jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2018, VOL. 64, NO. 2, PP. 173–178
Manuscript received January 24, 2018; revised March, 2018. DOI: 10.24425/119366

Block Cipher Based Public Key Encryption via Indistinguishability Obfuscation

Aleksandra Horubała, Daniel Waszkiewicz, Michał Andrzejczak and Piotr Sapiecha

***Abstract*—The article is devoted to generation techniques of the new public key crypto-systems, which are based on application of indistinguishability obfuscation methods to selected private key crypto-systems. The techniques are applied to symmetric key crypto-system and the target system is asymmetric one. As an input for our approach an implementation of symmetric block cipher with a given private-key is considered. Different obfuscation methods are subjected to processing. The target system would be treated as a public-key for newly created public crypto-system. The approach seems to be interesting from theoretical point of view. Moreover, it can be useful for information protection in a cloud-computing model.**

***Keywords*—indistinguishability obfuscation, public key cryptosystems, homomorphic encryption, cloud-computing, information protection**

I. Introduction

PROGRAM obfuscation is the process of making it unintelligible for user without changing its functionality. There are many code obfuscators that make mechanical changes in the code (for example Stunnix [11], Zelix KlassMaster [12] or Opy [13]). They change special words, order of instructions or precompile programs. But those mechanical changes are not proven to be non invertible. In this article by *program obfuscation* we mean a cryptographic scheme that is computationally secure, what can be proven in mathematical way. Known program obfuscators rely on two cryptograhic primitives: fully homomorphic encryption (proposed by Craig Gentry in 2009 [6]) and multilinear maps (proposed by Sanjam Garg in 2013 [7]). Thanks to homomorphic encryption one can perform operations on ciphertexts, execute programs in their obfuscated (encrypted) form and get the proper result. It is essential property that programs are being executed in their obfuscated form without being decrypted. Usage of multilinear maps guarantee that only entity that can be decrypted is the output of the program.

The security of obfuscation is usually defined as Indistinguishability Obfuscation (IO). Having two programs with the same input-output behaviour we obfuscate one of them and give it to the user. Indistinguishability means that user will not be able to decide which of those programs was obfuscated. Indistinguishability obfuscation is not possible for arbitrary circuits - it has been proven by Boaz Barak in 2011 [2]. We need to specify certain classes of programs that are indistinguishable from each other.

There are many applications of program obfuscation such as protecting intellectual property or providing security for cloud computations. Let us introduce the software producer who wants to protect his intellectual property but also needs to sell his programs. If he sells obfuscated program, everyone can use it but cannot modify it and does not know what they are actually computing (which operations are performed, in which order). The situation is illustrated in the Fig. 1. Growth of interest in cloud computing results in growing need of new secure solutions for clients that want to execute their programs on the external servers. If clients put their programs in the cloud in obfuscated form no one will know what they are computing.

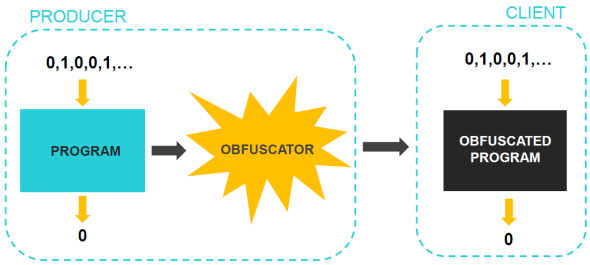

Fig. 1. Application of program obfuscation.

In cryptography obfuscation can be used to accomplish other cryptographic primitives, among others creating asymmetric cryptosystems from symmetric ones, which was the *dream of Diffie and Hellman* in 1980's and is showed in the Fig. 2. In this work we give the first approach to realize this dream. We introduce method of obfuscating block ciphers with many rounds and exchangeable key by implementing changes in Alex Malozemoff software (published on github [9]) allowing to obfuscate a block cipher. We also performed obfuscation of two round of Mini-AES cipher. The computational result are presented at the end of the article.

II. Cryptograhic background

To be obfuscated, program needs to be presented in the special form. Every program executable on the Turing Machine can be described by boolean formula (using Shannon theorem). Then boolean formulas are represented as a product of matrices, in the form of so called matrix branching programs. Matrix branching program is a set of pairs of matrices. Every pair of matrices is connected with one input boolean variable, one matrix from pair for each variable value. Execution of

D. Waszkiewicz, A. Horubała and P. Sapiecha are with Warsaw University of Technology, Poland (e-mail: d.waszkiewicz@tele.pw.edu.pl, aleksandra.horubala@gmail.com, p.sapiecha@krypton-polska.com).

M. Andrzejczak is with Military University of Technology in Warsaw, Poland (e-mail: michal.andrzejczak@wat.edu.pl).

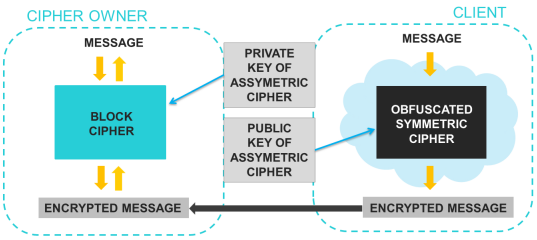

Fig. 2. Creating assymetric cryptograhic scheme from symmetric one.

a program consists of choosing one matrix from each pair (depended on the input value) and taking the product of them. Boolean function can be converted to matrix branching program in many ways for example: by constructing binary decision diagrams (described by Malozemoff and Katz [1]) or by using bilinear or multilinear forms (described by Sahai, Zhandry [10]). When we have our program represented in the matrix form, we need to encrypt it. To do that private and public parameters of obfuscation scheme must be generated. Then matrices are encrypted element by element using private parameters of the scheme. Encrypted matrices with public parameters (that allow to decrypt result of the program) form obfuscated program. Execution of encrypted program consists of multiplying encrypted matrices. In the end result of the multiplication can be decrypted with public parameters of the scheme. Steps described above are presented on the diagram in Fig. 3.

When matrix multiplication is performed two operations are performed on matrix elements: addition and multiplication. Thanks to homomorphic encryption one can perform addition and multiplication on ciphertexts and get proper results after decryption. To ensure that only output of the program can be decrypted we use multilinear maps and graded encoding scheme. Every ciphertext (matrix element) is associated with a level of encryption. After every multiplication of matrix elements the level of result grows. Only the ciphertext with maximal level - received after all planned multiplications - can be decrypted. Idea is illustrated in Fig. 4. Next subsection contains exact description of this process.

A. *Graded encoding scheme*

Fully homomorphic encryption allows to perform any operations on ciphertexts. In obfuscation scheme it is necessary to limit users possibilities, so that he can only perform operations planned by software producer. For security of obfuscated program it is crucial that user cannot examine its properties and draw conclusions about its structure. What is even more important it must be impossible to decrypt any part of a program, but its result must be easily decryptable. To implement these functionalities we need additional structure connected with ciphertexts, such structure can be added by using technique called *encryption over sets*. With every ciphertext is connected a set of variables. Ciphertexts can be added if they are connected with the same set and the sum is also connected with these set. Ciphertexts can be multiplied

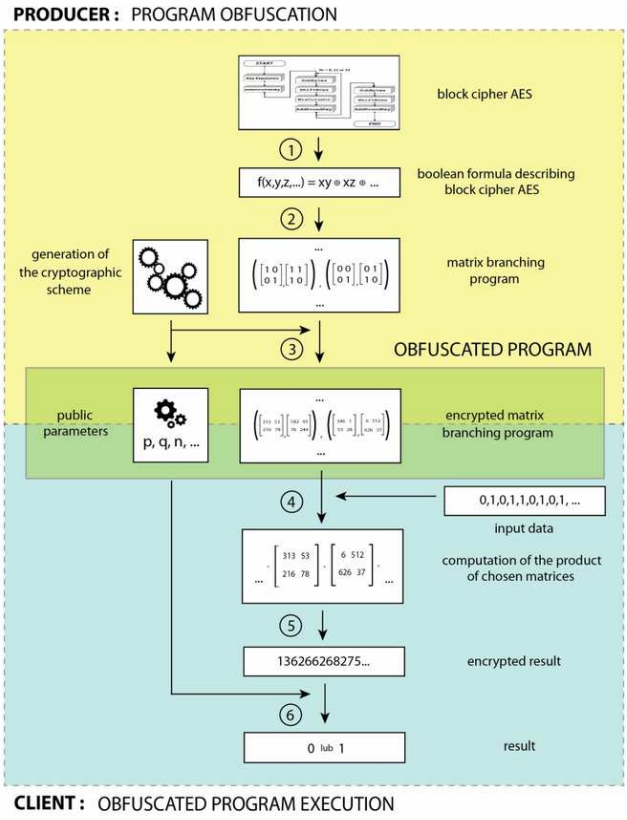

Fig. 3. Obfuscation scheme in general.

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} c_{11} & \cdots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{l1} & \cdots & c_{lp} \end{bmatrix}$$

PRODUCT MATRIX HAS ELEMENTS OF THE FORM:

$$a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + \cdots + a_{1n} \cdot b_{n1}$$

HOMOMORPHIC ENCRYPTION ALLOWS TO ADD AND MULTIPLY CIPHERTEXTS!

GRADED ENCODING ALLOWS TO DECRYPT OUTPUT OF THE PROGRAM!

Fig. 4. Cryptographic primitives used in matrix multiplication.

if their sets are disjoint and their product is connected with the sum of sets connected with factors.

Example. Let $Z = \{z_1, z_2, z_3, z_4\}$ be a finite set. Let s_1 be a homomorphic ciphertext connected with set $\{z_1, z_2\} \in Z$, s_2 - homomorphic ciphertext connected with set $\{z_1, z_2\} \in Z$ and s_3 - homomorphic ciphertext connected with set $\{z_3, z_4\} \in Z$. Ciphertexts over sets are expressed as follows:

$$c_1 = \frac{s_1}{z_1 \cdot z_2},$$

$$c_2 = \frac{s_2}{z_1 \cdot z_2},$$

$$c_3 = \frac{s_3}{z_3 \cdot z_4}.$$

Ciphertexts c_1 and c_2 can be added - denominators are the same, nominators can be added due to homomorphic encryptions. Notice that sum $c_1 + c_2$ is connected with set $\{z_1, z_2\}$:

$$c_1 + c_2 = \frac{s_1}{z_1 \cdot z_2} + \frac{s_2}{z_1 \cdot z_2} = \frac{s_1 + s_2}{z_1 \cdot z_2}.$$

Ciphertexts c_1 and c_3 can be multiplied - their sets are disjoint so after multiplying all factors of the product denominator will be different, nominators can be multiplied due to homomorphic encryption. Product is connected with the sum of sets - $\{z_1, z_2, z_3, z_4\}$:

$$c_1 \cdot c_2 = \frac{s_1}{z_1 \cdot z_2} \cdot \frac{s_3}{z_3 \cdot z_4} = \frac{s_1 \cdot s_2}{z_1 z_2 z_3 z_4}.$$

Now let us consider multiplying matrices presented in Fig. 4. If we connect different sets with elements of different matrices, elements of different matrices can be multiplied and then products of element can be sumed (because they are ciphertext over the same sets). So matrix multiplication can be performed with specified above rules. If we create a public parameter of a system that has product of every element of set Z (here: $z_1 z_2 z_3 z_4$) in the nominator we would be able to shorten denominator of output ciphertext - that is the idea of decryption of the output. Details of this process may be found in papers [1], [4], [7]. The security of these systems is based on the hard problems in multilinear maps: MDL (Multilinear Discrete Logarithm Problem) and MDDH (Multilinear Decisional Diffie Hellman Problem).

B. Choosing homomorphic scheme

As a homomorphic scheme different cryptographic systems can be chosen. There are two main groups of graded encoding systems: CLT13, CLT15 by Coron, Tibbouchi and Lepoint published in papers [4], [5] (based on homomorphic scheme DGVH, based on integer numbers) and GGH13, GGH15 by Garg, Gentry and Halevi published in papers [7], [8] (based on Gentry's homomorphic scheme [6], based on ideal lattices). In our experiment we used CLT13 scheme, but we plan to test other schemes in the future.

C. Choosing matrix branching program form

As we mentioned at the beginning there are many ways to obtain matrix branching program from a boolean formula. We compared to solutions: using BDD trees proposed by Katz, Malozemoff and others in [1] and using multilinear forms proposed by Sahai and Zhandry in [10]. Using BDD trees bigger much matrices are obtain, they are square matrices and their dimension is the number of inputs to the program. Because every matrix has the same size this solution is more secure (the structure of the program is well hidden) but highly inefficient (matrices are bigger than it is needed). Using multilinear forms instead dimension of matrices can be diverse. Smaller sizes of matrices make solution more efficient, but imposes responsibility on system designer - he must examine how different dimensions of matrices narrows down the class of indistiguishable functions. In our work experiments we decided to use multilinear forms because of efficiency of the solution and because we are more concerned about hiding the secret key than the cipher structure (as we will explain in the next section).

III. Design and implementation

To introduce our approach we begin with defining the problem being solved and with description of the tools we have used.

A. Problem definition

The main aim of these work was obfuscation of block cipher with many rounds and exchangeable key. During research we have faced and solved following dilemmas and problems:

- many round problem - how to choose matrices depend on ciphertexts,
- exchangeable key problem - how to store matrices connected with the secret key bits,
- whitebox dilemma - what is the profit of not hiding structure of the cipher (that is often public) but only the secret key,
- matrices storage dilemma - is it more efficient to premultiply some of stored matrices.

We describes these problems and proposed solutions in next subsections.

B. Used tools

For our computations we used Microsoft Corporation cloud server machine with 32 cores and 448 GB RAM. We used implementation of obfuscation created by Alex Malozemoff and published on github [9]. We used variant of this implementation based on CLT13 graded encoding scheme proposed by Coron, Lepoint and Tibouchi [4]. For creating matrix branching program from boolean formula we used method of Sahai and Zhandry based on bilinear forms [10] (which we extended in our implementation to multilinear case).

C. Our approach

To obfuscate Mini-AES cipher firstly we needed to describe the round of it by boolean formulas. Then formulas were minimized and presented in the optimal form. On the base of these formulas matrix branching program was created. See Fig. 5.

1) Many rounds problem: Having obfuscated version of one round of the cipher one faces important question: how to perform the next round not knowing the output from the previous round? Let us notice that decryption of the outputs from the inner rounds cannot be possible - otherwise there would be no point in performing many rounds. But how to choose one matrix of pair depend on ciphertext?

To this problem we propose solution, which introduces one more multiplication to computations. Consider the output bit from previous round - bit w_1 - and consider two variables: $y_1 = w_1$ and $y_2 = encrypted(1) - w_1$. Note that if w_1 is

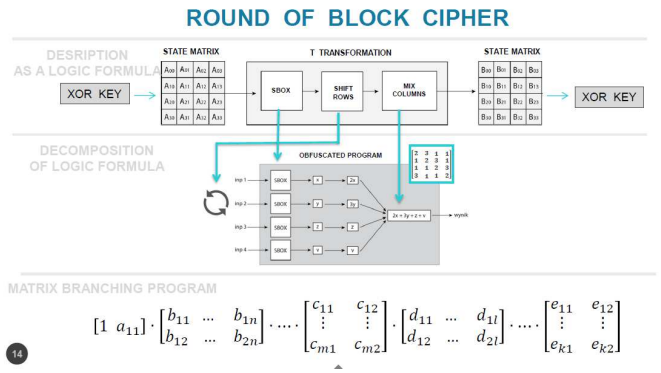

Fig. 5. Obfuscating a round of a block cipher.

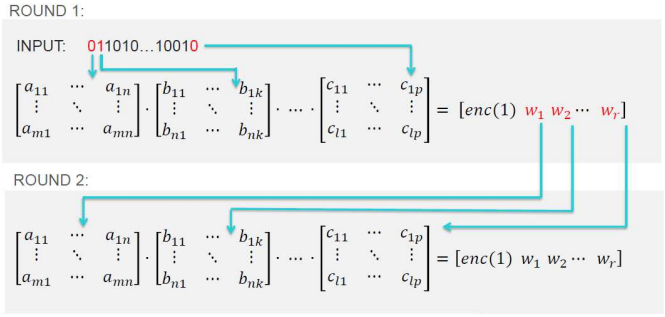

Fig. 6. Problem of many rounds

encrypted one then y_1 is encrypted one and y_2 is encrypted zero. In the other case, when w_1 is encrypted zero, then y_1 is encrypted zero and y_2 is encrypted one. So let us multiply every element of matrix connected with zero with y_2 and every element of matrix connected with one with y_1 and sum these matrices. Then if w_1 is encrypted one then all elements of matrix connected with zero are multiplied with encrypted zero (and by this action deleted) and all elements of matrix connected with one are multiplied by encrypted identity. In result sum of matrices is exactly the matrix connected with one. Analogically when w_1 is encrypted zero matrix connected with one is deleted and matrix connected with zero is chosen. Fig. 7 illustrates this situation.

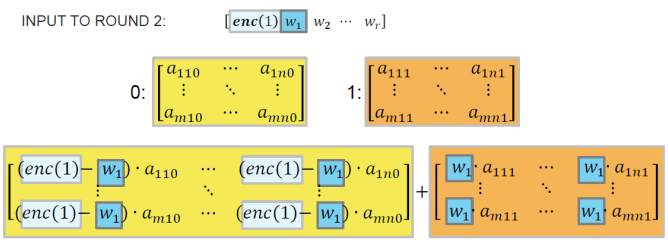

Fig. 7. Choosing matrices conditionally depend on the value of ciphertext

2) Exchangeable key problem.: So far we have not discussed how to perform operations on the secret key. For ciphers publicly used their structure is known and does not need to be hidden. A real secret is a key - sequence of bits that is a private parameter of cryptosystem. In Mini-AES cipher key bits are added to the input of the round by exclusive OR operation. Let us notice that the key is different in every round and it has influence on boolean formulas describing rounds. So we cannot obfuscate one round and use encrypted matrices periodically in successive rounds - we need to store every encrypted round in the memory. This case is illustrated in the Fig. 8.

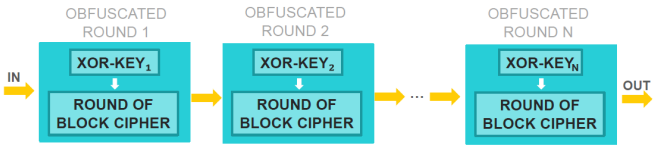

Fig. 8. Key placed inside round block results in different round blocks.

It would more convenient to change only the part of the program connected with the secret key (matrices connected with key-bits). To do that we need to extract key part and treat it as separate block of program. It means that we have to choose inputs to the round conditionally two times: one time in the key-block and one time in round-block, and because of it we need to perform one more multiplication in the round for every matrix. Now we can store much less matrices in memory, but the size of ciphertexts grows (in homomorphic encryption size of ciphertext depends on the number of multiplications). Fig. 9 shows discussed variant.

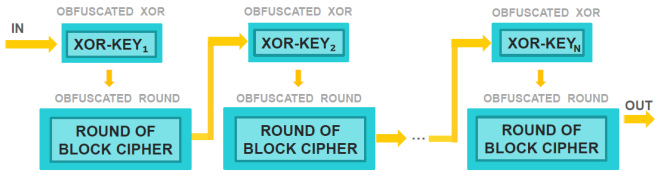

Fig. 9. Separated key block, external from the round description.

Choice of key storage method should be the subject of optimization in every special case. For Mini-AES cipher key storage methods will be compared in Table I and in section IV.

3) Whitebox dilemma: As we have already mentioned Mini-AES cipher is a wide known cipher, its structure is public and does not need to be hidden. In many cases we do not want to reveal structure of a cipher or other obfuscated program, but for now consider the case when only important secret is the key. Presentation of the cipher that hides only the secret key is called *whitebox* implementation of a cipher. Let's have a look at the Mini-AES round structure. One round has 16 inputs and 16 outputs - so in *blackbox* case (when structure needs to be hidden) the obfuscation of a round should consist of 16 pairs of matrices and 16 multiplication must be performed to execute obfuscated version of a round. But the structure of Mini-AES cipher is very symmetrical, we can decompose it to much smaller blocks. It can be noticed (Fig. 10) that it is possible

to separate blocks of the same structure varying only by input bits. This repeatedly occurring parts has only 8 input bits. Of course we give extra information for potential attacker - which bits influence which blocks. But if the structure of a cipher is public that is not a problem. What we gain is performing less multiplications - now only 8 multiplications need to be performed in one round.

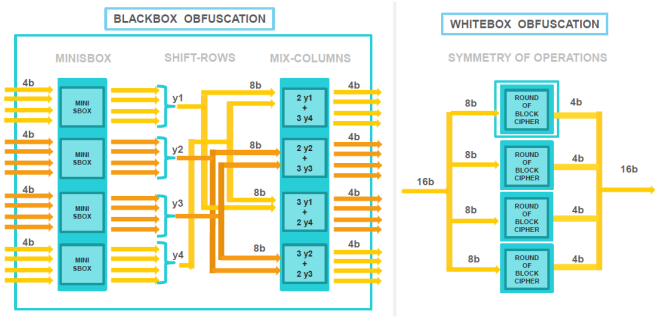

Fig. 10. Whitebox version of a cipher uses symmetry of its structure.

4) Matrix storage dilemma: One more decision needs to be made: how to store encrypted matrices. We can store all encrypted matrices that forms obfuscated program but it is not optimal solution. Better idea is to group matrices in pairs and pre-multiply these pairs (proposed by Sahai). Now we need to choose one of four matrices depend on four possible inputs (00, 01, 10, 11) instead of choosing two times one of the pair. It is better because we have the same number of matrices but we perform one less multiplication. But why group matrices in pairs? Maybe it would be better to group it in fours or eights. The bigger are matrices groups the more matrices need to be stored, but less multiplications need to be performed (because we pre-multiply whole parts of our obfuscated program). Fig. 11 shows the numbers.

$$[M_{x_0=0}], [M_{x_0=1}]; [M_{x_1=0}], [M_{x_1=1}]; \ldots; [M_{x_7=0}], [M_{x_7=1}] \qquad 2^1 \cdot 8 = 16$$

$$[M_{x_0=0,\,x_1=0}], [M_{x_0=1,\,x_1=0}], [M_{x_0=0,\,x_1=1}], [M_{x_0=1,\,x_1=1}]; \ldots \qquad 2^2 \cdot 4 = 16$$

$$\ldots \qquad \ldots$$

$$[M_{x_0=0,\,x_1=0,\,x_2=0,\,x_3=0,\,x_4=0,\,x_5=0,\,x_6=0,\,x_7=0}], \ldots \qquad 2^8 \cdot 1 = 256$$

Fig. 11. Different ways to store matrix branching programs.

In our implementation of Mini-AES obfuscation we decided to store whole round pre-multiplied. For example in the whitebox case the output of a round depends on 8 input variables. If we pre-multiply a round we need to choose the output matrix depend on 8 variables, so one of $2^8 = 256$ matrices which all need to be stored in memory. Thanks to that we do not need to perform any matrix multiplication during a round (but we still need to perform conditional multiplication to choose matrices depend on ciphertexts). We have chosen this case because time of computation is crucial for us.

For Mini-AES cipher we compared multiplicative complexity for cases discussed in this sections. Results can be seen in the Table I.

TABLE I
COMPARISON OF MULTIPLICATIVE COMPLEXITY IN OBFUSCATION METHODS FOR MINI-AES CIPHER

Obfuscation method	whitebox internal key	whitebox external key	blackbox external key
one round	0	9	17
two rounds	9	81	289
three rounds	73	657	4641

Multiplicative complexity grows rapidly with the number of rounds. It is because we need to choose every bit of a round conditionally depend on output bits from previous rounds.

IV. EXPERIMENT

All experiments where performed on Microsoft Azure cloud server machine with 32 cores and 448 GM RAM memory. We have obfuscated two rounds of Mini-AES cipher with pre-compiled rounds on the level of security 2^6. We compared three obfuscation variants: whitebox version with internal key, whitebox version with external key and blackbox version with external key. Results can be seen on Table II.

TABLE II
RESULTS OF OBFUSCATION OF MINI-AES CIPHER.

2 rounds of Mini-AES cipher	whitebox internal key	whitebox external key	blackbox external key
size of obfuscated program [kB]	11 000	557 000	$\approx 35.7 \cdot 2^{16}$
evaluation time [s]	11	3557	$\approx 13466 \cdot 2^{16}$
multiplicative complexity	9	81	289

Computation were performed on low security level (2^6) to check correctness of reasoning and implementation. We also examined how security parameter influences sizes of ciphertext (and in consequence obfuscated program). We decide to carry out tests for multiplicative complexity 8. It is shown in Table III.

TABLE III
SIZE OF CIPHERTEXTS DEPEND ON THE SECURITY PARAMETER FOR 8 MULTIPLICATIONS PERFORMED

security level	2^8	2^{16}	2^{24}	2^{32}
size of single ciphertext [kB]	12.6	58.9	145.1	275.0
speed of growth	–	4.67	2.44	1.88

We have also tested influence of multiplicative complexity on the program size. To carry out test we set security parameter to 2^{16}. We present results in Table IV.

TABLE IV
SIZE OF CIPHERTEXTS DEPEND ON MULTIPLICATIVE COMPLEXITY FOR SECURITY PARAMETER 2^{16}

number of multiplications	4	8	16	32
size of single ciphertext [kB]	58.9	189.3	393.6	672.0
speed of growth	–	3.21	2.07	1.7

V. CONCLUSION

We performed obfuscation on many-round block cipher Mini-AES and proven that it is possible. High complexity of obfuscation scheme and long time of computation is caused by homomorhic encryption scheme. Future work will focus on changing homomorphic scheme to more efficient one (testing new solutions based on lattices and LWE scheme). If more efficient homomorphic scheme will be adopted to obfuscation techniques methods proposed in this article will allow to obfuscate practically used block ciphers like AES cipher.

ACKNOWLEDGMENT

The authors would like to thank Microsoft Corporation for the possibility to use Azure cloud machine with 32 cores and 448 GB RAM.

REFERENCES

[1] D. Apon, Y. Huang, J.Katz and A. Malozemoff, "Implementing cryptographic program obfusctaion", *Cryptology ePrint Archive*, 2014.
[2] B. Barak, O. Goldreich, R.Impagliazzo, S.Rudich, A.Sahai, S. Vadhan and K.Yang, "On the (im)possibility of obfuscating programs", Advences in Cryptology - EUROCRYPT 2011, 2011.
[3] G. Boole, "An investigation of the laws of thought: On which are founded the mathematical theories of logic and probabilities", 1854.
[4] J-S. Coron, T. Lepoint and M. Tibouchu, "Practical multilinear maps over the integers", Advances in Cryptology - CRYPTO 2013, vol. 1, 8042, pp. 476-493, 2013.
[5] J-S. Coron, T. Lepoint and M. Tibouchu, "New multilinear maps over the integers", Cryptology ePrint Archive, Report 2015/162, 2015.
[6] C. Gentry, "A fully homomorphic encryption scheme", PhD thesis, Stanford University, 2009,
[7] S. Garg, C. Gentry and S. Halevi, "Candidate multilinear maps from ideal lattices", Advences in Cryptology - EUROCRYPT 2013, 2013, 7881 of Lecture Notes in Computer Science pp. 1-17],
[8] C. Gentry, S. Gorbunov, and S. Halevi, "Graph-induced multilinear maps from lattices", TCC 2015, Part II, volume 9015 of LNCS, pp. 498–527. Springer, 2015,
[9] A. Malozemoff, Implementation of program obfuscation published on github: https://github.com/amaloz/obfuscation. Accesed: 2017-09-20.
[10] A. Sahai and M. Zhandry, "Obfuscating low-rank matrix branching programs", Cryptology ePrint Archive, Report 2014/772, 2014.
[11] Stunnix - a tool designed for obfuscation of C/C++ programs. http://stunnix.com/prod/cxxo. Accesed: 2018-01-20.
[12] Zelix KlassMaster - a tool designed for obfuscation of JAVA programs. http://stunnix.com/prod/cxxo. Accesed: 2018-01-20.
[13] Opy - a tool designed for obfuscation of PYTHON programs. https://github.com/QQuick/Opy. Accesed: 2018-01-20.