only 1,720 Adaptive Logic Modules (ALM). This implies, at 50 MHz, an execution time of under 50μs!

Compare this to an ECDSA implementation, such as that in [23]. They implemented ECDSA on a Xilinx Virtex 4 platform and computed a point multiplication would take 304μs at 171.247MHz. When you normalize to a 50MHz fabric speed, this equates to 1041μs 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.

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 634 bits and the private key length ranges from 752 to 1040 bits long (with a maximum theoretical length of 1056 bits). The signatures, after using BKL and Dehornoy braid rewriting techniques, range from 3080 to 7704 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 5232-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 (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. J. Birman, *Braids, Links and Mapping Class Groups*, Annals of Mathematics Studies, Princeton University Press, 1974.
9. 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.
10. 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.
11. 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).
12. P. Dehornoy, *A fast method for comparing braids,* Adv. Math. 125 (1997), no. 2, 200–235.

13. 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).

14. 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.

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

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

17. 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.

18. 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.

19. 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] .

20. 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).

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

22. 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.

23. 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.

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

25. 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.

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

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

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

29. 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).

30. 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.

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

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

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

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

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

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

37. 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.

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

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

40. 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.

41. 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**
8051 (8b)	24.5	3370	312	864101	35.3	?	?	?	?	?
MSP430 (16b)	8	3244	236	370944	46	?	?	?	2000	43x
ARM Cortex M3 (32b)	48	2952	272	275563	5.7	7168	540	?	233	40x
FPGA	50	1720(ALM)		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 $N = 8$, $q = 2^5 = 32$, $L = 15$, and $\ell = 132$. 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. [35]). 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:

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

The public data:

- T-values: `1 1 6 9 19 14 29 30`

- Pub(S):
  - Matrix:
    $$\begin{pmatrix} 30 & 30 & 0 & 0 & 0 & 0 & 0 & 0 \\ 30 & 30 & 0 & 0 & 31 & 31 & 0 & 0 \\ 2 & 22 & 11 & 21 & 29 & 5 & 30 & 4 \\ 27 & 14 & 15 & 16 & 10 & 7 & 11 & 30 \\ 6 & 24 & 3 & 4 & 2 & 13 & 25 & 17 \\ 14 & 14 & 26 & 23 & 3 & 25 & 9 & 18 \\ 15 & 15 & 31 & 29 & 31 & 23 & 23 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
  - Permutation: `4 1 8 5 7 2 6 3`
- Pub(S'):
  - Matrix:
    $$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 6 & 6 & 0 \\ 0 & 30 & 30 & 0 & 0 & 6 & 6 & 0 \\ 14 & 15 & 8 & 24 & 16 & 21 & 7 & 24 \\ 25 & 5 & 20 & 11 & 17 & 1 & 15 & 6 \\ 14 & 5 & 19 & 28 & 19 & 23 & 3 & 22 \\ 24 & 18 & 22 & 31 & 4 & 23 & 27 & 17 \\ 2 & 6 & 24 & 29 & 17 & 1 & 16 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
  - Permutation: `6 4 2 7 5 8 1 3`

An astute observer will notice there are a lot of zeros, specifically in the first, second, and last rows of this matrix. This is to be expected, and is taken into account for our level-of-security calculation. When a T-value is set to 1 there is a significant amount of duplication between that row and the row before. So when $a = 1$, the first row will roughly duplicate the "previous row", which of course is all zeros. Similarly, with $b = 2$, the second row will roughly duplicate the first row, which is mostly zeros, resulting in a second row with lots of zeros. Finally, the last row is always all zeros except for the last element. On average we expect there to be approximately $3N$ zeros in the $N \times N$ matrix, resulting in $q^{N(N-3)}$ possible states. For $N = 8$ and $q = 32$, this results in 2^{200} possible states, well greater than the expected 2^{128} security level.

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:

```
21 a4 b8 e3 d4 92 31 6a cd 27 1d ac 6e 59 62 05
14 f2 5d 77 c6 b6 02 c8 c0 94 8d a6 84 89 7d 95
```

Example Signature and Verification

For this example we use the generators $g_{1,8}, g_{3,8}, g_{5,8}, g_{7,8}$ from (7) for encoding. After free reduction, we find that the message becomes the following braid $E(\mathcal{M})$:

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

Notice the long runs of the generators 1, 3, 5, 7. These occur because we take the $g_{k,N}$, where $k = 1, 3, 5, 7$ to nontrivial powers during the encoding process, and because cancellations occur upon performing the free reduction.

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

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

After running the raw signature through both BKL Normal Form and then Dehornoy reduction we obtain the following 1298-generator braid:

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

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

Notice that one sees runs of the generators 1, ..., 4 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:

$$\begin{pmatrix} 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 27 & 26 & 5 & 5 & 29 & 4 & 25 \\ 27 & 29 & 29 & 9 & 8 & 14 & 1 & 15 \\ 19 & 22 & 22 & 7 & 6 & 23 & 12 & 27 \\ 9 & 26 & 26 & 15 & 15 & 21 & 2 & 22 \\ 19 & 31 & 31 & 31 & 31 & 28 & 20 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Note the zeros in the first, second, and last rows. This, too, is expected because of the choices of $a = 1, b = 2$ and the resulting duplication from the previous rows while performing E-Multiplication. Due to this duplication we expect to see approximately $3N$ zeros in the matrix. See the previous discussion about the public key.

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

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 & 0 & 30 & 30 & 0 \\ 1 & 28 & 7 & 1 & 3 & 1 & 23 & 3 \\ 1 & 26 & 12 & 4 & 14 & 4 & 11 & 27 \\ 29 & 5 & 1 & 12 & 27 & 22 & 8 & 0 \\ 17 & 18 & 17 & 4 & 27 & 25 & 22 & 11 \\ 31 & 0 & 30 & 26 & 17 & 16 & 4 & 28 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

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

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 & 0 & 30 & 30 & 0 \\ 1 & 28 & 7 & 1 & 3 & 1 & 23 & 3 \\ 1 & 26 & 12 & 4 & 14 & 4 & 11 & 27 \\ 29 & 5 & 1 & 12 & 27 & 22 & 8 & 0 \\ 17 & 18 & 17 & 4 & 27 & 25 & 22 & 11 \\ 31 & 0 & 30 & 26 & 17 & 16 & 4 & 28 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

which is obviously equal to the previous matrix by inspection. Again, we expect there to be approximately $3N$ zeros in the resulting matrix, yielding $q^{N(N-3)} = 2^{200}$ possible matrices. An astute reader will notice there are only 19 zeros, which is less than the expected 24.