UDC 004.056.53

Kostiantyn V. Zashcholkin[1], Candidate of Technical Sciences, Associate Professor, Department of Computer Intellectual Systems and Networks, E-mail: const-z@te.net.ua, ORCID ID: 0000-0003-0427-9005
Oleksandr V. Drozd[1], Doctor of Technical Sciences, Professor, Department of Computer Intellectual Systems and Networks, E-mail: drozd@ukr.net, ORCID ID: 0000-0003-2191-6758
Olena M. Ivanova[1], Senior Lecturer, Department of Computer Systems, E-mail: en.ivanova.ua@gmail.com, ORCID ID: 0000-0002-4743-6931
Yulian Y. Sulima[2], Candidate of Technical Sciences, Head of the Computer Systems Department, E-mail: mr_lemur@ukr.net, ORCID ID: 0000-0003-3986-7296
[1]Odessa National Polytechnic University, Shevchenko Avenue, 1, Odessa, Ukraine
[2]Odessa Technical College of the Odessa National Academy of Food Technologies, Balkovskaya st., 54, Odessa, Ukraine

COMPOSITIONAL METHOD OF FPGA PROGRAM CODE INTEGRITY MONITORING BASED ON THE USAGE OF DIGITAL WATERMARKS

Abstract. *The paper considers a problem of provision of the programmable component integrity of computer systems. First the basic stages of the programmable components life cycle are presented. The authors note that the program code modification gives the opportunity to maliciously violate its (program code) integrity. The traditional methods of integrity modification are based on the usage of monitoring hash sums. However the main disadvantage of the traditional methods is that they are not able to hide the fact of integrity monitoring execution itself. This fact cannot be hidden and becomes obvious. Even under the conditions of extra encrypting of monitoring hash sum the very existence of it demonstrates that the integrity monitoring is carried out. The paper presents a class of methods which offer the hash sum embedding into program code in the form of digital watermark. This class of methods is considered with reference to monitoring the chip FPGA (Field Programmable Gate Array) program code integrity. For embedding the features of LUT-oriented FPGA architecture are used. The monitoring digital watermark embedding is performed due to the usage of equivalent program codes conversions in a set of LUT-units included in FPGA. The peculiarities of the digital watermark embedding are as follows – such kind of embedding does not change the program code size and does not modify the chip FPGA operation. As a result of embedding it is impossible to distinguish the monitoring hash sum in the program code in an evident way. The extraction of digital watermark including hash sum can be carried out only in the presence of special steganographic key, which sets the rules of watermark location in the FPGA program code space. In the given paper a compositional method of embedding the monitoring digital watermark into the FPGA program code is offered. The method combines the features of ones providing the recovery of initial program code state and the ones (methods), which implement the embedding on the basis of syndrome decoding. The proposed method incorporates the useful features of two classes of methods mentioned above and serves to reduce the amount of equivalent conversions applied to the program code in the course of the digital watermark embedding. This demonstrates the advantage of the proposed method as compared to the base ones of the digital watermark embedding in the FPGA program code. The description and results of experimental research of the proposed method are also presented.*

Keywords: *integrity monitoring of the program code; programmable hardware components; FPGA; LUT-oriented architecture; monitoring hash sum; digital watermark, steganographic approach to integrity monitoring*

Introduction

Among the hardware components used to build digital computer systems, two large classes can be separate: a) integrated circuits with hard logic of functioning; b) program-controlled (programmable) integrated circuits.

The first of these classes is formed by so-called ASIC (Application Specific Integrated Circuit) chips [1]. These integrated circuits are focused on solving one specific computational or control task. Their functioning does not change during the life cycle of the system in which they are included.

Integrated circuits belonging to the class of program-controlled [2], on the contrary, allow to customize (program) them to solve an arbitrary range of tasks. The operation of the integral circuits of this class can potentially be changed at any stage of their life cycle.

Programmable hardware components are not initially configured to solve any particular task. In the process of designing a computer system, a program (program code) is created for such components, which custom them to solve the required task. This program is placed in the memory of the programmable component, thereby setting it up for a given functioning.

The functioning of programmable components can be modified at all stages of their life cycle. This

 DOI://10.15276/aait.02.2019.5 ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

modification is carried out by changing the program code of the components.

A typical (but not exhaustive) set of reasons leading to the need to change the functioning of programmable components is:

a) detection of errors in the initial version of the program code;

b) the need to optimize the system at a certain stage of its operation;

c) planned upgrade of the program code;

d) the need to adapt the system to changes in conditions determined by the environment external to it.

The ability to modify the program code of programmable components creates vulnerability in ensuring the integrity of the system. For components with hard logic (ASIC), potential integrity violations are possible mainly through physical intervention in their structure. For programmable components, the possibility of integrity violations at the level of program code arises. Integrity is further understood as the ability of the system to exclude unforeseen changes to the system and the services it provides [3].

The prerequisites for the occurrence of this vulnerability are that:

a) there is (can be used each of legitimately and not illegitimate) the technical possibility of modifying the program code, which leads to a change in the operation of the components;

b) provided for legitimate (made by the developer or the person operating the system) changes in the program code.

Integrity violation of the program code, caused by both the action of natural forces and malicious acts of humans is an extremely dangerous phenomenon that can lead to technological disasters and financial losses [4].

So programmable components are part of the systems for managing high-risk technical objects [5] (in safety-critical systems) [6]: energy facilities, chemical industry, aviation objects and high-speed ground transportation. Disruption of the functioning of these objects can lead to unacceptable consequences.

An important area of application of programmable components, for which functional safety requirements are one of the main development factors, is medical equipment (including wearable and implantable in the human body) [7]. Violation of the integrity of the software code for components of such equipment at the very least degrades the quality of life of its users, and, at most, can affect the vital functions of the body of users.

Also, programmable components are part of systems that are not characterized by a critical area of application, but are used massively [8]. The improper functioning of such systems can lead to financial and reputational losses, both for companies producing systems and for end users.

The presence these factors makes ensuring the integrity of the program code for programmable components of one the priorities in the process of creating safe systems.

One of the types of programmable components of modern computer systems is FPGA (Field Programmable Gate Array) chips [9], [10]. These integrated circuits differ from microprocessors and microcontrollers in the way they change functioning. FPGA chips have a variable (programmable) structure that can be modified by a program code to solve a specific task.

FPGAs are a matrix of programmable elementary units of both universal and specialized purposes. Each of these units is configured by the FPGA program code to implement a specific function. The connections of the units between themselves and with the external outputs of the chip are also determined by the program code. Thus, unlike microprocessors and microcontrollers, FPGAs can change their functioning by changing the internal structure and functions of the elements of this structure. This makes it possible to ensure the distribution (paralleling) of the problem solving process in the FPGA chip space.

Due to the above features, FPGA chips have greater performance, as compared to the other, frequently used the type of programmable components – microprocessors and microcontrollers.

Typically, FPGA chips are used in cases:

a) the specificity of the computing tasks that need to be solved is that microprocessors cannot be used for performance reasons;

b) this requires the implementation of the solution of the problem on programmable components (it is assumed that at further stages of the life cycle of the system, modification of its functioning will be required).

The problem of ensuring and monitoring the integrity of the software code for such programmable components of computer systems as microprocessors and microcontrollers is worked out much deeper than for FPGAs. This is due to the earlier occurrence of microprocessors and microcontrollers and, accordingly, a longer stage of studying the problem of the integrity of their program code. Significant differences in the principles of operation and programming of these two classes of programmable components (microprocessors and FPGA) do not allow extending the methods used to monitoring integrity in one class to another class. Therefore, the problem of ensuring the integrity of the FPGA software code is currently significant.

Analysis of recent research and publications

Currently, the most effective and frequently used mechanism underlying integrity monitoring is the use of hash sums [11]. In contrast to the checksums used in online-testing of computer systems [12], hash sums have a number of additional properties.

So, the hash functions that help to calculate monitoring hash sums provide the properties used in the integrity monitoring process, among which we can highlight:

a) non-invertibility – the extreme computational complexity of obtaining an argument of a hash function by its value;

b) a significant change in the hash sum with a slight change in the data block for which this hash sum was calculated;

c) the impossibility, knowing the argument and the corresponding hash sum, to find another argument that gives the same hash sum.

The main approach to the monitoring of program code is to double calculation of the hash sum in the framework of following base procedure [13] (Fig. 1).

1) At the stage of preparing the program code for integrity monitoring, the hash sum H of the program code is calculated. This hash sum is further considered to be a standard. The standard hash sum H should be available at the time of integrity monitoring of the program code. To ensure the availability of the hash sum, H is attached to the information object of the program code or is in some way associated with it.

2) Immediately at the moment of performing the integrity check, the hash sum H^* is again calculated for the information object of the program code. The calculated hash sum H^* is compared with the standard hash sum H. Based on the comparison of the specified hash sums, it is decided whether the integrity of the program code is violation. Any change to the information object of the program code or / and the standard hash sum leads to a mismatch between the hash sums H^* and H.

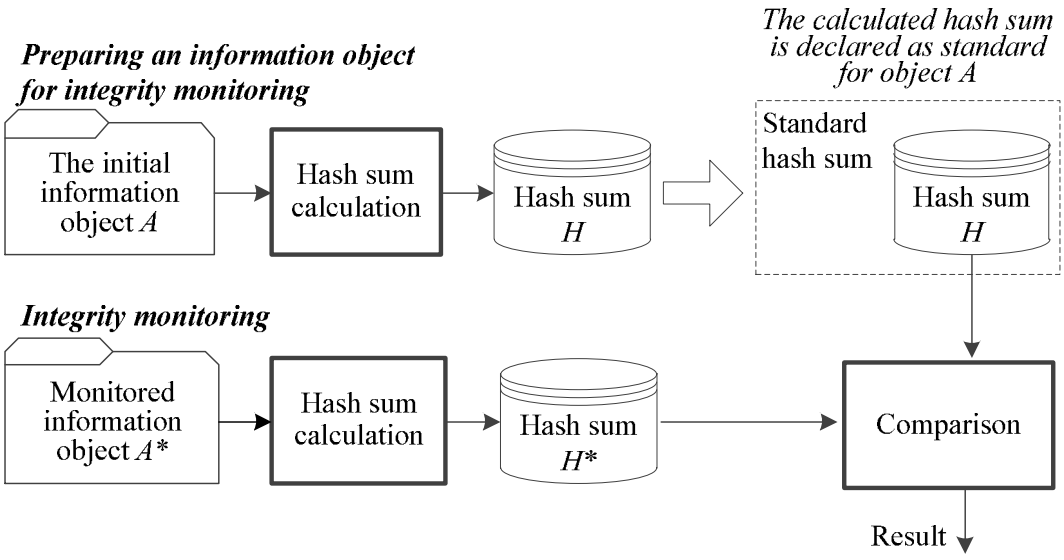

Fig. 1. Basic integrity monitoring procedure

Possible options in the case of the mismatch of H^* and H hash sums (not an exhaustive list): stopping the system; overwriting the initial program code from a reliable source; switch to backup system.

Depending on the cause of the integrity violation, the specificity of individual details for the integrity ensuring process arises.

So for integrity monitoring, aimed at countering the expected violation caused by malicious interference in the program code, the essential aspects are:

a) the storage location of the standard hash sum;

b) the method of storing the standard hash sum and the technique for accessing it;

c) the extent of openness for fact of the integrity monitoring.

A known approach to integrity monitoring, which involves storing the monitoring hash sum in open form in a separate information object from the program code [14]. This approach is acceptable when processing the expected integrity violation as a result of natural phenomena. However, for the case of malicious interference in program code, the appropriateness of using this approach is questionable. The reason for this is the openness of the storage of the standard hash sum, which creates a potential for falsification.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

The traditional integrity monitoring approach described above (based on the double calculation of the hash sum and its storage in open form) is used to monitoring the integrity of the program code, both microprocessors and FPGA chips. However, this approach has a significant disadvantage. This disadvantage is due to two interrelated factors:

Factor 1: the standard hash sum is stored in such a way that it is available for reading, analysis, and possible falsification.

Factor 2: the fact of performing integrity monitoring is open to an outside observer.

Despite the fact that hash functions have the property of non-invertibility, access to the value of the standard hash sum creates the possibility of manipulating the integrity monitoring process. A whole range of methods has been developed to accelerate the search for the preimage for the hash sum: search by rainbow tables [15]; dictionary methods [16]; methods focused on frequency analysis [17] and various compositions of these methods. However, the main problem generated by the availability of a hash sum is the ability to use insider manipulation methods [18]. Within these methods, the capabilities of persons who have access to the processes of monitoring the integrity or legal modification of program code are used. These persons can potentially perform falsification of the standard hash sum: replace it with the hash sum calculated for the object of the program code that has been illegitimate changes.

There are known approaches to minimizing the influence of the first factor mentioned above. In particular, it is proposed to store the standard hash sum not in the clear, but to pre-encrypt it using the agreed cryptographic method [19], [20]. In this case, to access the value of this hash sum, it is necessary to decrypt it (Fig. 2). This requires information on the encryption method used and the encryption key. When using this approach, hash sum falsification requires an additional procedure — obtaining the initial hash sum value. In the absence of a encryption key, this procedure is extremely computationally complex. However, this approach inherits the disadvantages of the basic approach. In the case of using such an approach, the fact that integrity monitoring is performed remains open, which makes it possible to apply cipher-hacking techniques to falsify the standard hash sum. Also, this approach does not eliminate the possibility of insider manipulation of monitoring information. It only narrows the circle of persons capable of performing such a manipulation.

Another approach [21] is known to eliminate the influence of the first of the above factors. The standard hash sums for the information objects of the program code are not distributed together with the information objects themselves (they are not attached to them), but are stored in a certain centralized database of the subject of control. The main disadvantage of this approach is the difficulty of protecting this database from information leaks. Mass information leaks from such databases are very frequent [22]. Such leaks put at threat all integrity monitoring systems that store standard hash sums in compromised databases. Similar to the previous approach, this approach does not eliminate the possibility of insider manipulation of standard hash sums.

The approach based on the application of the theory of digital steganography [23] eliminates the indicated disadvantages of traditional approaches. Steganography is a field in the theory of information security, based on information hiding. The main mechanism of steganography is the hidden embedding of information objects of one type into information objects of another type. Digital steganography has various practical applications, the main of which are: hidden data transmission, hidden data marking, and hidden tracking of data distribution paths. To solve the problem of integrity monitoring, one of the steganographic-oriented technologies is used – the technology of digital watermarks [24].

Digital watermark is used as information medium [25] of the standard hash sum within the framework of a steganographic approach to integrity monitoring. In such a case a digital watermark is a data block that contains monitoring hash sum and optionally additional utility information fields. This digital watermark is embedded in the information object of the program code in such a way that the fact of this embedding becomes hidden from an outside observer. The fact that integrity monitoring is performed is also hidden. In this case, the standard hash sum is distributed over the information object of the program code in such a way that it (hash sum) can be accessed only with a special steganographic key (stego-key) [26].

Thus, when using the steganographic approach, the control hash sum is not attached to the information object of the program code, but is embedded in it in the form of a digital watermark.

The advantages of this approach are that:

1) a digital watermark does not increase the volume of an information object by the size of the hash sum;

2) there is no possibility for an external observer to identify the fact that the program code is monitored, as well as to identify parts of the information object that contain program code and parts that contain the hash sum;

3) the embedding of a digital watermark in the program code is performed in such a way that the operation of the FPGA chip does not change.

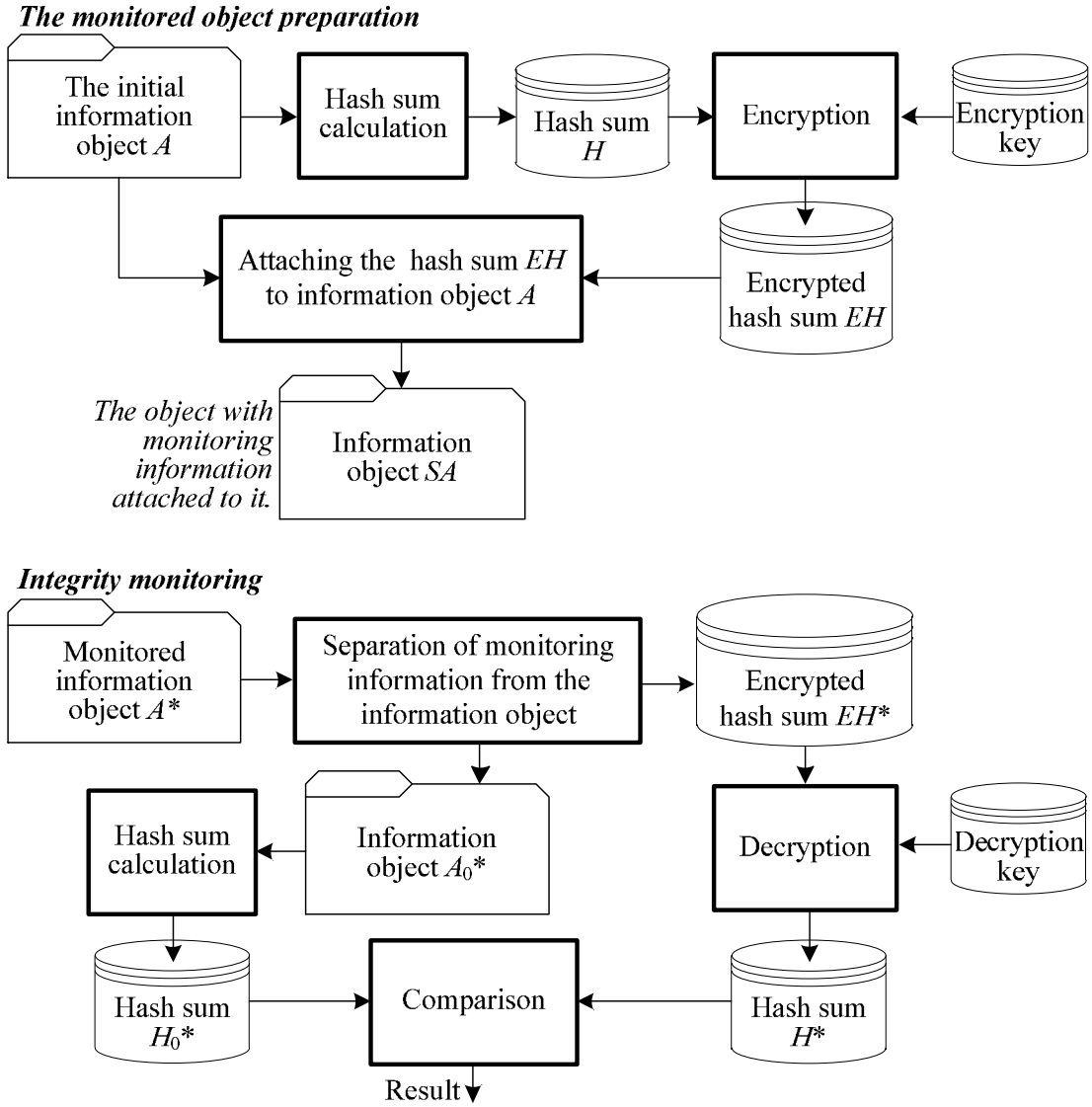

Fig. 2. Integrity monitoring procedure with encryption of standard hash sum

Methods [27], [28] that implement the concept of using digital watermarks to integrity monitoring the program code of FPGA-based components use the program codes of the LUT (Look Up Table) units [29], [30] as the information medium in which the digital watermark is embedding. LUT units (Fig. 3) are the most mass elementary calculating units of FPGA. Their number in modern FPGA chips can vary from tens of thousands to several millions.

The LUT unit is a programmable module for calculating an n-arguments (usually from 4 to 8) logical function. Each LUT unit is configured to implement a specific logic function using 2n-bit program code. In accordance with the provisions of the methods [27], [28], the set of program codes of the LUT units is used as the information medium for embedding a monitoring digital watermark.

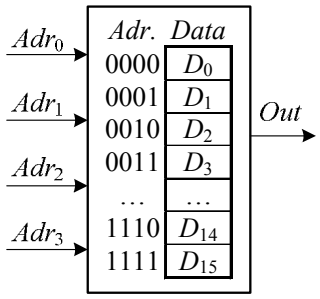

Fig. 3. Structure of the 4-input LUT unit of FPGA chip

Embedding within the framework of these methods is performed using equivalent conversions [31], [32], which do not change the logic functions implemented by the LUT units and do not affect the operation of the FPGA chip. Methods [27], [28] determine that for embedding a digital watermark from general set of LUT units an ordered set of units is

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

formed, each element of which uniquely corresponds to a bit of a digital watermark and is used to store this bit. The specified ordered set is called the steganographic path (stego-path) in the space of a LUT-oriented information object. In this case, the stego-path formation rule is a part of the stego-key – a set of secret information that defines the formal rules for extracting a digital watermark from the program codes of the LUT units.

In Fig. 4 shows a diagram of the steganographic procedure for preparing an information object of the FPGA program code for integrity monitoring. In accordance with this scheme, the considered methods [27], [28] are functioning. This diagram describes the formation and use of a digital watermark (denoted as a DWM in the diagram). A digital watermark includes a mandatory component – a standard hash sum H of the code and two optional components: the usage monitoring tag (marker) and the legitimacy monitoring tag of the program code information object. Optional components of a digital watermark provide a solution to specific problems for the protection of FPGA software code. These solutions can be implemented along with integrity monitoring by storing special tags in a digital watermark.

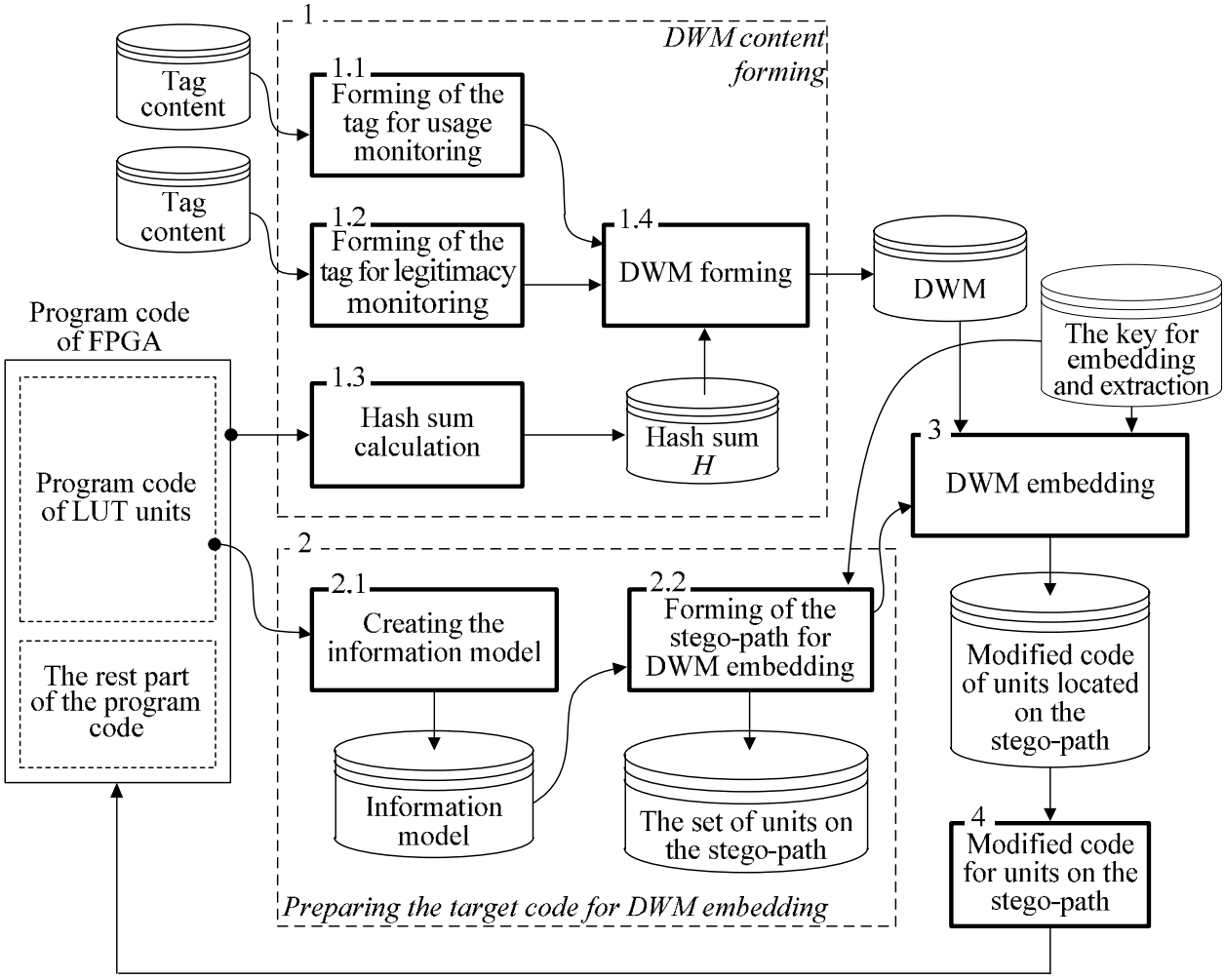

Fig. 4. Steganographic procedure for preparing an information object for integrity monitoring

The use monitoring tag is designed to track the distribution of the program code. This procedure is necessary to identify the point of leakage and the illegal distribution of FPGA code. The legitimacy monitoring tag can be used to confirm the legality or authenticity of the program code.

Programmable LUT units are the most mass in the FPGA structure. Because of this, the program code of these units makes up the largest part of the entire program code FPGA. It is for the program code of LUT units that the equivalent conversions [31], [32] are proposed, which are used in the methods [27], [28] as the basis for the embedding of a digital watermark. In addition to LUT units, FPGA contains specialized programmable units: lumped memory units, multiplication units, I/O units, etc. The program code of these units is not used in the process of embedding a digital watermark. In accor-

dance with the considered diagram (Fig. 4), the standard hash sum is calculated for the entire FPGA program code, and the destination place of the monitoring digital watermark is the program code of the LUT units.

However, the use of the steganographic approach imposes an additional requirement on the integrity monitoring method. This requirement is the need to restore the initial state of the information object of the program code. Indeed, if a standard hash sum is calculated for the program code, then any change in the program code will be manifested at the time the integrity monitoring is performed. When embedding (even performed by equivalent conversions) monitoring watermark in the program code, its (code) integrity is violated. Thus, there is a contradiction between the method of storing the standard hash sum and the main method of integrity monitoring.

To resolve this contradiction, the following procedure is usually used:

1) the standard hash sum for the information object of the program code is calculated;

2) the digital watermark is formed, which includes the standard hash sum;

3) the state that the information object had at the time of calculating the standard hash sum (the initial state of the information object) is in some way stored. Moreover, the storing is done in such a way that only an information object and a stego-key are required for monitoring. Based on this, the initial state can be stored as part of the same digital watermark in which the standard hash sum is placed;

4) the monitoring digital watermark is embedded in the information object of the program code;

5) at the time of performing the integrity monitoring, the digital watermark is extracted and at the same time the initial state of the information object is recovered (the state for which the standard hash sum was calculated).

It is a pair of actions “storing - recovering” of the initial state of the information object that eliminates this contradiction.

There is a method [33] that provides this recovery procedure in order to monitoring the integrity of the FPGA program code. This method is based on the Friedrich method [34], [35] proposed for embedding digital watermarks in multimedia information objects. The disadvantage of the Friedrich method is that it requires a relatively large number of changes made in the program codes. These changes are performed using equivalent conversions; do not change the size of the program code and the operation of the device. However, any massive code changes (even equivalent ones) could potentially be used in the future to compromise the method and search for vulnerabilities in it. Therefore, the task of minimizing the changes in the program code resulting from the embedding is very important and significant.

The steganographic method F5 [36], [37] is known, which is characterized by a small value of the ratio of information object bits number changed during the embedding to the total number of bits. The F5 method is based on the joint use of the theory of steganography and the theory of error check coding [38]. This method is intended only for use in relation to multimedia information objects: raster images, digital video and sound. There is an adaptation of this method to the environment of LUT-oriented information objects (to embedding digital watermarks in the FPGA program code) [39]. However, neither the basic nor the FPGA-oriented methods have the ability to ensure the recovery of the initial state of an information object in the process of extracting a digital watermark.

Thus, the method [33] and the methods derived from it have a property that is useful for the task of monitoring integrity – they provide the ability to recovery the initial state of an information object in the process of extracting a digital watermark. However, the method [33] in its practical applications shows a relatively large value of the ratio of the number of modified bits of the program code to their total number.

On the other hand, the method [39] and methods derived from it, on average, gives fewer modified code bits, but does not support the ability to recovery the initial state of an information object.

Under these conditions, we consider significant the task of obtaining a method that combines the useful properties of the methods [33] and [39] in solving the problem of monitoring the integrity for the FPGA program code.

The goal and objectives of the work

The goal of this work is to reduce the number of modifiable bits of the FPGA program code in the process of monitoring integrity by combining digital watermark embedding methods that:

a) have the property of recovering the initial state for the program code of information object;

b) generate a small (relative to other similar methods) number of changes for bits of the program code.

To achieve this goal in the work the *following objectives are set*:

– to formalize the method that allows to perform the recovering of the initial state for the information object and at the same time provides the

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

number of changes for program code bits at the method [39] level;

– develop a procedure for applying the proposed method in the process of the FPGA program code integrity monitoring;

– perform an experimental comparison of the proposed method with known methods and draw conclusions about the appropriateness of its use.

Main part of the work

A compositional integrity monitoring method is proposed that combines the property of ensuring the recovering of the initial state for an information object with the property (characteristic for methods based on the use of syndrome decoding) of the minimal change for bits of the FPGA program code. The proposed method is based on the following six key principles.

The first principle of the method. The information medium of a digital watermark is the stego-key-specified bits of the LUT units, which are along the stego-path (target bits for embedding). In the following, the ordered sequence of such bits will be denoted by $M_{SPath} = <m_1, m_2, \dots, m_p>$. Each of the $m_i \in M_{SPath}$ bits can be inverted by equivalent conversions [31], [32] used in the methods [27], [28].

The second principle of the method. To embed the bits of a digital watermark, a change is made to the syndromes that are associated with *n*-bit fragments (parameter n is set by the description of the error-correcting code used in the embedding process) of the M_{SPath} sequence, using the error-correcting coding method specified by the steganographic key.

The third principle of the method. Storing of the initial state of the information object of the program code is achieved by lossless compression and embedding the compression results as part of a digital watermark (similarly to how this is implemented in the Friedrich method [34], [35] and the FPGA-oriented method [33]).

The procedure for embedding a digital watermark involves changing the values of the bits in the M_{SPath} sequence. Because of this, within the framework of this provision, it is ensured that the initial state of this part of the FPGA program code is maintained.

The fourth principle of the method is that the digital watermark being embedded into the program code is formed as a set:

a) compressed initial state of the M_{SPath} binary sequence;

b) the monitoring part containing the standard hash sum;

c) optional additional information fields.

The fifth principle is that the initial state of the M_{SPath} bit sequence is stored by changing the M_{SPath} bits, which (changing) leads to the replacement of the original *n*-bit M_{SPath} fragment syndromes by the *n*-bit fragments obtained after compression.

The sixth principle of the method is that the modification of the syndromes is performed by equivalent conversions [31], [32], used in the methods [27], [28].

Based on the presented basic theoretical principles, the following sequence of actions is proposed, which leads to the embedding of the monitoring digital watermark in the FPGA program code.

Stage 1. Stego-path is formed in the space of the program code of LUT units. To perform this stage, we use the stego-path formation procedure proposed in [40]. The result of this stage is an ordered sequence $M_{SPath} = <m_1, m_2, \dots, m_p>$ bits of the program codes of LUT units located on the stego-path. The M_{SPath} sequence bits are information mediums of the monitoring digital watermark being implemented in the program code.

Stage 2. The M_{SPath} sequence is divided into n-bit fragments.

Let the stego-key *Skey* as one of the components contain the description of the error-correcting code $ECode \in Skey$ given by three parameters: *n*, *k*, *H*, where *n*, *k* are the parameters of the (*n*, *k*)-code, *n* is the length of the code word, k is the number information bits in the code word; *H* – some rule for performing syndrome decoding. Further, for simplicity, the rule *H* will be specified by the check matrix of the block code. However, in the general case, this rule can be specified in any other way of describe the procedure for obtaining the error syndrome for error-correcting coding.

The sequence of M_{SPath} binary bits is represented as a sequence of concatenated fragments:

$$M_{SPath} = M_1 \mid M_2 \mid \dots \mid M_q, \quad (1)$$

where, M_i is a fragment of the M_{SPath} sequence, with a length of *n* bits (the *n* parameter is specified by the description of the used error-correcting code $Ecode \in Skey$);

«|» – designation for the operation of concatenation of the binary sequences.

If the length of the binary sequence M_{Spath} is not a multiple of the parameter *n*, then the sequence is supplemented to the nearest multiple of the length of the M_{Spath} by the specified placeholder.

Stage 3. Each fragment M_i with the help of the check matrix $H \in Ecode$ is assigned the n - k bit S_i syndrome.

Stage 4. From the resulting syndromes, a binary sequence is formed by concatenation:

$$S_{Spath} = S_1 \mid S_2 \mid \ldots \mid S_q. \quad (2)$$

Stage 5. S_{SPath} binary sequence compression is performed. In this case, the loss base compression method specified by the stego-key is used. In the process of compression, the syndromes that make up the S_{SPath} sequence are considered as symbols of the primary alphabet, and the sequence itself, as a message consisting of these symbols. As a result, a compressed sequence of syndromes $S_{SpathCom}$ is formed.

Stage 6. A digital watermark is formed:

$$DWM = S_{SpathCom} \mid HashSum \mid ExtraFields, \quad (3)$$

where,

HashSum – monitoring hash sum;

ExtraFields – optional additional information fields;

«|» – designation for the operation of concatenation of the binary sequences.

The length of a digital watermark *DWM* must not exceed the length of the S_{SPath} sequence. Since $S_{SPathCom}$ is a compressed version of the S_{SPath} sequence, the valid size of the control hash sum and additional fields cannot exceed the difference between the lengths of the S_{SPath} and $S_{SPathCom}$ sequences.

Stage 7. The resulting digital watermark *DWM* is divided into fragments of S_i* by $n - k$ bits, i.e. the same length as the length of the S_i syndromes in expression (2):

$$DWM = S_1^* \mid S_2^* \mid \ldots \mid S_q^* \quad (4)$$

Stage 8. The bit values of the M_{SPath} sequence are modified in such a way as to ensure that for M_i fragments replace S_i syndromes with S_i* syndromes. To perform such a replacement, the current S_i and the required S_i* syndromes are summed modulo two: $b_i = S_i \oplus S_i^*$. The obtained value b_i sets the position of the bit that needs to be inverted in the M_i fragment to replace S_i syndrome with S_i* syndrome. Fragments of M_i*, resulting from changes in syndromes concatenate in form a binary sequence M^*_{SPath}, which coincides in length with the initial M_{SPath} sequence.

Thus, the resulting sequence M^*_{SPath} contains:

a) information about the original state of the sequence M_{SPath};

b) monitoring hash sum;

c) optional additional information fields.

Stage 9. The target bit values are modified in the program codes of the LUT units. Modifications are made in such a way as to change the source sequence of the target M_{SPath} bits to the M^*_{SPath} sequence obtained in the previous step. Modification is performed by applying equivalent conversions [31], [32].

In Fig. 5 shows an example of the implementation of the proposed stages of the integrity control method in part of embedding a digital watermark. The figure shows an M_{SPath} sequence consisting of 42 target bits. This sequence is obtained at the *first stage* of the method from the program codes of the LUT units that are on the stego-path.

Let for this example, an error-correcting code with the parameters (n, k) = (7, 4) and the check matrix of the following form is used:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

At the *second stage* of the method execution, the M_{SPath} sequence is divided into six fragments of n = 7 bits each.

At the *third stage* of the method, for each of the obtained fragments, $n - k = 3$ bit S_i error syndrome is calculated using the check matrix H.

In the *fourth stage*, the resulting syndromes concatenate, forming a binary sequence.

At the *fifth stage*, the resulting sequence is compressed. In this example, the Huffman method [41] is used for compression. This method is used in the example for clarity. In real applications, it is advisable to use more efficient lossless compression methods. A compressible sequence is considered as a set of symbols of the primary alphabet, which is formed by the values of S_i syndromes. As a result of the compression, a code table is formed that assigns a code combination to each syndrome. The length of code combination is proportional to the frequency of occurrence of the syndrome in binary sequence. Codes resulting from replacing the primary alphabet symbols form the binary sequence $S_{SpathCom}$.

At the *sixth stage*, a digital watermark is formed to be embedded. Let, for the considered example, the monitoring hash sum is a binary sequence “0000011”. At this stage, hash sum (in the figure, the bits of the hash sum are shown by accentuation) is concatenated with the sequence $S_{SpathCom}$. The result is an 18-bit binary sequence of digital watermark *DWM*.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

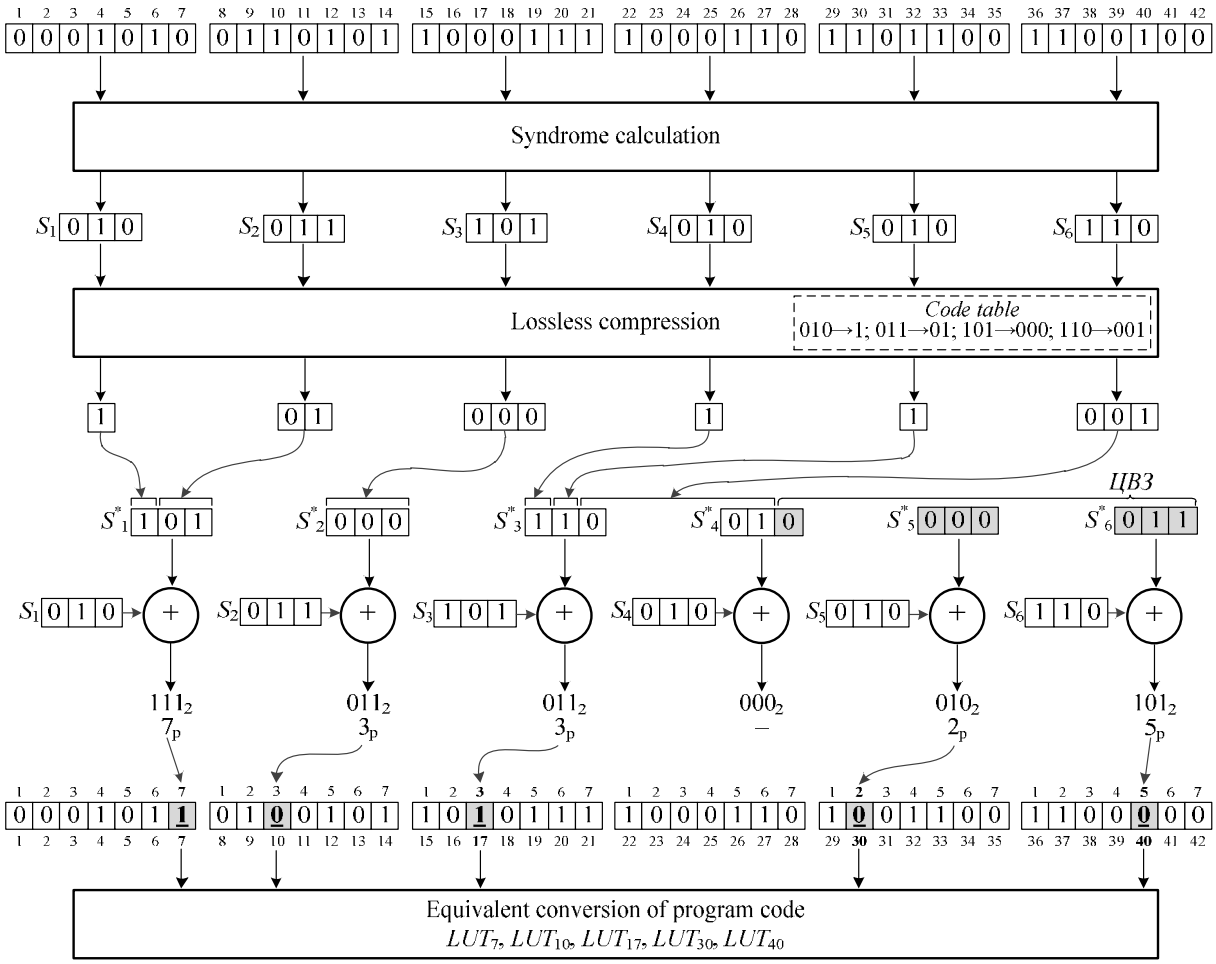

Fig. 5. Example of the proposed method: the steps of embedding a digital watermark in the FPGA program code

At the *seventh stage* of the method, the resulting digital watermark DWM is divided into fragments of n - k = 3 bits each: 6 fragments S_i* are formed.

At the *eighth stage*, the current S_i syndromes for M_i fragments are replaced with newly calculated S_i* syndromes. If the syndromes S_i and S_i* coincide, then there is no need to replace them and no additional actions are taken with respect to the M_i fragment. In this example, the equality holds for $S_4 = S_4* = 010$.

In the case of differences between S_i and S_i*, one bit of the fragment M_i is modified, which leads to a change in the syndrome. The position of this bit is equal to the sum modulo-two syndromes S_i and S_i*.

At the *ninth stage*, the bits are inverted for each of the fragments M_i. As a result, the binary sequence $M*_{SPath}$ is obtained from the M_{SPath} binary sequence. Further, in the target bits of the program code of the LUT units, equivalent conversions are performed, resulting in values corresponding to the sequence $M*_{SPath}$. As a result of these actions, the digital watermark is embedded in the program code of the LUT units.

In the considered example, to save the initial 42-bit state of the target bits and the 7-bit monitoring hash sum, we needed an equivalent inversion of the five target bits (one bit each in the program code of five LUT units).

The following procedure is proposed for extracting a digital watermark from the FPGA program code in the integrity monitoring process.

Stage 1. Similar to the first stage of a method of embedding the set of the LUT units which are on the stego-path is formed. From the program codes of this units set, an ordered sequence M_{SPath} of target bits is selected.

Stage 2. The sequence $M*_{SPath}$ is divided into n-bit fragments $M*_i$.

Stage 3. For each of the fragments $M*_i$, the n – k bit syndrome $S*_i$ is calculated using the check matrix $H \in Ecode$.

Stage 4. By concatenating syndromes S^*_i, a binary sequence is formed.

Stage 5. The monitoring hash sum is separated from the sequence obtained in the previous step. The rest of the sequence is subjected to a decompression procedure. To do this, use the decompression method, the inverse of the compression method used in the process of embedding a digital watermark. As a result of decompression, a set of S_i syndromes is obtained, the number of which coincides with the number of M^*_i fragments.

Stage 6. The procedure is similar to the one that was performed in step 8 in the process of embedding a digital watermark. The values of S^*_i syndromes are converted to S_i values by modifying one of the bits in each of the M^*_i fragments. In the case of equality of the syndromes S^*_i and S_i, the modification of the bits of the corresponding M^*_i fragments is not performed.

Stage 7. The initial state for the target bits of the program code is restored from the modified values of the fragments M^*_i by equivalent conversions.

To test the effectiveness of the proposed method, an experiment was performed. An experimental comparison of the proposed method with a well-known integrity monitoring method [33] (which performs the recovering of an information object, but does not use syndrome decoding during embedding) was performed.

The experiment involved five FPGA projects of various volumes and design mission. Synthesis of projects was carried out with the help of the CAD system Intel (Altera) Quartus Prime [42]. FPGA Intel Cyclone IV was used as target synthesis chips [43]. The aim of the experiment was to determine the number of bits modified in the process of embedding a digital watermark.

The experiment procedure consisted in the fact that a digital watermark was embedded in each of the five experimental projects. This watermark contained a monitoring hash sum and a compressed initial state for the target bits of program code of the LUT units. Embedding was performed twice: using the known and proposed methods. After each embedding, the number of target bits of the LUT units that were modified during the embedding process was counted.

The results of the experiment are shown in the table. FPGA-projects (which were used in the experiment) in the table are ordered by increasing their volume (total number of LUT units). For each of the projects, it is shown by how much, due to the application of the proposed method, the number of LUT units whose program code has been modified has decreased.

Table. Experimental results

Project No	Total number of LUT units	Reducing the number of modified LUT units
1	3280	6,1 %
2	3837	7,6 %
3	4589	12,9 %
4	7403	15,1 %
5	8265	19,4 %

From the results of the experiment it can be seen that the proposed method allows reducing the number of modified bits of program codes in LUT units. It can also be seen that a greater decrease in this number is achieved on projects that have a larger volume. This reduction in the number of modifications to the program code confirms the validity and effectiveness of the proposed method.

Conclusions and directions for future research

The proposed method is an integrated part of the technology for the integrity monitoring of FPGA chips program code. The method is based on embedding a standard hash sum in a program code in the form of a digital watermark. Integrity monitoring in the framework of the proposed method is possible due to the property of recovering the initial state for an information object of a program code in the process of extracting a digital watermark from it.

The proposed method of integrity monitoring for each n-bit fragment of the target bits sequence, where n is the parameter used by the (n, k)-code (specified by the stego-key):

a) requires changing the code for only one of the n LUT units in the fragment in case of a mismatch between the syndromes S_i and S_i^*;

b) does not require changes to the codes of the LUT units in the fragment in case these syndromes coincide.

The visible problem (which causes the need for future research) of the proposed method is the need to include a table of prefix codes in the stego-key. This need is caused by the fact that the table is calculated at the stage of embedding a digital watermark and cannot be calculated in advance. The remaining parameters of the key can be specified in advance by the embedding side and the digital watermark extraction side. This requirement increases the size of the stego-key. In the case of the inclusion of a code table in the composition of a digital watermark, the potential effective volume of a digital watermark is reduced. In this regard, there is a need for future research on the possibility of using (within the framework of the proposed approach) compression methods that do not require the preservation of a code table as part of a stego-key or digital watermark.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

References

1. Mehta, A. (2018). "ASIC/SoC Functional Design Verification", *Publ. Springer*, Cham, Switzerland.

2. Amano, H. (2018). "Principles and Structures of FPGAs", *Publ. Springer,* USA, New-York.

3. Kharchenko, V., Gorbenko, A., Sklyar V., & Phillips, C. (2013). "Green Computing and Communications in Critical Application Domains: Challenges and Solutions", In: 9th International Conference on Digital Technologies (DT2013), pp. 191-197, Zhilina, *Slovak Republic.*

4. Kharchenko, V., Illiashenko, O., Kovalenko, A., & Sklyar, V. (2014). Boyarchuk, A. "Security Informed Safety Assessment of NPP I&C Systems: GAP-IMECA Technique", In: 22nd International Conference on Nuclear Engineering, pp. 1-9. Prague, *Czech Republic.*

5. Drozd, A., Drozd, M., & Antonyuk, V. (2015). "Features of Hidden Fault Detection in Pipeline Components of Safety-Related System", CEUR Workshop Proceedings, Vol. 1356, pp. 476-485.

6. Drozd, A., Antoshchuk, S., Drozd, J., Zashcholkin, K., Drozd, M., Kuznietsov, M., Al-Dhabi, M., & Nikul, V. (2019). "Checkable FPGA Design: Energy Consumption, Throughput and Trustworthiness", In: Kharchenko, V., Kondratenko, Y., Kacprzyk, J. (eds.) "Green IT Engineering: Social, Business and Industrial Applications, Studies in Systems, Decision and Control", Vol. 171, pp. 73-94. *Publ. Springer,* Heidelberg, doi: 10.1007/978-3-030-00253-4_4.

7. Mukhopadhyay, D., & Chakraborty, R. (2014). "Hardware Security: Design, Threats, and Safeguards", *Publ. Chapman and CRC*, USA, Boca Raton.

8. Maevsky, D., Bojko, A., Maevskaya, E., Vinakov, O., & Shapa, L. (2017). "Internet of things: Hierarhy of smart systems", In: 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Vol. 2, pp. 821-827.

9. Andina, J. (2017). "FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics", *Publ. CRC Press*, USA, Boca Raton.

10. Vanderbauwhede, W., & Benkrid, K. (2016). "High-performance computing using FPGAs", *Publ. Springer,* USA, New-York.

11. Yang, Y., Chen, F., Zhang, X., Yu, J., & Zhang, P. (2016). "Research on the Hash Function Structures and its Application", In: International Conference Wireless Personal Communications.

12. Drozd, A., Drozd, J., Antoshchuk, S., Nikul, V., & Al-Dhabi, M. (2016). "Objects and methods of on-line testing: Main requirements and perspectives of development", In: IEEE East-West Design and Test Symposium, EWDTS-2016, pp. 1-9, doi: 10.1109/EWDTS.2016.7807750.

13. Stallings, W. (2017). "Cryptography and Network Security: Principles and Practice", 7th edn. *Publ. Pearson Education Limited*, United Kingdom, Harlow.

14. Habli, I., Hawkins, R., & Kelly, T. (2010). "Software safety: relating software assurance and software integrity", International Journal of Critical Computer-Based Systems 1(4), pp. 364-383.

15. Vacca, J. (2013). "Computer and information security", 2nd edn. *Publ. Morgan Kaufmann Publishers,* USA, Waltham.

16. Ferguson, N., Schneier, B., & Kohno, T. (2013). "Cryptography engineering", *Publ. Wiley*, USA, Hoboken.

17. Katz, J. (2010). "Digital signatures. Advances in Information Security", *Publ. Springer,* USA, New York.

18. Mishra, P., Bhunia, S., & Tehranipoor, M. (2017). "Hardware IP Security and Trust", *Publ. Springer,* USA, New-York.

19. Bishop, M. (2018). "Computer Security. 2nd edn.", *Publ. Addison-Wesley*, USA, Boston.

20. Kleidermacher, D., & Kleidermacher, M. (2012). "Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development", *Publ.* Newnes, USA, Boston.

21. Sklavos, N., Chaves, R., Natale, G., & Regazzoni, F. (2017). "Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment", *Publ. Springer,* Switzerland, Cham.

22. Berchtold, W., Schafer, M., & Steinebach, M. (2013). "Leakage detection and tracing for databases", In: ACM Information Hiding and Multimedia Security Workshop.

23. Ching-Nung Yang, Chia-chen Lin, & Chin-chen Chang. (2013). "Steganography and Watermarking", *Publ. Nova Science Publishers*, USA New York.

24. Shih, F. (2017). “Digital Watermarking and Steganography: Fundamentals and Techniques”, 2nd edn., *Publ. CRC Press*, USA, Boca Raton.

25. Cox, I., Miller, M., Bloom, J., & Fridrich, J. (2008). “Digital Watermarking and Steganography”, *Publ. Morgan Kaufmann Publishers,* Amsterdam.

26. Arnold, M., Schmucker, M., & Wolthusen, S. (2003). “Techniques and Applications of Digital Watermarking and Content Protection”, *Publ. Artech House*, Boston.

27. Zashcholkin, K. V. & Ivanova, E. N. (2013). Metod steganograficheskogo skrytiya dannykh v LUT-oriyentirovannykh apparatnykh konteynerakh, [Method of steganographical hiding of information in LUT-oriented hardware containers], Electrotechnic and Computer Systems, No. 12 (88), pp. 83-90 (in Russian).

28. Zashcholkin, K. V. & Ivanova, E. N. (2014). Informatsionnaya tekhnologiya vnedreniya samovosstanavlivayushchikh tsifrovykh vodyanykh znakov v LUT-oriyentirovannyye konteynery, [Information technology of embedding self-recovery digital watermark in LUT-oriented containers], Electrotechnic and Computer Systems, No. 16 (92), pp. 78-84 (in Russian).

29. Sklyarov, V., Skliarova, I., Barkalov, A., & Titarenko, L. (2014). “Synthesis and Optimization of FPGA-Based Systems”, *Publ. Springer,* Berlin.

30. Barkalov, A., Titarenko, L., Zeleneva, I., & Hrushko, S. (2018). “Implementing on the field programmable gate array of combined finite state machine with counter”, In: Conference Proceedings of 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies DESSERT-2018, pp. 247-251.

31. Drozd, A., Drozd, M., & Kuznietsov, M. (2016). “Use of Natural LUT Redundancy to Improve Trustworthiness of FPGA Design”, CEUR Workshop Proceedings, Vol. 1614, pp. 322-331.

32. Drozd, A., Drozd, M., Martynyuk, O., & Kuznietsov, M. (2017). “Improving of a Circuit Checkability and Trustworthiness of Data Processing Results in LUT-based FPGA Components of Safety-Related Systems”, CEUR Workshop Proceedings, Vol. 1844, pp. 654-661.

33. Zashcholkin, K., & Ivanova, O. (2015). “The control technology of integrity and legitimacy of LUT-oriented information object usage by self-recovering digital watermark”, CEUR Workshop Proceedings, Vol. 1356, pp. 498-506.

34. Fridrich, J. (2010). “Steganography in Digital Media”, *Publ. Cambridge University Press*, USA, New York.

35. Bossuet, L., Torres, L. (2018). “Foundations of Hardware IP Protection”, *Publ. Springer,* USA, New-York.

36. Westfeld, A. (2001). “F5 – A Steganographic algorithm. High capacity despite better steganalysis”, In Proceeding of 4th International Workshop on Information Hiding, 2001, Vol. 2137, pp. 289-302.

37. Fridrich, J., Goljan, M. & Du, R. (2002). “Lossless Data Embedding – New Paradigm in Digital Watermarking”, EURASIP Journal on Advances Signal Processing, pp. 185-196.

38. Morelos-Zaragoza, R. (2006). “The art of error correcting coding, *Publ. Wiley*, Chichester.

39. Zashcholkin, K., & Ivanova, O. (2018). “LUT-object integrity monitoring methods based on low impact embedding of digital watermark”, In: 14th International Conference “Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET-2018)”, pp. 519-523.

40. Zashcholkin, K. V., Drozd, A. V., Sulima, J. J. & Ivanova, E. N. (2018). Metod formirovaniya stego-puti pri reshenii zadachi kontrolya tselostnosti programmnogo koda FPGA-bazirovannykh ustroystv, [The method for stego-path formation in solving the problem of monitoring the integrity of the program code of FPGA-based devices], Systems and Technologies, No. 1 (56), pp. 5-17 (in Russian).

41. Salomon, D., & Motta, G. (2010). “Handbook of data compression”, *Publ. Springer,* London.

42. (2019).“Intel Quartus” [Electronic Resource]. – Access mode: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview. Html, Title from the screen. – Active link – 26.02.2019.

43. (2019). “Intel Cyclone FPGA series” [Electronic Resource]. – Access mode: https://www.intel.com/content/www/us/en/products/programmable/cyclone-series.html, Title from the screen. – Active link – 26.02.2019.

Received 05.03.2019

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

УДК 004.056.53

[1]**Защолкін, Костянтин Вячеславович,** кандидат технічних наук, доцент, доцент кафедри комп'ютерних інтелектуальних систем та мереж, E-mail: const-z@te.net.ua,
ORCID ID: 0000-0003-0427-9005
[1]**Дрозд, Олександр Валентинович,** доктор технічних наук, професор, професор кафедри комп'ютерних інтелектуальних систем та мереж, E-mail: drozd@ukr.net,
ORCID ID: 0000-0003-2191-6758
[1]**Іванова, Олена Миколаївна,** старший викладач кафедри комп'ютерних систем,
E-mail: en.ivanova.ua@gmail.com, ORCID ID: 0000-0002-4743-6931
[2]**Суліма, Юліан Юрійович,** кандидат технічних наук, відділенням комп'ютерних систем,
E-mail: mr_lemur@ukr.net, ORCID ID: 0000-0003-3986-7296
[1]Одеський національний політехнічний університет, проспект Шевченка, 1, Одеса,Україна
[2]Одеський технічний коледж Одеської національної академії харчових технологій, вул. Балківська, 54, Одеса,Україна

КОМПОЗИЦІЙНИЙ МЕТОД КОНТРОЛЮ ЦІЛІСНОСТІ ПРОГРАМНОГО КОДУ FPGA, БАЗОВАНИЙ НА ВИКОРИСТАННІ ЦИФРОВИХ ВОДЯНИХ ЗНАКІВ

Анотація. *В роботі розглянута проблема забезпечення цілісності програмованих компонентів комп'ютерних систем. Показані основні етапи життєвого циклу програмованих компонентів. Відзначено, що можливість модифікації програмного коду відкриває шляхи до зловмисному порушення його цілісності. Традиційні методи контролю цілісності базуються на використанні контрольних хеш-сум. Однак недолік традиційних методів полягає в тому, що вони не дозволяють приховати факт виконання контролю цілісності. Цей факт є відкритим. Навіть в умовах додаткового шифрування контрольної хеш-суми її наявність свідчить про те, що проводиться контроль цілісності. В роботі виділяється клас методів, в рамках яких контрольна хеш-сума вбудовується в програмний код у вигляді цифрового водяного знаку. Цей клас методів розглядається стосовно контролю цілісності програмного коду мікросхем FPGA (Field Programmable Gate Array). Для вбудовування використовуються особливості LUT-орієнтованої архітектури FPGA. Вбудовування контрольного цифрового водяного знаку виконується за рахунок застосування еквівалентних перетворень програмних кодів на множині блоків LUT, що входять до складу FPGA. Особливістю вбудовування цифрового водяного знаку є те, що таке вбудовування не змінює розмір програмного коду і не модифікує функціонування мікросхеми FPGA. В результаті вбудовування явно виділити контрольну хеш-суму в програмному коді стає неможливим. Витягання цифрового водяного знаку, який включає до свого складу хеш-суму можливо тільки при наявності спеціального стеганографічного ключа (що задає правила розміщення водяного знака в просторі програмного коду FPGA). В даній роботі пропонується композиційний метод вбудовування контрольного цифрового водяного знаку в програмний код FPGA. Метод поєднує властивості методів, що забезпечують відновлення первісного стану програмного коду, і методів, які здійснюють вбудовування на основі синдромного декодування. Пропонований метод поєднує корисні властивості зазначених двох класів методів і спрямований на зменшення кількості еквівалентних перетворень, що застосовуються до програмного коду в ході вбудовування цифрового водяного знаку. Представлено опис і результати експериментального дослідження запропонованого методу. Показано переваги запропонованого методу в порівнянні з базовими методами, вбудовування цифрових водяних знаків в програмний код FPGA.*

Ключові слова: *контроль цілісності програмного коду; програмовані апаратні компоненти; FPGA; LUT-орієнтована архітектура; контрольна хеш-сума; цифровий водяний знак; стеганографічний підхід до контролю цілісності*

УДК 004.056.53

[1]**Защелкин, Константин Вячеславович,** кандидат технических наук, доцент, доцент кафедры компьютерных интеллектуальных систем и сетей, E-mail: const-z@te.net.ua,
ORCID ID: 0000-0003-0427-9005
[1]**Дрозд, Александр Валентинович,** доктор технических наук, процессор, профессор кафедры компьютерных интеллектуальных систем и сетей, E-mail: drozd@ukr.net,
ORCID ID: 0000-0003-2191-6758
[1]**Иванова, Елена Николаевна,** старший преподаватель кафедры компьютерных систем,
E-mail: en.ivanova.ua@gmail.com, ORCID ID: 0000-0002-4743-6931

[2]Сулима, Юлиан Юрьевич, кандидат технических наук, заведующий отделением компьютерных систем, E-mail: mr_lemur@ukr.net, ORCID ID: 0000-0003-3986-7296
[1]Одесский национальный политехнический университет, проспект Шевченко, 1, Одесса, Украина
[2]Одесский технический колледж Одесской национальной академии пищевых технологий, ул. Балковская, 54, Одесса, Украина

КОМПОЗИЦИОННЫЙ МЕТОД КОНТРОЛЯ ЦЕЛОСТНОСТИ ПРОГРАММНОГО КОДА FPGA, ОСНОВАННЫЙ НА ИСПОЛЬЗОВАНИИ ЦИФРОВЫХ ВОДЯНЫХ ЗНАКОВ

Аннотация. *В работе рассмотрена проблема обеспечения целостности программируемых компонентов компьютерных систем. Показаны основные этапы жизненного цикла программируемых компонентов. Отмечено, что возможность модификации программного кода открывает пути к злонамеренному нарушению его целостности. Традиционные методы контроля целостности, основаны на использовании контрольных хэш-сумм. Однако недостаток традиционных методов состоит в том, что они не дают возможность скрыть факт выполнения контроля целостности. Этот факт является открытым. Даже в условиях дополнительного шифрования контрольной хэш-суммы ее наличие свидетельствует о том, что производится контроль целостности. В работе выделяется класс методов, в рамках которых контрольная хэш-сумма внедряется в программный код в виде цифрового водяного знака. Этот класс методов рассматривается применительно к контролю целостности программного кода микросхем FPGA (Field Programmable Gate Array). Для встраивания используются особенности LUT-ориентированной архитектуры FPGA. Встраивание контрольного цифрового водяного знака выполняется за счет применения эквивалентных преобразований программных кодов на множестве блоков LUT, входящих в состав FPGA. Особенностью встраивания цифрового водяного знака является то, что такое встраивание не изменяет размер программного кода и не модифицирует функционирование микросхемы FPGA. В результате встраивания явным образом выделить контрольную хэш-сумму в программном коде становится невозможным. Извлечение цифрового водяного знака, который включает в свой состав хэш-сумму возможно только при наличии специального стеганографического ключа (который задает правила размещения водяного знака в пространстве программного кода FPGA). В данной работе предлагается композиционный метод встраивания контрольного цифрового водяного знака в программный код FPGA. Метод совмещает свойства методов, обеспечивающих восстановление инициального состояния программного кода, и методов, осуществляющих встраивание на основе синдромного декодирования. Предлагаемый метод сочетает полезные свойства указанных двух классов методов и направлен на уменьшение количества эквивалентных преобразований, применяемых к программному коду в ходе встраивания цифрового водяного знака. Представлено описание и результаты экспериментального исследования предлагаемого метода. Показаны преимущества предлагаемого метода в сравнении с базовыми методами, встраивания цифровых водяных знаков в программный код FPGA.*

Ключевые слова: *контроль целостности программного кода; программируемые аппаратные компоненты; FPGA; LUT-ориентированная архитектура; контрольная хэш-сумма; цифровой водяной знак; стеганографический подход к контролю целостности*

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)