Research Article

Iris Anshel, Derek Atkins, Dorian Goldfeld* and Paul E. Gunnells

A class of hash functions based on the algebraic eraser™

DOI: 10.1515/gcc-2016-0004
Received June 26, 2015

Abstract: This paper introduces a novel braid based cryptographic hash function candidate which is suitable for use in low resource environments. It is shown that the new hash function performed extremely well on a range of cryptographic test suites.

Keywords: Algebraic eraser, colored Burau key agreement protocol, group theoretic cryptography, braid groups, hash functions

MSC 2010: 20F36, 94A60

1 Introduction

At the core of the Algebraic Eraser™ Key Agreement Protocol [1] (denoted AEKAP) lies the novel one-way function, e-multiplication. This paper introduces the AEHash [2], a cryptographic hash function based on this one-way function. Various test suites, including the NIST Statistical Test Suite [5] were run, and AEHash was shown to perform above and beyond expectations.

The AEKAP has been implemented in a range of constrained and low footprint devices where there is very little space for additional functionality. Since the new AEHash runs on the same already implemented e-multiplication engine it can also fit on such devices, furthering their cryptographic capability.

2 The braid group, colored Burau matrices, and the algebraic eraser™

Let B_N denote the N-strand braid group. Each element in B_N can be expressed as a word in the Artin generators $\{b_1, b_2, \dots, b_{N-1}\}$, which are subject to the following identities: for $i = 1, \dots, N-1$, we have

$$b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1},$$

and for all i, j with $|i-j| \geq 2$, we have

$$b_i b_j = b_j b_i.$$

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

$$\beta = b_{i_1}^{\epsilon_1} b_{i_2}^{\epsilon_2} \cdots b_{i_k}^{\epsilon_k}, \tag{2.1}$$

where $i_j \in \{1, \dots, N-1\}$, and $\epsilon_j \in \{\pm 1\}$.

Iris Anshel, Derek Atkins, Paul E. Gunnells: SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484, USA, e-mail: ianshel@securerf.com, datkins@securerf.com, pgunnells@securerf.com
***Corresponding author: Dorian Goldfeld:** SecureRF Corporation, 100 Beard Sawmill Rd #350, Shelton, CT 06484, USA, e-mail: goldfeld@optonline.net

Each braid $\beta \in B_N$ determines a permutation in S_N, the group of permutations of N letters, as follows. Let $\sigma_i \in S_N$ be the ith simple transposition, which maps $i \to i+1$, $i+1 \to i$, and leaves $\{1, \dots, i-1, i+2, \dots, N\}$ fixed. Then if $\beta \in B_N$ is written as in (2.1), we take β to the permutation $\sigma_\beta = \sigma_{i_1} \cdots \sigma_{i_k}$.

Each braid generator b_i also determines an $N \times N$ matrix $\mathrm{CB}(b_i)$, called the ith colored Burau matrix. Let $\{t_1, \dots, t_N\}$ be a set of N indeterminates. Then the matrix $\mathrm{CB}(b_i)$ is defined by

$$\mathrm{CB}(b_i) = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & t_i & -t_i & 1 & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}, \tag{2.2}$$

where the indicated variables appear in row i, and if $i = 1$ the leftmost t_1 is omitted. We similarly define $\mathrm{CB}(b_i^{-1})$ by modifying (2.2) slightly:

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

where again the indicated variables appear in row i, and if $i = 1$ the leftmost 1 is omitted. For example, here are the colored Burau matrices when $N = 4$:

$$\mathrm{CB}(b_1) = \begin{pmatrix} -t_1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathrm{CB}(b_2) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ t_2 & -t_2 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathrm{CB}(b_3) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & t_3 & -t_3 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathrm{CB}(b_1^{-1}) = \begin{pmatrix} -\frac{1}{t_2} & \frac{1}{t_2} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathrm{CB}(b_2^{-1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & -\frac{1}{t_3} & \frac{1}{t_3} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathrm{CB}(b_3^{-1}) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -\frac{1}{t_4} & \frac{1}{t_4} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Thus each braid generator b_i (respectively, inverse generator b_i^{-1}) determines a colored Burau/permutation pair $(\mathrm{CB}(b_i), \sigma_i)$ (resp., $(\mathrm{CB}(b_i^{-1}), \sigma_i)$). We now wish to define a multiplication of colored Burau pairs such that the natural mapping from the braid group to the group of matrices with entries in the ring of Laurent polynomials in the t_i is a homomorphism. To accomplish this, we require the following observation. Given a Laurent polynomial

$$f(t_1, \dots, t_N) \in \mathbb{Z}[t_1^\pm, t_2^\pm, \dots, t_N^\pm],$$

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

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

We extend this action to $N \times N$ matrices over $\mathbb{Z}[t_1^\pm, t_2^\pm, \dots, t_N^\pm]$, denoted $\mathcal{M}$, by acting on each entry in the matrix, and denote the action in the same way. The general definition for multiplying two colored Burau pairs is now defined as follows from the definition of $\mathcal{M} \rtimes S_N$: given $b_i^\pm, b_j^\pm$, the colored Burau/permutation pair associated with the product $b_i^\pm \cdot b_j^\pm$ is

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

Given any braid

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

as in (2.1), the colored Burau pair $(\mathrm{CB}(\beta), \sigma_\beta)$ is given by

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

As the length of the braid β increases, the entries in the matrix $\mathrm{CB}(\beta)$ become very high degree Laurent polynomials. Thus an additional step is needed to make the colored Burau representation suitable for cryptographic applications. Let q be a prime power, and let F_q be the finite field F_q of q elements. A vector of t-values is a collection of non-zero field elements:

$$\{\tau_1, \tau_2, \ldots, \tau_N\} \subset F_q.$$

Given a vector of t-values, we can evaluate any Laurent polynomial $f(t_1, t_2, \ldots, t_N)$ to obtain an element of 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. Further, if $\sigma \in S_N$ and $\mathfrak{m} \in \mathcal{M}$, we define

$$^{\sigma}\mathfrak{m}\downarrow_{t\text{-values}} = (^{\sigma}\mathfrak{m})\downarrow_{t\text{-values}}.$$

With all these components in place we can now define e-multiplication which is in essence a right action of $\mathcal{M} \rtimes S_N$ on $N_q \times S_N$. By definition, e-multiplication is an operation that takes as input two ordered pairs,

$$(M, \sigma), \quad (\mathrm{CB}(\beta), \sigma_\beta),$$

where $\beta \in B_N$ and $\sigma_\beta \in S_N$ as before, and where $M \in N_q$, the group of $N \times N$ matrices with entries in F_q, and $\sigma \in S_N$. We denote e-multiplication with a star: $\star$. The result of e-multiplication, denoted

$$(M', \sigma') = (M, \sigma) \star (\mathrm{CB}(\beta), \sigma_\beta),$$

will be another ordered pair $(M', \sigma') \in N_q \times S_N$.

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

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

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

$$(M, \sigma) \star (\mathrm{CB}(\beta), \sigma_\beta) = (M, \sigma) \star (\mathrm{CB}(b_{i_1}^{\epsilon_1}), \sigma_{b_{i_1}}) \star (\mathrm{CB}(b_{i_2}^{\epsilon_2}), \sigma_{b_{i_2}}) \star \cdots \star (\mathrm{CB}(b_{i_k}^{\epsilon_k}), \sigma_{b_{i_k}}), \tag{2.3}$$

where we interpret the right of (2.3) by associating left-to-right. One can check that this is independent of the expression of β in the Artin generators.

The above definition lies at the core of the Algebraic Eraser Key Agreement Protocol; for details and examples, we refer to [1]. We will require a slight modification in the definition of e-multiplication to construct the AEHash. In the definition of $\star$ above the t-values remain the same at every step of the iterative process (2.3). Now we wish to define a new operation, denote $\star'$, in which the t-values themselves are permuted along the way. Assuming as above that $\beta = b_{i_1}^{\epsilon_1} b_{i_2}^{\epsilon_2} \cdots b_{i_k}^{\epsilon_k}$, we define

$$T_1 = t\text{-values} = \{\tau_1, \ldots, \tau_N\},$$

and let

$$T_2 = {}^{\sigma_0 \cdot \sigma_{b_{i_1}}} T_1.$$

The second step of the e-multiplication,

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

is given by

$$(M \cdot^{\sigma_0} (\mathrm{CB}(b_{i_1}^{\epsilon_1}))\downarrow_{T_1}, \sigma_0 \cdot \sigma_{b_{i_1}}) \star (\mathrm{CB}(b_i^{\epsilon_2}), \sigma_{b_{i_2}}). \tag{2.4}$$

We modify the original definition of e-multiplication by defining a new operation $\star'$ in the following way. Modify the second step of the e-multiplication in (2.4) by using the set T_2 for t-values to obtain

$$(M \cdot^{\sigma_0} (\mathrm{CB}(b_{i_1}^{\epsilon_1}))\downarrow_{T_1} \cdot^{\sigma_0 \sigma_{b_{i_1}}} (\mathrm{CB}(b_{i_2}^{\epsilon_2}))\downarrow_{T_2}, \sigma_0 \cdot \sigma_{b_{i_1}} \cdot \sigma_{b_{i_2}}).$$

Iterating this process we obtain the operation $\star'$. It is this variation of e-multiplication that we will use to define our hash function. We remark that one can also define an algebraic eraser-based hash function by replacing the operation $\star'$ with the original e-multiplication operation $\star$. Unlike $\star$, the $\star'$-operation does not define an action of $\mathcal{M} \rtimes S_N$ on $N_q \times S_N$. We plan to investigate $\star'$ in future work.

3 The AEHash function

Let S denote a string of bits and let λ denote a fixed non-zero positive integer. By padding S we can assume that the length of S, denoted $\mathrm{Card}(S)$, is divisible by λ. Thus, $D_S = \mathrm{Card}(S)/\lambda$ is the number of blocks assuming that S is padded. We see that S can be viewed as a union of blocks, each of which has length λ:

$$S = \bigcup_{i=1}^{D_S} \mathrm{Block}(i).$$

Let $\nu(i)$ denote the integer that the binary string $\mathrm{Block}(i)$ represents. By construction, we have $0 \le \nu(i) \le 2^\lambda - 1$.

The AEHash function H_{AE} is specified by the following data:

$$\{B_N, q, \lambda, t\text{-values} = \{\tau_1, \dots, \tau_N\}, \{c_0, c_1, \dots, c_{2^\lambda-1}\} \subset B_N, (n_0, \sigma_0) \in N_q \times S_N\},$$

where

- B_N is the braid group on N strands,
- q is a power of a 2, the t-values are invertible elements in F_q,
- the collection of braid group elements $\{c_0, c_1, \dots, c_{2^\lambda-1}\}$ is fixed and assumed to generate a free submonoid of B_N,
- $(n_0, \sigma_0) \in N_q \times S_N$ is an ordered pair.

The output of the AEHash is defined to be the sequence of bits that specify the matrix, which is evaluated through a sequence of e-multiplications. The length of the AEHash is given by $N^2 \cdot \mathrm{ceil}(\log_2(q))$, where for $x > 0$, the function $\mathrm{ceil}(x)$ (denotes the ceiling of x) which is the smallest integer n such that $x \le n$.

The lengths of the elements c_i, will impact the efficiency of the hash function. In our initial testing we chose the length to be in the range of $2N$.

Each element c_i is explicitly given as a word in the Artin braid generators and is thus associated with a sequence of colored Burau matrices and permutations. We evaluate the operation $\star'$ using this explicit string of colored Burau pairs and, abusing the notation slightly, we denote the output by $(\mathrm{CB}(c_i), \sigma_{c_i})$. It is important to remark that since $\star'$ is not an action, the braids c_i must be used as specified and not rewritten using the braid relations. Were we to use $\star$ instead of $\star'$, this would not be an issue.

The string S, having been broken into blocks of length λ, is associated with a sequence of braid words:

$$c_{\nu(1)}, c_{\nu(2)}, \dots, c_{\nu(D_S)}.$$

Thus S is associated with a sequence colored Burau/permutation pairs:

$$(\mathrm{CB}(c_{\nu(1)}), \sigma_{c_{(\nu(1))}}), (\mathrm{CB}(c_{\nu(2)}), \sigma_{c_{\nu(2)}}), \dots, (\mathrm{CB}(c_{\nu(D_S)}), \sigma_{c_{\nu(D_S)}}).$$

The hash of the string S, denoted $H_{\mathrm{AE}}(S)$, is defined to be the matrix part of the output of the iterative modified e-multiplication

$$(n_0, \sigma_0) \star' (\mathrm{CB}(c_{\nu(1)}), \sigma_{c_{\nu(1)}}) \star' (\mathrm{CB}(c_{\nu(2)}), \sigma_{c_{\nu(2)}}) \star' \cdots \star' (\mathrm{CB}(c_{\nu(D_S)}), \sigma_{c_{\nu(D_S)}}).$$

4 Basic analysis of the AEHash

As detailed above, the output of AEHash is a string of bits of length $N^2 \cdot \mathrm{ceil}(\log_2(q))$. An upper bound for the size of the set of all possible hashes the AEHash produces is given by q^{N^2}. Recall the assumption that the subgroup of B_N generated by $\{c_0, c_1, \dots, c_{2^\lambda-1}\}$ is free on the set of fixed braids $\{c_0, c_1, \dots, c_{2^\lambda-1}\}$. The number of possible sequences in the fixed braids of length D_S is given by $2^{D_S \cdot \lambda}$, and thus reversing the first step of the AEHash has $D_S \cdot \lambda$-bit security. Note that $D_S \cdot \lambda$ coincides with the length of the message, so the security will only be high for long messages.

At present, the only known method for reversing e-multiplication is a brute force procedure, resulting in $D_S \cdot \lambda$-bit security. Furthermore, given two distinct strings S_1, S_2 whose block decomposition is given by

$$S_j = \bigcup_{i=1}^{D_{S_j}} \mathrm{Block}(j_i), \quad j = 1, 2,$$

where $D_{S_j} = \frac{\mathrm{Card}(S_j)}{\lambda}$, the braids defined by the associated sequences of the fixed braids, say

$$\{c_{S_1,\nu(1)}, c_{S_1,\nu(2)}), \ldots, c_{S_1,\nu(D_{S_1})}\}, \quad \{c_{S_2,\nu(1)}, c_{S_2,\nu(2)}), \ldots, c_{S_2,\nu(D_{S_2})}\},$$

will necessarily be distinct. The essential property of a free group is that two (reduced) words are equal if and only if they are identical. Thus any collisions the AEHash might have cannot emerge when the string is replaced by the sequence of elements in the fixed braids, and must stem from an element in the kernel of the function that takes braids to colored Burau/permutations pairs, or colored Burau/permutation pairs to matrices with field entries/permutation pairs.

An upper bound for the running time of the AEHash can be obtained as follows. Let

$$L_C = \mathrm{Max}\{\mathrm{ArtinLength}(c_i) \mid i = 0, \ldots, 2^\lambda - 1\}.$$

Then AEHash(S) requires at most

$$L_C \cdot \frac{\mathrm{Card}(S)}{\lambda}$$

e-multiplications in order to obtain the $N \times N$ matrix. Once this matrix is in place the entries must be converted to bits. The individual braids, c_i, must have sufficiently long length so that the initial matrix the AEHash produces does not have too many zeros.

5 Statistical testing of the AEHash function

We now describe various statistical tests we performed on the AEHash function.

5.1 Bit flip test

In this test we first choose a random message of length from 1 to 256 bytes. We computed the hash of the original message and then iterated through the full message, flipping one bit at a time. After each bitflip we compute another hash (and then revert the bitflip before continuing on). Then we compare the Hamming distance of each hash result to the original hash.

Focussing on the case of the braid group B_8 and the finite field F_{16} we obtained the following test results.

Case 1. $\lambda = 5$ bit-flip test (2,088,408 checks):
- min: 47,
- max: 166,
- mean: 127.87,
- median: 128,
- stddev: 8.24.

Case 2. $\lambda = 7$ bit-flip test (2,056,088 checks):
- min: 50,
- max: 165,
- mean: 127.91,
- median: 128,
- stddev: 8.14.

For $N = 8$ and $q = 16 = 2^4$, the AEHash length is $N^2 \log_2(q) = (8^2) \cdot 4 = 256$. Hence, for a good hash function, the median of the Hamming distance (of the hash before and after bit flips) should be $256/2 = 128$.

5.2 Collision test

This test is looking for collisions at specified offsets in the hash. In particular, this test chooses the offset(s) into the hash output to check by iterating through every possible offset or offset-combination depending on the test configuration. Once the offset values are fixed, it loops for a number of trials. In each trial it chooses two random messages of length twice that of a hash, then hashes them, and compares the hash values at the chosen offsets. We expect this to find a collision every 1 in 2^b trials, where $b = 8^o$ and o is the number of offset bytes checked, i.e., we expect to find a collision every 1 in 256 trials with a single offset, and every 1 in 65,536 trials when checking for two offsets.

Focussing on the case of the braid group B_8 and the finite field F_{16} we obtained the following test results.

Case 1. $\lambda = 5$:
- 1-octet collisions (25,600 iterations per offset, so expecting 100):
  - min: 81,
  - max: 124,
  - mean: 101.63,
  - median: 101.5,
  - stddev: 9.97.
- 2-octet collisions (655,360 iterations, expecting 10):
  - min: 1,
  - max: 23,
  - mean: 9.96,
  - median: 10,
  - stddev: 3.16.

Case 2. $\lambda = 7$:
- 1-octet collisions (25,600 iterations per offset, so expecting 100):
  - min: 88,
  - max: 127,
  - mean: 100.16,
  - median: 97,
  - stddev: 8.99.
- 2-octet collisions (655,360 iterations, expecting 10):
  - min: 1,
  - max: 22,
  - mean: 10.03,
  - median: 10,
  - stddev: 3.29.

5.3 NIST statistical test suite

We ran the AEHash through the NIST Statistical Test Suite [5]. This is a statistical package that was developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. The tests focus on a variety of different types of non-randomness that could exist within a given sequence. The package then applies standard statistical significance methodology to produce a p-value for each test, which can then be compared to a (standard) level of significance α. For our tests, we chose $\alpha = 0.01$, as recommended by [5].

Altogether there are fifteen different tests in the suite, some of which are broken up into various subtests. Among the full set of tests are the following:

- The *Frequency (Monobit) Test*, which gives a most basic test of randomness of a sequence. It determines whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.
- The *Runs Test*. By definition a run in a sequence is a subsequence of identical elements. This test decides whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, it detects whether the oscillation between such zeros and ones is too fast or too slow.
- The *Binary Matrix Rank Test*, which checks for linear dependencies among fixed length substrings of the original sequence.
- The *Non-overlapping Template Matching Test*. This test counts the number of occurrences of a fixed set of pre-specified target strings, and rejects sequences exhibiting too many or too few occurrences of a given aperiodic pattern For a given target string of length m, an m-bit window is used to search for this pattern. If the pattern is not found, the window slides one bit position to the right, and the search is continued. If the pattern is found, then the window is reset to the bit after the found pattern, and again the search continues. The NIST suite uses a collection of 149 aperiodic patterns for this test.

For full details of all the tests in the suite, we refer to [5].

In our tests we generated the test data by creating an 8-byte input message using a counter and then hashing the input message (counter) until we had sufficient data to produce 10 streams of 100,000 bits each. These streams were run through the test suite. Overall the AEHash function performed extremely well. For only six of the 149 non-overlapping template tests did any bitstream appear nonrandom according to the test, and in each case it was only 1 of the 10 bitstreams. All 161 computed p-values were above $\alpha = 0.01$ save for two, which were 0.0043 and 0.0088. To put these results in perspective, we also tested 10 bitstreams of 100,000 bits generated using output from `/dev/random`, which is a special file found on Unix-like systems that gathers environmental noise from device drivers and other sources to produce a blocking pseudorandom number generator. These bitstreams exhibited similar behavior, and in fact failed 22 of the non-overlapping template tests. Two p-values from the bitstreams from `/dev/random` were also found to be lower than our threshold.

References

[1] I. Anshel, M. Anshel, D. Goldfeld and S. Lemieux, Key agreement, the algebraic eraser™, and lightweight cryptography, in: *Algebraic Methods in Cryptography*, Contemp. Math. 418, American Mathematical Society, Providence (2006), 1–34.
[2] I. Anshel and D. Goldfeld, Cryptographic hash function, US Patent number 8,972,715, March 3, 2015.
[3] J. Birman, K. H. Ko and S. J. Lee, A new approach to the word and conjugacy problems in the braid groups, *Adv. Math.* **139** (1998), no. 2, 322–353.
[4] P. Dehornoy, A fast method for comparing braids, *Adv. Math.* **125** (1997), no. 2, 200–235.
[5] National Institute of Standards and Technology, NIST Statistical Test Suite, available from http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html.