Edon–$\mathcal{R}(256, 384, 512)$ — an efficient implementation of Edon–$\mathcal{R}$ family of cryptographic hash functions

Danilo Gligoroski, Svein Johan Knapskog

Abstract. We have designed three fast implementations of a recently proposed family of hash functions Edon–$\mathcal{R}$. They produce message digests of length $n = 256, 384, 512$ bits and project security of $2^{\frac{n}{2}}$ hash computations for finding collisions and 2^n hash computations for finding preimages and second preimages. The design is not the classical Merkle-Damgård but can be seen as wide-pipe iterated compression function. Moreover the design is based on using huge quasigroups of orders 2^{256}, 2^{384} and 2^{512} that are constructed by using only bitwise operations on 32 bit values (additions modulo 2^{32}, XORs and left rotations). Initial Reference C code achieves processing speeds of 16.18 cycles/byte, 24.37 cycles/byte and 32.18 cycles/byte on x86 (Intel and AMD microprocessors). In this paper we give their full description, as well as an initial security analysis.

Keywords: hash function, Edon–$\mathcal{R}$, quasigroup

Classification: 94A60, 20N05, 05B05, 68P30

1. Introduction

On the Second NIST Hash Workshop a family of hash functions Edon–$\mathcal{R}$ was proposed [11]. The initial design was by general quasigroups of relatively small order (up to 256), and the approach was without concrete realization of those hash functions. No concrete measurements about the speed of those hash functions were given, although the authors admitted that the computational speed of their design is slow.

In this paper we describe three concrete realizations of Edon–$\mathcal{R}$ that produce hash outputs of 256, 384 and 512 bits. We use bitwise operations on 32 bit values (additions modulo 2^{32}, XORs and left rotations) to construct quasigroups of huge order (2^{256}, 2^{384} and 2^{512}) and then we use those quasigroups as a basis for implementing the compression function of Edon–$\mathcal{R}$. We will show that the designed quasigroups lack some of the laws that are satisfied in groups such as commutativity and associativity. That is similar to the approach in the original proposal for the Edon–$\mathcal{R}$ family of cryptographic hash functions. Thus, we are relying our claims about the security of our concrete realization of Edon–$\mathcal{R}$ hash functions on the difficulty of solving general quasigroup equations.

The organization of the paper is as follows: In Section 2 we give some basic mathematical definitions, a definition of a general compression function of Edon–$\mathcal{R}$ with only three blocks, and a definition of three huge quasigroups of orders 2^{256}, 2^{384} and 2^{512}, in Section 3 we define three hash functions Edon–$\mathcal{R}(256, 384, 512)$, in Section 4 we give a design rationale, in Section 5 we give some implementation characteristics, in Section 6 we give an initial security analysis of the proposed hash functions and we conclude the paper by Section 7.

2. Mathematical preliminaries and notation

In this section we need to repeat some parts of the definition of the class of one-way candidate functions $\mathcal{R}_1$ recently defined in [11], [12]. For that purpose we need also several brief definitions for quasigroups and quasigroup string transformations.

A quasigroup $(Q, *)$ is an algebraic structure consisting of a nonempty set Q and a binary operation $* : Q^2 \to Q$ with the property that each of the equations

$$
\begin{aligned}
a * x &= b \\
y * a &= b
\end{aligned} \tag{1}
$$

has unique solutions x and y in Q. Closely related combinatorial structures to finite quasigroups are Latin squares, since the main body of the multiplication table of a quasigroup is just a Latin square. More detailed information about theory of quasigroups, quasigroup string processing, Latin squares and hash functions can be found in [1], [19], [20], [21].

For the description of the algorithm we use the following definitions:

Definition 1 ([12] Quasigroup reverse string transformation $\mathcal{R}_1 : Q^r \to Q^r$)**.**

Let r be a positive integer, let $(Q, *)$ be a quasigroup and $a_j, b_j \in Q$. For each fixed $m \in Q$ define first the transformation $Q_m : Q^r \to Q^r$ by

$$
Q_m(a_0, a_1, \dots, a_{r-1}) = (b_0, b_1, \dots, b_{r-1}) \iff b_i := \begin{cases} m * a_0, & i = 0 \\ b_{i-1} * a_i, & 1 \le i \le r-1. \end{cases}
$$

Then define $\mathcal{R}_1$ as composition of transformations of kind Q_m, for suitable choices of the indexes m, as follows:

$$
\mathcal{R}_1(a_0, a_1, \dots, a_{r-1}) := Q_{a_0}(Q_{a_1} \dots (Q_{a_{r-1}}(a_0, a_1, \dots, a_{r-1}))).
$$

Note that the word "reverse" in the definition of $\mathcal{R}_1$ comes from the fact that the order of the indexes m is reverse to the original ordering of the letters in the string that is transformed by $\mathcal{R}_1$. It was conjectured in [11], [12] that $\mathcal{R}_1$ is one-way function (under some assumptions about the underlying quasigroup $(Q, *)$)

	a_0	a_1	a_2
a_2	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$
a_1	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$
a_0	b_0	b_1	b_2

a.

	x_0	x_1	x_2
x_2	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$
x_1	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$
x_0	b_0	b_1	b_2

b.

Table 1. **a.** Schematic presentation of the function $\mathcal{R}_1$ for $r = 3$; **b.** The conjectured one-wayness of $\mathcal{R}_1$ comes from the difficulty to solve a system of three equations where b_0, b_1 and b_2 are given, and $a_0 = x_0$, $a_1 = x_1$ and $a_2 = x_2$ are indeterminate variables.

and that the complexity of its inverting is exponential i.e. that inverting $\mathcal{R}_1$ has a complexity $O(|Q|^{\frac{r}{3}})$, where $|Q|$ is the size of the set Q.

In our construction of Edon–$\mathcal{R}(n)$, $n = 256, 384, 512$, we use the function $\mathcal{R}_1$ with $r = 3$. The transformation can be schematically presented by Table 1a.

The conjectured one-wayness of $\mathcal{R}_1$ can be explained by Table 1b. Namely, let us take that only the values b_0, b_1 and b_2 are given. In order to find pre-image values $a_0 = x_0$, $a_1 = x_1$ and $a_2 = x_2$ we can use Definition 1 and obtain the following equalities for the elements of Table 1b:
$x_0^{(1)} = x_2 * x_0$; $x_1^{(1)} = (x_2 * x_0) * x_1$; $x_2^{(1)} = ((x_2 * x_0) * x_1) * x_2$; $x_0^{(2)} = x_1 * (x_2 * x_0)$; $x_1^{(2)} = (x_1 * (x_2 * x_0)) * ((x_2 * x_0) * x_1)$; $x_2^{(2)} = \big((x_1 * (x_2 * x_0)) * ((x_2 * x_0) * x_1)\big) * \big(((x_2 * x_0) * x_1) * x_2\big)$.

From them, we can obtain the following system of quasigroup equations with indeterminates x_0, x_1, x_2:

$$\begin{cases} b_0 = x_0 * (x_1 * (x_2 * x_0)) \\ b_1 = b_0 * \big((x_1 * (x_2 * x_0)) * ((x_2 * x_0) * x_1)\big) \\ b_2 = b_1 * \Big(\big((x_1 * (x_2 * x_0)) * ((x_2 * x_0) * x_1)\big) * \big(((x_2 * x_0) * x_1) * x_2\big)\Big). \end{cases}$$

One can show that for any given $a_0 = x_0 \in Q$ either there are values of $a_1 = x_1$ and $a_2 = x_2$ as a solution or there is no solution. However, if the quasigroup operation is non-commutative, non-associative, the quasigroup operations are not linear in the underlying algebraic structure, and if the size of the quasigroup is very big (for example 2^{256}, 2^{384} or 2^{512}) then solving this simple system of three quasigroup equations is hard. Actually there is no known efficient method for solving such systems of quasigroup equations.

Of course, one inefficient method for solving that system would be to try every possible value for $a_0 = x_0 \in Q$ until obtaining other two indeterminates $a_1 = x_1$ and $a_2 = x_2$. That brute force method would require in average $\frac{1}{2}|Q|$ attempts to guess $a_0 = x_0 \in Q$ before solving the system.

2.1 Definition of quasigroups of huge order

In this section we describe the construction of quasigroups of huge orders (2^{256}, 2^{384} and 2^{512}). We use the following notation: Q is a set of cardinality 2^n, and elements $x \in Q$ are represented in their bitwise form as n-bit words

$$x \equiv (\overline{x}_0, \overline{x}_1, \ldots, \overline{x}_{n-2}, \overline{x}_{n-1}) \equiv \overline{x}_0 \cdot 2^{n-1} + \overline{x}_1 \cdot 2^{n-2} + \ldots + \overline{x}_{n-2} \cdot 2 + \overline{x}_{n-1}$$

where $\overline{x}_i \in \{0,1\}$.

Actually, we shall be constructing quasigroups $(Q, *)$ as isotopes of $\mathbb{Z}_2^n$. We shall thus define $\pi_i \in \mathrm{Sym}(\mathbb{Z}_2^n)$, $1 \leq i \leq 3$ so that

$$a * b \equiv \pi_1(\pi_2(a) \oplus_n \pi_3(b))$$

for all $a, b \in \mathbb{Z}_2^n$. Note that $\oplus_n$, the operation of $\mathbb{Z}_2^n$, can be identified with the Bitwise eXclusive OR (XOR) upon bit strings of length n.

Let us denote by $Q_{256} = \{0,1\}^{256}$, $Q_{384} = \{0,1\}^{384}$ and $Q_{512} = \{0,1\}^{512}$ the corresponding sets of 256–bit, 384–bit and 512–bit words. Our intention is to define Edon–$\mathcal{R}$ by the following bitwise operations on 32 bit values: 1. Rotation of 32 bits to the left for r positions, 2. Bitwise XOR operations on 32–bit words, 3. Addition between 32–bit words modulo 2^{32}.

Further, we introduce the following convention: elements $X \in Q_{256}$ are represented as $X = (X_0, X_1, \ldots, X_7)$, elements $X \in Q_{384}$ are represented as $X = (X_0, X_1, \ldots, X_{11})$, and elements $X \in Q_{512}$ are represented as $X = (X_0, X_1, \ldots, X_{15})$, where X_i are 32–bit words.

The left rotation of a 32–bit word Y by r positions will be denoted by $\mathrm{ROT}\,L_r(Y)$. Note that this operation can be expressed as a linear matrix-vector multiplication over the ring $(\mathbb{Z}_2, +, \times)$ i.e. $\mathrm{ROT}\,L_r(Y) = \mathbf{E}_r \cdot Y$ where $\mathbf{E}_r \in \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$ is a matrix obtained from the identity matrix by rotating its columns by r positions in the direction top to bottom. Further on, if we have a vector $X \in Q_{256}$ represented as $X = (X_0, X_1, \ldots, X_7)$ and we want to rotate all X_i by r_i $(0 \leq i \leq 7)$ positions to the left, then we denote that operation by $\mathrm{ROT}\,L_{\mathbf{r}}(X)$, where $\mathbf{r} = (r_0, \ldots, r_7) \in \{0, 1, \ldots, 31\}^7$ is called the rotation vector. The operation $\mathrm{ROT}\,L_{\mathbf{r}}(X)$ can also be represented as a linear matrix-vector multiplication over the ring $(\mathbb{Z}_2, +, \times)$ i.e. $\mathrm{ROT}\,L_{\mathbf{r}}(X) = \mathbf{D}_{\mathbf{r}} \cdot X$ where $\mathbf{D}_{\mathbf{r}} \in \mathbb{Z}_2^{256} \times \mathbb{Z}_2^{256}$,

$$\mathbf{D}_{\mathbf{r}} = \begin{pmatrix} \mathbf{E}_{r_0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_{r_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_4} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_5} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_6} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{E}_{r_7} \end{pmatrix},$$

submatrices $\mathbf{E}_{r_i} \in \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$, $0 \leq i \leq 7$ are obtained from the identity matrix by rotating its columns by r_i positions in the direction top to bottom, and the submatrices $\mathbf{0} \in \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$ are the zero matrix.

Similar notation will be used for the operations in Q_{384} and Q_{512} but here the range of indexes i will be $0 \leq i \leq 11$ and $0 \leq i \leq 15$.

Further, we use the following notation:

- $\mathbf{A}_{1,n}, \mathbf{A}_{3,n}, n = 256, 384, 512$ are nonsingular matrices of order $\frac{n}{32} \times \frac{n}{32}$, over the ring $(\mathbb{Z}_{2^{32}}, +, \times)$. The values of the elements in $\mathbf{A}_{1,n}$ and $\mathbf{A}_{3,n}$ will be only 0 or 1, since we want to avoid the operations of multiplication (as more costly microprocessor operations) in the ring $(\mathbb{Z}_{2^{32}}, +, \times)$, and stay only with operations of addition.
- $\mathbf{A}_{2,n}, \mathbf{A}_{4,n}$ are nonsingular matrices of order $n \times n$ over the ring $(\mathbb{Z}_2, +, \times)$. Since we want to apply XOR operations on 32–bit registers, the matrices $\mathbf{A}_{2,n}$ and $\mathbf{A}_{4,n}$ will be of the form

$$\begin{pmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} & \dots & \mathbf{B}_{1,\frac{n}{32}} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} & \dots & \mathbf{B}_{2,\frac{n}{32}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{\frac{n}{32},1} & \mathbf{B}_{\frac{n}{32},1} & \dots & \mathbf{B}_{\frac{n}{32},\frac{n}{32}} \end{pmatrix},$$

where $\mathbf{B}_{i,j} \in \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$, $1 \leq i, j \leq \frac{n}{32}$ are either the identity matrix or the zero matrix.

Now we give the formal definitions for the permutations: $\pi_{1,n}$, $\pi_{2,n}$, $\pi_{3,n}$.

Definition 2. Transformations $\pi_{1,n} : Q_n \to Q_n$ $(n = 256, 384, 512)$ are defined as:

$$\begin{aligned} &\pi_{1,256}(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7) = \\ &\qquad (X_5, X_6, X_7, X_0, X_1, X_2, X_3, X_4), \\ &\pi_{1,384}(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}) = \\ &\qquad (X_7, X_8, X_9, X_{10}, X_{11}, X_0, X_1, X_2, X_3, X_4, X_5, X_6), \\ &\pi_{1,512}(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}) = \\ &\qquad (X_9, X_{10}, X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8). \end{aligned}$$

Lemma 1. *Transformations $\pi_{1,n}$ are permutations.* □

Definition 3. Transformations $\pi_{2,n} : Q_n \to Q_n$ and $\pi_{3,n} : Q_n \to Q_n$ are defined as:

$$\pi_{2,n} \equiv \mathbf{A}_{1,n} \circ \mathrm{ROT}\, L_{\mathbf{r}_{1,n}} \circ \mathbf{A}_{2,n} \circ \mathrm{ROT}\, L_{\mathbf{r}_{2,n}},$$
$$\pi_{3,n} \equiv \mathbf{A}_{3,n} \circ \mathrm{ROT}\, L_{\mathbf{r}_{3,n}} \circ \mathbf{A}_{4,n} \circ \mathrm{ROT}\, L_{\mathbf{r}_{4,n}},$$

n	$\mathbf{r}_{1,n}$	$\mathbf{r}_{2,n}$	$\mathbf{r}_{3,n}$	$\mathbf{r}_{4,n}$
256	(1, 3, 4, 5, 7, 8, 10, 13)	(1, 4, 8, 9, 10, 12, 13, 14)	(2, 5, 6, 7, 8, 10, 11, 14)	(3, 4, 6, 8, 9, 11, 12, 13)
384	(1, 3, 4, 5, 7, 8, 10, 13, 0, 0, 0, 0)	(1, 4, 8, 9, 10, 12, 13, 14, 0, 0, 0, 0)	(2, 5, 6, 7, 8, 10, 11, 14, 0, 0, 0, 0)	(3, 4, 6, 8, 9, 11, 12, 13, 0, 0, 0, 0)
512	(1, 3, 4, 5, 7, 8, 10, 13, 0, 0, 0, 0, 0, 0, 0, 0)	(1, 4, 8, 9, 10, 12, 13, 14, 0, 0, 0, 0, 0, 0, 0, 0)	(2, 5, 6, 7, 8, 10, 11, 14, 0, 0, 0, 0, 0, 0, 0, 0)	(3, 4, 6, 8, 9, 11, 12, 13, 0, 0, 0, 0, 0, 0, 0, 0)

TABLE 2. Rotation vectors for definition of $\pi_{2,n}$ and $\pi_{3,n}$

where the rotation vectors are given in Table 2 and the matrices $\mathbf{A}_{i,n}$, $i = 1,2,3,4$, $n = 256, 384, 512$, are given in Table 3. The matrices $\mathbf{A}_{1,n}$ and $\mathbf{A}_{3,n}$ act in the ring $(\mathbb{Z}_{2^{32}}, +, \times)$ where the operation $+$ is addition modulo 2^{32} and matrices $\mathbf{A}_{2,n}$ and $\mathbf{A}_{4,n}$ act in the ring $(\mathbb{Z}_2, +, \times)$, where $\mathbf{1}, \mathbf{0} \in \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32}$ are the identity matrix and the zero matrix, and where the operation $+$ is bitwise XOR.

Lemma 2. *Transformations $\pi_{2,n}$ and $\pi_{3,n}$ are permutations on Q_n.*

PROOF: This follows immediately from the fact that all transformations $\mathbf{A}_{i,n}$ and $\mathrm{ROT}\, L_{\mathbf{r}_{i,n}}$, $i = 1,2,3$, $n = 256, 384, 512$ are expressed by nonsingular matrices over the rings $(\mathbb{Z}_{2^{32}}, +, \times)$ or $(\mathbb{Z}_2, +, \times)$. $\square$

Theorem 1. *Operations $*_n : Q_n^2 \to Q_n$ defined as:*

$$a *_n b = \pi_{1,n}(\pi_{2,n}(a) \oplus_n \pi_{3,n}(b))$$

are non-commutative quasigroup operations that are not loops.

PROOF: We give a proof for $n = 256$ and the other two cases are similar.

To show that $*_{256}$ is not a loop we have to show that there is no unit element $e \in Q_{256}$ such that for every $a \in Q_{256}$, $a *_{256} e = a = e *_{256} a$. Let us suppose that there is a neutral element $e \in Q_{256}$. Let us first put

$$\pi_{2,256}(e) \oplus_{256} \pi_{3,256}(e) = \mathrm{Const}_e$$

where $\mathrm{Const}_e \in Q_{256}$ is a constant element.

From the definition of the quasigroup operation $*_{256}$ for the neutral element e we get:

$$\pi_{1,256}(\pi_{2,256}(e) \oplus_{256} \pi_{3,256}(a)) = \pi_{1,256}(\pi_{2,256}(a) \oplus_{256} \pi_{3,256}(e)).$$

Since $\pi_{1,256}$ is a permutation we can remove it from the last equation and we get:

$$\pi_{2,256}(e) \oplus_{256} \pi_{3,256}(a) = \pi_{2,256}(a) \oplus_{256} \pi_{3,256}(e)$$

n	$\mathbf{A}_{1,n}$	$\mathbf{A}_{2,n}$
256	$\begin{pmatrix} 0&1&1&0&1&0&1&1 \\ 1&1&0&1&1&0&0&1 \\ 1&1&1&0&0&0&1&1 \\ 0&1&0&1&1&1&1&0 \\ 1&0&0&1&1&1&1&0 \\ 1&0&1&0&1&1&0&1 \\ 1&1&1&1&0&1&0&0 \\ 0&0&1&1&0&1&1&1 \end{pmatrix}$	$\begin{pmatrix} 1&0&0&1&0&1&0&0 \\ 0&0&1&0&0&1&1&0 \\ 0&0&0&1&1&1&0&0 \\ 1&0&1&0&0&0&0&1 \\ 0&1&1&0&0&0&0&1 \\ 0&1&0&1&0&0&1&0 \\ 0&0&0&0&1&0&1&1 \\ 1&1&0&0&1&0&0&0 \end{pmatrix}$
384	$\begin{pmatrix} 0&1&1&1&0&0&0&1&1&0&1&1 \\ 1&1&0&0&1&1&1&0&1&0&1&0 \\ 1&0&1&0&1&1&0&0&0&1&1&1 \\ 0&1&0&1&1&1&1&1&0&0&0&1 \\ 1&1&0&1&0&0&0&1&1&1&0&1 \\ 1&0&1&0&1&1&1&1&1&0&0&0 \\ 1&1&1&1&1&0&0&0&0&1&1&0 \\ 1&0&1&0&0&1&1&0&1&1&0&1 \\ 0&0&0&1&1&1&0&1&1&1&1&0 \\ 1&0&0&1&1&0&1&1&0&0&1&1 \\ 0&1&1&0&0&1&1&1&0&1&1&0 \\ 0&1&1&1&0&0&1&0&1&1&0&1 \end{pmatrix}$	$\begin{pmatrix} 1&0&0&0&1&1&1&0&0&1&0&0 \\ 0&0&1&1&0&0&0&1&0&1&0&1 \\ 0&1&0&1&0&0&1&1&1&0&0&0 \\ 1&0&1&0&0&0&0&0&1&1&1&0 \\ 0&0&1&0&1&1&1&0&0&0&1&0 \\ 0&1&0&1&0&0&0&0&0&1&1&1 \\ 0&0&0&0&0&1&1&1&1&0&0&1 \\ 0&1&0&1&1&0&0&1&0&0&1&0 \\ 1&1&1&0&0&0&1&0&0&0&0&1 \\ 0&1&1&0&0&1&0&0&1&1&0&0 \\ 1&0&0&1&1&0&0&0&1&0&0&1 \\ 1&0&0&0&1&1&0&1&0&0&1&0 \end{pmatrix}$
512	$\begin{pmatrix} 1&0&1&1&1&0&1&0&0&1&1&0&0&1&0&1 \\ 1&1&0&1&1&0&1&0&1&0&1&0&0&1&0&1 \\ 1&1&0&1&0&0&1&1&0&0&1&1&1&0&1&0 \\ 0&1&0&1&1&0&1&0&1&0&1&1&0&0&1&1 \\ 1&1&1&0&1&0&0&1&1&1&0&1&0&1&0&0 \\ 0&1&1&0&1&1&0&1&1&1&1&0&1&0&0&0 \\ 1&0&0&0&0&1&0&1&0&1&1&1&1&1&1&0 \\ 1&0&0&1&0&1&0&0&1&1&0&1&1&1&1&0 \\ 1&1&1&1&1&1&0&0&0&1&0&0&0&1&0&1 \\ 1&1&1&1&0&1&1&0&1&1&0&0&0&0&0&1 \\ 0&1&1&0&0&0&1&0&1&0&0&1&1&1&1&1 \\ 0&0&0&1&0&1&0&1&0&0&1&1&1&1&1&1 \\ 0&0&1&1&0&1&1&1&0&1&0&0&1&1&1&0 \\ 1&0&0&0&1&1&1&1&0&1&0&1&1&0&0&1 \\ 0&0&1&0&1&1&1&1&1&0&1&1&0&0&1&0 \\ 0&1&1&0&1&0&0&1&1&0&1&0&1&0&1&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&0&1&0&1&1&0&0&1&1&0&1&0 \\ 0&0&1&0&0&1&0&1&0&1&0&1&1&0&1&0 \\ 0&0&1&0&1&1&0&0&1&1&0&0&0&1&0&1 \\ 1&0&1&0&0&1&0&1&0&1&0&0&1&1&0&0 \\ 0&0&0&1&0&1&1&0&0&0&1&0&1&0&1&1 \\ 1&0&0&1&0&0&1&0&0&0&0&1&0&1&1&1 \\ 0&1&1&1&1&0&1&0&1&0&0&0&0&0&0&1 \\ 0&1&1&0&1&0&1&1&0&0&1&0&0&0&0&1 \\ 0&0&0&0&0&0&1&1&1&0&1&1&1&0&1&0 \\ 0&0&0&0&1&0&0&1&0&0&1&1&1&1&1&0 \\ 1&0&0&1&1&1&0&1&0&1&1&0&0&0&0&0 \\ 1&1&1&0&1&0&1&0&1&1&0&0&0&0&0&0 \\ 1&1&0&0&1&0&0&0&1&0&1&1&0&0&0&1 \\ 0&1&1&1&0&0&0&0&1&0&1&0&0&1&1&0 \\ 1&1&0&1&0&0&0&0&0&1&0&0&1&1&0&1 \\ 1&0&0&1&0&1&1&0&0&1&0&1&0&1&0&0 \end{pmatrix}$

n	$\mathbf{A}_{3,n}$	$\mathbf{A}_{4,n}$
256	$\begin{pmatrix} 0&1&0&0&1&1&1&1 \\ 0&1&1&0&1&0&1&1 \\ 1&1&0&1&0&0&1&1 \\ 1&0&1&1&0&1&1&0 \\ 0&0&1&1&1&1&0&1 \\ 1&0&0&1&1&1&1&0 \\ 1&1&1&1&1&0&0&0 \\ 1&1&1&0&0&1&0&1 \end{pmatrix}$	$\begin{pmatrix} 1&0&1&1&0&0&0&0 \\ 1&0&0&1&0&1&0&0 \\ 0&0&1&0&1&1&0&0 \\ 0&1&0&0&1&0&0&1 \\ 1&1&0&0&0&0&1&0 \\ 0&1&1&0&0&0&0&1 \\ 0&0&0&0&0&1&1&1 \\ 0&0&0&1&1&0&1&0 \end{pmatrix}$
384	$\begin{pmatrix} 0&0&0&0&1&1&1&1&0&1&1&1 \\ 0&1&1&1&0&0&0&1&1&0&1&1 \\ 1&1&1&1&0&0&0&1&0&1&1&0 \\ 1&0&1&0&1&1&0&1&1&1&0&0 \\ 0&1&0&1&1&0&1&1&0&1&0&1 \\ 1&1&1&0&0&1&1&0&0&1&1&0 \\ 1&1&0&0&1&0&1&0&1&0&1&1 \\ 1&0&1&1&1&0&0&0&1&1&0&1 \\ 1&1&0&0&0&1&1&1&1&0&0&1 \\ 1&0&0&1&0&1&1&0&1&1&1&0 \\ 0&0&1&1&1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0&0&0&1&1 \end{pmatrix}$	$\begin{pmatrix} 1&1&1&1&0&0&0&0&1&0&0&0 \\ 1&0&0&0&1&1&1&0&0&1&0&0 \\ 0&0&0&0&1&1&1&0&1&0&0&1 \\ 0&1&0&1&0&0&1&0&0&0&1&1 \\ 1&0&1&0&0&1&0&0&1&0&1&0 \\ 0&0&0&1&1&0&0&1&1&0&0&1 \\ 0&0&1&1&0&1&0&1&0&1&0&0 \\ 0&1&0&0&0&1&1&1&0&0&1&0 \\ 0&0&1&1&1&0&0&0&0&1&1&0 \\ 0&1&1&0&1&0&0&1&0&0&0&1 \\ 1&1&0&0&0&0&0&0&0&1&1&1 \\ 1&0&0&0&0&0&1&1&1&1&0&0 \end{pmatrix}$
512	$\begin{pmatrix} 0&1&1&1&1&0&0&0&1&0&1&1&0&1&1&0 \\ 1&1&0&1&0&0&1&1&0&0&0&1&0&1&1&1 \\ 1&0&0&1&0&1&1&0&1&0&1&0&1&0&1&1 \\ 1&0&1&0&1&0&1&0&0&1&1&1&1&0&0&1 \\ 1&0&0&1&1&1&0&1&0&1&1&0&1&1&0&0 \\ 0&0&1&0&1&1&1&0&1&0&1&0&1&0&1&1 \\ 0&1&0&1&1&0&0&1&1&1&0&1&1&0&1&0 \\ 0&0&1&0&1&1&0&1&0&1&0&1&0&1&1&1 \\ 0&1&0&0&0&1&1&1&0&1&1&1&0&1&1&0 \\ 1&1&1&1&0&0&0&1&1&1&1&0&0&0&0&1 \\ 1&1&1&1&0&1&1&0&0&0&0&1&1&0&1&0 \\ 1&1&0&0&0&1&1&1&1&0&0&1&1&1&0&0 \\ 0&1&0&1&1&1&0&0&1&0&1&0&1&1&0&1 \\ 1&1&1&0&1&0&1&1&1&1&1&0&0&0&0&0 \\ 1&0&1&0&0&0&0&1&1&1&0&0&1&1&1&1 \\ 0&0&1&1&1&1&1&0&0&1&0&1&0&1&0&1 \end{pmatrix}$	$\begin{pmatrix} 1&0&0&0&0&1&1&1&0&1&0&0&1&0&0&1 \\ 0&0&1&0&1&1&0&0&1&1&1&0&1&0&0&0 \\ 0&1&1&0&1&0&0&1&0&1&0&1&0&1&0&0 \\ 0&1&0&1&0&1&0&1&1&0&0&0&0&1&1&0 \\ 0&1&1&0&0&0&1&0&1&0&0&1&0&0&1&1 \\ 1&1&0&1&0&0&0&1&0&1&0&1&0&1&0&0 \\ 1&0&1&0&0&1&1&0&0&0&1&0&0&1&0&1 \\ 1&1&0&1&0&0&1&0&1&0&1&0&1&0&0&0 \\ 1&0&1&1&1&0&0&0&1&0&0&0&1&0&0&1 \\ 0&0&0&0&1&1&1&0&0&0&0&1&1&1&1&0 \\ 0&0&0&0&1&0&0&1&1&1&1&0&0&1&0&1 \\ 0&0&1&1&1&0&0&0&0&1&1&0&0&0&1&1 \\ 1&0&1&0&0&0&1&1&0&1&0&1&0&0&1&0 \\ 0&0&0&1&0&1&0&0&0&0&0&1&1&1&1&1 \\ 0&1&0&1&1&1&1&0&0&0&1&1&0&0&0&0 \\ 1&1&0&0&0&0&0&1&1&0&1&0&1&0&1&0 \end{pmatrix}$

TABLE 3. Matrices $\mathbf{A}_{i,n}$, $i = 1, 2, 3, 4$, $n = 256, 384, 512$

and if we rearrange the last equation we get:

$$\pi_{2,256}(a) \oplus_{256} \pi_{3,256}(a) = \pi_{2,256}(e) \oplus_{256} \pi_{3,256}(e) = \mathrm{Const}_e$$

The last equation states that for every $a \in Q_{256}$ the expression $\pi_{2,256}(a) \oplus_{256} \pi_{3,256}(a)$ is a constant. This is not true. For example $\pi_{2,256}(1) \oplus_{256} \pi_{3,256}(1) \neq \pi_{2,256}(2) \oplus_{256} \pi_{3,256}(2)$. Thus we conclude that $*_{256}$ is not a loop. $\square$

Note that the quasigroups cannot be associative since every associative quasigroup is a group and every group possesses a unit element.

Having defined three quasigroup operations $*_{256}$, $*_{384}$ and $*_{512}$ we define three one-way functions $\mathcal{R}_{1,256}$, $\mathcal{R}_{1,384}$ and $\mathcal{R}_{1,512}$ as follows:

Definition 4.

1. $\mathcal{R}_{1,256} : Q_{256}^3 \to Q_{256}^3 \equiv \mathcal{R}_1$ where $\mathcal{R}_1$ is defined as in Definition 1 over Q_{256} with the quasigroup operation $*_{256}$.
2. $\mathcal{R}_{1,384} : Q_{384}^3 \to Q_{384}^3 \equiv \mathcal{R}_1$ where $\mathcal{R}_1$ is defined as in Definition 1 over Q_{384} with the quasigroup operation $*_{384}$.
3. $\mathcal{R}_{1,512} : Q_{512}^3 \to Q_{512}^3 \equiv \mathcal{R}_1$ where $\mathcal{R}_1$ is defined as in Definition 1 over Q_{512} with the quasigroup operation $*_{512}$.

3. Edon–$\mathcal{R}(256, 384, 512)$ hash algorithm

Having one-way quasigroup functions $\mathcal{R}_{1,256}$, $\mathcal{R}_{1,384}$ and $\mathcal{R}_{1,512}$, we now define three hash algorithms Edon–$\mathcal{R}(256)$, Edon–$\mathcal{R}(384)$ and Edon–$\mathcal{R}(512)$ that map messages M of arbitrary length of l bits ($l \leq 2^{128}$) into hash values of 256, 384 or 512 bits.

3.1 Padding

Padding of the messages M of arbitrary length of l bits is done by the standard Merkle-Damgård strengthening [9], [22]. Let us shortly denote all three hash functions as Edon–$\mathcal{R}(n)$ where the parameter n can take the values 256, 384 or 512.

The padding of a message M that is l bits long is done by the following procedure:

1. Append the bit 1 at the end of the message.
2. Append the smallest amount l_1 of zero bits, such that $l+1+l_1+128 \equiv 0 \ (\texttt{mod}\ n)$.
3. Represent the original length l of the message M as an 128–bit number and append it at the end of the message. The length of the appended message M' becomes a multiple of n bits. Let the appended message be represented as $M' = M_1 M_2 \dots M_N$ where M_i is an n–bit long block.

3.2 Initial predetermined values

The definition of Edon–$\mathcal{R}(n)$ hash function includes one initial string H_0 of length $2n$ bits. That initial string is given as follows (represented in hexadecimal notation as concatenation of 32-bits chunks).

1. For $n = 256$, $H_0 =$ `0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0F10, 0x11121314, 0x15161718, 0x191A1B1C, 0x1D1E1F20, 0x21222324, 0x25262728, 0x292A2B2C, 0x2D2E2F30, 0x31323334, 0x35363738, x393A3B3C, 0x3D3E3F40.`
2. For $n = 384$, $H_0 =$ `0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0F10, 0x11121314, 0x15161718, 0x191A1B1C, 0x1D1E1F20, 0x21222324, 0x25262728, 0x292A2B2C, 0x2D2E2F30, 0x31323334, 0x35363738, 0x393A3B3C, 0x3D3E3F40, 0x41424344, 0x45464748, 0x494A4B4C, 0x4D4E4F50, 0x51525354, 0x55565758, 0x595A5B5C, 0x5D5E5F60.`

3. For $n = 512$, $H_0 =$ `0x01020304, 0x05060708, 0x090A0B0C, 0x0D0E0F10, 0x11121314, 0x15161718, 0x191A1B1C, 0x1D1E1F20, 0x21222324, 0x25262728, 0x292A2B2C, 0x2D2E2F30, 0x31323334, 0x35363738, 0x393A3B3C, 0x3D3E3F40, 0x41424344, 0x45464748, 0x494A4B4C, 0x4D4E4F50, 0x51525354, 0x55565758, 0x595A5B5C, 0x5D5E5F60, 0x61626364, 0x65666768, 0x696A6B6C, 0x6D6E6F70, 0x71727374, 0x75767778, 0x797A7B7C, 0x7D7E7F80.`

The initial values are obtained by concatenation of the 8–bit representation of the numbers $1, 2, \ldots, 128$.

3.3 Edon–$\mathcal{R}(n)$ hash function

Input: n and M, where: n is 256, 384 or 512, and M is the message to be hashed.
Output: A hash of length n bits.

1. **Pad** the message M, so the length of the padded message M' is multiple of n–bit words i.e. $|M'| = N \times n$.
2. **Initialize** H_0.
3. **Compute** the hash with the following iterative procedure:

$$\text{For } i = 1 \text{ to } N \text{ do}$$
$$H_i = \mathcal{R}_{1,n}(H_{i-1}||M_i) \ \mathtt{mod}\ 2^{2n};$$

Output:

$$Edon\text{-}\mathcal{R}(n)(M) = H_N \ \mathtt{mod}\ 2^n.$$

Since the one-way functions $\mathcal{R}_{1,n}$ are considered as transformations $\{0,1\}^{3n} \to \{0,1\}^{3n}$ for obtaining the intermediate value H_i, we apply the operation $\mathtt{mod}\ 2^{2n}$ that takes the last two n–bit words from the result of $\mathcal{R}_{1,n}$. Then, that value is concatenated (the operation "$||$") with the next message block, and so on. Finally, since the requested output from the hash function is n bits, we take just the last n–bit word from the H_N, denoted as the operation $\mathtt{mod}\ 2^n$.

4. Design rationale

4.1 Choosing basic 32–bit operations

We have decided to choose 32–bit operations of addition modulo 2^{32}, XOR-ing and left rotations as an optimum choice that can be efficiently implemented both on low-end 8–bit and 16–bit processors, as well as on modern 32–bit and 64–bit CPUs. In the past, several other cryptographic primitives have been designed following the same rationale as well, such as: Salsa20 [2], The Tiny Encryption Algorithm [27], or IDEA [15] — to name a few.

4.2 Choosing permutations π_1, π_2 and π_3

Our goal was to design a structure that is a non-commutative, non-associative, highly nonlinear quasigroup of huge order ($2^{256}, 2^{384}$ and 2^{512}) in order to apply the principles of the hash family Edon–$\mathcal{R}$. We have found a way how to construct such structures as isotopes of $\mathbb{Z}_2^n$, by applying some basic permutations π_1, π_2 and π_3 on the sets $\{0,1\}^{256}$, $\{0,1\}^{384}$ and $\{0,1\}^{512}$.

The permutations $\pi_{1,256}$, $\pi_{1,384}$ and $\pi_{1,512}$ are simple rotations on 256, 384 or 512–bit words. They can be effectively realized just by appropriate referencing of the 32–bit variables (after performing permutations π_2 and π_3). While the permutations π_2 and π_3 do the work of diffusion and nonlinear mixing separately on the first and the second argument of the quasigroup operations, after their outputs are XORed, the permutations π_1 introduce additional diffusion on the whole n–bit word. That diffusion then have influence on the next application of the quasigroup operation $*_n$ (since we apply three such operations in every row). The nonlinear mixing is achieved because we perform operations in two different rings: $(\mathbb{Z}_{2^{32}}, +, \times)$ and $(\mathbb{Z}_2, +, \times)$.

For the choice of the permutations π_2 and π_3 we had plenty of possibilities. However, since our design is based on quasigroups, it was a natural choice to use Latin squares in the construction of those permutations. Actually there is a long history of using Latin squares in the randomized experimental design (see for example [10]) as well as in cryptography [4], [5], [6], [24], [25].

Since for the permutations $\pi_{2,256}$ and $\pi_{3,256}$ we wanted to mix bijectively eight 32–bit variables, we have used the following 8×8 Latin squares:

$$L_1 = \begin{pmatrix} 2\,1\,7\,6\,3\,4\,0\,5 \\ 4\,3\,2\,5\,0\,7\,1\,6 \\ 7\,0\,1\,4\,6\,2\,5\,3 \\ 6\,7\,0\,1\,4\,5\,3\,2 \\ \underline{1\,4\,6\,3\,5\,0\,2\,7} \\ 0\,6\,5\,2\,1\,3\,7\,4 \\ 5\,2\,3\,0\,7\,6\,4\,1 \\ 3\,5\,4\,7\,2\,1\,6\,0 \end{pmatrix} = \begin{pmatrix} L_{1,1} \\ L_{1,2} \end{pmatrix} \qquad L_2 = \begin{pmatrix} 5\,7\,0\,3\,4\,6\,1\,2 \\ 6\,2\,1\,0\,7\,3\,4\,5 \\ 7\,1\,3\,6\,5\,4\,2\,0 \\ 4\,6\,7\,5\,2\,0\,3\,1 \\ \underline{1\,4\,6\,2\,3\,5\,0\,7} \\ 2\,5\,4\,1\,0\,7\,6\,3 \\ 3\,0\,5\,4\,1\,2\,7\,6 \\ 0\,3\,2\,7\,6\,1\,5\,4 \end{pmatrix} = \begin{pmatrix} L_{2,1} \\ L_{2,2} \end{pmatrix}$$

By splitting L_1 and L_2 on two (upper and lower) Latin rectangles $L_{1,1}$, $L_{1,2}$, $L_{2,1}$ and $L_{2,2}$ and taking columns of those rectangles as sets, we actually constructed four symmetric non-balanced block designs (for an excellent brief introduction on block designs see for example [10], [23]). The non-balanced symmetric block designs corresponding to $L_{1,1}$ and $L_{2,1}$ are with parameters $(v, k, \lambda) = (8, 5, \lambda)$ where $\lambda \in \{2, 3, 4\}$, and those corresponding to $L_{1,2}$ and $L_{2,2}$ are with parameters $(v, k, \lambda) = (8, 3, \lambda)$ where $\lambda \in \{0, 1, 2\}$. We used the incidence matrix obtained by $L_{1,1}$ to transform bijectively the variables by addition modulo 2^{32} (in the ring $(\mathbb{Z}_{2^{32}}, +, \times)$) and the incidence matrix obtained by $L_{1,2}$ to transform bijectively the variables by XORing of 32–bit variables (in the ring $(\mathbb{Z}_2, +, \times)$).

As we mentioned in Section 2.1, the matrix $\mathbf{A}_{1,256}$ is an 8×8 nonsingular matrix in the ring $(\mathbb{Z}_{2^{32}}, +, \times)$ and the matrix $\mathbf{A}_{2,256}$ is a 256×256 nonsingular matrix in the ring $(\mathbb{Z}_2, +, \times)$. Similarly from Latin rectangles $L_{2,1}$ and $L_{2,2}$ we got the nonsingular incidence matrices $\mathbf{A}_{3,256}$ and $\mathbf{A}_{4,256}$.

It is an interesting observation that we split the Latin rectangles in 5:3 ratio, not in 4:4 ratio. It comes from the fact that the symmetry of the corresponding formulas for calculation determinant of the incidence matrices when the splitting is 4:4, always gives the result 0 (singular value) in the ring $(\mathbb{Z}_2, +, \times)$.

$$L_3 = \left(\begin{array}{cccccccccccc}
11 & 0 & 9 & 6 & 3 & 4 & 10 & 8 & 5 & 7 & 1 & 2 \\
3 & 10 & 2 & 11 & 8 & 7 & 1 & 6 & 4 & 0 & 5 & 9 \\
1 & 5 & 0 & 7 & 9 & 8 & 4 & 11 & 10 & 3 & 2 & 6 \\
2 & 4 & 10 & 1 & 7 & 5 & 0 & 9 & 8 & 11 & 6 & 3 \\
10 & 1 & 11 & 5 & 0 & 6 & 3 & 2 & 9 & 4 & 7 & 8 \\
7 & 8 & 5 & 4 & 1 & 2 & 9 & 0 & 3 & 6 & 10 & 11 \\
8 & 6 & 4 & 3 & 11 & 0 & 2 & 5 & 7 & 10 & 9 & 1 \\
\hline
0 & 9 & 6 & 10 & 5 & 3 & 7 & 1 & 2 & 8 & 11 & 4 \\
6 & 2 & 3 & 8 & 10 & 1 & 5 & 4 & 11 & 9 & 0 & 7 \\
4 & 11 & 7 & 2 & 6 & 9 & 8 & 10 & 0 & 1 & 3 & 5 \\
5 & 7 & 1 & 9 & 4 & 10 & 11 & 3 & 6 & 2 & 8 & 0 \\
9 & 3 & 8 & 0 & 2 & 11 & 6 & 7 & 1 & 5 & 4 & 10
\end{array}\right) \quad
L_4 = \left(\begin{array}{cccccccccccc}
11 & 10 & 9 & 5 & 7 & 0 & 4 & 8 & 1 & 6 & 2 & 3 \\
4 & 7 & 0 & 8 & 11 & 2 & 10 & 9 & 6 & 5 & 3 & 1 \\
9 & 1 & 3 & 2 & 4 & 5 & 6 & 0 & 8 & 10 & 7 & 11 \\
5 & 11 & 1 & 9 & 6 & 10 & 8 & 3 & 7 & 0 & 4 & 2 \\
6 & 8 & 2 & 7 & 3 & 1 & 11 & 4 & 0 & 9 & 5 & 10 \\
7 & 3 & 10 & 4 & 1 & 9 & 0 & 2 & 11 & 8 & 6 & 5 \\
10 & 2 & 7 & 0 & 9 & 6 & 1 & 11 & 5 & 3 & 8 & 4 \\
\hline
2 & 9 & 11 & 1 & 8 & 7 & 3 & 5 & 10 & 4 & 0 & 6 \\
8 & 0 & 4 & 6 & 5 & 11 & 9 & 10 & 3 & 2 & 1 & 7 \\
3 & 6 & 5 & 10 & 0 & 8 & 2 & 1 & 4 & 7 & 11 & 9 \\
1 & 5 & 8 & 3 & 10 & 4 & 7 & 6 & 2 & 11 & 9 & 0 \\
0 & 4 & 6 & 11 & 2 & 3 & 5 & 7 & 9 & 1 & 10 & 8
\end{array}\right)$$

$$L_5 = \left(\begin{array}{cccccccccccccccc}
4 & 10 & 11 & 1 & 2 & 5 & 7 & 3 & 13 & 0 & 8 & 14 & 9 & 12 & 6 & 15 \\
0 & 15 & 1 & 10 & 8 & 7 & 13 & 12 & 9 & 3 & 14 & 11 & 6 & 5 & 2 & 4 \\
15 & 3 & 6 & 4 & 1 & 9 & 10 & 14 & 0 & 2 & 11 & 12 & 13 & 7 & 5 & 8 \\
6 & 1 & 3 & 11 & 0 & 2 & 14 & 8 & 5 & 9 & 15 & 13 & 7 & 4 & 10 & 12 \\
10 & 4 & 0 & 6 & 9 & 8 & 12 & 13 & 1 & 5 & 2 & 15 & 3 & 11 & 14 & 7 \\
13 & 0 & 14 & 3 & 4 & 10 & 9 & 11 & 15 & 8 & 1 & 5 & 12 & 6 & 7 & 2 \\
2 & 13 & 7 & 8 & 11 & 12 & 5 & 9 & 3 & 15 & 6 & 10 & 14 & 0 & 4 & 1 \\
3 & 6 & 10 & 15 & 13 & 4 & 11 & 0 & 2 & 1 & 12 & 7 & 5 & 9 & 8 & 14 \\
9 & 8 & 12 & 14 & 7 & 1 & 0 & 5 & 4 & 6 & 13 & 3 & 2 & 15 & 11 & 10 \\
\hline
5 & 9 & 15 & 2 & 12 & 14 & 8 & 6 & 11 & 4 & 7 & 1 & 10 & 13 & 3 & 0 \\
14 & 5 & 13 & 9 & 10 & 15 & 6 & 7 & 8 & 11 & 4 & 0 & 1 & 2 & 12 & 3 \\
1 & 11 & 5 & 13 & 14 & 0 & 2 & 4 & 7 & 12 & 3 & 6 & 8 & 10 & 15 & 9 \\
8 & 12 & 2 & 7 & 5 & 11 & 3 & 10 & 14 & 13 & 9 & 4 & 15 & 1 & 0 & 6 \\
11 & 7 & 8 & 5 & 3 & 6 & 1 & 15 & 12 & 10 & 0 & 2 & 4 & 14 & 9 & 13 \\
12 & 14 & 9 & 0 & 15 & 13 & 4 & 2 & 6 & 7 & 10 & 8 & 11 & 3 & 1 & 5 \\
7 & 2 & 4 & 12 & 6 & 3 & 15 & 1 & 10 & 14 & 5 & 9 & 0 & 8 & 13 & 11
\end{array}\right)$$

$$L_6 = \left(\begin{array}{cccccccccccccccc}
3 & 14 & 8 & 12 & 4 & 15 & 7 & 11 & 6 & 10 & 0 & 5 & 1 & 2 & 13 & 9 \\
1 & 3 & 5 & 0 & 10 & 4 & 9 & 7 & 11 & 2 & 14 & 12 & 13 & 6 & 8 & 15 \\
2 & 11 & 6 & 9 & 12 & 5 & 8 & 14 & 10 & 3 & 1 & 13 & 15 & 7 & 0 & 4 \\
4 & 13 & 10 & 11 & 9 & 14 & 3 & 15 & 1 & 7 & 2 & 6 & 8 & 0 & 12 & 5 \\
11 & 0 & 15 & 10 & 7 & 6 & 14 & 4 & 13 & 1 & 12 & 8 & 5 & 9 & 2 & 3 \\
8 & 15 & 12 & 6 & 0 & 2 & 4 & 13 & 5 & 9 & 3 & 7 & 10 & 1 & 14 & 11 \\
13 & 7 & 0 & 2 & 3 & 10 & 1 & 9 & 14 & 8 & 5 & 11 & 12 & 4 & 15 & 6 \\
10 & 1 & 14 & 4 & 5 & 12 & 11 & 2 & 9 & 15 & 6 & 0 & 3 & 8 & 7 & 13 \\
14 & 6 & 3 & 15 & 13 & 8 & 12 & 5 & 7 & 0 & 11 & 1 & 4 & 10 & 9 & 2 \\
\hline
7 & 9 & 11 & 3 & 1 & 13 & 2 & 6 & 15 & 4 & 8 & 14 & 0 & 12 & 5 & 10 \\
12 & 10 & 7 & 5 & 2 & 3 & 13 & 8 & 0 & 11 & 9 & 4 & 14 & 15 & 6 & 1 \\
9 & 5 & 13 & 8 & 11 & 7 & 6 & 0 & 4 & 12 & 15 & 10 & 2 & 3 & 1 & 14 \\
0 & 4 & 2 & 14 & 15 & 1 & 5 & 12 & 8 & 6 & 10 & 3 & 9 & 13 & 11 & 7 \\
5 & 8 & 4 & 1 & 6 & 9 & 0 & 10 & 2 & 13 & 7 & 15 & 11 & 14 & 3 & 12 \\
6 & 12 & 9 & 13 & 14 & 0 & 15 & 1 & 3 & 5 & 4 & 2 & 7 & 11 & 10 & 8 \\
15 & 2 & 1 & 7 & 8 & 11 & 10 & 3 & 12 & 14 & 13 & 9 & 6 & 5 & 4 & 0
\end{array}\right)$$

Analogously, we have chosen two Latin squares L_3 and L_4 of order 12×12 for Edon–$\mathcal{R}(384)$ and two Latin squares L_5 and L_6 of order 16×16 for Edon–$\mathcal{R}(512)$. The non-balanced symmetric block designs corresponding to $L_{3,1}$ and $L_{4,1}$ are with parameters $(v, k, \lambda) = (12, 7, \lambda)$ where $\lambda \in \{2, 3, 4, 5\}$, and those

n	MSVS 2005 Pro	Intel C++ 9.1
256	17.56	16.18
384	28.64	24.37
512	37.91	32.18

TABLE 4. Speed (cycles/byte) of the Reference C code for Edon–$\mathcal{R}(n)$ on x86 platforms in 32–bit mode obtained from Microsoft Visual Studio 2005 Pro and Intel C++ 9.1 for Windows.

corresponding to $L_{3,2}$ and $L_{4,2}$ are with parameters $(v, k, \lambda) = (12, 5, \lambda)$ where $\lambda \in \{0, 1, 2, 3\}$. The non-balanced symmetric block designs corresponding to $L_{5,1}$ and $L_{6,1}$ are with parameters $(v, k, \lambda) = (16, 9, \lambda)$ where $\lambda \in \{3, 4, 5, 6, 7\}$, and those corresponding to $L_{5,2}$ and $L_{6,2}$ are with parameters $(v, k, \lambda) = (16, 7, \lambda)$ where $\lambda \in \{1, 2, 3, 4, 5\}$.

5. Implementation characteristics of Edon–$\mathcal{R}(256, 384, 512)$

We have initial implementation of all three functions Edon–$\mathcal{R}(256, 384, 512)$ in C. We have run tests compiling both on Microsoft Visual Studio 2005 Pro and Intel C++ 9.1 for Windows. The code was tested only for x86 processors in 32–bit mode. Intel compiler was producing 8.5% – 17.8% faster code. However, in both cases we did not use 64 or 128 bit SSE and SSE2 registers as well as their SIMD capabilities. The initial processing speeds (in cycles/byte) are given in Table 2.

We project that significant improvements (at least twofold increasing) in the speed can be achieved by using SIMD instructions and capabilities of modern CPUs.

On the other hand, measuring the performances of Edon–$\mathcal{R}(256, 384, 512)$ on 8–bit platforms still has to be done, but we expect that the speeds will be relatively fast due to the fact that we are using only basic 32–bit operations such as addition modulo 2^{32}, eXlusive OR and rotations.

By careful analysis of the order of operations performed in Edon–$\mathcal{R}(256, 384, 512)$ one can notice that there are two types of parallelism of operations:

1. Operations inside the permutations π_2 and π_3 can be executed in parallel.
2. Pipelining of quasigroup operations: after the first quasigroup operation in the firs row, two quasigroup operations can be performed in parallel (one on the first row and one on the second row), and then similarly three quasigroup operations (in all three rows) can be performed in parallel.

This property can lead to hardware implementation of Edon–$\mathcal{R}(256, 384, 512)$ that can achieve even higher speeds.

6. Security analysis of the algorithm

The design of Edon–$\mathcal{R}(n)$ is “a wide-piped iterated compression function”. Although it is similar to the classical Merkle-Damgård iterated design [8], [9], [22], in the light of latest attacks with multi-collisions, it is also essentially different from it. We have chosen in the design of Edon–$\mathcal{R}$ to incorporate the suggestions of Lucks [18] and Coron et al. [7]. Namely, by setting the size of the internal memory of the iterated compression function to be twice as large as the output length, the weaknesses against generic attacks of Joux [13], and Kelsy and Schneier [14] are eliminated.

Doubling of the internal memory in our design is done by the fact that in every iterative step of its compression function, the strings of length $3n$ bits are mapped to strings of length $3n$ bits and then only the least significant $2n$ bits are kept for the next iterative step.

6.1 Natural resistance of Edon–$\mathcal{R}(n)$ against generic length extension attacks

Generic length extension attacks on iterated hash function based upon Merkle-Damgård iterating principles [9], [22] works as follows:

Let $M = M_1||M_2||\dots||M_N$ be a message consisting of exactly N blocks that will be iteratively digested by some compression function $C(A, B)$ according to the Merkle-Damgård iterating principles, and where A and B are messages (input parameters for the compression function) that have same length as the final message digest. Let P_M be the padding block of M obtained according to the Merkle-Damgård strengthening. Then, the digest H of the message M, is computed as

$$H(M) = C(\dots C(C(IV, M_1), M_2)\dots, P_M),$$

where IV is the initial fixed value for the hash function.

Now suppose that the attacker does not know the message M but knows (or can easily guess) the length of the message M. So the attacker actually knows the padding block P_M. Now, the attacker can construct a new message $M' = P_M||M'_1$ such that he knows the hash digest of the message $M||M'$. Namely,

$$H(M||M') = C(C(H(M), M'_1), P_{M'}),$$

where $P_{M'}$ is the padding (Merkle-Damgård strengthening) of the message $M||M'$.

Edon–$\mathcal{R}(n)$ has natural resistance against this generic attack due to the fact that it is iterated with the chaining variables that has length that is two times wider that the final digest value (see also the work of Lucks [18]).

6.2 Testing avalanche properties of Edon–$\mathcal{R}(n)$

First we show the avalanche propagation of the initial one bit differences of the compression function of Edon–$\mathcal{R}(n)$ during their evolution in all 9 quasigroup operations $*_n$, $(n = 256, 384, 512)$.

We have used two experimental settings:

1. Examining the propagation of the initial 1–bit difference in a message consisting of all zeroes.

2. Examining the propagation of the initial 1–bit difference in a randomly generated messages of n–bits.

The results for $n = 256$ are shown in Table 5. Notice that the level of Hamming distance equal to $\frac{1}{2}n = 128$, which would be the expected outcome in theoretical models of ideal random functions, is achieved after applying quasigroup operations that lie on the down-right half of the tables (in bold).

Min=15 Avr=15 Max=15	Min=86 Avr=108.44 Max=133	Min=107 **Avr=127.43** Max=153
Min=80 Avr=110.84 Max=142	Min=103 **Avr=128.17** Max=160	Min=100 **Avr=127.43** Max=151
Min=103 **Avr=127.54** Max=148	Min=102 **Avr=127.25** Max=146	Min=105 **Avr=127.86** Max=148

a.

Min=15 Avr=26.59 Max=74	Min=76 Avr=113.68 Max=149	Min=102 **Avr=128.11** Max=154
Min=73 Avr=115.93 Max=155	Min=103 **Avr=128.09** Max=158	Min=95 **Avr=127.75** Max=155
Min=101 **Avr=128.07** Max=153	Min=100 **Avr=128.01** Max=154	Min=95 **Avr=127.67** Max=155

b.

TABLE 5. **a.** Avalanche propagation of the Hamming distance between two 256–bit words M_1 and M_2 that initially differs in one bit and where $M_1 = 0$ (minimum, average and maximum) **b.** Avalanche propagation of the Hamming distance between two 256–bit words M_1 and M_2 that initially differs in one bit (minimum, average and maximum)

Similar results are obtained for $n = 384$ and $n = 512$.

The results for $n = 384$ are shown in Table 6. Notice again that the level of Hamming distance equal to $\frac{1}{2}n = 192$, which would be the expected outcome in theoretical models of ideal random functions, is achieved after applying quasigroup operations that lie on the down-right half of the tables (in bold), but some close values are obtained also after the second quasigroup operation (in italic).

Min=23 Avr=30.33 Max=35	Min=162 Avr=*190.28* Max=255	Min=166 **Avr=190.89** Max=219
Min=162 Avr=*190.87* Max=218	Min=166 **Avr=192.17** Max=218	Min=160 **Avr=192.40** Max=222
Min=162 **Avr=191.40** Max=225	Min=168 **Avr=192.11** Max=223	Min=160 **Avr=192.15** Max=221

a.

Min=23 Avr=52.54 Max=103	Min=157 Avr=*191.69* Max=227	Min=163 **Avr=192.31** Max=222
Min=166 Avr=*192.17* Max=225	Min=164 **Avr=191.41** Max=222	Min=166 **Avr=191.88** Max=222
Min=166 **Avr=192.68** Max=217	Min=160 **Avr=191.90** Max=216	Min=167 **Avr=191.99** Max=218

b.

TABLE 6. **a.** Avalanche propagation of the Hamming distance between two 384–bit words M_1 and M_2 that initially differs in one bit and where $M_1 = 0$ (minimum, average and maximum) **b.** Avalanche propagation of the Hamming distance between two 384–bit words M_1 and M_2 that initially differs in one bit (minimum, average and maximum)

The results for $n = 512$ are shown in Table 7. Also, the level of Hamming distance equal to $\frac{1}{2}n = 256$, which would be the expected outcome in theoretical models of ideal random functions, is achieved after applying quasigroup operations that lie on the down-right half of the tables (in bold), but some close values are obtained also after the second quasigroup operation (in italic).

One possible explanation about the reasons why Edon–$\mathcal{R}(384)$ and Edon–$\mathcal{R}(512)$ come slightly faster to the level of ideal random function than Edon–$\mathcal{R}(256)$ may lie in the fact that permutations π_2 and π_3 for $n = 384, 512$ are defined by bigger Latin squares of order 12×12 and 16×16 (see the Section 4). Thus they are more complex then corresponding permutations π_2 and π_3 for $n = 256$.

Min=27 *Avr=39.50* Max=51	Min=199 *Avr=252.46* Max=289	Min=222 **Avr=256.031** Max=296
Min=220 *Avr=254.93* Max=293	Min=222 **Avr=255.25** Max=283	Min=227 **Avr=257.01** Max=288
Min=224 **Avr=256.36** Max=287	Min=222 **Avr=255.54** Max=290	Min=227 **Avr=255.89** Max=295

a.

Min=27 *Avr=73.00* Max=142	Min=209 *Avr=254.54* Max=288	Min=222 **Avr=255.34** Max=288
Min=214 *Avr=255.49* Max=287	Min=226 **Avr=255.85** Max=290	Min=226 **Avr=256.50** Max=287
Min=217 **Avr=255.35** Max=286	Min=225 **Avr=256.38** Max=288	Min=221 **Avr=256.402** Max=297

b.

Table 7. **a.** Avalanche propagation of the Hamming distance between two 512–bit words M_1 and M_2 that initially differs in one bit and where $M_1 = 0$ (minimum, average and maximum) **b.** Avalanche propagation of the Hamming distance between two 512–bit words M_1 and M_2 that initially differs in one bit (minimum, average and maximum)

6.3 Description of all possible collision paths in the compression function $\mathcal{R}_1$ and infeasibility of finding local collisions

The design of the compression function $\mathcal{R}_1$ in Edon–$\mathcal{R}(n)$ is clearly different from the design of compression functions of known hash functions that are designed from scratch. While other compression functions have 64, 80 or even more iterating steps, $\mathcal{R}_1$ has 9 steps. So far, all successful attacks against the MDx and SHA families of hash functions exploited local collisions in the processing of the data block. Local collisions are collisions that can be found within few steps of the compression function.

$*_n$	$B_1 = \{b_1\}$	$B_2 = \{b_1, b_2\}$		
$A_1 = \{a_1\}$	$C_1=\{c_1\}$ where $a_1 *_n b_1 = c_1$	$C_2=\{c_1,c_2\}$ where $a_1 *_n b_1 = c_1$ and $a_1 *_n b_2 = c_2$		
$A_2 = \{a_1, a_2\}$	$C_2=\{c_1,c_2\}$ where $a_1 *_n b_1 = c_1$ and $a_2 *_n b_1 = c_2$	$C_2=\{c_1,c_2\}$ where $a_1 *_n b_1 = c_1$ and $a_2 *_n b_2 = c_2$	or	$C_1=\{c_1\}$ where $a_1 *_n b_1 = c_1$ and $a_2 *_n b_2 = c_1$

Table 8. Definition of quasigroup operation between one or two-element sets

The small number of steps in the compression function $\mathcal{R}_1$ as well as the algebraic properties of quasigroup operations allow us to describe all possible collision paths within the compression function which, we emphasize again, is a unique property among all known hash functions that are designed from scratch.

In order to track the collision paths for the compression function $\mathcal{R}_1$ we introduce a definition for quasigroup operation between sets of cardinality one and two.

Definition 5. Let $A_1 = \{a_1\}$, $A_2 = \{a_1, a_2\}$, $B_1 = \{b_1\}$, $B_2 = \{b_1, b_2\}$, $C_1 = \{c_1\}$, $C_2 = \{c_1, c_2\}$ be sets of cardinality one or two, where a_i, b_i and $c_i \in Q_n$ ($n = 256, 384, 512$). The operation of quasigroup multiplication $*_n$ between these sets is defined by Table 8.

Following directly by the properties of unique solutions of equations of type (1) it is easy to prove the following two propositions:

Proposition 1. *If* $b_1 \neq b_2$ *then* $\{a_1\} *_n \{b_1, b_2\} = \{c_1, c_2\}$ *such that* $c_1 \neq c_2$. □

Proposition 2. *If* $a_1 \neq a_2$ *then* $\{a_1, a_2\} *_n \{b_1\} = \{c_1, c_2\}$ *such that* $c_1 \neq c_2$. □

However if both $a_1 \neq a_2$ and $b_1 \neq b_2$ then $\{a_1, a_2\} *_n \{b_1, b_2\}$ can be either $\{c_1, c_2\}$ or $\{c_1\}$ and that is formulated in the following proposition:

Proposition 3. *If* $a_1 \neq a_2$ *and* $b_1 \neq b_2$ *then* $\{a_1, a_2\} *_n \{b_1, b_2\}$ *can be either* $\{c_1, c_2\}$ *(where* $c_1 \neq c_2$*) or* $\{c_1\}$. □

We formalize the notion of collisions for the compression function $\mathcal{R}_1$ by the following definition:

Definition 6. Let $(a_0, a_1, x_1), (a_0, a_1, x_2) \in Q_n \times Q_n \times Q_n$ where a_0 and a_1 are initial constants defined in Subsection 3.2. If $\mathcal{R}_1(a_0, a_1, x_1) = (c_0, c_1, y)$ and $\mathcal{R}_1(a_0, a_1, x_2) = (d_0, d_1, y)$ then we say that the pair $\{x_1, x_2\}$ is a collision for $\mathcal{R}_1$.

	$\{a_0\}$	$\{a_1\}$	$\{x_1, x_2\}$
$\{x_1, x_2\}$	$\{c_1, c_2\}$	$\{c_3, c_4\}$	$\{c_9, c_{10}\}$
$\{a_1\}$	$\{c_5, c_6\}$	$\{c_{11}, c_{12}\}$	$\{c_{13}, c_{14}\}$
$\{a_0\}$	$\{c_7, c_8\}$	$\{c_{15}, c_{16}\}$	$\{c_{17}\}$

a.

	$\{a_0\}$	$\{a_1\}$	$\{x_1, x_2\}$
$\{x_1, x_2\}$	$\{c_1, c_2\}$	$\{c_3, c_4\}$	$\{c_9, c_{10}\}$
$\{a_1\}$	$\{c_5, c_6\}$	$\{c_{11}, c_{12}\}$	$\{c_{13}\}$
$\{a_0\}$	$\{c_7, c_8\}$	$\{c_{14}\}$	$\{c_{15}\}$

b.

	$\{a_0\}$	$\{a_1\}$	$\{x_1, x_2\}$
$\{x_1, x_2\}$	$\{c_1, c_2\}$	$\{c_3, c_4\}$	$\{c_9, c_{10}\}$
$\{a_1\}$	$\{c_5, c_6\}$	$\{c_{11}\}$	$\{c_{12}, c_{13}\}$
$\{a_0\}$	$\{c_7, c_8\}$	$\{c_{14}, c_{15}\}$	$\{c_{16}\}$

c.

	$\{a_0\}$	$\{a_1\}$	$\{x_1, x_2\}$
$\{x_1, x_2\}$	$\{c_1, c_2\}$	$\{c_3, c_4\}$	$\{c_9\}$
$\{a_1\}$	$\{c_5, c_6\}$	$\{c_{10}, c_{11}\}$	$\{c_{12}, c_{13}\}$
$\{a_0\}$	$\{c_7, c_8\}$	$\{c_{14}, c_{15}\}$	$\{c_{16}\}$

d.

TABLE 9. Description of all possible differential paths in the compression function $\mathcal{R}_1$ that can give collisions.

Using the Definition 5 and Definition 6 we can trace all possible paths that can produce collisions in the compression function $\mathcal{R}_1$. That is formulated in the following theorem:

Theorem 2. *If $x_1 \neq x_2$ are two values in Q_n, then all possible differential paths starting with the set $\{x_1, x_2\}$ that can produce collisions in the compression function $\mathcal{R}_1$ are described in Table 9.* $\square$

$$\text{a. } \left\{ \begin{array}{rcl} c_{17} &=& c_{15} *_n c_{13} \\ c_{17} &=& c_{16} *_n c_{14} \\ c_{15} &=& c_7 *_n c_{11} \\ c_{13} &=& c_{11} *_n c_9 \\ c_{16} &=& c_8 *_n c_{12} \\ c_{14} &=& c_{12} *_n c_{10} \\ c_7 &=& a_0 *_n c_5 \\ c_{11} &=& c_5 *_n c_3 \\ c_9 &=& c_3 *_n x_1 \\ c_8 &=& a_0 *_n c_6 \\ c_{12} &=& c_6 *_n c_4 \\ c_{10} &=& c_4 *_n x_2 \\ c_5 &=& a_1 *_n c_1 \\ c_3 &=& c_1 *_n a_1 \\ c_6 &=& a_1 *_n c_2 \\ c_4 &=& c_2 *_n a_1 \\ c_1 &=& x_1 *_n a_0 \\ c_2 &=& x_2 *_n a_0 \end{array} \right. \qquad \text{b. } \left\{ \begin{array}{rcl} c_{15} &=& c_{14} *_n c_{13} \\ c_{14} &=& c_7 *_n c_{11} \\ c_{14} &=& c_8 *_n c_{12} \\ c_{13} &=& c_{11} *_n c_9 \\ c_{13} &=& c_{12} *_n c_{10} \\ c_7 &=& a_0 *_n c_5 \\ c_{11} &=& c_5 *_n c_3 \\ c_8 &=& a_0 *_n c_6 \\ c_{12} &=& c_6 *_n c_4 \\ c_9 &=& c_3 *_n x_1 \\ c_{10} &=& c_4 *_n x_2 \\ c_5 &=& a_1 *_n c_1 \\ c_3 &=& c_2 *_n a_1 \\ c_6 &=& a_1 *_n c_2 \\ c_4 &=& c_2 *_n a_1 \\ c_1 &=& x_1 *_n a_0 \\ c_2 &=& x_2 *_n a_0 \end{array} \right.$$

$$\text{c. } \left\{ \begin{array}{rcl} c_{16} &=& c_{14} *_n c_{12} \\ c_{16} &=& c_{15} *_n c_{13} \\ c_{14} &=& c_7 *_n c_{11} \\ c_{12} &=& c_{11} *_n c_9 \\ c_{15} &=& c_8 *_n c_{11} \\ c_{13} &=& c_{11} *_n c_{10} \\ c_7 &=& a_0 *_n c_5 \\ c_{11} &=& c_5 *_n c_3 \\ c_{11} &=& c_6 *_n c_4 \\ c_9 &=& c_3 *_n x_1 \\ c_8 &=& a_0 *_n c_6 \\ c_{10} &=& c_4 *_n x_2 \\ c_5 &=& a_1 *_n c_1 \\ c_3 &=& c_1 *_n a_1 \\ c_6 &=& a_1 *_n c_2 \\ c_4 &=& c_2 *_n a_1 \\ c_1 &=& x_1 *_n a_0 \\ c_2 &=& x_2 *_n a_0 \end{array} \right. \qquad \text{d. } \left\{ \begin{array}{rcl} c_{16} &=& c_{14} *_n c_{12} \\ c_{16} &=& c_{15} *_n c_{13} \\ c_{14} &=& c_7 *_n c_{10} \\ c_{12} &=& c_{10} *_n c_9 \\ c_{15} &=& c_8 *_n c_{11} \\ c_{13} &=& c_{11} *_n c_9 \\ c_7 &=& a_0 *_n c_5 \\ c_{10} &=& c_5 *_n c_3 \\ c_9 &=& c_3 *_n x_1 \\ c_9 &=& c_4 *_n x_2 \\ c_8 &=& a_0 *_n c_6 \\ c_{11} &=& c_6 *_n c_4 \\ c_5 &=& a_1 *_n c_1 \\ c_3 &=& c_1 *_n a_1 \\ c_4 &=& c_2 *_n a_1 \\ c_6 &=& a_1 *_n c_2 \\ c_1 &=& x_1 *_n a_0 \\ c_2 &=& x_2 *_n a_0 \end{array} \right.$$

TABLE 10. Concrete systems of quasigroup equations that can give collisions in the compression function $\mathcal{R}_1$

From Table 9 it is clear that for the collision in Table 9a., there are no local collisions. For the other three cases there are local collisions $\{c_{13}\}$ and $\{c_{14}\}$ in Table 9b., $\{c_{11}\}$ in Table 9c. and $\{c_9\}$ in Table 9d. In Table 10 we give four systems of quasigroup equations that are following directly from collision paths described in Table 9. From the complexity of the given quasigroup equations we can say that in this moment we see that it is infeasible even to find local collisions. As a support for that claim we can point out that the position of all local collisions lie in the areas that are reaching the level of randomness that is characteristic for a random Boolean functions (see bolded parts in Table 5, 6 and 7 and a position of local collisions in Table 9b., 9c. and 9d.).

6.4 Fixed points for the compression function $\mathcal{R}_1$

From the definition of the permutations π_1, π_2 and π_3 over Q_{256}, Q_{384} and Q_{512} it is clear that 0 is the fixed point of the compression function $\mathcal{R}_1$, i.e. $\mathcal{R}_1(0) = 0$ where $0 \in Q_{256}$ or $0 \in Q_{384}$ or $0 \in Q_{512}$.

We had (and still have) a dilemma: should we put some constants in π_2 and π_3 that will have an effect that $\mathcal{R}_1(0) \neq 0$.

At this moment we do not see any argument how the fact that $\mathcal{R}_1(0) = 0$ jeopardize the security of the whole hash function Edon–$\mathcal{R}(n)$, i.e., how can it be used to find collisions, preimages and second preimages.

Of course there is always concern that the property of the compression function $\mathcal{R}_1(0) = 0$ is not a "typical" random behavior, and hash functions are often used as random functions. A counter argument for this can be that there is clear distinction between the whole hash function (in this case Edon–$\mathcal{R}(n)$ which seems to act as a random function) and its compression function.

6.5 Getting all the additions to behave as XORs

Having a compression function $\mathcal{R}_1$ defined only by additions modulo 2^{32}, XORs and left rotations, it is a natural idea to try to find values for which additions in $\mathcal{R}_1$ behave as XORs [26].

In such a case, one would have a completely linear system in the ring $(\mathbb{Z}_2^n, +, \times)$ for which collisions, preimages and second preimages can easily be found. However, getting all the additions to behave as XORs is a challenge.

Here we can point out several significant works that are related with analysis of differential probabilities of operations that combine additions modulo 2^n, XORs and left rotations. In 1993, Berson made a differential cryptanalysis of addition modulo 2^{32} and applied it on MD5 [3], in 2001 Lipmaa and Moriai, have constructed efficient algorithms for computing differential properties of addition modulo 2^n [16], and Lipmaa, Wallén and Dumas in 2004 have constructed linear-time algorithm for computing the additive differential probability of exclusive-or [17].

All these works are determining the additive differential probability of exclusive-or:

$$\Pr[((x + \alpha) \oplus (y + \beta)) - (x \oplus y) = \gamma]$$

and exclusive-or differential probability of addition:

$$\Pr[((x \oplus \alpha) + (y \oplus \beta)) \oplus (x + y) = \gamma]$$

where probability is computed for all pairs $(x, y) \in \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$ and for any predetermined triplet $(\alpha, \beta, \gamma) \in \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^n}$.

In the case of Edon–$\mathcal{R}(n)$, instead of simple combination of two 32-bit variables once by additions modulo 2^n then by xoring, we have a linear transformation of 8, 12 or 16 32-bit variables described by transformations defined in Definition 3. Additionally, having in mind that $\mathcal{R}_1 : \{0,1\}^{3n} \to \{0,1\}^{3n}$, in this moment we do not see how these results will help in finding concrete values of arguments for the function $\mathcal{R}_1$ for which additions behave as XORs.

6.6 Infeasibility of going backward and infeasibility of finding free start collisions

According to the conjectured one-wayness of the function $\mathcal{R}_1$, iterating backward Edon–$\mathcal{R}(n)$ is infeasible. The conjecture is again based on the infeasibility of solving nonlinear quasigroup equations in non-commutative and non-associative quasigroups. From this it follows that the workload for finding preimages and second-preimages for any hash function of the family Edon–$\mathcal{R}(n)$ is 2^n hash computations.

Moreover, inverting the one-way function $\mathcal{R}_1$ would imply that finding free start collisions is feasible for the whole function Edon–$\mathcal{R}(n)$. Consequently, we base our conjecture that it is infeasible to find free start collisions for Edon–$\mathcal{R}(n)$ on the infeasibility of inverting the one-way function $\mathcal{R}_1$.

We elaborate our claims by the following discussion.

Definition 7. Let $(a_0, a_1, x_1), (b_0, b_1, x_2) \in Q_n \times Q_n \times Q_n$. If $\mathcal{R}_1(a_0, a_1, x_1) = (c_0, c_1, y)$ and $\mathcal{R}_1(b_0, b_1, x_2) = (d_0, d_1, y)$ then we say that the pair $((a_0, a_1, x_1), (b_0, b_1, x_2))$ is a free start collision for Edon–$\mathcal{R}(n)$.

The free start collision situation is described in the Table 11.

	a_0	a_1	x_1
x_1	$x_0^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$
a_1	$x_0^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$
a_0	c_0	c_1	y

a.

	b_0	b_1	x_2
x_2	$y_0^{(1)}$	$y_1^{(1)}$	$y_2^{(1)}$
b_1	$y_0^{(2)}$	$y_1^{(2)}$	$y_2^{(2)}$
b_0	d_0	d_1	y

b.

TABLE 11. **a.** Schematic presentation of the function $\mathcal{R}_1(a_0, a_1, x_1) = (c_0, c_1, y)$, **b.** Schematic presentation of the function $\mathcal{R}_1(b_0, b_1, x_2) = (d_0, d_1, y)$

Here we see two ways how to find free start collisions for Edon–$\mathcal{R}(n)$:

1. Generate a random $y \in Q_n$. Construct vectors (c_0, c_1, y) and (d_0, d_1, y) where $c_0, c_1, d_0, d_1 \in Q_n$ are randomly chosen. Try to find $\mathcal{R}_1^{-1}(c_0, c_1, y)$ and $\mathcal{R}_1^{-1}(d_0, d_1, y)$.
2. Generate a random (a_0, a_1, x_1) and compute $\mathcal{R}_1(a_0, a_1, x_1) = (c_0, c_1, y)$. Construct vector (d_0, d_1, y) where $d_0, d_1 \in Q_n$ are randomly chosen. Try to find $\mathcal{R}_1^{-1}(d_0, d_1, y)$.

Both ways need inversion of $\mathcal{R}_1$ and as we already said we see that as an infeasible task.

7. Conclusions

We have designed a concrete realization of the family of hash functions Edon–$\mathcal{R}$ with message digests of 256, 384 and 512 bits by defining huge quasigroups of orders 2^{256}, 2^{384} and 2^{512} that are non-commutative, and that are not loops (and consequently they are non-associative). The definition of quasigroups involve 32–bit operations of addition modulo 2^{32}, bitwise XORing and left rotations. Those operations are very fast on most modern microprocessors but they can be also efficiently realized on low-end 8–bit and 16–bit processors. By our reference C code implementation on x86 platforms we have achieved processing speeds of 16.18 cycles/byte, 24.37 cycles/byte and 32.18 cycles/byte.

In the forthcoming period we will do additional security analysis and we will try to develop some optimized implementations for different platforms.

References

[1] Belousov V.D., *Osnovi teorii kvazigrup i lup*, Nauka, Moskva, 1967.

[2] Bernstein D., *Salsa20*, eSTREAM – ECRYPT Stream Cipher Project, Report 2005/025, `http://www.ecrypt.eu.org/stream`.

[3] Berson T.A., *Differential Cryptanalysis Mod* 2^{32} *with Applications to MD5*, in Advances in Cryptology - EUROCRYPT '92, Lecture Notes in Comput. Sci. 658, 1993, pp. 71–80.

[4] Carter G., Dawson E., Nielsen L., *A latin square version of DES*, in Proc. Workshop of Selected Areas in Cryptography, Ottawa, Canada, 1995.

[5] Cooper J., Donovan D., Seberry J., *Secret sharing schemes arising from Latin squares*, Bull. Inst. Combin. Appl. **12** (1994), 33–43.

[6] Dénes J., Keedwell A.D., *A new authentication scheme based on Latin squares*, Discrete Math. **106/107** (1992), 157–161.

[7] Coron J.-S., Dodis Y., Malinaud C., Puniya P., *Merkle-Damgård revisited: How to construct a hash function*, in Advances in Cryptology – CRYPTO 2005, Lecture Notes in Comput. Sci. 3621, Springer, Berlin, 2005.

[8] Damgård I.B., *Collision free hash functions and public key signature schemes*, in Advances in Cryptology – EUROCRYPT 87, Lecture Notes in Comput. Sci. 304, Springer, Berlin, 1988, pp. 203–216.

[9] Damgård I.B., *A design principle for hash functions*, in Advances in Cryptology – CRYPTO 89, Lecture Notes in Comput. Sci. 435, Springer, New York, 1990, pp. 416-427.

[10] DesignTheory.org, `http://designtheory.org/`.

[11] Gligoroski D., Markovski S., Kocarev L., *Edon–$\mathcal{R}$, an infinite family of cryptographic hash functions*, Second NIST Cryptographic Hash Workshop, University of California - Santa Barbara, August, 2006, `http://www.csrc.nist.gov/pki/HashWorkshop/2006/Papers/GLIGOROSKI_EdonR-ver06.pdf`.

[12] Gligoroski D., *On a family of minimal candidate one-way functions and one-way permutations*, in print, International Journal of Network Security, ISSN 1816-3548, (see also ePrint archive `http://eprint.iacr.org/2005/352.pdf` for an early version of the paper).

[13] Joux A., *Multicollisions in iterated hash functions. Application to cascaded constructions*, in M. Franklin, ed., Advances in Cryptology – CRYPTO 2004, Lecture Notes in Comput. Sci. 3152, Springer, Berlin, 2004, pp. 306–316.

[14] Kelsey J., Schneier B., *Second preimages on n-bit hash functions for much less than 2^n work*, in R. Cramer, ed., Advances in Cryptology — EUROCRYPT 2005, Lecture Notes in Comput. Sci. 3494, Springer, Berlin, 2005, pp. 474-490.

[15] Lai X., Massey J.L., Murphy S., *Markov ciphers and differential cryptanalysis*, in Advances in Cryptology – EUROCRYPT '91, Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques (Brighton, April 1991), Lecture Notes in Comput. Sci. 547, Springer, Berlin, 1991, pp. 17–38.

[16] Lipmaa H., Moriai S., *Efficient algorithms for computing differential properties of addition*, in Fast Software Encryption 2001, Lecture Notes in Comput. Sci. 2355, Springer, Berlin, 2002, pp. 336–350.

[17] Lipmaa H., Wallén J., Dumas P., *On the additive differential probability of exclusive-or*, in Fast Software Encryption 2004, Lecture Notes in Comput. Sci. 3017, Springer, Berlin, 2004, pp. 317–331.

[18] Lucks S., *Design principles for iterated hash functions*, Cryptology ePrint Archive, report 2004/253.

[19] Markovski S., Gligoroski D., Bakeva V., *Quasigroup string processing, Part 1*, Makedon. Akad. Nauk Umet. Oddel. Mat.-Tehn. Nauk. Prilozi **20** (1999), no. 1–2, 13–28.

[20] Markovski S., Gligoroski D., Bakeva V., *Quasigroup and hash functions*, Discrete Mathematics and Applications, Sl. Shtrakov and K. Denecke, eds., Proceedings of the 6th ICDMA, Bansko, 2001, pp. 43–50.

[21] McKay B.D., Rogoyski E., *Latin squares of order* 10, Electron. J. Comb. **2** (1995), `http://ejc.math.gatech.edu:8080/Journal/journalhome.html`.

[22] Merkle R., *One way hash functions and DES*, Advances in Cryptology — Crypto'89, Lecture Notes in Comput. Sci. 435, Springer, New York, 1990, pp. 428–446.

[23] Rosen K.H., Michaels J.G., Gross J.L., Grossman J.W., Shier D.R., *Handbook of Discrete and Combinatorial Mathematics*, ISBN 0-8493-0149-1, CRC Press, Boca Raton, Florida, 2000.

[24] Schnorr C.P., Vaudenay S., *Black Box Cryptanalysis of hash networks based on multipermutations*, Advances in Cryptology — EUROCRYPT '94, Lecture Notes in Comput. Sci. 950, Springer, Berlin, 1995, pp. 47–57.

[25] Shannon C.E., *Communication theory of secrecy systems*, Bell System Tech. J. **28** (1949), 656–715.

[26] Thomsen S.S., *Personal communication*, May 2007.

[27] Wheeler D.J., Needham R.M., *TEA, a tiny encryption algorithm*, Fast software encryption: second international workshop, Leuven, Belgium, December 1994, Proceedings, Lecture Notes in Comput. Sci. 1008, 1995, pp. 363–366.

Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and Technology, O.S. Bragstads plass 2E, N-7491 Trondheim, Norway

E-mail: Danilo.Gligoroski@q2s.ntnu.no
Svein.J.Knapskog@q2s.ntnu.no

(*Received* October 15, 2007, *revised* January 28, 2008)