Three-Dimensional Data Stochastic Transformation Algorithms for Hybrid Supercomputer Implementation

Chugunkov I.V., Ivanov M.A., Rovnyagin M.M., Skitev A.A., Spiridonov A.A., Vasilyev N.P.
Muleys R.B., Michaylov D.M.
National Research Nuclear University “MEPhI”
Moscow, Russian Federation
IVChugunkov@mephi.ru, MAIvanov@mephi.ru, NPVasilyev@mephi.ru

Abstract — **This paper describes new three-dimensional algorithms of stochastic data transformation, offering a solution for information security problems. The most important feature of these algorithms is a high degree of parallelism at the level of elementary operations. In this paper we present a new 3D stochastic transformation called DOZEN, inspired by AES cipher, and two new constructions of S-box, called 2D and 3D S-boxes respectively.**

Keywords* — *Pseudorandom number generator, Hybrid computing systems, S-box, Square architecture, Cube architecture.

I. Introduction

An essential element of any information security system is the pseudorandom number generators (PRNG). The main functions of the PRNG are:

- key information and passwords generation;
- random requests generation during remote objects authentication procedure;
- introduction of indeterminacy into security means and objects operation, etc.

Unpredictable PRNG are the basis of stochastic methods which are used for the following tasks: ensuring the secrecy or confidentiality of information, authenticity confirmation of the information interaction subjects, programs control flow monitoring, value integrity of interacting information objects (messages, data arrays). Generators implemented for information security features should be compliant to very stringent requirements for unpredictability, statistical security and value of generated sequence period [1].

II. Problem Statement

The main problem with the PRNG design is the difficulty to solve the contradiction between the quality of the generated pseudorandom sequences, on the one hand, and efficiency of software and hardware implementation (defining generators' performance) on the other hand. The most rational solution in terms of these criteria is a PRNG with iterative block encryption functions which essentially are the nonlinear feedback or output functions.

Fig. 1 shows a general block diagram of a PRNG, where S_0 – the initial state, S_i – i^{th} state of the memory elements of the generator. In practice, it uses two variations –OFB (output feedback) scheme when the quality of the generator is determined by a nonlinear function of the feedback, and two-stage scheme Counter, when the first stage (counter) provides the maximum period of generated sequences and the quality of generator is determined by a nonlinear output function of the second stage [1].

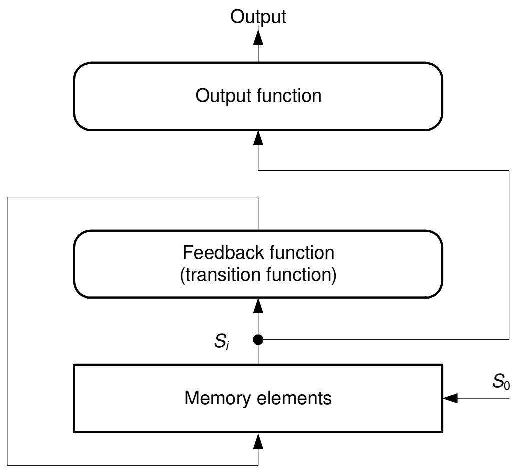

Fig. 1. PRNG structure.

One of the most promising modern architectures for building a qualitative PRNG is the architecture of «Square», proposed by the authors of the Square and the Rijndael block ciphers [2-4]. Rijndael won the competition for adoption of a new symmetric-key encryption standard AES (Advanced Encryption Standard). It has also rapidly became a worldwide standard within financial commercial institutions, and is the default cipher employed in many software and hardware applications that use encryption. NIST predicts that the cipher will remain secure for at least 20-30 years [5].

AES is an algorithm with a simple and very elegant design. It has been designed to offer strong resistance against known attacks. It has always seemed unlikely that its security could be affected by conventional methods of cryptanalysis.

AES is based on a design principle known as a Substitution Permutation Network (SPN). SPN consists of two basic cryptographic operations (substitution and permutation) that provide diffusion and confusion of data and key. Confusion is intended to make the relationship between the key and ciphertext as complex as possible. Diffusion refers to rearranging or spreading out of bits in the message so that any redundancy in the plaintext is spread out over the ciphertext [6].

Most clearly the advantages of the square architecture occur in AES-128. In this case all input and output data blocks, all the intermediate results of the transformations, all the round keys are represented in the form of a 4x4 square array of bytes. So AES-128 has a fixed block size of 128 bits and a key size of 128 bits. AES operates on a 4×4 matrix of bytes, termed as the State. The State consists of 4 rows and 4 columns of bytes. Most AES calculations are done in a finite field GF(2^8). The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps, including one that depends on the encryption key.

A round function consists of four transformations – SubBytes, ShiftRows, MixColumns and AddRoundKey. Every transformation affects all bytes of the State. The SubBytes transformation is nonlinear byte substitution that operates on each bytes of the State using a look-up table called an S-box. The ShiftRows transformation is a circular shifting operation, which rotates the rows of the State with different numbers of bytes. The MixColumns transformation mixes the bytes in each columns of the State. While performing this transformation the column bytes are considered as the coefficients of the polynomial $A(x)$ of degree 3 over GF(2^8). MixColumns is the multiplication in modulus $x^4 + 1$ of the polynomial $A(x)$ by the polynomial $03x^3 + x^2 + x + 02$. The AddRoundKey transformation is a simple XOR of the round key with the State.

The main advantages of stochastic algorithms with the Square architecture are:

- simple and clear logic of work, convenient for analysis and justification of resistibility;
- simple justification of data diffusion and confusion while two-round structure is used,;
- high degree of parallelism operation-level;
- byte-oriented structure.

Square architecture is interesting and active area of research. The purpose of the paper is further development of this architecture.

III. Areas of Improvement of Square Architecture

There are the following ways of Square architecture improvement:

- replacement of the ShiftRows transformation by the MixRows transformation will ensure the complete data diffusion and mixing during single round;
- mutual use of Feistel Network and Square architectures to provide reversibility of data transformation;
- conversion from the Square architecture to the Cube architecture [1, 7-9].

IV. 3D Transformation DOZEN

Consider a nonlinear transformation with the Cube architecture named DOZEN (because algorithm has twelve basic steps as discussed below). The DOZEN is based on the following principles:

- representation of the input and output data blocks, all the intermediate transformation results and round keys K_0, K_1, K_2, K_3 as cubic array of bytes $4 \times 4 \times 4$ (Fig. 2);
- definition of the layer (or slice) concept – a square array of bytes 4×4 (Fig. 3);
- representation of the i^{th} round key in the form of four subkeys (RoundSubKeys) K_{i0}, K_{i1}, K_{i2}, K_{i3}, i =1, 2, 3, each of which is basically a square array of bytes 4×4;
- three-dimensional data block transformation by the layers along the axes x, y, z;
- inclusion of the layer transformation (MixLayer) consisting four steps – SubBytes, MixRows, MixColumns, AddRoundSubKey;
- usage of Rijndael cryptographic algorithms during MixRow and MixColumn transformations when implementing the MixColumn transformation.

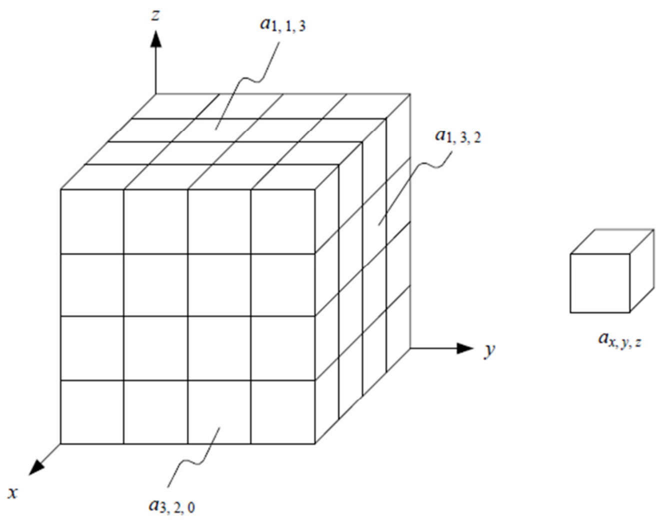

Fig. 2. DOZEN: a data block and a single byte of data block.

The transformations sequence of 512 bits (4 × 4 × 4 × 8) data block with with the structure shown in Fig. 2, where $a_{x, y, z}$, $x = 0, 1, 2, 3$, $y = 0, 1, 2, 3$, $z = 0, 1, 2, 3$, – bytes, are as follows:

- decomposition of the data block into L_{x0}, L_{x1}, L_{x2}, L_{x3} layers along the *x* axis (Fig. 3);
- first round: the stochastic transformation of layers L_{x0}, L_{x1}, L_{x2}, L_{x3} by performing SubBytes, MixRows, MixColumns and AddRoundSubKey transformations for each L_{xk} layer;
- decomposition of the data block into L_{y0}, L_{y1}, L_{y2}, L_{y3} layers along the *y* axis (Fig. 3);
- second round: the stochastic transformation of layers L_{y0}, L_{y1}, L_{y2}, L_{y3} by performing SubBytes, MixRows, MixColumns and AddRoundSubKey transformations for each L_{yk} layer;
- decomposition of the data block into L_{z0}, L_{z1}, L_{z2}, L_{z3} layers along the *z* axis (Fig. 3);
- third round: the stochastic transformation of layers L_{z0}, L_{z1}, L_{z2}, L_{z3} by performing SubBytes, MixRows, MixColumns and AddRoundSubKey transformations for each L_{zk} layer.

Finally, the following sequence of 3D-transformations is obtained (Fig. 4a):

Step 0. Addition with a round key K_0 (AddRoundKey K_0).

First round:

Step 1. Transformation of L_{x0} layer (MixLayer L_{x0}).

Step 2. Transformation of L_{x1} layer (Mix*Layer* L_{x1}).

Step 3. Transformation of L_{x2} layer (Mix*Layer* L_{x2}).

Step 4. Transformation of L_{x3} layer (Mix*Layer* L_{x3}).

Second round:

Step 5. Transformation of L_{y0} layer (Mix*Layer* L_{y0}).

Step 6. Transformation of L_{y1} layer (Mix*Layer* L_{y1}).

Step 7. Transformation of L_{y2} layer (Mix*Layer* L_{y2}).

Step 8. Transformation of L_{y3} layer (Mix*Layer* L_{y3}).

Third round:

Step 9. Transformation of L_{z0} layer (Mix*Layer* L_{z0}).

Step 10. Transformation of L_{z1} layer (Mix*Layer* L_{z1}).

Step 11. Transformation of L_{z2} layer (Mix*Layer* L_{z2}).

Step 12. Transformation of L_{z3} layer (Mix*Layer* L_{z3}).

Fig. 4b shows an example of PRNG construction by the CTR scheme, i.e. example of conversion DOZEN as a function of PRNG output.

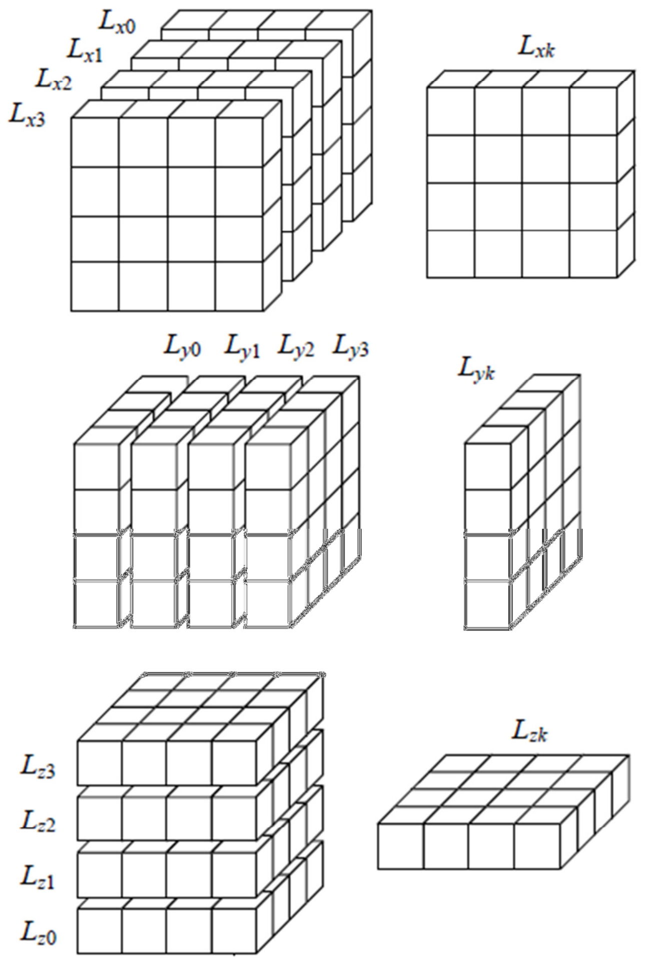

Fig. 3. DOZEN: layering along the x, y, z axeses and the individual layers L_{xk}, L_{yk}, L_{zk}.

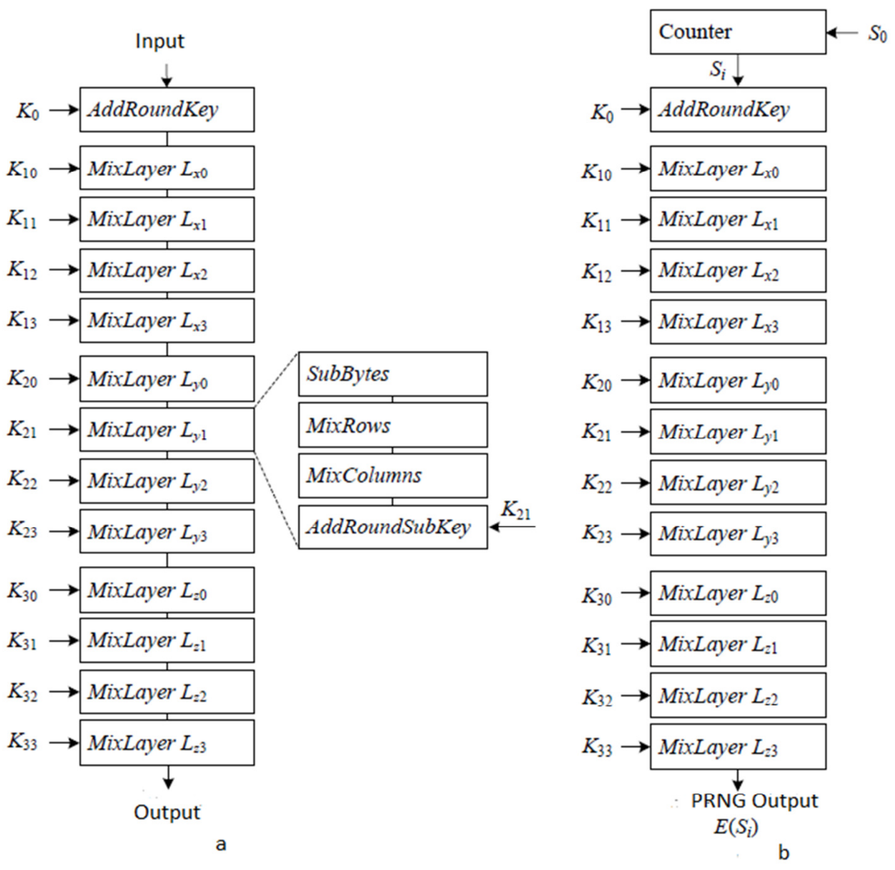

Fig. 4. Three-dimensional transformation DOZEN: a – sequence of 3D-transformations; b – example of PRNG construction a scheme CTR.

V. Implementation of DOZEN Using Hybrid Computing Technologies

Hybrid (or heterogeneous) computer systems combine the convenience of classical central processing unit (CPU) computing with massively-parallel computing on graphics processors (GPU) [10, 11]. GPU is characterized by the presence of a large number (hundreds and thousands) of cores working in parallel. The performance of CPU/GPU system is up to several ten times compared with the classical CPU-system. A number of the most productive modern supercomputers also have hybrid architecture of CPU/GPU.

Hybrid CPU systems solve the problem of program execution control and perform simple calculations. The most critical parts of a program segments are presented in the form of special functions (kernels) running on GPU. Modern manufacturers of GPUs, in particular, NVIDIA Company provide software developers for CPU/GPU systems with powerful tools. The useful feature of these tools is that they are free. For example, CUDA Toolkit [12] and the Parallel NSight [13] integrated with popular software development systems, such as Microsoft Visual Studio and NetBeans.

For the software implementation of the proposed algorithm DOZEN the most appropriate technology is CUDA (Compute Unified Device Architecture) from NVIDIA Company [10, 11]. The minimal processing unit in CUDA is termed a thread. In fact, the thread has a set of specific actions over the data element. The threads are grouped in warps; all the threads of a single warp are physically executed in parallel on a streaming multiprocessor; GPU has multiple streaming multiprocessors. An important feature of CUDA is that the threads can be combined in one-, two- or three-dimensional structures called blocks. Blocks, in their turn, are grouped into a larger multidimensional structure called a grid. In other words, a grid is the set of all threads performing parallel processing. It is a flexible multidimensional hierarchical structure. Therefore, CUDA-programmer can operate with one-, two- or three-dimensional structures for parallel processing of input data including combination of dimensions of these structures.

It is evident that within the each work round of the PRNG DOZEN all layers can be processed in parallel (Appendix A), and the use of CUDA will significantly simplify the process of software development based on DOZEN algorithm.

Fig.5 gives an illustration of the performance of Counter mode encryption for different input size.

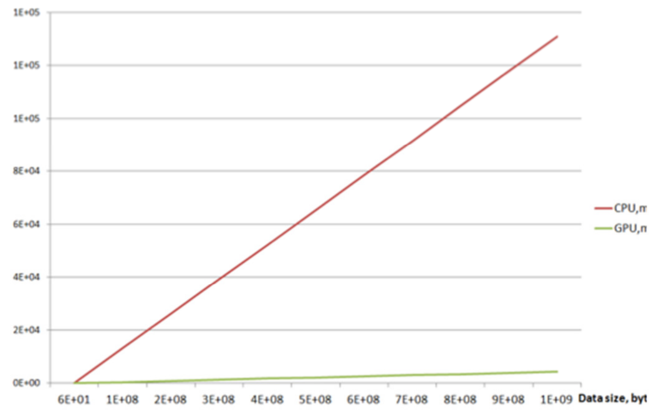

Fig. 5. DOZEN performance of different input size.

VI. THREE-DIMENSIONAL S-BOX

A substitution box (S-box) is one of the basic components of the symmetric cryptography. S-box may be defined as nonlinear transformation which map a number of N input bits into N output bits by using $N \times 2^N$ look-up table. The quality of S-boxes determines the unpredictability of pseudorandom number generators and cryptographic strength of block and stream ciphers, hash functions. In order to resist to all possible cryptographic attacks, S-boxes have to satisfy a set of criteria, that are used to build proper S-boxes. A criterion of non-linearity is needed for the cryptographic primitive to resist against linear cryptanalysis, strict avalanche criteria (SAC) is necessary to resist against differential cryptanalysis. The high order strict avalanche criteria is an extension of SAC [14-18].

Let us consider two new ways of constructing S-boxes; corresponding devices may be called 2D and 3D S-boxes respectively.

A. *The algorithm of 2D S-box functioning*

We represent input and output data blocks, as well as all intermediate transformation results in a square array bits $N \times N$, where N – number of input and output bits of S-boxes. Therefore, the amount of key information uniquely defining the logic of each S-box equals $N \times 2^N$.

The sequence of substitution transformation of the square array A of bits $N \times N$ (A = SubSquare[A] or $A = S_{sq}[A]$) is the following (see Figure 6):

- division of an input array A into N rows R_i with length of N, $i = 0, 1, \ldots, (N-1)$;
- substitution of rows (the SubRows transformation), i.e. the substitution of each i^{th} N-bit binary set R_i using the corresponding S_i-box: $R_i = S_i[R_i]$;
- division of resulting array A = SubRows[A] into N columns C_i with length of N;
- substitution of columns (the SubColumns transformation), i.e. the substitution of each ith N-bit binary set of C_i using the corresponding S_{i+N}-box: $C_i = S_{i+N}[C_i]$;
- result is A = SubColumns[A].

In the particular case when only one substitution table is used, i.e. $S_i = S$, the following algorithm is obtained:

- division of an input array A into N rows R_i with length of N;
- *SubRows* transformation, i.e. the substitution of each i^{th} N-bit binary set R_i:

  $R_i = S[R_i]$, $i = 0, 1, \ldots, (N-1)$;
- division of resulting array A = SubRows[A] into N columns C_i with length of N;
- *SubColumns* transformation, i.e. the substitution of each i^{th} N-bit binary set of C_i: $C_i = S[C_i]$;
- result is A = SubColumns[A].

B. *The algorithm of 3D S-box functioning*

We represent input and output data blocks, as well as all intermediate transformation results in the form of cubic bit array $N \times N \times N$, where N – number of input and output bits of S-boxes. Therefore, the amount of key information uniquely defining the logic of each S-box equals $N \times 2^N$.

The sequence of substitution transformation of a cubic bit array A of bits $N \times N \times N$ (A = SubCube[A] or $A = S_{cu}[A]$) is as follows:

- division of an input array A into N layers L_{xi} $N \times N$ along the x axis, $i = 0, 1, \ldots, (N-1)$;
- 2D substitution layers L_{xi}, i.e. transformation performs L_{xi} = SubSquare[L_{xi}] of each i^{th} layer L_{xi} using corresponding S-boxes;
- division of resulting array A = SubLayersX[A] into N layers L_{yi} $N \times N$ along the y axis;
- 2D substitution layers L_{yi}, i.e. transformation perform L_{yi} = SubSquare[L_{yi}] of each i^{th} layer L_{yi} using corresponding S-boxes;
- division of resulting array A = SubLayersY[A] into N layers L_{zi} $N \times N$ along the z axis;
- 2D substitution layers L_{zi}, i.e. transformation perform L_{zi} = SubSquare[L_{zi}] of each i^{th} layer L_{zi} using the corresponding S-boxes;
- result is A = SubLayersZ[A].

The most evident aim of the proposed algorithms is the conversion of N^2- and N^3-bit data blocks using the bit replacement tables N x 2^N.

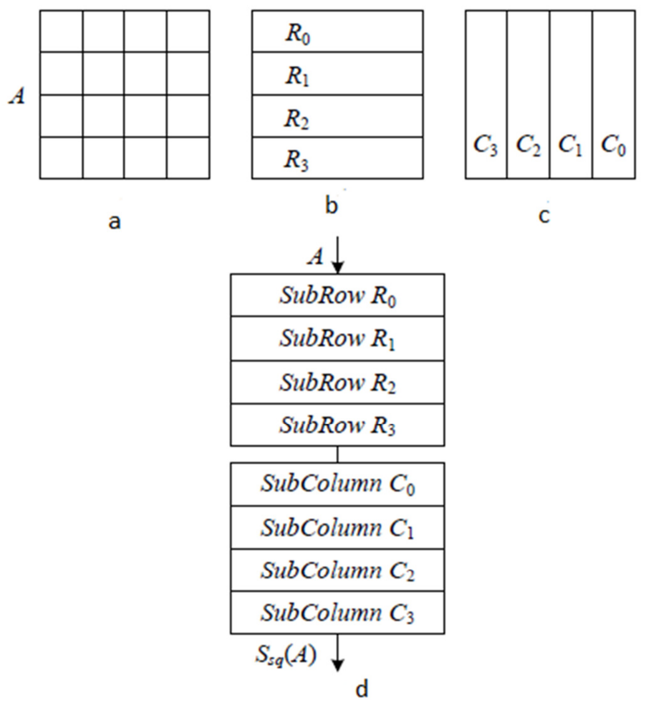

Fig. 6. 2D S-box with N = 4; a – the data block; b – the division of the data block by rows; c – dividing of the data block by columns; d – the sequence of steps of transformation.

VII. Conclusion

This paper describes the proposed three-dimensional algorithm DOZEN for stochastic data transformation. The essence of the proposed solution is the use of Cube architecture. The composition of the resulting 3D-transformation includes a zero step of adding round key and 12 steps of 2D-transformations of data block layers: four for each of coordinates *x*, *y* and *z*. The most important features of the proposed transformation are byte-oriented structure and a high degree of parallelism that allows achieving a high performance of the algorithm for hybrid computing systems CPU/GPU based of CUDA technology.

The same idea is used to build 2D and 3D substitution boxes. The essence of the proposed solutions is N^2- and N^3-bit data blocks stochastic transformation using $N \times 2^N$ S-boxes.

The most obvious area of use of the considered algorithms (in addition with the generation of unpredictable pseudorandom sequences) is building cryptographic primitives of data hashing, block and stream ciphering.

Appendix A: Kernel function gpu_dozen_encrypt

Algorithm 1

```
__global__ void gpu_dozen_encrypt(unsigned char *pre, const int size){
__shared__ unsigned char
cache[THREADS_PER_BLOCK*DOZEN_SQUARE_SIZE];
unsigned char line[4];//used for gpu_dozen_mixline function
int tid = blockIdx.x*blockDim.x  + threadIdx.x;
while(tid<size/DOZEN_SQUARE_SIZE){
        //load next data chunk to SHMEM
        for(int     t=threadIdx.x*DOZEN_SQUARE_SIZE,k=0,
        l=t+DOZEN_SQUARE_SIZE;t<l;t++,k++){
                cache[t]=pre[tid*DOZEN_SQUARE_SIZE+k];
        }
        __syncthreads();//used before changing working slice
        ...
        //AddRoundKeyCube and X-Transformation
        ...
        //SubBytesY
        for(int t=threadIdx.x*DOZEN_SQUARE_SIZE,
        l=t+DOZEN_SQUARE_SIZE;t<l;t++){
                cache[t]=sbox[cache[t]];
        }
        __syncthreads();
        int dozenBlockId=threadIdx.x/4;
        int dozenLayerId=threadIdx.x%4;
        //each thread works with its own square
        int effectiveIdx=dozenBlockId*DOZEN_BLOCK_SIZE+
        dozenLayerId*DOZEN_LINE_SIZE;
        line[0]=cache[effectiveIdx+3];
        line[1]=cache[effectiveIdx+19];
        line[2]=cache[effectiveIdx+35];
        line[3]=cache[effectiveIdx+51];
        gpu_dozen_mixline(line);//similar to Rijndael mix columns algorithm
        cache[effectiveIdx+3]=line[0];
        cache[effectiveIdx+19]=line[1];
        cache[effectiveIdx+35]=line[2];
        cache[effectiveIdx+51]=line[3];
        ...
        //MixRowsY1,2,3
        ...
        //MixColumsY0
        line[0]=cache[effectiveIdx+3];
        line[1]=cache[effectiveIdx+2];
        line[2]=cache[effectiveIdx+1];
        line[3]=cache[effectiveIdx+0];
        gpu_dozen_mixline(line);
        cache[effectiveIdx+3]=line[0];
        cache[effectiveIdx+2]=line[1];
        cache[effectiveIdx+1]=line[2];
        cache[effectiveIdx+0]=line[3];
        ...
        //MixColumsY1,2,3
        ...
        __syncthreads();
        //AddRoundKeyY
        for(int t=threadIdx.x*DOZEN_SQUARE_SIZE,
        k=(threadIdx.x%4)*DOZEN_SQUARE_SIZE,
        l=t+DOZEN_SQUARE_SIZE;t<l;t++,k++){
                cache[t]=cache[t]^key[k];
        }
        ...
        //Z-Transformation
        ...
        __syncthreads();
        //store results to global memory
        for(int t=threadIdx.x*DOZEN_SQUARE_SIZE,k=0,
        l=t+DOZEN_SQUARE_SIZE;t<l;t++,k++){
                pre[tid*DOZEN_SQUARE_SIZE+k]=cache[t];
        }
        tid+=blockDim.x*gridDim.x;
}
}
```

Acknowledgement

This paper was made with financial support of the Ministry of Education and Science of the Russian Federation.

References

[1] M.A. Ivanov, M.D. Michailov, N.A., N.A. Matsuk and other. Stochastic Methods and Means of Information Protection in Computer Systems and Networks. M.: KUDIC-PRESS, 2009.

[2] J. Daemen, L. Knudsen, V. Rijmen, The Block Cipher Square, 4th International Workshop on Fast Software Encryption, LNCS 1267, pp. 149–165, Springer, 1997.

[3] J. Daemen, V. Rijmen, “AES Proposal: Rijndael, Version 2”, http://www.esat.kuleuven..ac.be/vijmen/rijndael, 1999.

[4] A. Kaminsky, M. Kurdziel, S. Radziszowski, An Overview of Cryptanalysis Research for the Advanced Encryption Standard. http://www.cs.rit.edu/~spr/PUBL/aes.pdf

[5] NIST: Advanced Encryption Standard (AES) (FIPS PUB 197). National Institute of Standards and Technology (Nov 2001)

[6] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of Applied Cryptography, CRC Press; Fifth Printing (August 2001).

[7] O.V. Burdaev, M.A. Ivanov, I.I. Teterin, Assembler in the Tasks of Data Protection. 2th Edition. M.: KUDIC-OBRAZ, 2004.

[8] Nakahara Jr., J.: 3D: A Three-Dimensional Block Cipher. In: Franklin, M.K., Hui, L.C.K., Wong, D.S. (eds.) CANS 2008. LNCS, vol. 5339, pp. 252–267. Springer, Heidelberg (2008).

[9] M.A. Ivanov, N.P. Vasiliev, I.V. Chugunkov and other. Three-dimensional pseudo-random number generator, focused on the implementation of a hybrid computing systems // Vestnik NRNU MEPhI, 2012, № 2.

[10] A.V. Boreskov, A.A. Kharlamov. Technology basics CUDA. Moscow: DMK Press, 2011.

[11] CUDA Zone. http://developer.nvidia.com/cat-egory/zone/cuda-zone.

[12] CUDA Toolkit. http://developer.nvidia.com/cuda-toolkit.

[13] NVIDIA Parallel Nsight. http://developer.nvidia.com/nvidia-parallel-nsight.

[14] A.F. Webster and S.E. Tavares. On the design of S-boxes. Advances in Cryptology : Proc. of CRYPTO'85, Springer-Verlag, 523–534, (1986).

[15] R. Forré, The strict avalanche criterion: spectral properties of boolean functions and an extended definition, Proceeding of CRYPTO '88, Springer-Verlag, 450-468, (1988).

[16] K. Kim, T. Matsumoto, H. A. Imai, A Recursive Construction Method of S-boxes Satisfying Strict Avalanche Criterion, Proceeding of CRYPTO '90, 564 - 574, (1990).

[17] A. Lineham, Heuristic S-box Design, Contemporary Engineering Sciences, Vol. 1, no. 4, 147 – 168, (2008).

[18] Phyu Phyu Mar, Khin Maung Latt, New Analysis Methods on Strict Avalanche Criterion of SBoxes, World Academy of Science, Engineering and Technology 48, (2008).