Cryptographic hash function EDON-R′

Conference Paper · June 2009

Source: IEEE Xplore

CITATIONS
28

READS
337

8 authors, including:

Danilo Gligoroski
Norwegian University of Science and Technology
174 PUBLICATIONS **1,728** CITATIONS
SEE PROFILE

Marija Mihova
Ss. Cyril and Methodius University in Skopje
66 PUBLICATIONS **164** CITATIONS
SEE PROFILE

Ales Drapal
Charles University in Prague
123 PUBLICATIONS **704** CITATIONS
SEE PROFILE

Vlastimil Klíma
Charles University in Prague, Prague, Czech Republic
41 PUBLICATIONS **520** CITATIONS
SEE PROFILE

Cryptographic Hash Function EDON-$\mathcal{R}'$

Danilo Gligoroski, Rune Steinsmo Ødegård, Marija Mihova, Svein Johan Knapskog,
Aleš Drápal, Vlastimil Klima, Jørn Amundsen, Mohamed El-Hadedy
Department of Telematics, Norwegian University of Sceince and Technology, Trondheim, Norway
danilog@item.ntnu.no

Abstract

In this paper we describe in details the tweaked cryptographic hash function EDON-$\mathcal{R}$ *that we denote as* EDON-$\mathcal{R}'$. EDON-$\mathcal{R}$ *was submitted as a candidate for SHA-3 hash competition organized by National Institute of Standards and Technology (NIST). The difference between originally submitted version of* EDON-$\mathcal{R}$ *and version* EDON-$\mathcal{R}'$ *is in the added feedback to the original compression function* $\mathcal{R}$. *The feedback consist of xoring the output of the function* $\mathcal{R}$ *with the previous double pipe value and the value of the current message block. Now,* EDON-$\mathcal{R}'$ *can be seen as a double-pipe PGV7 hash scheme.*

The introduced tweak does not invalidates the cryptanalytic efforts to analyze the quasigroup operations used in EDON-$\mathcal{R}'$, *as well as its function* $\mathcal{R}$. *It also does not affect much the speed of the function. However, this tweak prevents finding free-start collisions and prevents all attacks based on free-start collisions.*

EDON-$\mathcal{R}'$ *is a cryptographic hash function with output size of* n *bits where* $n = 224, 256, 384$ *or* 512. *Its conjectured cryptographic security is:* $O(2^{\frac{n}{2}})$ *hash computations for finding collisions,* $O(2^n)$ *hash computations for finding preimages,* $O(2^{n-k})$ *hash computations for finding second preimages for messages shorter than* 2^k *bits. Additionally, it is resistant against length-extension attacks, resistant against multicollision attacks and it is provably resistant against differential cryptanalysis.*

EDON-$\mathcal{R}'$ *performance has been measured with Microsoft Visual Studio 2005, and with Intel C++ v 11.0.072.* EDON-$\mathcal{R}'$ *has been designed to be much more efficient than SHA-2 cryptographic hash functions, while in the same time offering same or better security. The speed of the optimized 32-bit version on defined reference platform with Intel C++ v 11.0.072 is 6.71 cycles/byte for* $n = 224, 256$ *and 10.74 cycles/byte for* $n = 384, 512$. *The speed of the optimized 64-bit version on defined reference platform with Intel C++ v 11.0.072 is 4.90 cycles/byte for* $n = 224, 256$ *and 2.74 cycles/byte for* $n = 384, 512$.

1. Introduction

In cryptography and information security, hash functions are considered as the "Swiss army knife". They are used in countless protocols and algorithms. In 2005 we have witnessed significant theoretical breakthrough in breaking the current cryptographic standard SHA-1 [16]. Although there is another family of standardized hash functions called SHA-2, ready to replace SHA-1 hash function, at the end of 2007, the National Institute of Standards and Technology (NIST) decided to start a 4 year world-wide competition process for choosing the next cryptographic hash standard SHA-3 [1]. Currently, the new hash standard SHA-3, is under development - the function will be selected via an open competition running between 2008 and 2012.

The EDON-$\mathcal{R}'$ hash function is the fastest proposed new designs in the SHA-3 competition [4]. In this paper we describe its construction in details.

2. General design properties of EDON-$\mathcal{R}'$

EDON-$\mathcal{R}'$ follows the general design pattern that is common for most of the known hash functions. It means that it has two stages (and several sub-stages within every stage):

1. Preprocessing
   (a) padding a message,
   (b) parsing the padded message into m–bit blocks, and
   (c) setting initialization values to be used in the hash computation.
2. Hash computation
   (a) using a conjectured one-way operation with huge quasigroups iteratively generates series of double pipe values,
   (b) The n Least Significant Bits (LSB) of the final double pipe value generated by the hash computation are used to determine the message digest.

Depending on the context we will sometimes refer to the hash function as EDON-$\mathcal{R}'$ and sometimes as EDON-$\mathcal{R}'$224, EDON-$\mathcal{R}'$256, EDON-$\mathcal{R}'$384 or EDON-$\mathcal{R}'$512.

In Table 1, we give the basic properties of all four variants of the EDON-$\mathcal{R}'$ hash algorithms.

2.1 Preprocessing

Preprocessing consists of three steps:

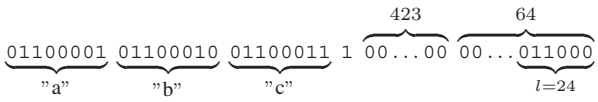
1. padding the message M,
2. parsing the padded message into message blocks, and
3. setting the initial double pipe value, $P^{(0)}$.

2.1.1 Padding the message

The message M, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512 or 1024 bits, depending on the size of the message digest n.

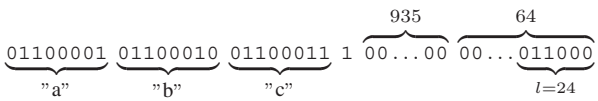
2.1.2 EDON-$\mathcal{R}'$224 and EDON-$\mathcal{R}'$256

Suppose that the length of the message M is l bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $l + 1 + k \equiv 448 \mod 512$. Then append the 64–bit block that is equal to the number l expressed using a binary representation. For example, the (8–bit ASCII) message "abc" has length $8 \times 3 = 24$, so the message is padded with the bit "1", then $448 - (24 + 1) = 423$ zero bits, and then the 64–bit binary representation of the number 24, to become the 512–bit padded message.



2.1.3 EDON-$\mathcal{R}'$384 and EDON-$\mathcal{R}'$512

Suppose that the length of the message M is l bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $l + 1 + k \equiv 960 \mod 1024$. Then append the 64–bit block that is equal to the number l expressed using a binary representation. For example, the (8–bit ASCII) message "abc" has length $8 \times 3 = 24$, so the message is padded with the bit "1", then $960 - (24 + 1) = 935$ zero bits, and then the 64–bit binary representation of the number 24, to become the 1024–bit padded message.



2.2 Parsing the message

After a message has been padded, it must be parsed into N m–bit blocks before the hash computation can begin.

2.2.1 EDON-$\mathcal{R}'$224 and EDON-$\mathcal{R}'$256

For EDON-$\mathcal{R}'$224 and EDON-$\mathcal{R}'$256, the padded message is parsed into N 512–bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32–bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

2.2.2 EDON-$\mathcal{R}'$384 and EDON-$\mathcal{R}'$512

For EDON-$\mathcal{R}'$384 and EDON-$\mathcal{R}'$512, the padded message is parsed into N 1024–bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 1024 bits of the input block may be expressed as sixteen 64–bit words, the first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

2.3 Initial double pipe value $P^{(0)}$

Before the hash computation begins for each of the hash algorithms, the initial double pipe value $P^{(0)}$ must be set. The size and the value of words in $P^{(0)}$ depend on the message digest size n. As it is shown in the following subsections the constants consist of a concatenation of the consecutive 8-bit natural numbers. Since EdonR224 is the same as EdonR256 except for the final truncation, they have to have different initial values. Thus, the initial double pipe of EdonR224 starts from the byte value `0x00` and takes all 64 successive byte values up to the value `0x3F`. Then, the initial double pipe of EdonR256 starts from the byte value `0x40` and takes all 64 successive byte values up to the value `0x7F`. The situation is the same with EdonR384 and EdonR512, but since now the variables are 64-bit long, the initial double pipe of EdonR384 starts from the byte value `0x00` and takes all 128 successive byte values up to the value `0x7F` and the initial double pipe of EdonR512 starts from the byte value `0x80` and takes all 128 successive byte values up to the value `0xFF`.

3 Mathematical preliminaries and notation

In this section we need to repeat some parts of the definition of the class of one-way candidate functions recently defined in [11, 8]. For that purpose we need also several brief definitions for quasigroups and quasigroup string transformations.

Table 1. Basic properties of all four variants of the EDON-$\mathcal{R}'$

Algorithm abbreviation	Message size l (in bits)	Block size m (in bits)	Word size w (in bits)	Endianess	Digest size n (in bits)	Support of "one-pass" streaming mode
EDON-$\mathcal{R}'$224	$< 2^{64}$	512	32	Little-endian	224	Yes
EDON-$\mathcal{R}'$256	$< 2^{64}$	512	32	Little-endian	256	Yes
EDON-$\mathcal{R}'$384	$< 2^{64}$	1024	64	Little-endian	384	Yes
EDON-$\mathcal{R}'$512	$< 2^{64}$	1024	64	Little-endian	512	Yes

Definition 1. *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.

Definition 2. *A Latin square is an $n \times n$ table filled with n different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column.*

Definition 3. *A pair of Latin squares is said to be orthogonal if the n^2 pairs formed by juxtaposing the two squares are all distinct.*

More detailed information about theory of quasigroups, quasigroup string processing, Latin squares and hash functions can be found in [3, 5, 15, 14, 13].

3.1 Algorithmic definition of quasigroups of orders 2^{256} and 2^{512}

First we give an algorithmic description of an operation that takes two eight-component vectors $\mathbf{X} = (X_0, X_1, \ldots, X_7)$ and $\mathbf{Y} = (Y_0, Y_1, \ldots, Y_7)$ where X_i and Y_i are either 32–bit or 64–bit variables, and computes a new eight-component vector $\mathbf{Z} = (Z_0, Z_1, \ldots, Z_7)$. Operation "+" denotes addition modulo 2^{32} or modulo 2^{64}, the operation $\oplus$ is the logical operation of bitwise exclusive or and the operation $ROTL^r(X_i)$ is the operation of bit rotation of the 32–bit or 64–bit variable X_i, to the left for r positions. We give the description of the quasigroup operation of order 2^{256} in Table 2, while the quasigroup operation of order 2^{512} being similar and in order to save the space is omitted. The reader can find its detailed description in [4].

3.2 Algebraic definition of quasigroups of orders 2^{256} and 2^{512}

In this subsection we will present the same quasigroups that have been described in the previous subsection in an algebraic way.

For that purpose we use the following notation: we identify Q as a set of cardinality 2^q, $q = 256, 512$, and elements $x \in Q$ are represented in their bitwise form as q-bit words

$$\begin{aligned} \mathbf{X} &\equiv (\widetilde{x}_0, \widetilde{x}_1, \ldots, \widetilde{x}_{q-2}, \widetilde{x}_{q-1}) \equiv \\ &\equiv \widetilde{x}_0 \cdot 2^{q-1} + \widetilde{x}_1 \cdot 2^{q-2} + \ldots + \widetilde{x}_{q-2} \cdot 2 + \widetilde{x}_{q-1} \end{aligned}$$

where $\widetilde{x}_i \in \{0, 1\}$, but also as the set $\left(\mathbb{Z}_{2^w}\right)^8$, $w = 32, 64$.

Actually, we shall be constructing quasigroups $(Q, *)$ as isotopes of $\left(\left(\mathbb{Z}_{2^w}\right)^8, +_8\right)$, $w = 32, 64$ where $+_8$, is the operation of componentwise addition of two 8–dimensional vectors in $\left(\mathbb{Z}_{2^w}\right)^8$. We shall thus define three permutations $\pi_i : \mathbb{Z}_2^q \to \mathbb{Z}_2^q$, $1 \le i \le 3$ so that

$$\mathbf{X} * \mathbf{Y} \equiv \pi_1(\pi_2(\mathbf{X}) +_8 \pi_3(\mathbf{Y}))$$

for all $\mathbf{X}, \mathbf{Y} \in \left(\mathbb{Z}_{2^w}\right)^8$.

Let us denote by $Q_{256} = \{0, 1\}^{256}$ and $Q_{512} = \{0, 1\}^{512}$ the corresponding sets of 256–bit and 512–bit words. Our intention is to define EDON-$\mathcal{R}'$ by the following bitwise operations on w–bit values (where $w = 32$ or $w = 64$):

1. Addition between w–bit words modulo 2^w,
2. Rotation of w–bits to the left for r positions,
3. Bitwise XOR operations $\oplus$ on w–bit words.

Further, we introduce the following convention:

- Elements $\mathbf{X} \in Q_{256}$ are represented as $\mathbf{X} = (X_0, X_1, \ldots, X_7)$, where X_i are 32–bit words,
- Elements $\mathbf{X} \in Q_{512}$ are represented as $\mathbf{X} = (X_0, X_1, \ldots, X_7)$ where X_i are 64–bit words.

The left rotation of a w–bit word Y by r positions will be denoted by $ROTL^r(Y)$. Note that this operation can be expressed as a linear matrix–vector multiplication over the ring $(\mathbb{Z}_2, +, \times)$ i.e. $ROTL^r(Y) = \mathbf{E}^r \cdot Y$ where $\mathbf{E}^r \in \mathbb{Z}_2^w \times \mathbb{Z}_2^w$ 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 $X \in Q_q$ where $q = 256, 512$ represented as $\mathbf{X} = (X_0, X_1, \ldots, X_7)$ and

Table 2. An algorithmic description of a quasigroup of order 2^{256}.

Quasigroup operation of order 2^{256}

Input: $\mathbf{X} = (X_0, X_1, \ldots, X_7)$ and $\mathbf{Y} = (Y_0, Y_1, \ldots, Y_7)$
where X_i and Y_i are 32–bit variables.
Output: $\mathbf{Z} = (Z_0, Z_1, \ldots, Z_7)$ where Z_i are 32–bit variables.
Temporary 32–bit variables: $T_0, \ldots, T_{15}$.

1.
$$
\begin{array}{lcl}
T_0 & \leftarrow & ROTL^0(\texttt{0xAAAAAAAA} + X_0 + X_1 + X_2 + X_4 + X_7);\\
T_1 & \leftarrow & ROTL^4(X_0 + X_1 + X_3 + X_4 + X_7);\\
T_2 & \leftarrow & ROTL^8(X_0 + X_1 + X_4 + X_6 + X_7);\\
T_3 & \leftarrow & ROTL^{13}(X_2 + X_3 + X_5 + X_6 + X_7);\\
T_4 & \leftarrow & ROTL^{17}(X_1 + X_2 + X_3 + X_5 + X_6);\\
T_5 & \leftarrow & ROTL^{22}(X_0 + X_2 + X_3 + X_4 + X_5);\\
T_6 & \leftarrow & ROTL^{24}(X_0 + X_1 + X_5 + X_6 + X_7);\\
T_7 & \leftarrow & ROTL^{29}(X_2 + X_3 + X_4 + X_5 + X_6);
\end{array}
$$

2.
$$
\begin{array}{lcl}
T_8 & \leftarrow & T_3 \oplus T_5 \oplus T_6;\\
T_9 & \leftarrow & T_2 \oplus T_5 \oplus T_6;\\
T_{10} & \leftarrow & T_2 \oplus T_3 \oplus T_5;\\
T_{11} & \leftarrow & T_0 \oplus T_1 \oplus T_4;\\
T_{12} & \leftarrow & T_0 \oplus T_4 \oplus T_7;\\
T_{13} & \leftarrow & T_1 \oplus T_6 \oplus T_7;\\
T_{14} & \leftarrow & T_2 \oplus T_3 \oplus T_4;\\
T_{15} & \leftarrow & T_0 \oplus T_1 \oplus T_7;
\end{array}
$$

3.
$$
\begin{array}{lcl}
T_0 & \leftarrow & ROTL^0(\texttt{0x55555555} + Y_0 + Y_1 + Y_2 + Y_5 + Y_7);\\
T_1 & \leftarrow & ROTL^5(Y_0 + Y_1 + Y_3 + Y_4 + Y_6);\\
T_2 & \leftarrow & ROTL^9(Y_0 + Y_1 + Y_2 + Y_3 + Y_5);\\
T_3 & \leftarrow & ROTL^{11}(Y_2 + Y_3 + Y_4 + Y_6 + Y_7);\\
T_4 & \leftarrow & ROTL^{15}(Y_0 + Y_1 + Y_3 + Y_4 + Y_5);\\
T_5 & \leftarrow & ROTL^{20}(Y_2 + Y_4 + Y_5 + Y_6 + Y_7);\\
T_6 & \leftarrow & ROTL^{25}(Y_1 + Y_2 + Y_5 + Y_6 + Y_7);\\
T_7 & \leftarrow & ROTL^{27}(Y_0 + Y_3 + Y_4 + Y_6 + Y_7);
\end{array}
$$

4.
$$
\begin{array}{lcl}
Z_5 & \leftarrow & T_8 + (T_3 \oplus T_4 \oplus T_6);\\
Z_6 & \leftarrow & T_9 + (T_2 \oplus T_5 \oplus T_7);\\
Z_7 & \leftarrow & T_{10} + (T_4 \oplus T_6 \oplus T_7);\\
Z_0 & \leftarrow & T_{11} + (T_0 \oplus T_1 \oplus T_5);\\
Z_1 & \leftarrow & T_{12} + (T_2 \oplus T_6 \oplus T_7);\\
Z_2 & \leftarrow & T_{13} + (T_0 \oplus T_1 \oplus T_3);\\
Z_3 & \leftarrow & T_{14} + (T_0 \oplus T_3 \oplus T_4);\\
Z_4 & \leftarrow & T_{15} + (T_1 \oplus T_2 \oplus T_5);
\end{array}
$$

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 $ROTL^{\mathbf{r}}(\mathbf{X})$, where $\mathbf{r} = (r_0, \ldots, r_7) \in \{0, 1, \ldots, w-1\}^7$ is called the rotation vector. The operation $ROTL^{\mathbf{r}}(\mathbf{X})$ can also be represented as a linear matrix–vector multiplication over the ring $(\mathbb{Z}_2, +, \times)$ i.e. $ROTL^{\mathbf{r}}(\mathbf{X}) = \mathbf{D}^{\mathbf{r}} \cdot \mathbf{X}$ where $\mathbf{D}^{\mathbf{r}} \in \mathbb{Z}_2^q \times \mathbb{Z}_2^q$,

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

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

Further on, we use the following notation:

- $\widehat{\mathbb{A}}_1, \widehat{\mathbb{A}}_3 : \left(\mathbb{Z}_{2^w}\right)^8 \rightarrow \left(\mathbb{Z}_{2^w}\right)^8$ are two bijective transformations in $\left(\mathbb{Z}_{2^w}\right)^8$ over the ring $(\mathbb{Z}_{2^w}, +, \times)$ where $w = 32$ or $w = 64$. The mappings $\widehat{\mathbb{A}}_i, i = 1, 3$ can be described as:

$$\widehat{\mathbb{A}}_i(\mathbf{X}) = \mathbf{C}_i + \mathbb{A}_i \cdot \mathbf{X},$$

where $\mathbf{C}_i \in \left(\mathbb{Z}_{2^w}\right)^8, i = 1, 2$ are two constant vectors and $\mathbb{A}_1$ and $\mathbb{A}_3$ are two 8×8 invertible matrices over the ring $(\mathbb{Z}_{2^w}, +, \times)$. Since they look like affine transformations in vector fields, sometimes we will call these two transformations also "*affine bijective transformations*" although strictly speaking we are not working in any vector field. All elements in those two matrices are either 0 or 1, since we want to avoid the operations of multiplication (as more costly microprocessor operations) in the ring $(\mathbb{Z}_{2^w}, +, \times)$, and stay only with operations of addition.

- $\mathbb{A}_2, \mathbb{A}_4 : \left(\mathbb{Z}_{2^w}\right)^8 \rightarrow \left(\mathbb{Z}_{2^w}\right)^8$ are two linear bijective transformations of Q_q that are described by two invertible matrices (we use the same notation: $\mathbb{A}_2, \mathbb{A}_4$)

Table 3. Rotation vectors defined in π_2 and π_3.

q	$\mathbf{r}_{1,q}$	$\mathbf{r}_{2,q}$
256	(0, 4, 8, 13, 17, 22, 24, 29)	(0, 5, 9, 11, 15, 20, 25, 27)
512	(0, 5, 15, 22, 31, 40, 50, 59)	(0, 10, 19, 29, 36, 44, 48, 55)

of order $q \times q$ over the ring $(\mathbb{Z}_2, +, \times)$ ($q = 256$ or $q = 512$). Since we want to apply XOR operations on w–bit registers, the matrices $\mathbb{A}_2$ and $\mathbb{A}_4$ will be of the form

$$\begin{pmatrix} \mathbb{B}_{1,1} & \mathbb{B}_{1,2} & \dots & \mathbb{B}_{1,8} \\ \mathbb{B}_{2,1} & \mathbb{B}_{2,2} & \dots & \mathbb{B}_{2,8} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{B}_{8,1} & \mathbb{B}_{8,2} & \dots & \mathbb{B}_{8,8} \end{pmatrix},$$

where $\mathbb{B}_{i,j} \in \mathbb{Z}_2^w \times \mathbb{Z}_2^w$, $1 \leq i, j \leq 8$ are either the identity matrix or the zero matrix i.e. $\mathbb{B}_{i,j} \in \{\mathbf{0}, \mathbf{1}\}$.

Now we give the formal definitions for the permutations: π_1, π_2 and π_3.

Definition 4. *Transformations $\pi_1 : Q_q \to Q_q$ ($q = 256, 512$) are defined as:*

$$\pi_1(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7) = \\ = (X_5, X_6, X_7, X_0, X_1, X_2, X_3, X_4)$$

Lemma 1. *Transformations π_1 are permutations.* □

Definition 5. *Transformations $\pi_2 : Q_q \to Q_q$ and $\pi_3 : Q_q \to Q_q$ are defined as:*

$$\pi_2 \equiv \widehat{\mathbb{A}}_1 \circ ROTL^{\mathbf{r}_{1,q}} \circ \mathbb{A}_2$$
$$\pi_3 \equiv \widehat{\mathbb{A}}_3 \circ ROTL^{\mathbf{r}_{2,q}} \circ \mathbb{A}_4$$

where the rotation vectors $\mathbf{r}_{i,q}$, $i = 1, 2$, $q = 256, 512$ are given in Table 3, and the information about $\widehat{\mathbb{A}}_1$, $\mathbb{A}_2$, $\widehat{\mathbb{A}}_3$ and $\mathbb{A}_4$ is given in Table 4. There, the symbols $\mathbf{1}, \mathbf{0} \in \mathbb{Z}_2^w \times \mathbb{Z}_2^w$ are the identity matrix and the zero matrix, and the constants $const_{i,q}$, $i = 1, 2$, $q = 256, 512$ have the following values (given in hexadecimal notation): $const_{1,256} =$ `0xAAAAAAAA`, $const_{2,256} =$ `0x55555555`, $const_{1,512} =$ `0xAAAAAAAAAAAAAAAA` and $const_{2,512} =$ `0x5555555555555555`.

Lemma 2. *Transformations π_2 and π_3 are permutations on Q_q, $q = 256, 512$.*

Proof. The proof follows immediately from the fact that all transformations $\mathbb{A}_i$, $i = 1, 2, 3, 4$ and $ROTL^{\mathbf{r}_{i,q}}$, $i = 1, 2$, $q = 256, 512$ are expressed by invertible matrices over the rings $(\mathbb{Z}_{2^w}, +, \times)$, $w = 32, 64$ or over the ring $(\mathbb{Z}_2, +, \times)$. □

Theorem 1. *Operations $*_q : Q_q^2 \to Q_q$ defined as:*

$$\mathbf{X} *_q \mathbf{Y} = \pi_1(\pi_2(\mathbf{X}) \ +_8 \ \pi_3(\mathbf{Y}))$$

are non-commutative quasigroup operations that are not loops.

Proof. We give a proof for $q = 256$ and the other case for $q = 512$ is similar.

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

$$\pi_2(\mathbf{E}) -_8 \pi_3(\mathbf{E}) = \mathbf{Const}_E$$

where $\mathbf{Const}_E \in Q_{256}$ is a constant element and the operation $-_8$ is the componentwise subtraction modulo 2^{32}.

From the concrete definition of the quasigroup operation $*_{256}$ for the neutral element $\mathbf{E}$ we get:

$$\pi_1(\pi_2(\mathbf{E}) +_8 \pi_3(\mathbf{A})) = \pi_1(\pi_2(\mathbf{A}) +_8 \pi_3(\mathbf{E}))$$

Since π_1 is a permutation we can remove it from the last equation and we get:

$$\pi_2(\mathbf{E}) +_8 \pi_3(\mathbf{A}) = \pi_2(\mathbf{A}) +_8 \pi_3(\mathbf{E})$$

and if we rearrange the last equation we get:

$$\pi_2(\mathbf{A}) -_8 \pi_3(\mathbf{A}) = \pi_2(\mathbf{E}) -_8 \pi_3(\mathbf{E}) = \mathbf{Const}_E$$

The last equation states that for every $\mathbf{A} \in Q_{256}$ the expression $\pi_2(\mathbf{A}) -_8 \pi_3(\mathbf{A})$ is a constant. This is not true. For example $\pi_2(1) -_8 \pi_3(1) \neq \pi_2(2) -_8 \pi_3(2)$. Thus we conclude that $*_{256}$ is not a loop. □

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

Having defined two quasigroup operations $*_{256}$ and $*_{512}$ we define two functions $\mathcal{R}_{256}$ and $\mathcal{R}_{512}$ as follows:

Definition 6.

1. *$\mathcal{R}_{256} : Q_{256}^4 \to Q_{256}^2 \equiv \mathcal{R}$ where $\mathcal{R}$ is defined as in Definition 8 over Q_{256} with the quasigroup operation $*_{256}$.*

2. *$\mathcal{R}_{512} : Q_{512}^4 \to Q_{512}^2 \equiv \mathcal{R}$ where $\mathcal{R}$ is defined as in Definition 8 over Q_{512} with the quasigroup operation $*_{512}$.*

Table 4. Matrices $\widehat{\mathbb{A}}_1$, $\mathbb{A}_2$, $\widehat{\mathbb{A}}_3$ and $\mathbb{A}_4$.

$\widehat{\mathbb{A}}_1$

$$\begin{pmatrix} const_{1,q} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1&1&1&0&1&0&0&1 \\ 1&1&0&1&1&0&0&1 \\ 1&1&0&0&1&0&1&1 \\ 0&0&1&1&0&1&1&1 \\ 0&1&1&1&0&1&1&0 \\ 1&0&1&1&1&1&0&0 \\ 1&1&0&0&0&1&1&1 \\ 0&0&1&1&1&1&1&0 \end{pmatrix}$$

$\mathbb{A}_2$

$$\begin{pmatrix} \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0} \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0} \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0} \\ \mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0} \\ \mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1} \\ \mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1} \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0} \\ \mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1} \end{pmatrix}$$

$\widehat{\mathbb{A}}_3$

$$\begin{pmatrix} const_{2,q} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1&1&1&0&0&1&0&1 \\ 1&1&0&1&1&0&1&0 \\ 1&1&1&1&0&1&0&0 \\ 0&0&1&1&1&0&1&1 \\ 1&1&0&1&1&1&0&0 \\ 0&0&1&0&1&1&1&1 \\ 0&1&1&0&0&1&1&1 \\ 1&0&0&1&1&0&1&1 \end{pmatrix}$$

$\mathbb{A}_4$

$$\begin{pmatrix} \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0} \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1} \\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1} \\ \mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0} \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1} \\ \mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0} \\ \mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0} \\ \mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{0} \end{pmatrix}$$

3.3 The function $\mathcal{R}$: A reverse quasigroup string transformation

The reverse quasigroup string transformation as a candidate one-way function has been introduced in [7], and a generic hash function with reverse quasigroup string transformation has been described in [11, 8]. A concrete hash function with similar name: Edon-R(n) for $n = 256, 384, 512$ has been described in [9]. Many properties from that function are present in the design of EDON-$\mathcal{R}'$, but we can say that without loosing security properties of the hash function, the design of EDON-$\mathcal{R}'$ is now simplified and performance is much better compared to the older Edon-R(n). Additionally, we can say that the concept of reverse quasigroup string transformation is present also in another cryptographic primitive - the stream cipher Edon80 [10]. Edon80 IV Setup procedure is a conjectured one-way function and so far no cryptographic weaknesses have been found for the Edon80 IV Setup, although Edon80 have been under the public scrutiny from the cryptographic community for more than 3 years.

Definition 7. *For a given $\mathbf{X} \in Q_q, q = 256, 512$, which can be represented as an eight component vector $\mathbf{X} = (X_0, X_1, \ldots, X_7) \in \left(\mathbb{Z}_{2^w}\right)^8, w = 32, 54$, the reversed vector $\overline{\mathbf{X}}$ is defined as:*

$$\overline{\mathbf{X}} = (X_7, X_6, \ldots, X_0)$$

Definition 8. *For a given quasigroup $*_q, q = 256, 512$, (shortly denoted as $*$ i.e. without the index q) the function $\mathcal{R} : Q_q^4 \to Q_q^2$ used in* EDON-$\mathcal{R}'$ *hash function is defined as:*

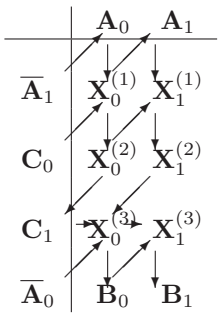
$$\mathcal{R}(\mathbf{C}_0, \mathbf{C}_1, \mathbf{A}_0, \mathbf{A}_1) \quad = \quad (\mathbf{B}_0, \mathbf{B}_1)$$

where

$$\mathbf{B}_0 = \overline{\mathbf{A}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1)$$
$$\mathbf{B}_1 = (\overline{\mathbf{A}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1)) *$$
$$*((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) *$$
$$*((\overline{\mathbf{A}}_1 * \mathbf{A}_0) * \mathbf{A}_1) * (\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1).$$

The reasons why the expressions for $\mathbf{B}_0$ and $\mathbf{B}_1$ are as they are given in Definition 8 can be easily understood by looking at the Table 5a. The diagonal arrows can be interpreted as quasigroup operations between the source and the destination, and the vertical or the horizontal arrows as equality signs "=".

Table 5. a. Schematic representation of the function $\mathcal{R}$, b. It is difficult to solve a system of two equations where $\mathbf{B}_0$, $\mathbf{B}_1$, $\mathbf{C}_0$ and $\mathbf{C}_1$ are given, and $\mathbf{A}_0 = X_0$ and $\mathbf{A}_1 = X_1$ are indeterminate variables.



a.

	$\mathbf{X}_0$	$\mathbf{X}_1$
$\overline{\mathbf{X}}_1$	$\mathbf{X}_0^{(1)}$	$\mathbf{X}_1^{(1)}$
$\mathbf{C}_0$	$\mathbf{X}_0^{(2)}$	$\mathbf{X}_1^{(2)}$
$\mathbf{C}_1$	$\mathbf{X}_0^{(3)}$	$\mathbf{X}_1^{(3)}$
$\overline{\mathbf{X}}_0$	$\mathbf{B}_0$	$\mathbf{B}_1$

b.

The conjectured one-wayness of $\mathcal{R}$ if we assume that only the values $\mathbf{B}_0$, $\mathbf{B}_1$, $\mathbf{C}_0$ and $\mathbf{C}_1$ are given, can be explained by Table 5b. In order to find pre-image values $\mathbf{A}_0 = \mathbf{X}_0$ and $\mathbf{A}_1 = \mathbf{X}_1$ we can use Definition 8 and obtain the following relations for the elements of Table 5b:

$$\mathbf{X}_0^{(1)} = \overline{\mathbf{A}}_1 * \mathbf{A}_0$$
$$\mathbf{X}_1^{(1)} = \mathbf{X}_0^{(1)} * \mathbf{A}_1 = (\overline{\mathbf{A}}_1 * \mathbf{A}_0) * \mathbf{A}_1$$
$$\mathbf{X}_0^{(2)} = \mathbf{C}_0 * \mathbf{X}_0^{(1)} = \mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)$$
$$\mathbf{X}_1^{(2)} = \mathbf{X}_0^{(2)} * \mathbf{X}_1^{(1)} = (\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * ((\overline{\mathbf{A}}_1 * \mathbf{A}_0) * \mathbf{A}_1)$$
$$\mathbf{X}_0^{(3)} = \mathbf{X}_0^{(2)} * \mathbf{C}_1 = (\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1$$
$$\mathbf{X}_1^{(3)} = \mathbf{X}_1^{(2)} * \mathbf{X}_0^{(3)} = ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) *$$
$$*((\overline{\mathbf{A}}_1 * \mathbf{A}_0) * \mathbf{A}_1)) * ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1)$$
$$\mathbf{B}_0 = \overline{\mathbf{A}}_0 * \mathbf{X}_0^{(3)} = \overline{\mathbf{A}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1)$$
$$\mathbf{B}_1 = \mathbf{B}_0 * \mathbf{X}_1^{(3)} = (\overline{\mathbf{A}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1)) *$$
$$*(((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * ((\overline{\mathbf{A}}_1 * \mathbf{A}_0) * \mathbf{A}_1)) *$$
$$*((\mathbf{C}_0 * (\overline{\mathbf{A}}_1 * \mathbf{A}_0)) * \mathbf{C}_1))$$

From them, we can obtain the following system of quasigroup equations with indeterminates $\mathbf{X}_0, \mathbf{X}_1$:

$$\begin{cases} \mathbf{B}_0 = \overline{\mathbf{X}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{X}}_1 * \mathbf{X}_0)) * \mathbf{C}_1) \\ \mathbf{B}_1 = (\overline{\mathbf{X}}_0 * ((\mathbf{C}_0 * (\overline{\mathbf{X}}_1 * \mathbf{X}_0)) * \mathbf{C}_1)) * \\ \quad *(((\mathbf{C}_0 * (\overline{\mathbf{X}}_1 * \mathbf{X}_0)) * ((\overline{\mathbf{X}}_1 * \mathbf{X}_0) * \mathbf{X}_1)) * \\ \quad *((\mathbf{C}_0 * (\overline{\mathbf{X}}_1 * \mathbf{X}_0)) * \mathbf{C}_1)). \end{cases}$$

One can show that for any given $\mathbf{A}_0 = \mathbf{X}_0 \in Q$ either there are values of $\mathbf{A}_1 = \mathbf{X}_1$ 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} or 2^{512}) then solving this simple system of just two 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 $\mathbf{A}_0 = \mathbf{X}_0 \in Q$ until obtaining the other indeterminate $\mathbf{A}_1 = \mathbf{X}_1$. That brute force method would require in average $\frac{1}{2}|Q|$ attempts to guess $\mathbf{A}_0 = \mathbf{X}_0 \in Q$ before solving the system.

3.4 Generic description for all variants of the EDON-$\mathcal{R}'$

First we are giving a generic description for all variants of the EDON-$\mathcal{R}'$ hash algorithm. Then, in the following subsections we are giving some concrete specifics for four different message digest sizes: $n = 224$, $n = 256$, $n = 384$ and $n = 512$. The generic description of EDON-$\mathcal{R}'$ hash algorithm is given in Table 6.

In the generic description the words of the initial double pipe $P_0^{(i-1)}, P_1^{(i-1)}, \ldots, P_{15}^{(i-1)}$ are represented as two vectors of length 8 i.e. $(P_0^{(i-1)}, P_1^{(i-1)}, \ldots, P_{15}^{(i-1)}) \equiv (\mathbf{P}_0^{(0)}, \mathbf{P}_1^{(0)}) \equiv P^{(0)}$. Then, by each iteration, they are replaced by intermediate double pipe value, $P^{(i)} = (\mathbf{P}_0^{(i)}, \mathbf{P}_1^{(i)})$, ending with the final double pipe value $P^{(N)} = (\mathbf{P}_0^{(N)}, \mathbf{P}_1^{(N)})$. The final result of EDON-$\mathcal{R}'$ is a n–bit message digest that are the least significant n bits from the final double pipe.

Similar notation is used for the values of the padded message $M' = (M^{(1)}, M^{(2)}, \ldots, M^{(N)})$. Namely, every message block $M^{(i)}$ is represented as a pair of two vectors of length 8, $M^{(i)} \equiv (\mathbf{M}_0^{(i)}, \mathbf{M}_1^{(i)})$ and the notation $\overline{M^{(i)}}$ denotes the swaped order of $\mathbf{M}_0^{(i)}$ and $\mathbf{M}_1^{(i)}$ i.e. $\overline{M^{(i)}} \equiv (\mathbf{M}_1^{(i)}, \mathbf{M}_0^{(i)})$. A graphic representation of the EDON-$\mathcal{R}'$ hash algorithm is given in the Figure 1.

Preneel, Govaerts, and Vandewalle in [2] have located 12 secure schemes for constructing hash functions from block ciphers. Black et. al., [12] have proved (in an ideal cipher model) that those schemes are collision-resistant too.

Table 6. A generic description of the EDON-$\mathcal{R}'$ hash algorithm

Algorithm: EDON-$\mathcal{R}'$
Input: Message M of length l bits, and the message digest size n. **Output:** A message digest $Hash$, that is long n bits.
1. Preprocessing (a) Pad the message M. (b) Parse the padded message into N, m-bit message blocks, $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$. (c) Set the initial value of the double pipe $P^{(0)}$. 2. Hash computation For $i = 1$ to N $P^{(i)} = \mathcal{R}(P^{(i-1)}, M^{(i)}) \oplus P^{(i-1)} \oplus \overline{M^{(i)}}$; 3. $Hash$ =Take_n_Least_Significant_Bits($P^{(N)}$).

The basic iterative relation for the scheme number 7 (PGV7) is:

$$H_i = E(M_i, H_{i-1}) \oplus M_i \oplus H_{i-1}$$

where the notation $E(K, X)$ denotes a block cipher operation with a key K and a plaintext X.

It is relatively easy to prove the following theorem:

Theorem 2. *The function $\mathcal{R}(\mathbf{C}_0, \mathbf{C}_1, \mathbf{A}_0, \mathbf{A}_1) = (\mathbf{B}_0, \mathbf{B}_1)$ as it is defined in Definition 8, is a bijection if the values $\mathbf{A}_0$ and $\mathbf{A}_1$ are kept fixed.* □

From there it follows directly that the compression function in EDON-$\mathcal{R}'$ is a double-pipe PGV7 scheme, where the function $\mathcal{R}(\cdot)$ has the role of the block cipher, the double-pipe has the role of the plaintext and the message block has the role of the key.

4. Operating characteristics of EDON-$\mathcal{R}'$

4.1 Speed of EDON-$\mathcal{R}'$ on NIST SHA-3 Reference Platform

We have developed and measured the performances of EDON-$\mathcal{R}'$ on a platform with the following characteristics:

CPU: Intel Core 2 Duo,

Clock speed: 2.4 GHz,

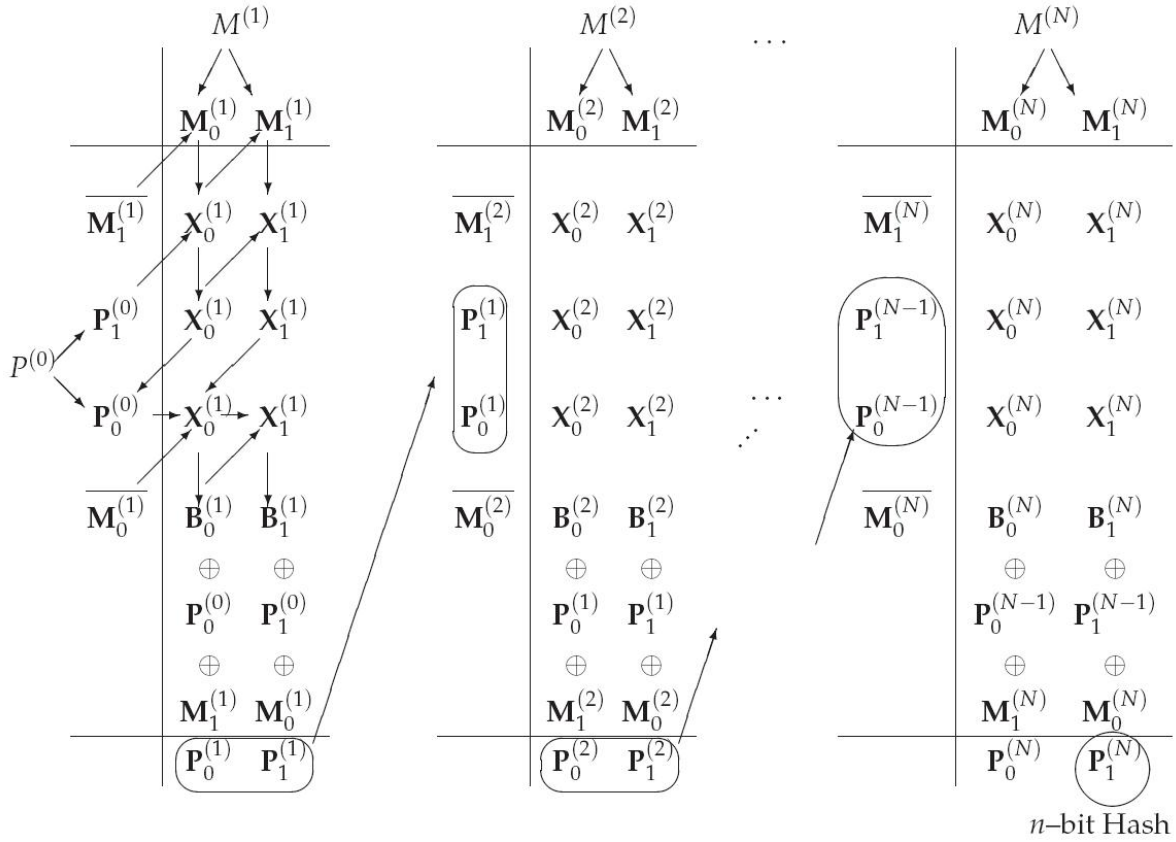


Figure 1. A graphic representation of the EDON-$\mathcal{R}'$ hash algorithm.

Memory: 4GB RAM,

Operating system: Windows Vista Enterprise 64-bit (x64) Edition with Service Pack 1,

Compiler: ANSI C compiler in the Microsoft Visual Studio 2005 Professional Edition.

Compiler: ANSI C compiler in the Intel C++ v 11.0.072.

Generally, the Intel compiler was giving us 10% to 50% faster code, so in this paper in Table Table 7 and in Table Table 8 we are giving only the speeds obtained by the Intel C++ v 11.0.072 compiler. For measuring the speed of the hash function expressed as cycles/byte we have used the `rdtsc()` function and a modified version of a source code that was given to us by Dr. Brian Gladman from his optimized realization of SHA-2 hash function [6].

4.2 Memory requirements of EDON-$\mathcal{R}'$ on NIST SHA-3 Reference Platform

When processing the message block $M^{(i)} = (M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)})$, we need the current value of the double pipe $P^{(i-1)} = (P_0^{(i-1)}, P_1^{(i-1)}, \ldots, P_{15}^{(i-1)})$, the values of the new double pipe – in the reference source code indexed as $P^{(i)} = (P_{16}^{(i)}, P_{17}^{(i)}, \ldots, P_{31}^{(i)})$ and 16 temporary variables (in the reference source code denoted as `t0`, ..., `t15`).

Table 7. The performance of optimized 32–bit version of EDON-$\mathcal{R}'$ in machine cycles per data byte on Intel Core 2 Duo for different hash data lengths, obtained by Intel C++ v 11.0.072.

	Speed in cycles/byte for different lengths (in bytes) of the digested message.				
MD Size	10	100	1000	10,000	100,000
224	64.90	10.93	7.08	6.78	6.74
256	58.90	10.69	7.07	6.78	6.71
384	153.70	15.49	13.03	10.96	10.74
512	154.90	15.49	12.77	10.93	10.74

The need of memory is thus:

- 16 words of $M^{(i)}$,

Table 8. The performance of optimized 64–bit version of EDON-$\mathcal{R}'$ in machine cycles per data byte on Intel Core 2 Duo for different hash data lengths, obtained by Intel C++ v 11.0.072.

	Speed in cycles/byte for different lengths (in bytes) of the digested message.				
MD Size	10	100	1000	10,000	100,000
224	49.30	8.53	5.52	4.85	4.90
256	46.90	8.05	5.13	4.85	4.90
384	55.30	5.17	2.90	2.72	2.74
512	51.70	5.17	2.93	2.73	2.74

- 32 words of $P^{(i)}$.
- 16 temporary words `t0`, ..., `t15`.

which is in total 64 words. That means that EDON-$\mathcal{R}'$224/256 use 256 bytes and EDON-$\mathcal{R}'$384/512 use 512 bytes.

5. Conclusions

The design of EDON-$\mathcal{R}'$ heavily uses combinations of bitwise operations of XORing, rotating and operations of addition in $\mathbb{Z}_{2^{32}}$ or in $\mathbb{Z}_{2^{64}}$ (which are mutually nonlinear operations). This strategy, combined with the conjectured one-wayness of the PGV7 compression function and the good differential properties of the underlying quasigroup operations used in $\mathcal{R}$ are the cornerstones of the EDON-$\mathcal{R}'$ strength.

EDON-$\mathcal{R}'$ is designed to have a security strength that is at least as good as the hash algorithms currently specified in FIPS 180-2, and this security strength is achieved with significantly improved efficiency. Having in mind the fact that EDON-$\mathcal{R}'$ design differs completely from the design of SHA-2 family of hash functions, we claim that any possibly successful attack on SHA-2 family of hash functions is unlikely to be applicable to EDON-$\mathcal{R}'$.

References

[1] *Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family*. NIST, 2007. `http://csrc.nist.gov/groups/ST/hash/index.html`.

[2] R. G. B. Preneel and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Proceedings of CRYPTO 1993*, volume 773 of *LNCS*, pages 368–378, 1994.

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

[4] M. M. S. J. K. A. D. V. K. Danilo Gligoroski, Rune Steinsmo Ødegård. Cryptographic hash function EDON-$\mathcal{R}$. In *SHA-3 Algorithm Submission*, 2008.

[5] J. Dénes and A. D. Keedwell. A new authentication scheme based on latin squares. *Discrete Math.*, 106-107:157–161, 1992.

[6] B. Gladman. Sha1, sha2, hmac and key derivation in c. `http://fp.gladman.plus.com/cryptography_technology/sha/index.htm`.

[7] D. Gligoroski. Candidate one-way functions and one-way permutations based on quasigroup string transformations. Cryptology ePrint Archive, Report 2005/352, 2005. `http://eprint.iacr.org/`.

[8] D. Gligoroski. On a family of minimal candidate one-way functions and one-way permutations. *International Journal of Network Security*, in print 2009.

[9] D. Gligoroski and S. J. Knapskog. Edon–$\mathcal{R}(256, 384, 512)$ – an efficient implementation of edon–$\mathcal{R}$ family of cryptographic hash functions. *Comment.Math.Univ.Carolin.*, 49,2:219–239, 2008.

[10] D. Gligoroski, S. Markovski, and S. J. Knapskog. The stream cipher edon80. In M. Robshaw and O. Billet, editors, *New Stream Cipher Designs*, volume 4986 of *LNCS*, pages 152–169. Springer-Verlag, 2008.

[11] D. Gligoroski, S. Markovski, and L. Kocarev. Edon–$\mathcal{R}$, an infinite family of cryptographic hash functions. August 2006. `http://www.csrc.nist.gov/pki/HashWorkshop/2006/Papers/GLIGOROSKI\_EdonR-ver06.pdf`.

[12] P. R. J. Black and T. Shrimpton. Black-box analysis of the block-cipher-based hash function constructions from pgv. In *Proceedings of CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335, 2002.

[13] S. Markovski, D. Gligoroski, and V. Bakeva. Quasigroup string processing. In *Part 1, Contributions, Sec. Math. Tech. Sci., MANU*, volume XX, pages 13–28, 1999.

[14] B. McKay and E. Rogoyski. Latin squares of order 10. *Electronic J. Comb.*

[15] J. D. H. Smith. *An introduction to quasigroups and their representations*. Chapman & Hall/CRC, 2007.

[16] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full sha-1. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005.