Contribution to Symmetric Cryptography by Convolutional Neural Networks

Conference Paper · October 2019

DOI: 10.23919/KIT.2019.8883490

2 authors:

Miloš Očkay
General Milan Rastislav stefanik Armed Forces Academy
22 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Radoslav Forgac
Slovak Academy of Sciences
19 PUBLICATIONS **131** CITATIONS
SEE PROFILE

Contribution to Symmetric Cryptography by Convolutional Neural Networks

Radoslav Forgáč
Institute of Informatics
Slovak Academy of Sciences
Bratislava, Slovak Republic
radoslav.forgac@savba.sk

Miloš Očkay
Department of Informatics
Armed Forces Academy of general M.R. Štefánik
Liptovský Mikuláš, Slovak Republic
milos.ockay@aos.sk

***Abstract*—This paper presents an implementation of Convolutional Neural Network (CNN) symmetric encryption. The results based on selected criteria are compared with Advanced Encryption Standard (AES). Authors are testing both implementations using the set of differently sized and content independent files. Compared metrics include the execution time, memory occupancy and processor load. Robustness of CNN encryption against the crypto attacks was not tested and it is out of the scope of this paper. Concluded results offer the usability prospects of selected neural network for symmetric cryptographic purposes. The results of the experiments have shown that CNN has the potential for cryptographic purposes under the conditions specified in this paper.**

Keywords— Advanced Encryption Standard, Convolutional Neural Network, cryptography, encryption, decryption, algorithm, crypto key

I. INTRODUCTION

Many different encryption algorithms have been created and implemented over the past years. Some of them have become standards. Authors decided to perform the test and compare well settled conventional symmetric encryption algorithm with experimental convolutional neural network algorithm. The test was performed as a part of a larger problem, where it needs to be decided if the CNN is suitable for the encryption purposes or if the AES implementation is a better option. Python implementation was used for both algorithms. It is included in a larger project also coded in Python language. It is known that Python does not offer the fastest possible solution, but it is appropriate for the comparison purpose [1]. Sequential implementation with the Keras [3] frontend and TensorFlow [4] backend was used. GPU accelerated parallel implementation based on TensorFlow [12] is planned in the future. The paper presents the high-level algorithm explanation of the both algorithms. It also offers the comparison of both Python implementations in the terms of execution time, memory occupancy and CPU load requirements.

The aim of our research is to specify the conditions under which CNN has a cryptographic potential. The overview of deep learning cryptography implementations in the context of side channel attacks are outlined in [14].

The paper is divided into four sections. After the introduction, section II describes CNN encryption and decryption algorithm implemented with the Keras frontend and TensorFlow backend. Section III presents high-level AES algorithm. Section IV shows the preliminaries of the testing, presents and evaluates collected results.

II. CNN ENCRYPTION AND DECRYPTION

CNN is a multi-layer feedforward neural network. Symmetric CNN encryption is based on 3D convolution. This approach requires shaping data accordingly and generating the appropriate form of secret crypo key [6].

A. Data Shaping

A file selected for encryption is stored in an array (char by char) as eight-bit values elements. This approach allows to process the file on bit level without taking the content type into account. The array is divided by 3D *block_size* (8 or 16) to determine the number of processed blocks. The final step in data shaping stage is padding of the last block. The last block is padded with randomly generated values. The whole array is reshaped according to the number of blocks. Data is ready for input of neural network (Fig. 1).

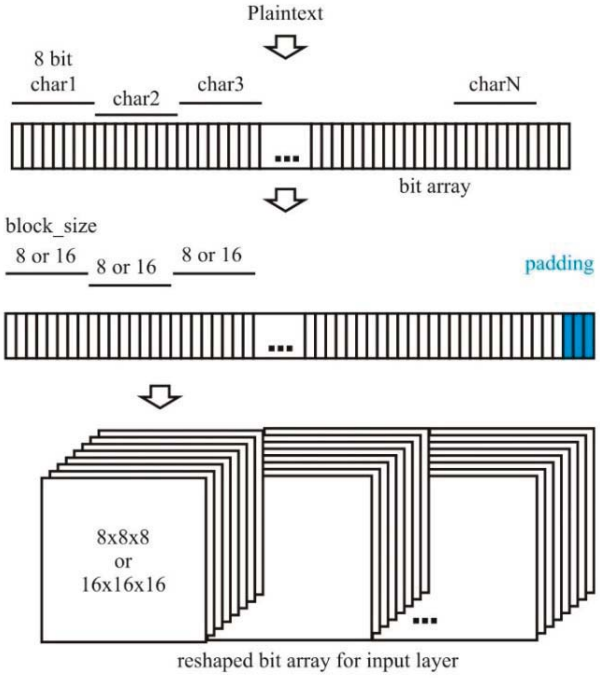

Fig. 1. Data shaping block_size 8 or 16

B. Crypto Key Generation

The crypto key is defined as the binary cube with randomly generated binary values. The crypto key can be possibly generated from a password provided by a user. The size of the cube dimensions is derived from the *block_size* input parameter. Each dimension has the same size as the *block_size* parameter. Final crypto key is saved to the file in a

flattened one dimensional character form. The crypto key is used as a convolutional kernel for CNN.

C. Neural Network

Neural network sequential model is based on linear stuck of Keras layers [7]. The structure of CNN is designed by [6] and consists of four layers (Fig. 2): Input layer, Conv3D, Flatten layer and Output layer by [6]. It is a minimal configuration defined by Keras layers for cryptographic purposes to reduce the computational requirements. Keras is a high-level neural network API written in Python and capable of running on the top of TensorFlow, Microsoft Distributed Machine Learning Toolkit and Theano [3].

Input layer initiates Keras tensor and prepares it for underlying backend, in our case TensorFlow. TensorFlow is the core open source library to help develop and train machine learning models [4]. Keras use input shape already created in data shaping stage.

3D convolutional layer Conv3D [8] represents spatial convolution over data volumes [13]. Convolutional kernel, i.e. the crypto key, is convoluted with the layer input and produces output tensor. Dimensionality of output space is set to 8 and kernel size is reduced in configuration. Linear activation function is applied:

$$f(x) = x \tag{1}$$

Output of the Conv3D layer is flattened by the Flatten layer and produces one dimensional vector. Output layer applies sigmoid activation function to the output:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Neural network is trained to convolve crypto key into all data blocks. Each data block is trained independently. Weights of each neural network are saved in a temporary file and have to be transferred to provide a successful decryption. Learning process is configured to use Adam optimizer [3], categorical cross-entropy as loss function, and accuracy as a metric. Model is trained on Numpy arrays using the *fit* function. The final encrypted data are saved to the output file.

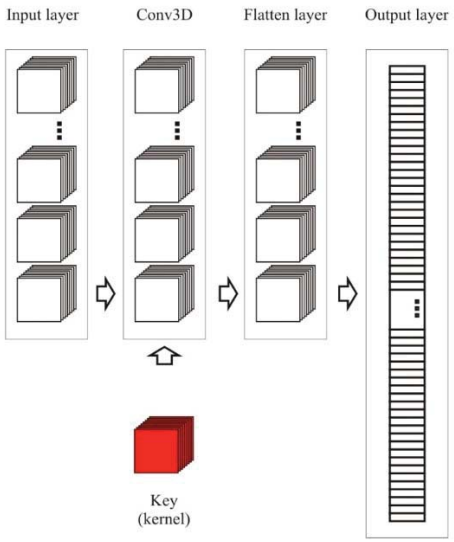

Fig. 2. The structure of CNN for encryption

In the decryption process an encrypted file is loaded, input string is split and blocks are reconstructed to provide correctly shaped input. Subsequently, the number of blocks, block size, padding size and dimensions are identified. The crypto key is loaded and a key cube is reconstructed. A model is created for each block and the weights are loaded from the temporary file. The model is compiled with the same configuration as it was for the encryption. The known padding is striped and the output is flattened and stored as a final decrypted string.

III. Advanced Encryption Standard

AES also known as Rijndael is a specification for the data encryption established by U.S. National Institute of Standards and Technology (NIST) in 2001. High level algorithm description is provided in this section [9].

A. Data Shaping

Initially AES separates data into the blocks. The block size is 128 bits and it is shaped as 4 by 4 array including 8 bits per array element (Fig. 3).

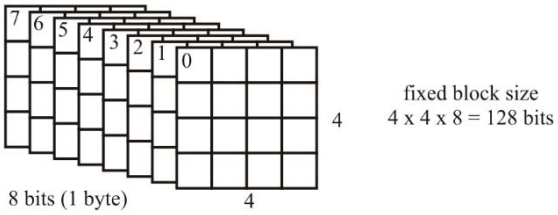

Fig. 3. AES 128 bits block

B. Initial Key and Round Keys

Initial key can be 128, 192 or 256 bits long. The crypto key size specifies the number of rounds (transformations) that convert the plaintext into the ciphertext [9]:

- 128 bits – 10 rounds
- 192 bits – 12 rounds
- 256 bits – 14 rounds

Key expansion takes the initial key and creates a series of keys for each round plus one by using Rijndael's key schedule.

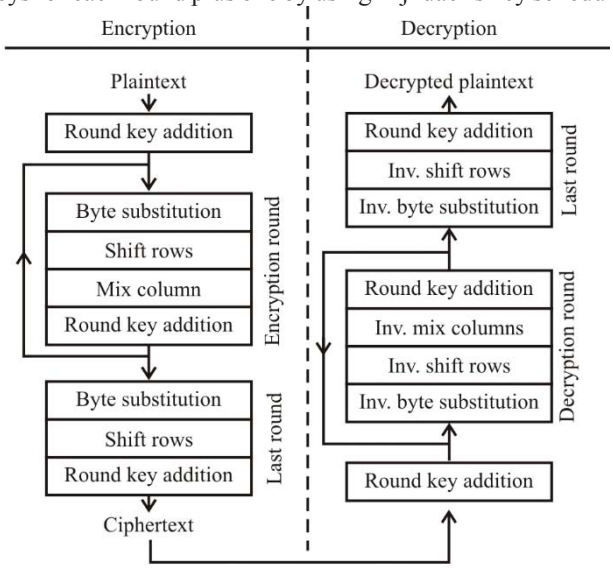

Fig. 4. AES encryption and decryption algorithm

C. Encryption

AES encryption and decryption algorithm consists of several stages, transformations and their inverse forms (Fig. 4) [11].

Initial round key addition stage combines each byte with a block of the round key with bitwise XOR function.

The round key addition is followed by nine, eleven or thirteen rounds which consist of byte substitution, rows shifting, column mixing and a round key addition. Byte substitution is a non-linear substitution where each byte is substituted with different one selected from the substitution table S-box. S-box is created as the combination of inverse function and an invertible affine transformation (Fig. 5).

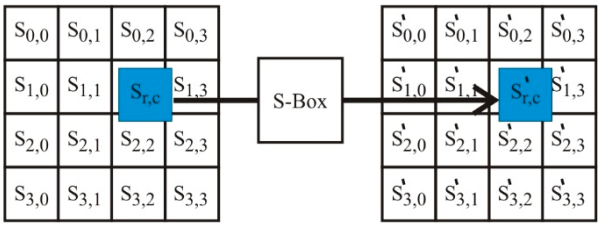

Fig. 5. Byte substitution

Rows shifting stage cyclically shifts the bytes in rows by specific offset. The first row is not shifted. The second row is shifted one to the left. The third one is shifted by offset of two. Next rows are following the same pattern. This transformation guaranties that each output column consists of bytes from each input column (Fig. 6).

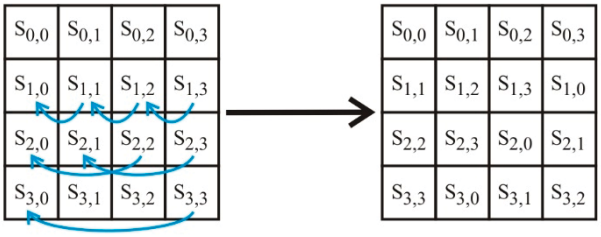

Fig. 6. Rows shifting

Columns mixing stage takes four bytes from each column, created in previous stage and combines them by using an invertible linear transformation (the multiplication with fixed polynomial). Each input byte affects all four output bytes. In the next stage (round key addition) the round key is derived from the main key using Rijndael's schedule. Derived key is combined with each corresponding byte using bitwise XOR function (Fig. 7).

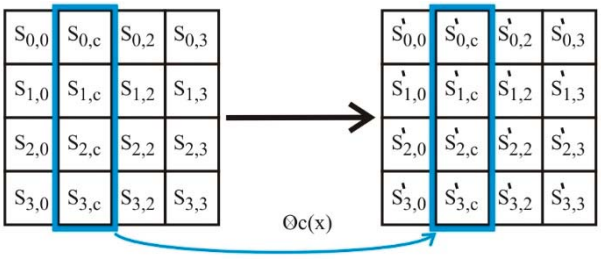

Fig. 7. Columns mixing

The last round consists of the same stages excluding the columns mixing stage. The output of the last stage is encrypted and it is called ciphertext.

Cipher Block Chaining (CBC) mode is used in tested implementation. Each cipher block depends on all previous plaintext blocks (XOR). The initialization vector in the first block makes each message unique. This mode makes encryption more robust but inner dependencies among the blocks prevent effective parallelization (Fig. 8) [10].

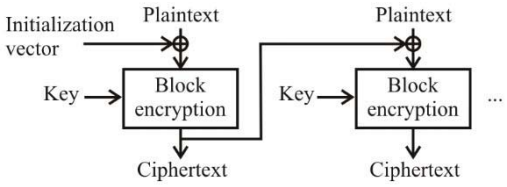

Fig. 8. Cipher Block Chaining

D. Decryption

Decryption process of AES is constructed in the reverse order to encryption process (Fig. 4). All the stages of round are performed in inverse form. Main decryption rounds are also performed in a reverse order. Round key addition is followed by a byte substitution, inverted rows shifting, inverted columns mixing, and again a round key addition. The last round follows the same rules. Inverted columns mixing stage performs the same operation as encryption column mixing, but with altered fixed polynomial. The row shifting stage cyclically shifts the bytes in rows by specific offset. The first row is not shifted. The second row is shifted one to the right. The third one is shifted by offset of two. Next rows are following the same right sifting pattern. The inverse byte substitution uses an inverse S-box to substitute each byte. Since all stages take the reversed form, the encryption and decryption algorithms require separated implementations [9].

IV. Experiments and Results

This section summarizes testing conditions and the preliminaries of the testing. Collected results are presented in the table and chart form. The following hardware, software and dataset configuration were used.

A. Hardware

Both approaches use the following test bed hardware configuration: HP 640 Workstation, Intel(R) Xeon(R) CPU E5-2643 v3 @ 3.40GHz 6 cores/12 threads, 64 GB RAM.

B. Software

The following software and modules were used for CNN: MS Windows 10 64bit, Python 3.6.3 (64bit), Tensorflow 1.13.1, Keras 2.2.4. AES implementation uses the following module: pyAesCrypt (AES256-CBC) ver.2

C. Dataset

The files used for the testing are generated with the file generator [5]. Files have exact size of 512 b, 1024 b, 2048 b, 3 KB, 10 KB and 3 MB. All files are filed with the randomly generated strings of characters (UTF-8 code), 8 bits per character. Batches of files were generated for each file size to reduce testing error. Dataset was used for the testing of both CNN and AES implementations.

D. CNN Results

CNN encryption testing consists of five measurements for each file size, one for each randomly generated file in batch. Average value is evaluated as final value for execution time, and peak values are used for memory occupancy and CPU load. Deviations, if occurred, are excluded from the evaluation of final values. Because of very long execution time, the values for larger files are estimated by linear approximation. Two different block sizes (8, 16) were used to generate the final results (Tab. 1, Tab. 2).

Python CNN encryption phase parameters:

- Input file (plaintext)
- Output file (ciphertext)
- Crypto key output file
- Block size (8, 16).

Table 1. CNN encryption results with block size 8

File size	**512 b**	**1024 b**	**2048 b**
Execution time [s]	2,797	4,689	11,162
Memory occupancy [MB]	99	110	129
CPU load [%]	17	17	17
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	210,651	1598	162000 est.
Memory occupancy [MB]	596	2071	X
CPU load [%]	17	17	X

Table 2. CNN encryption results with block size 16

File size	**512 b**	**1024 b**	**2048 b**
Execution time [s]	16,608	17,170	16,616
Memory occupancy [MB]	97	97	97
CPU load [%]	67	67	67
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	96,237	321,930	108000 est.
Memory occupancy [MB]	151	293	X
CPU load [%]	67	67	X

CNN decryption uses files encrypted in the previous phase. The files with estimated results were excluded (Tab. 3, Tab. 4).

Python CNN decryption phase parameters:

- Encrypted input file
- Decrypted output file
- File containing crypto key

Table 3. CNN decryption results with block size 8

File size	**512 b**	**1024 b**	**2048 b**
Execution time [s]	0,169	0,272	0,509
Memory occupancy [MB]	71	88	98
CPU load [%]	8	8	8
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	16,012	165,914	X
Memory occupancy [MB]	206	465	X
CPU load [%]	8	8	X

Table 4. CNN decryption results with block size 16

File size	**512 b**	**1024 b**	**2048 b**
Execution time [s]	0,162	0,164	0,160
Memory occupancy [MB]	81	81	81
CPU load [%]	5	5	5
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	0,891	4,084	X
Memory occupancy [MB]	97	142	X
CPU load [%]	5	5	X

E. AES Results

AES encryption and decryption follow the same rules as previous CNN testing. AES256-CBC implementation module pyAesCrypt [2] are used as an AES algorithm for generating results. The results consist of five runs for each file size, one for each randomly generated file in batch. The average value is used as the final value for execution time, and peak values are used for memory occupancy and CPU load. Deviations (if occurred) are excluded from the evaluation of final values.

Table 5. AES encryption results

File size	**512 b**	**1024 b**	**2048 b**
Execution time [s]	0,169	0,160	0,156
Memory occupancy [MB]	24	24	24
CPU load [%]	4	4	4
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	0,175	0,172	0,175
Memory occupancy [MB]	24	24	24
CPU load [%]	4	4	4

Table 6. AES decryption results

File size	512 b	1024 b	2048 b
Execution time [s]	0,138	0,138	0,135
Memory occupancy [MB]	24	24	24
CPU load [%]	4	4	4
File size	**3KB**	**10 KB**	**3 MB**
Execution time [s]	0,156	0,138	0,162
Memory occupancy [MB]	24	24	24
CPU load [%]	4	4	4

Encryption results display the following characteristics of CNN and AES256-CBC algorithms. CNN with both block sizes (8,16), in terms of execution time, on selected hardware and software configuration, is usable till the size of plaintext file reaches 2048 b (256 B) (Fig. 9). Beyond that point CNN execution time quickly peaks and renders the encryption unusable because of very long execution time. What is considered for CNN large file (beyond 10 KB in our case) with the long execution time, it takes AES short time to encrypt. This gap between the CNN and AES execution times stretches even more significantly for the bigger files (Fig. 9). For the files with the size of few hundreds of MB, AES needs a few seconds to encrypt the file and CNN encryption lasts for the days (not included in the results).

Decryption results display characteristics similar to the CNN encryption. CNN decryption execution time values are not as high, because the training stage is not required. However, CNN execution times start to rise significantly beyond the 2048 b (256 B) file size and they are outperformed by the AES. What was already mentioned for encryption, gap between CNN and AES encryption execution times is even more significant for larger files (Fig. 10).

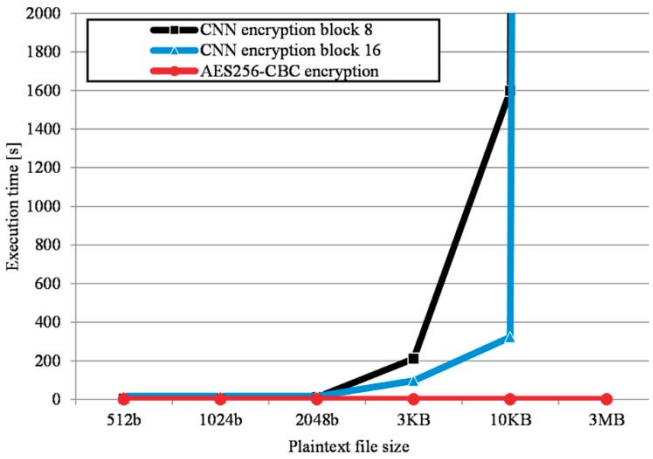

Fig. 9. Encryption execution time

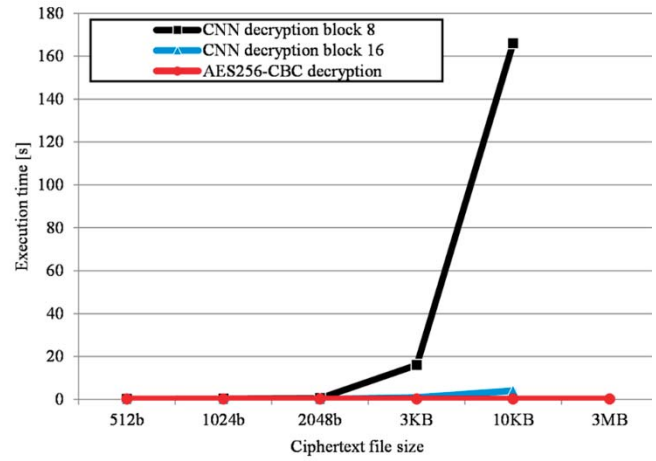

Fig. 10. Decryption execution time

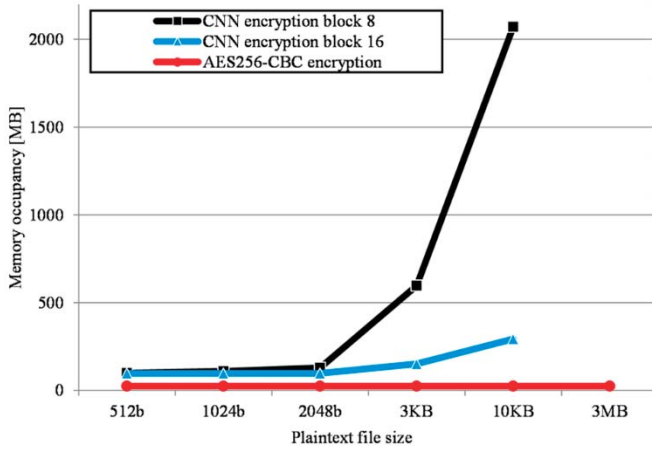

Fig. 11. Encryption memory occupancy

CNN memory occupancy is block size dependent. Because it is used sequential implementation of algorithm, blocks are processed in sequential fashion. Larger block consumes less memory than several small ones (Fig. 11). It also depends on how the underlying APIs handle memory management. Without further optimization it is possible to deplete a system memory. However, our test bed hardware configuration has enough system memory to handle the testing task. AES also works with blocks of small size (128 b) and the memory occupancy takes up only the fraction of our system memory. The difference between CNN and AES memory occupancy follows the same scheme as execution time for encryption and decryption (Fig. 11, Fig. 12). AES absolutely outperformed CNN behind the 2048 bits barrier.

It has to be taken into account that CPU load for sequential CNN and AES encryption and decryption reflects the block size and only employs CPU for processing the blocks in sequential way. It means the file size does not affect CPU load (Fig. 13). It can be possibly altered by using the parallel processing approach.

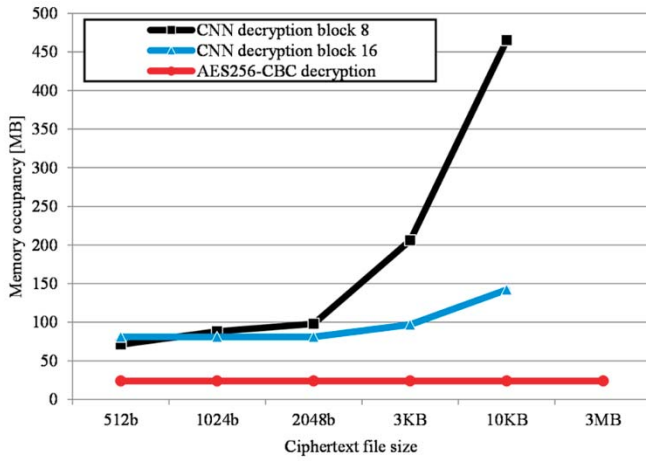

Fig. 12. Decryption memory occupancy

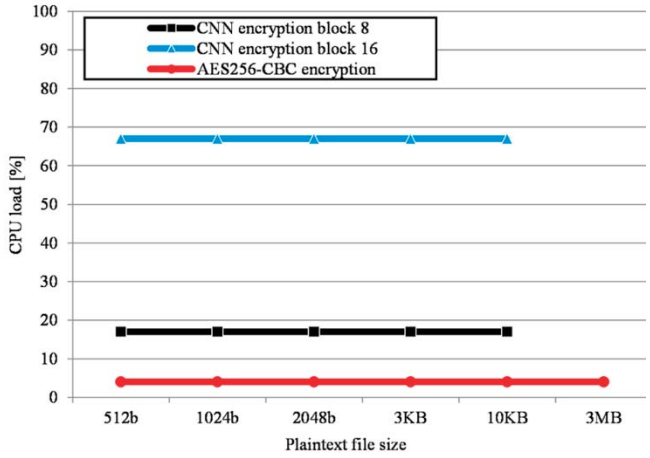

Fig. 13. Encryption CPU load

V. CONCLUSIONS

The purpose of this paper was to compare selected characteristics of two implementations of encryption and decryption algorithms.

AES is well accommodated standard in the cryptography field. Its implementations have been developed and analyzed for many years. From the prospective of cryptanalysis there have been some lists of known attacks published and documented. AES high speed and low memory consumption are well known characteristics of the algorithm and it was also proven by our testing results.

On the other hand, CNN cryptography is the proof of concept with no cryptanalysis. The structure of CNN, the weights and crypto key are required for encryption and decryption. This means that also the process of their exchange has to be considered. Because of it, conclusions concerning security of the algorithm cannot be expressed and further analyses are advised.

It can be stated that CNN has a cryptographic potential using the files with relatively small size. Further CNN structure optimization and cryptanalysis also plays important role in the real-world application.

ACKNOWLEDGMENT

This work was supported by Slovak Scientific Grant Agency VEGA No. 2/0167/16.

REFERENCES

[1] A. Shaw, “Why is Python so slow?”, Hackersnoon, July 2018, https://hackernoon.com/why-is-python-so-slow-e5074b6fe55b

[2] M. Bellaccini, pyAesCrypt, ver. 2, Apache License 2.0, March 2019, https://pypi.org/project/pyAesCrypt/

[3] F. Chollet, Keras: The Python Deep Learning library, 2.2.4, MIT License, March 2019, https://keras.io/

[4] Google Brain, TensorFlow: An end-to-end open source machine learning platform, 1.13.1, Apache License 2.0, March 2019, https://www.tensorflow.org/

[5] Cubic Design, Disk Tools Toolkit, Tahionic, March 2019, https://www.soft.tahionic.com/download-file-generator/index.html

[6] S. Benoit, “Convcrypt: Data encryption using n-dimensional convolutional neural networks.”, MIT License, February 2018, https://github.com/santient/convcrypt

[7] F. Chollet, Keras: Layers, 2.2.4, MIT License, March 2019, https://keras.io/layers/about-keras-layers/

[8] F. Chollet, Keras: Convolutional Layers, 2.2.4, MIT License, March 2019, https://keras.io/layers/convolutional/

[9] NIST, Announcing the Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, United States National Institute of Standards and Technology, 2001, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[10] NIST, Block Cipher Techniques, United States National Institute of Standards and Technology, 2017, https://csrc.nist.gov/Projects/Block-Cipher-Techniques/BCM/Current-Modes

[11] International Organization for Standardization, Information technology -- Security techniques -- Encryption algorithms -- Part 3: Block ciphers, ISO/IEC 18033-3:2010, 2010-12.

[12] Google Brain, TensorFlow: GPU support, 1.13.1, Apache License 2.0, March 2019, https://www.tensorflow.org/install/gpu

[13] F. Chollet, Keras: Layers, 2.2.4, MIT License, March 2019, https://keras.rstudio.com/reference/layer_conv_3d.html

[14] H. Maghrebi, T. Portigliatti, E. Prouff: Breaking Cryptographic Implementations Using Deep Learning Techniques. In: 6th International Conference Proceedings SPACE 2016, Springer, Hyderabad India, December 14-18, 2016, p.3-26, ISBN 978-3-319-49445-6