

Article

Countering IoV Cyberattacks Using Encryption in a Polynomial Modular Code

Igor Kalmykov ¹, Gennady Slyusarev ², Vladimir Kopytov ³, Alexander Olenov ¹, Tatyana Peleshenko ¹ and Maria Lapina ^{1,*}

¹ Department of Computational Mathematics and Cybernetics, North-Caucasus Federal University, 355017 Stavropol, Russia; kia762@yandex.ru (I.K.); olenovalexandr@gmail.com (A.O.); tagish@ncfu.ru (T.P.)

² Center for Scientific Infrastructure and Export Control, North-Caucasus Federal University, 355017 Stavropol, Russia; gslyusarev@ncfu.ru

³ Department of Information Security Organization and Technology, North-Caucasus Federal University, 355017 Stavropol, Russia; v.kopytov@infocom-s.ru

* Correspondence: mlapina@ncfu.ru; Tel.: +7-91-8761-0038

Abstract

Despite significant improvements in vehicle management efficiency achieved through the integration of VANET and Internet of Things technologies, Internet of Vehicles (IoV) networks remain vulnerable to cyberattacks. This is because the wireless data exchange channel in IoV has several vulnerabilities that are exploited to carry out cyberattacks. The article suggested using the symmetric block cipher GOST 34.12-2015 (SBCG) to combat a variety of cyberattacks. This cipher was chosen because it can be efficiently implemented on low-power platforms and offers high cryptographic strength and encryption speed. Furthermore, implementing SBCG in polynomial modular codes (PMCs) enables detection of encryption errors caused by faults in encoder/decoder operation. The scientific novelty of the proposed solution is that it is the first method to increase the fault tolerance of an SBCG encoder, enabling real-time, effective countermeasures against faults caused by both Differential Fault Analysis (DFA) attacks and natural faults. The originality of the solution lies in the integration of cryptographic theory and the theory of constructing correcting modular codes. The goal of this study is to improve the resilience of SBCG encryptors/decoders to faults by using polynomial modular codes. Imparting fault-tolerant properties to SBCG encryption systems implemented in PMC will enable them to effectively mitigate real-time faults arising from both Differential Fault Analysis (DFA) attacks and natural faults.

Keywords: cyberattacks; IoV network; symmetric block cipher; polynomial modular codes; fault tolerance; error-correcting decoding algorithm



Academic Editor: Leandros Maglaras

Received: 18 May 2026

Revised: 5 June 2026

Accepted: 8 June 2026

Published: 10 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The major increase in vehicle numbers in modern cities and the growing demand for traffic safety have provided a powerful impetus for the development of Vehicle-to-Everything (VANET) networks, which consist of Roadside Units (RSUs) and On-Board Units (OBUs) [1–3]. This is clearly evident in the integration of VANET and Internet of Things (IoT) technologies. The emergence of Internet of Vehicles (IoV) technology enables drivers to receive real-time information on traffic congestion, road conditions, traffic accidents, and detour routes. Thanks to IoV technology, the burden on drivers is reduced, and comfort levels increase, including for passengers [4].

However, the adoption of new information technologies has increased cyberattacks on the IoV network [5,6]. An analysis of the works [5–8] has shown that most cyberattacks aim to compromise the confidentiality of data transmitted over the IoV. The effectiveness of such attacks can be mitigated through encryption. Among modern SPN ciphers, the symmetric block cipher GOST 34.12-2015 (SBCG) [9] holds a special place. This cipher was chosen because it can be efficiently implemented on low-power platforms and offers high cryptographic strength and encryption speed [10,11]. Consequently, this cipher has great potential for effectively preventing cyberattacks targeting confidentiality and can be used in IoV systems.

It is well known that technologies for creating high-speed vehicles and the intelligent real-time control systems for them are considered critical. One of the main requirements for such systems is ensuring their fault tolerance during operation. This requirement is fundamental for the operation of encoders/decoders, as in the event of malfunctions or failures, unprotected data will be transmitted over the radio channel. As a result, an attacker could launch a cyberattack by tampering with the transmitted information. Furthermore, due to malfunctions or failures in the encryption/decryption modules, the speed at which critical information reaches drivers is reduced because of the need for re-encryption. All of this can negatively impact the safety of high-speed road traffic.

There are two ways to improve the fault tolerance of IoV systems. First, introducing structural redundancy. Thus, in [12,13], a duplication method is proposed to improve fault tolerance. The works [14–16] consider the Triple Modular Redundancy (TMR) method. However, despite their simple circuit implementation, structural redundancy methods have a drawback: high hardware costs.

Second, this method utilizes information redundancy. In this case, error-correcting codes are used to mitigate the effects of failures or errors that distort computation results. Since the SBCG encryption algorithm operates in the field $GF(2^8)$, it is advisable to select a polynomial modular code (PMC) as the code. This code allows transitioning from encryption in the field $GF(2^8)$ to analogous procedures in the ring of $GF(2^4)$. At the same time, the developed decoding algorithm PMC eliminates the consequences of failures and errors directly during the operation of the encoders/decoders. This will prevent the transmission of critical information in plaintext. As a result, a cyberattack targeting the confidentiality of data transmitted over a VANET network will fail.

The scientific novelty of the proposed solution is that it is the first method to increase the fault tolerance of an SBCG encoder, enabling real-time, effective countermeasures against faults caused by both Differential Fault Analysis (DFA) attacks and natural faults. The originality of the solution lies in the integration of cryptographic theory and the theory of constructing correcting modular codes.

The goal of this study is to improve the resilience of SBCG encryptors/decoders to faults by using polynomial modular codes. Imparting fault-tolerance properties to SBCG encryption systems implemented in PMC will enable them to effectively mitigate real-time faults caused by both DFA attacks and natural faults. Equipping SBCG encryption systems with fault-tolerant properties will not reduce the IoV network's resilience to cyberattacks, even if failures or malfunctions occur during operation.

The scientific results presented in the article include:

- We develop the decoding algorithm in the PMC, in contrast to previously known algorithms, and show that our approach corrects distorted code residues at a lower circuit cost than these existing solutions.
- We present precise mathematical models for encryption transformations in SBCG, implemented in modular code.

- We introduce explicit mathematical models for decryption transformations in SBCG, realized in modular code.

The article is organized as follows. Section 1 presents the introduction and frames the research issue. Section 2 reviews related work and existing approaches to fault tolerance, modular coding, and error correction in redundant computational systems. Section 3 focuses on PMC redundancy codes and presents an explicit error-correcting decoding algorithm for these codes; it precisely defines mathematical models of fault-tolerant SBCG encoders and decoders constructed using modular code structures. Section 4 details the operation of a fault-tolerant system that implements the explicit SBCG algorithm in PMCs and demonstrates, step-by-step, how these exact models and decoding methods are used to achieve reliable system performance under fault conditions.

2. Related Works

It is advisable to divide the analysis of related works into two parts. The first part of this section will examine works describing methods for countering cyberattacks on VANET networks using encryption. The second part of this section is devoted to an analysis of works aimed at improving the fault tolerance of computing and information systems.

2.1. Analysis of Works on Countering Cyberattacks in VANET Networks

Currently, several known cyberattacks are targeting VANET networks. A classification of these attacks is provided in [7]. The main objectives of such attacks are to create conditions that prevent effective data reception, gain unauthorized access to information, modify it, and cause service delays. Obviously, different countermeasures are applied for each type of attack. Let us consider attacks whose effectiveness is reduced by encryption methods.

The first group of such cyberattacks consists of attacks on availability (Attack on Availability), among which DOS and DDoS attacks can be distinguished [17,18]. When these attacks occur, legitimate users are unable to access VANET network resources. This is achieved by attackers sending a large number of messages, resulting in a denial-of-service attack. One method for countering such attacks is encryption [19,20]. A Malware Attack can reduce the availability of the VANET communication channel. Such an attack disrupts the operation of the OBU and RSU transmitting systems. To counter such attacks, it is proposed to use message source authentication and encryption [21].

The second category of cyberattacks consists of attacks on authentication. Among these attacks, tunneling attacks stand out. Such an attack relies on a communication channel (tunnel) created by an attacker between two remote VANET nodes. As a result, the attacker gains unauthorized access to the transmitted data. This attack can be prevented using authentication protocols and encryption methods [22]. The goal of an eavesdropping attack is to analyze traffic between two VANET nodes. The attacker listens in on conversations between vehicle drivers using radio interception tools. The effectiveness of such an attack can be reduced using pseudorandom frequency hopping and encryption methods [23,24]. In a Man-in-the-Middle attack, an attacker intercepts the data stream from the source, modifies it, and then transmits it to the recipient. In this case, the source and recipient believe they are exchanging data directly without an intermediary. Authentication methods and encryption protocols can reduce the effectiveness of such an attack [25]. The goal of a social engineering attack is to steal personal information from vehicle owners. Methods proposed to reduce the effectiveness of such an attack include spam filtering and encryption of emails and passwords [26,27].

The third category of cyberattacks consists of attacks on data integrity. In a masquerading attack, an attacker impersonates another legitimate user. This attack results in unauthorized access to data stored in the OBU or RSU memory. In [28], the authors propose

using the AES encryption algorithm to prevent such attacks. The goal of an Illusion Attack is to create a false image of the traffic situation for the vehicle driver. One method for preventing this attack is the use of public-key cryptography [27,29].

The fourth group of cyberattacks consists of attacks on confidentiality. Encryption methods are widely used specifically to counter such attacks. The goal of such attacks on confidentiality is to provide an attacker with unauthorized access to confidential information. The paper [30] describes the use of the AES block symmetric cipher to counter brute-force cyberattacks. In [31], the authors chose traffic encryption as a method to reduce the effectiveness of Traffic Analysis attacks. To improve the effectiveness of countermeasures against confidentiality attacks, several papers propose the combined use of symmetric and asymmetric ciphers [32,33].

Side channel attacks occupy a special place among attacks on confidentiality. The targets of such attacks are cryptosystems where it is impossible to violate isolation or directly access the secret. A distinction is made between timing attacks and Differential Fault Analysis (DFA) attacks. In the first type of side channel attacks, the attacker identifies a correlation between the secret data used by an operation and the time of the operation. In the second type of DFA attacks, the attacker deliberately introduces a glitch in the encryption process. The purpose of this attack is to analyze the functioning of the encryption system after such a destructive effect. As a result of such a cyberattack, the attacker can obtain information about the key used for encryption. In [34], a DFA attack on the LED cipher used in VANETs to ensure confidentiality is described. Simulation modeling showed that the attacker was able to recover 64-bit and 128-bit LED secret keys, introducing, on average, 48 and 96 errors, respectively. In [35], the authors analyzed the resistance of the Khudra cipher, used in VANETs, to DFA attacks. The results of the study showed that the DFA attack allows an attacker to recover an 80-bit secret key by introducing only two errors. In VANET networks, the AES cipher is widely used to ensure confidentiality. Let us analyze the works describing DFA attacks against this SPN cipher. The effectiveness of these attacks largely depends on the choice of the attack model and the version of the encryption algorithm. In [36], a developed model of a DFA attack on the AES SPN cipher is described. Using this model, it was possible to obtain a 128-bit key from a set of six corrupted texts. In [37], the authors implemented a DFA attack on the AES-192 and AES-256 algorithms. The results showed that only 16 erroneous texts were used to find 192-bit and 256-bit keys when implementing this attack. In [38], it was shown that, using two pairs of correct and incorrect ciphertexts, the authors were able to calculate a 96-bit key from a 128-bit key. The authors claim to have obtained a 128-bit key using four pairs of correct and incorrect ciphertexts. In [39], the authors proposed a DFA attack model that uses two corrupted ciphertexts. In [40], the authors used a single-byte fault model to conduct a DFA attack. The results of the study showed that using this attack model to obtain a 192-bit key, the required number of errors is two, and the complete search is 210.

An analysis of the studies showed that DFA attacks can reduce the level of confidentiality in a VANET. Therefore, the development of protection methods against these attacks is a pressing issue. The following methods are used to protect against DFA attacks. To prevent external destructive influences, encoders are shielded. To reduce the effectiveness of the attack based on the power reduction method, voltage stabilizers are used. One of the most promising ways to prevent DFA attacks is to use methods that enable remediation of failures during encryption without re-encryption. This approach will be discussed in this article.

The analysis of the literature presented here shows that encryption methods are highly effective against a wide range of cryptographic attacks in IoV. At the same time, block ciphers are typically used to ensure confidentiality in VANET networks. Such ciphers may

include Blowfish, DES, 3DES, RSA, AES, and SBCG. Each of these has its own advantages and disadvantages.

Two conditions were set when selecting a cipher. The first condition is that the cipher must be efficiently implementable on low-power platforms and must offer high cryptographic strength and encryption speed.

The second condition is that the encryption algorithm must provide the ability to eliminate the consequences of failures and malfunctions that occur during the operation of encoders/decoders, using redundant arithmetic codes.

An analysis of known ciphers has shown that the AES and SBCG encryption algorithms satisfy these two conditions. Each is based on a set of cryptographic transformations $GF(2^8)$.

However, using the same SPN-based approach, these ciphers exhibit varying cryptographic resistance to different types of attacks. Let us consider the cryptographic strength of these ciphers against DFA attacks. An analysis of [36–40] has shown that the effectiveness of these attacks largely depends on the attack model and the encryption system's modes of operation. Thus, in [36], six distorted texts were required to restore a 128-bit key. The attack model presented in [37] uses 16 erroneous texts to find a 192-bit and 256-bit key. It was shown in [38] that, using two pairs of correct and erroneous ciphertexts, the authors could recover 96 bits of a 128-bit key. The DFA-type attack model presented in [39], which uses two distorted texts, made it possible to recover the 128-bit AES key. In [40], when conducting a DFA-type attack, the authors used the single-byte fault model. The results of the study showed that when using this attack model to obtain a 192-bit key, the required number of errors is two, and the full search is 2^{10} .

Let us consider the cryptographic strength of the SBCG cipher against DFA attacks. It should be noted that the number of papers devoted to these attacks on the AES cipher significantly exceeds the number of publications on DFA attacks on SBCG. In [41], a four-stage fault analysis (IFA) based on cipher replacement blocks was considered. The attack model presented in the article requires introducing 217 errors. Byte distortion attacks in rounds 7 and 8 of encryption are described in [42]. When implementing this attack, the intruder can recover the original key with an average of four errors. Let us consider the SBCG cipher's resistance to other attacks. It was shown in [43] that, to find the first-round 128-bit key for SBCG, it is necessary to encrypt 2128 modifications of a given array. At the same time, to calculate the first round 128-bit key for the AES cipher. The analysis of the works [36–43] shows the prospects for using the SBCG cipher in IoV from the perspective of ensuring network confidentiality. Let us perform a comparative analysis of AES and SBCG ciphers using other parameters.

It should be noted that these ciphers have different time costs. First, the number of encryption rounds differs. For example, with a 256-bit key length, the AES standard performs 14 encryption rounds, whereas SBCG performs only 10. Moreover, the final round in SBCG involves mixing with the round key. Second, there is a difference in the number of procedures performed per encryption round. Each SBCG round has an LSX structure. That is, during one encryption round, a linear L transformation, a nonlinear S transformation, and a mixing operation X with the round key are performed. A single round of the AES cipher contains more operations. These are SubBytes—the substitution S, ShiftRows—a cyclic byte shift, MixColumns—the linear transformation L, and AddRoundKey—mixing with the round key X. Third, there are different mathematical transformations when performing the linear transformation. Thus, in SBCG, the linear transformation is implemented using a shift register with linear feedback. In the AES cipher, calculating a single byte in MixColumns requires four modular multiplications and three additions. Summarizing the analysis of the two SPN ciphers, we conclude that, despite lower time costs for encryp-

tion/decryption while maintaining equivalent cryptographic strength, SBCG has greater potential to counter cyberattacks in VANET networks than AES.

2.2. Analysis of Works Devoted to Improving the Fault Tolerance of Computing and Information Systems

When implementing DFA-type attacks, the violator intentionally introduces a malfunction into the encryption process to analyze the behavior of the encryption system after such a destructive impact. In this case, the failure is anthropogenic. However, failures of encryption systems can occur for other reasons. These include operational overloads, defects in materials and manufacturing processes, and violations of operating standards. An important role in the occurrence of failures is played by the temperature conditions under which encryption systems are operated. A number of methods have been developed to eliminate the consequences of this type of failure. We will analyze the main types of methods for increasing the fault tolerance of infotelecommunication systems.

All methods designed to improve the fault tolerance of computing and information systems can be divided into two groups. The first group includes methods that use structural redundancy. The goal of introducing structural redundancy is to ensure the system's normal functioning after component failures. In this case, the redundant component must have characteristics similar to those of the primary component. In [12,13], the duplication method is proposed to improve fault tolerance. When implementing this method, redundancy is achieved through replacement. In this case, the primary element's functions are transferred to the backup only after the primary fails. The advantage of this method is its low circuit cost, since only two identical systems are used. However, this method also has drawbacks. The main drawback of the redundancy method is as follows. To switch to the backup encryption system, it is first necessary to detect the failure of the primary system, then activate the redundant system, and re-establish communication. All of this requires additional time. Furthermore, the redundancy method is not very effective when hardware malfunctions occur.

Methods involving multiple redundancy provide a higher level of fault tolerance. The works [14–16] discuss the Triple Modular Redundancy (TMR) method. In this case, three identical systems operate in parallel. A majority element is used to compare the output values of all three systems. This method increases the probability of failure-free operation and operational reliability by automatically mitigating the effects of random failures and malfunctions. However, despite their simple circuit implementation, structural redundancy methods have a drawback: high hardware costs.

The second group is based on methods that utilize information redundancy. In this case, modular error-correcting codes are used to mitigate the effects of errors or failures that distort calculation results. Among such codes, residue number system (RNS) codes have become the most widespread. To identify a distorted residue in a code combination and correct it, RNS codes use positional characteristics (PCs). In [44], a decoding algorithm was proposed that used the interval positional characteristic for error detection and correction. If the calculated interval is zero, the RNS code combination does not contain an erroneous residue. In [45], an algorithm for calculating this characteristic is described, in which Euler's function is applied. To reduce hardware costs, [46] proposes an algorithm for calculating the interval PC using the Chinese Remainder Theorem (CRT). However, this algorithm is performed over a composite modulus, which does not ensure the maximum possible reduction in hardware costs. To solve this problem, in [47], it was proposed to move from calculations based on a composite modulus to similar calculations based on two control bases.

Reference [48] describes an error-correction algorithm for RNS codes based on the computation of the coefficients of a generalized polyadic system. In Reference [49], the zeroing method was proposed for detecting and correcting a corrupted residue in a modular code.

However, the algorithms discussed above cannot be used for error detection and correction in SBCG encoders/decoders. This is because they were developed for RNS codes, which perform computations with integers. The SBCG cipher, however, operates in a field $GF(2^8)$ where bytes are polynomials. Therefore, it is necessary to develop a decoding algorithm that accounts for the principles of SBCG cipher construction when detecting and correcting errors.

3. Materials and Methods

3.1. Decoding Algorithm in a Polynomial Modular Code with Error Correction

Before we move on to developing a decoding algorithm for PMC with error correction, let us review the principles of modular coding. A polynomial modular code is an arithmetic code implemented as the sum of finite fields defined by irreducible polynomials $g_1(z), \dots, g_k(z)$, where z is a variable [50,51]. In this case, the PMC combination takes the form

$$R(z) = (R_1(z), \dots, R_k(z)), \quad (1)$$

where $R_i(z) \equiv R(z) \bmod g_i(z)$ is the residue of the polynomial $R(z)$; $i = 1, \dots, k$.

The irreducible polynomials $g_1(z), \dots, g_k(z)$ are the bases of the PMC, and their product defines the working range

$$P_k(z) = \prod_{i=1}^k g_i(z) \quad (2)$$

In this case, the set of valid code combinations will look like this:

$$N_k = 2^{\deg P_k(z)} \quad (3)$$

where $\deg P_k(z)$ is the degree $P_k(z)$.

It is known in [50–52] that if condition (4) is met for three polynomials

$$\{\deg R(z), \deg T(z), \deg H(z)\} < \deg P_k(z), \quad (4)$$

then the following operations are valid:

$$H(z) = R(z) + T(z) = (|R_1(z) + T_1(z)|_{g_1(z)}, \dots, |R_k(z) + T_k(z)|_{g_k(z)}), \quad (5)$$

$$H(z) = R(z) - T(z) = (|R_1(z) - T_1(z)|_{g_1(z)}, \dots, |R_k(z) - T_k(z)|_{g_k(z)}), \quad (6)$$

$$H(z) = R(z) \times T(z) = (|R_1(z) \times T_1(z)|_{g_1(z)}, \dots, |R_k(z) \times T_k(z)|_{g_k(z)}), \quad (7)$$

where $T_i(z) \equiv T(z) \bmod g_i(z)$; $H_i(z) \equiv H(z) \bmod g_i(z)$; $i = 1, \dots, k$.

Expressions (5)–(7) clearly illustrate the basic principle of performing modular operations in a polynomial modular code. In PMC, modulo-2 addition and subtraction, as well as multiplication, are performed in parallel with the corresponding remainders. In this case, computations in one base do not depend on computations in the other base. Since modular operations are performed on residues whose degree does not exceed $\deg g_i(z)$, where $i = 1, \dots, k$, expressions (5)–(7) can be implemented using LUTs. Thus, the first advantage of PMC is that the use of polynomial modular codes allows for increased computation speed in the sum of Galois fields.

The second advantage of PMC stems from the independence of computations across the code bases. As a result, an error occurring in the i -th base, where $i = 1, \dots, k$, does not adversely affect the other bases. In other words, errors do not propagate during computations in PMC. Thanks to this property, polynomial modular codes can not only detect but also correct computational errors.

To solve this problem, the number of PMC bases is increased. The added bases increase the code's redundancy and are considered check modules. As a result, the size of the code combination increases by ρ residues

$$R(z) = (R_1(z), \dots, R_k(z), R_{k+1}(z), \dots, R_{k+\rho}(z)), \quad (8)$$

where ρ —the number of check bases; $R_{k+j}(z) \equiv R(z) \bmod g_{k+j}(z); j = 1, \dots, \rho$.

If $\rho = 1$, then the polynomial modular code has minimal redundancy and can detect only a single error [51]. In the theory of constructing redundant PMCs, a single error is defined as the distortion of a single residue in the code combination.

To improve the error-correcting capabilities of PMCs, [47,51,53] propose increasing PMC redundancy to $\rho = 2$. Studies have shown that if the check bases are selected under the condition:

$$\deg g_{k-1}(z) + \deg g_k(z) \leq \deg g_{k+1}(z) + \deg g_{k+2}(z), \quad (9)$$

then the redundant code combination (10) can not only detect but also correct a single corrupted residue:

$$R(z) = (R_1(z), \dots, R_k(z), R_{k+1}(z), R_{k+2}(z)). \quad (10)$$

In other words, if the PMC combination contains two check residues, it can correct a single error.

The main condition for verifying code combination (10) is that the following inequality holds:

$$\deg R(z) < \deg P_k(z). \quad (11)$$

If this condition holds, then the code combination contains no errors. A violation of condition (11) indicates that the code combination contains an error.

During decoding, the redundant code combination is checked to identify corrupted residues. The previous section of this article discussed the main algorithms for detecting and correcting errors in modular codes. The first feature of these algorithms is that they were developed for the RNS modular code. To use these algorithms in a polynomial modular code, they must be modified. The second feature is a specific aspect of developing correction algorithms for the SBCG encoder/decoder. It is known that SBCG is related to SPN ciphers that operate over $GF(2^8)$. In [53], the possibility of implementing such an SPN cipher as a sum of two fields of lower dimension, $GF(2^4)$, is demonstrated. In this case, computations modulo $g(z) = z^8 + z^4 + z^3 + z + 1$ were replaced by computations modulo two modules. First module $g_1(z) = z^4 + z + 1$. Second module $g_2(z) = z^4 + z^3 + 1$. It is known that there are three irreducible fourth-degree polynomials. If two polynomials serve as information bases, then $g_3(z) = z^4 + z^3 + z^2 + z + 1$ was chosen as the check basis. In this case, the code combination takes the form (12):

$$R(z) = (R_1(z), R_2(z), R_3(z)), \quad (12)$$

where $R_i(z) \equiv R(z) \bmod g_i(z); i = 1, 2, 3$.

Let us consider the decoding algorithm [53] when performing the S-substitution operation. To implement this operation, the developed algorithm uses three 256×4 -bit tables. The combinations $(R_1(z), R_2(z))$, $(R_1(z), R_3(z))$ and $(R_2(z), R_3(z))$ are input to each of them, respectively. Three leftovers have been taken out of these tables:

$$R'(z) = (R'_1(z), R'_2(z), R'_3(z)), \quad (13)$$

where $R'(z)$ —the output byte of the S-table; $R'_i(z) \equiv R'(z) \bmod g_i(z)$; $i = 1, 2, 3$.

After that, the authors propose performing the inverse transformation using the Chinese Remainder Theorem (CRT):

$$R'(z) = \sum_{i=1}^3 R'_i(z) B_i(z) \bmod P_3(z), \quad (14)$$

where $B_1(z)$ —the orthogonal basis; $B_1(z) \equiv 1 \bmod g_i(z)$; $P_3(z) = \prod_{i=1}^3 g_i(z)$; $i = 1, 2, 3$.

If condition (11) is satisfied, then combination (12) contains no errors. If condition (11) is not satisfied, then combination (12) contains errors. In this case, the nonlinear S-substitution operation is performed again.

An advantage of the algorithm [53] is that it uses only irreducible fourth-degree polynomials. This allows for lower hardware costs compared to the algorithms [44–49]. However, this algorithm also has drawbacks. Since the number of check bases is $\rho = 1$, such a PMC can only detect single errors. Furthermore, the use of CRT in the decoding algorithm does not ensure minimal time complexity for error detection. Furthermore, repeating encryption rounds increases the time required for encryption/decryption, potentially compromising traffic safety. Therefore, developing a decoding algorithm in PMC that corrects computational errors in SBCG is a pressing task.

An analysis of the works [44–49] has shown that classic error-correcting decoding methods for modular codes are not suitable for ensuring fault tolerance in SBCG encoders/decoders. This is because, in this case, two check bases will be used as check bases $g_3(z)$, $g_4(z)$, for which $\deg g_3(z) = 4$, $\deg g_4(z) = 5$. In this case, the size of the LUT table size increases during hardware implementation of SBCG encoders/decoders. Thus, for $g_3(z)$, the LUT table will have 2^8 memory elements, and for $g_4(z)$, the size of the LUT table will increase to 2^{10} memory elements. Thus, it is necessary for the two check residues, which allow for the correction of a single error, to have a length of four bits. Furthermore, the use of algorithms [44–49] may lead to a reduction in the cryptographic strength of SBCG, since the check residues could be described as:

$$R'_3(z) \equiv R'(z) \bmod g_3(z), \quad R'_4(z) \equiv R'(z) \bmod g_4(z). \quad (15)$$

As a result, an attacker gains additional information about how transformations are executed in the cipher. Therefore, the error-correcting decoding algorithm under development must have checked residues that were not computed using expression (15).

It is known that, in PMCs, two conditions must be met simultaneously to correct an error during decoding. First, the location of the corrupted residue—that is, the bit position where the error occurred—must be known. Second, the specific bits in the residue that were corrupted—that is, the depth of the error—must be known. Only in this case can the decoding algorithm correct the corrupted PMC residue.

To implement the first condition, the developed decoding algorithm proposes representing the check residue $R_3(z)$ as a weighted sum of the first information residue and the product of the second residue by z . The calculation of the second check residue $R_4(z)$,

implementing the second condition, can be obtained by summing the two information residues modulo two. Then we have

$$\begin{aligned} R_3(z) &= R_1(z) + zR_2(z) \bmod g_3(z), \\ R_4(z) &= R_1(z) + R_2(z). \end{aligned} \quad (16)$$

The result is a redundant PMC combination describing a byte—the smallest unit in SBCG encryption:

$$R(z) = (R_1(z), R_2(z), R_3(z), R_4(z)). \quad (17)$$

To locate the corrupted residue and determine the error depth in the developed decoding algorithm, it is proposed to use a positional characteristic—the error syndrome. As a result, we obtain the following:

$$\begin{cases} \lambda_1(z) = R_3(z) + \tilde{R}_3(z), \\ \lambda_2(z) = R_4(z) + \tilde{R}_4(z), \end{cases} \quad (18)$$

where $R_3(z)$, $R_4(z)$ —the check residues of the valid PMC combination, which contains no errors; $\tilde{R}_3(z)$, $\tilde{R}_4(z)$ —parity residues calculated from the information residues using expressions (16).

It is clear that if the code combination contains no errors, then the following equality holds:

$$R_3(z) = \tilde{R}_3(z), R_4(z) = \tilde{R}_4(z). \quad (19)$$

Thus, expression (18) takes the form

$$\begin{cases} \lambda_1(z) = 0, \\ \lambda_2(z) = 0. \end{cases} \quad (20)$$

If an error occurs in the information residues of the code combination, then

$$R_3(z) \neq \tilde{R}_3(z), R_4(z) \neq \tilde{R}_4(z). \quad (21)$$

As a result, expression (18) will take the form

$$\begin{cases} \lambda_1(z) \neq 0, \\ \lambda_2(z) \neq 0. \end{cases} \quad (22)$$

In this case, the first part of the error syndrome $\lambda_1(z)$ will indicate the bit in which the error occurred, and the second part $\lambda_2(z)$ will indicate its depth.

Analysis of expressions (16)–(22) allows us to draw a number of conclusions.

First, if an error has corrupted the first information residue $R_1^*(z)$, and its depth is $\Delta R_1(z)$, $\deg \Delta R_1(z) < \deg g_1(z)$, then, in this case, the corrupted residue takes the form:

$$R_1^*(z) = R_1(z) + \Delta R_1(z). \quad (23)$$

Then, for the combination $R^*(z) = (R_1^*(z), R_2(z), R_3(z), R_4(z))$, the following holds:

$$\begin{cases} \lambda_1(z) = \Delta R_1(z), \\ \lambda_2(z) = \Delta R_1(z). \end{cases} \quad (24)$$

Thus, if the error syndromes coincide, the error vector takes the form $\bar{e}(z) = (\Delta R_1(z), 0, 0, 0)$. Then, the correction of the erroneous combination is performed as follows:

$$R(z) = R^*(z) + \bar{e}(z) = (R_1^*(z) + \Delta R_1(z), R_2(z) + 0, R_3(z) + 0, R_4(z) + 0) = (R_1(z), R_2(z), R_3(z), R_4(z)). \quad (25)$$

Second, if an error has corrupted the first information residue $R_2^*(z)$, and its depth is $\Delta R_2(z)$, for which $\deg \Delta R_2(z) < \deg g_2(z)$, then, in this case, the corrupted residue takes the form:

$$R_2^*(z) = R_2(z) + \Delta R_2(z). \quad (26)$$

Then, for the combination $R^*(z) = (R_1(z), R_2^*(z), R_3(z), R_4(z))$, the following holds:

$$\begin{cases} \lambda_1(z) = z \cdot \Delta R_2(z), \\ \lambda_2(z) = \Delta R_2(z). \end{cases} \quad (27)$$

Thus, if the error syndromes coincide, the error vector takes the form $\bar{e}(z) = (0, \Delta R_2(z), 0, 0, 0)$. Then, the correction of the erroneous combination is performed as follows:

$$R(z) = R^*(z) + \bar{e}(z) = (R_1(z) + 0, R_2^*(z) + \Delta R_2(z), R_3(z) + 0, R_4(z) + 0) = (R_1(z), R_2(z), R_3(z), R_4(z)). \quad (28)$$

Third, if an error has corrupted the first information residue $R_3^*(z)$, and its depth is $\Delta R_3(z)$, for which $\deg \Delta R_3(z) < \deg g_3(z)$, then, in this case, the corrupted residue takes the form:

$$R_3^*(z) = R_3(z) + \Delta R_3(z). \quad (29)$$

Then, for the combination $R^*(z) = (R_1(z), R_2(z), R_3^*(z), R_4(z))$, the following holds

$$\begin{cases} \lambda_1(z) = \Delta R_3(z), \\ \lambda_2(z) = 0. \end{cases} \quad (30)$$

Thus, if the error syndromes coincide, the error vector takes the form $\bar{e}(z) = (0, 0, \Delta R_3(z), 0)$. Then, the correction of the erroneous combination is performed as follows:

$$R(z) = R^*(z) + \bar{e}(z) = (\Delta R_1(z) + 0, R_2(z) + 0, R_3^*(z) + \Delta R_3(z), R_4(z) + 0) = (R_1(z), R_2(z), R_3(z), R_4(z)). \quad (31)$$

Fourth, if an error has corrupted the first information residue $R_4^*(z)$, and its depth is $\Delta R_4(z)$, for which $\deg \Delta R_4(z) < \deg g_3(z)$, then, in this case, the corrupted residue takes the form:

$$R_4^*(z) = R_4(z) + \Delta R_4(z). \quad (32)$$

Then, for the combination $R^*(z) = (R_1(z), R_2(z), R_3(z), R_4^*(z))$, the following holds:

$$\begin{cases} \lambda_1(z) = 0, \\ \lambda_2(z) = \Delta R_4(z). \end{cases} \quad (33)$$

Thus, if the error syndromes coincide, the error vector takes the form $\bar{e}(z) = (0, 0, 0, \Delta R_4(z))$. Then, the correction of the erroneous combination is performed as follows:

$$R(z) = R^*(z) + \bar{e}(z) = (R_1^*(z) + 0, R_2(z) + 0, R_3(z) + 0, R_4^*(z) + \Delta R_4(z)) = (R_1(z), R_2(z), R_3(z), R_4(z)). \quad (34)$$

Analysis of expressions (24), (27), (30), and (34) shows that by using the developed method, one can unambiguously determine the basis in which a single computational error occurred, as well as its depth.

Let us consider the choice of the check basis used to compute the first check residue, as in (16). It is known that there are three irreducible fourth-degree polynomials. If the two polynomials $g_1(z) = z^4 + z + 1$ and $g_2(z) = z^4 + z^3 + 1$ are used as the code's information bases, then $g_3(z) = z^4 + z^3 + z^2 + z + 1$ can be chosen as the check basis. However, choosing this polynomial will not result in minimal circuit complexity. This is because calculating the first error syndrome $\lambda_1(z)$ modulo this modulus requires three modulo-2 adders with four inputs and one three-input adder. In this case, the combinational circuit of the four-input adder contains eight four-input AND gates, as well as one OR gate with eight inputs.

The circuit complexity of implementing expression (16) can be reduced by using $g_1(z) = z^4 + z + 1$ a polynomial instead $g_3(z) = z^4 + z^3 + z^2 + z + 1$. In this case, calculating the first error syndrome $\lambda_1(z)$ modulo $g_1(z) = z^4 + z + 1$ requires one four-input and three three-input modulo adders. In this case, the combinational circuit of a three-input modulo-1 adder contains four three-input AND gates, as well as one OR gate with four inputs. Thus, it is evident that the implementation of a three-input modulo $g_1(z) = z^4 + z + 1$ adder will require half the circuit complexity compared to the implementation of a four-input adder. Therefore, the developed error-correcting decoding algorithm takes the following form:

$$\begin{aligned} R_3(z) &= R_1(z) + zR_2(z) \bmod g_1(z), \\ R_4(z) &= R_1(z) + R_2(z). \end{aligned} \quad (35)$$

3.2. Mathematical Models of the SBCG Encryption/Decryption Round, Implemented in Modular Code

3.2.1. Mathematical Models of Encryption Transformations, Implemented in Modular Code

According to [54], SBCG provides encryption of a 128-bit plaintext block over 10 rounds. The first nine rounds are complete, while the final tenth round consists only of the X-mixing operation with the tenth-round key. All nine rounds have the same structure, which includes the following operations:

- Mixing X;
- Transformation S;
- Transformation L.

Let us examine them in more detail. A 128-bit input block, consisting of 16 bytes of plaintext, is fed to the first inputs of the modulo-two adders. The first-round key is fed into the second input. The adders perform a mixing X operation with the first-round key.

$$a(u) = X(u) + K^1(u), \quad (36)$$

where $X(u)$ is the u -th byte of plaintext; $K^1(u)$ is the u -th byte of the first-round key; $u = 0, 1, \dots, 15$.

Since it was proposed to use PCM to improve the fault tolerance of the SBCG cipher, before performing the first round of encryption, the bytes must be represented in modular code over the base $g_1(z) = z^4 + z + 1$ and $g_2(z) = z^4 + z^3 + 1$. This operation is called the conversion from positional code (PC) to PCM code. It is implemented as follows:

$$X_i(u) = \sum_{n=0}^7 x_n(u) \beta_i^n, \quad (37)$$

where $X_i(u) \equiv X(u) \bmod g_i(z)$; $x_n(u)$ —the n -th bit of the u -th byte of the plaintext; β_i^n —a nonzero element of $GF(2^4)$; $u = 0, 1, \dots, 15$; $i = 1, 2$.

A similar operation is performed with the round keys:

$$K_i^s(u) = \sum_{n=0}^7 k_n^s(u) \beta_i^n, \quad (38)$$

where $K_i^s(u) \equiv K^s(u) \bmod g_i(z)$; $k_n^s(u)$ —the n -th bit of the u -th byte of the s -th round key; β_i^n —a nonzero element of $GF(2^4)$; $u = 0, 1, \dots, 15$; $i = 1, 2$; $s = 1, \dots, 10$.

Then, in the modular code, the mixing operation with the first-round key takes the form:

$$a_i(u) = X_i(u) + K_i^1(u). \quad (39)$$

The result, obtained using expression (39), is fed to the input of the S block. It performs a nonlinear transformation in SBCG. The nonlinear transformation involves replacing the input byte with another byte. As a result, the static dependencies between the plaintext and ciphertext are disrupted. Figure 1 shows a block diagram of the nonlinear transformer used in SBCG.

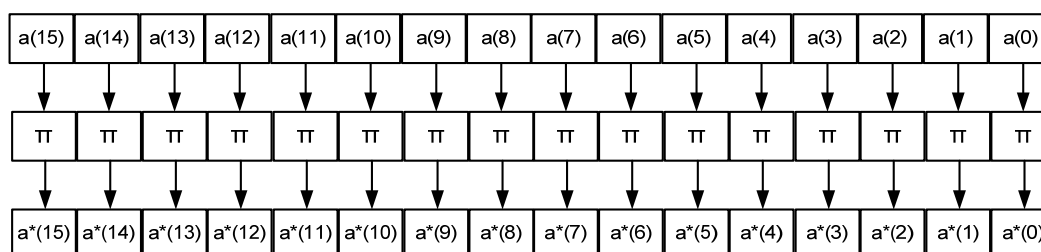


Figure 1. Structure diagram of the nonlinear converter block.

In SBCG, the nonlinear transformer replaces the input byte with another byte

$$\ddot{a}(u) = \pi(a(u)), \quad (40)$$

where $\ddot{a}(u)$ is the byte at the output of the S-transformer; $\pi(a(u))$ —the replacement operation.

The replacement $\pi(a(u))$ depends on the value of the input byte and is determined by the rule:

$$\pi = (252, 238, 221, \dots, 99, 182). \quad (41)$$

In this case, if $a(u) = 0$, then $\ddot{a}(u) = \pi(0) = 252$. And if $a(u) = 255$, then $\ddot{a}(u) = \pi(255) = 182$.

To improve the fault tolerance of the LFSR performing the nonlinear S transformation in the PMC, a table-based implementation of constant multiplication operations is proposed. In this case, four tables will be used. Let us examine their contents. Since the input byte arriving at the input of the block S contains two residues:

$$a(u) = (a_1(u), a_2(u)). \quad (42)$$

They determine the row and column of the tables. In the first table, at the intersection of row $a_1(u)$ and column, $a_2(u)$ the first information residue of the output byte is located:

$$\ddot{a}_1(u) = \pi(a(u)) \bmod g_1(z). \quad (43)$$

In the second table, at the intersection of the row $a_1(u)$ and the column, $a_2(u)$ the second information residue is located:

$$\ddot{a}_2(u) = \pi(a(u)) \bmod g_2(z), \quad (44)$$

In the third table, at the intersection of the row $a_1(u)$ and the column, $a_2(u)$ the first check residue is located, calculated based on the expression:

$$\ddot{a}_3(u) = \ddot{a}_1(u) + z \ddot{a}_2(u) \bmod g_1(z). \quad (45)$$

In this case, unlike in expression (16), the first information module $g_1(z) = z^4 + z + 1$ is used as the check basis. This allows for a reduction in circuit costs compared to $g_3(z) = z^4 + z^3 + z^2 + z + 1$, as confirmed by [55].

In the fourth table, at the intersection of the row $a_1(u)$ and the column $a_2(u)$ is the second check residue, calculated based on the expression:

$$\ddot{a}_4(u) = \ddot{a}_1(u) + \ddot{a}_2(u). \quad (46)$$

The check residues obtained using expressions (45) and (46) correspond to a valid PMC combination that contains no errors. Thus, the nonlinear transformation in SBCG is represented as the code combination

$$\pi(a(u)) = \ddot{a}(u) = (\ddot{a}_1(u), \ddot{a}_2(u), \ddot{a}_3(u), \ddot{a}_4(u)). \quad (47)$$

The approach to performing a nonlinear transformation in PMC discussed above was used to develop a mathematical model of the S block, operating in PMC and possessing the fault-tolerance property. It consists of the following steps:

1. The input of the fault-tolerant converter S of the SBCG cipher receives code combinations containing two remainders

$$a(0) = (a_1(0), a_2(0)), \dots, a(15) = (a_1(15), a_2(15)). \quad (48)$$

2. A nonlinear S block is performed in PMCs based on expression (47).
3. A procedure is performed to search for and correct distorted residues in the calculated byte $(\ddot{a}_1(u), \ddot{a}_2(u), \ddot{a}_3(u), \ddot{a}_4(u))$. According to the developed algorithm, using expression (35), the values of the check residues are determined

$$\begin{aligned} \tilde{a}_3(u) &= \ddot{a}_1(u) + z \ddot{a}_2(u) \bmod g_1(x), \\ \tilde{a}_4(u) &= \ddot{a}_1(u) + \ddot{a}_2(u). \end{aligned} \quad (49)$$

4. The error syndrome is calculated. Expression (18) is used for this. In this expression, the check residues of the valid PMC combination are taken from Tables 3 and 4. If the result is zero, the code combination is considered undistorted. In this case, no correction is performed. If the error syndrome satisfies condition (24), the first information residue is corrected according to (25). If condition (27) is satisfied, the second information residue is corrected. Expression (28) is used for this purpose. If the error syndrome satisfies condition (30), the first check residue is corrected according to (31). If condition (33) is satisfied, the second check residue is corrected. Expression (34) is used for this purpose.

After the nonlinear transformation is performed, the 16 bytes, which are represented as PMC combination (47), are fed to the input of the fault-tolerant block L. This block is required to perform linear transformations in the SBCG.

The L-to-SBCG transformation is performed using a linear feedback shift register (LFSR). Figure 2 shows a block diagram of the LFSR used in the SBCG cipher.

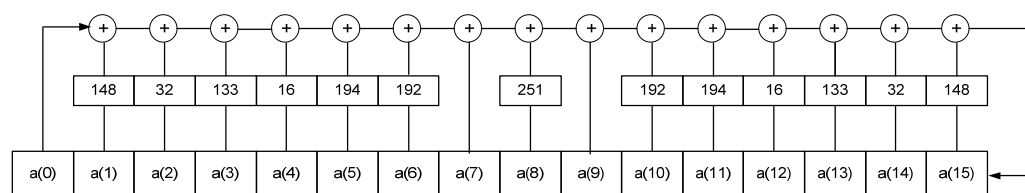


Figure 2. Structural model of the LFSR used in the SBCG encoder.

The linear feedback shift register (LFSR) contains 16 memory elements used to store bytes $a(0) - a(15)$. These bytes are the results of the X-mixing operation performed in the previous encryption round. The LFSR operates over 16 clock cycles. A new byte value is computed on each clock cycle. In this process, the bytes $a(0) - a(15)$ and the constants $h = (148, 32, 133, 16, 194, 192, 251)$ are represented as polynomials. The multiplication operation is performed modulo $g(x) = x^8 + x^7 + x^6 + x + 1$. The resulting products are added modulo two. To place the calculated byte in place of the high-order byte of the LFSR, a left shift of one byte is performed. This process is described by the expression

$$L(a(15) \dots a(0)) = \nabla(148\ddot{a}(15) + 32\ddot{a}(14) + 133\ddot{a}(13) + 16\ddot{a}(12) + 194\ddot{a}(11) + 192\ddot{a}(10) + \ddot{a}(9) + 251\ddot{a}(8) + \ddot{a}(7) + 192\ddot{a}(6) + 194\ddot{a}(5) + 16\ddot{a}(4) + 133\ddot{a}(3) + 32\ddot{a}(2) + 148\ddot{a}(1) + \ddot{a}(0)) \bmod g(x), \quad (50)$$

where ∇ —one-byte left shift; $\ddot{a}(0) - \ddot{a}(15)$ —current byte value.

To improve the fault tolerance of the LFSR performing the linear transformation L into PMC, a table-based implementation of constant multiplication is proposed. In this case, four tables will be used. Let us examine their contents.

Since the input byte received at the input of the LFSR contains four remainders

$$a(u) = (a_1(u), a_2(u), a_3(u), a_4(u)), \quad (51)$$

Then the information residues $a_1(u) = a(u) \bmod g_1(x)$ and $a_2(u) = a(u) \bmod g_2(x)$ will determine the row and column of the tables. In the first table, at the intersection of row $a_1(u)$ and column $a_2(u)$, the first information residue of the product by the feedback constant is located:

$$\ddot{a}_1(u) = \left(|\ddot{a}(u)C(u)|_{g(z)} \right) \bmod g_1(z), \quad (52)$$

where $C(u)$ is the LFSR feedback constant.

In the second table, at the intersection of row $a_1(u)$ and column $a_2(u)$, the second information residue of the product by the feedback constant is located:

$$\ddot{a}_2(u) = \left(|\ddot{a}(u)C(u)|_{g(z)} \right) \bmod g_2(z). \quad (53)$$

In the third table, at the intersection of row $a_1(u)$ and column $a_2(u)$, the first check residue is calculated based on expression (35). In this case, it is proposed to use the first information module $g_1(z) = z^4 + z + 1$ as the basis for the check. In the fourth table, at the intersection of row $a_1(u)$ and column $a_2(u)$, the second check residue, is calculated based on expression (35).

As a result, the product of the current byte and the feedback constant is represented as a code combination

$$|\ddot{a}(u)C(u)|_{g(z)} = (\ddot{a}_1(u), \ddot{a}_2(u), \ddot{a}_3(u), \ddot{a}_4(u)). \quad (54)$$

If the feedback constant is equal to one, then the product is represented as the code combination (51).

The approach to performing a linear transformation in PMC discussed above was used in developing a mathematical model of block L, operating in PMC and possessing the fault-tolerance property. It consists of the following stages:

1. The fault-tolerant LFSR receives 16 bytes at its input, which are the result of the X-mixing operation in the previous round. These bytes are represented as code combinations (51). The byte combinations are written into the LFSR.
2. To compute the new V -th byte $a^V(15)$, where $V = 1, \dots, 16$, the current bytes are first multiplied by the feedback constants. This yields the PMC combinations. Expressions (52) and (53) are used to compute the information residues. The parity residues are computed:

$$\begin{aligned}\ddot{a}_3(u) &= (\ddot{a}_1(u) + z\ddot{a}_2(u)) \bmod g_1(z), \\ \ddot{a}_3(u) &= \ddot{a}_1(u) + \ddot{a}_2(u).\end{aligned}\quad (55)$$

3. If the feedback constant is equal to one, then the redundant combination (51) is taken as the result of the multiplication.
4. The two resulting products are summed modulo two. All four residues are included in the calculations.

$$\ddot{a}_i^V(15) = \sum_{n=0}^{15} \ddot{a}_i(j), \quad (56)$$

where $i = 1, \dots, 4$; $V = 1, \dots, 16$.

5. A procedure is performed to search for and correct distorted residues in the calculated byte $\ddot{a}^V(15) = (\ddot{a}_1^V(15), \ddot{a}_2^V(15), \ddot{a}_3^V(15), \ddot{a}_4^V(15))$. According to the developed algorithm, the values of the check residues are determined using expression (35):

$$\begin{aligned}\tilde{a}_3(15) &= \ddot{a}_1^V(15) + z\ddot{a}_2^V(15) \bmod g_1(z), \\ \tilde{a}_4(15) &= \ddot{a}_1^V(15) + \ddot{a}_2^V(15).\end{aligned}\quad (57)$$

Using (18), the error syndrome is calculated. If the result is zero, the code combination is considered undistorted. In this case, no correction is performed. If the error syndrome satisfies condition (24), the first information residue is corrected according to (25). If condition (27) is satisfied, the second information residue is corrected. Expression (28) is used for this purpose. If the error syndrome satisfies condition (30), the first check residue is corrected according to (31). If condition (33) is satisfied, the second check residue is corrected. Expression (34) is used for this purpose.

6. A left shift of one byte is performed in the LFSR. In this case, the PMC combination of the first byte is written to the least significant byte $a_i(u)$ position of the LFSR. At the same time, the code combination $(\ddot{a}_1^V(15), \ddot{a}_2^V(15), \ddot{a}_3^V(15), \ddot{a}_4^V(15))$ written to the LFSR is considered the most significant byte.
7. The process returns to step 2. The number of iterations is 16.

After 16 clock cycles, all bytes in the LFSR will have been replaced. This concludes the round of operation for the fault-tolerant cipher.

3.2.2. Mathematical Models of Decryption Transformations Implemented in Modular Ciphers

In modern block SPN ciphers, decryption differs from encryption. Consequently, a single round of decryption consists of:

- The L^{-1} transformation, which is the inverse of the linear transformation;
- An S^{-1} transformation, which is the inverse of a nonlinear transformation;

- X-mixing (modulo-2 addition) with the round key.

When performing the inverse linear transformation, an LFSR is used, whose structure matches that of the encoder's LFSR. However, the register loading order has been changed. In this case, the plaintext input is fed into the shift register with the most significant bytes first. The process of computing the byte is defined by the expression:

$$\begin{aligned} L(\ddot{a}(0) \dots \ddot{a}(15)) = & \nabla(148a(0) + 32a(1) + 133a(2) + 16a(3) + \\ & + 194a(4) + 192a(5) + a(6) + 251a(7) + a(8) + 192a(9) + \\ & + 194a(10) + 16a(11) + 133a(12) + 32a(13) + 148a(14) + a(15)) \bmod g(x), \end{aligned} \quad (58)$$

where ∇ —one-byte left shift; $a(0) - a(15)$ —the current value of the bytes.

As in the cipher, when performing the inverse linear transformation in the PMC, it is proposed to use a table-based implementation of multiplication by constants. In this case, four tables will be used. The contents of the tables and their purpose coincide with those of the fault-tolerant LFSR of the encoder.

In this case, the mathematical model of the fault-tolerant LFSR block, the decoder, consists of the following stages:

1. The fault-tolerant LFSR receives 16 bytes at its input, which are the result of the X-mixing operation from the previous round. These bytes are represented as PMC combinations according to (51). The code combinations of the bytes are written into the LFSR.
2. To compute the new V -th byte $\ddot{a}^V(0)$, where $V = 1, \dots, 16$, the current bytes are first multiplied by the feedback constants. This yields PMC combinations. In this process, expressions (52) and (53) are used to compute the information residues. The parity residues are computed:

$$\begin{aligned} \ddot{a}_3(u) &= (\ddot{a}_1(u) + z\ddot{a}_2(u)) \bmod g_1(z), \\ \ddot{a}_3(u) &= \ddot{a}_1(u) + \ddot{a}_2(u). \end{aligned} \quad (59)$$

3. If the feedback constant is equal to one, then the redundant combination (51) is taken as the result of the multiplication.
4. The two resulting products are summed modulo two. All four residues are included in the calculations.

$$\ddot{a}_i^V(0) = \sum_{n=0}^{15} \ddot{a}_i(u), \quad (60)$$

where $i = 1, \dots, 4$; $V = 1, \dots, 16$.

5. A procedure is performed to search for and correct distorted residues in the calculated byte $\ddot{a}^V(0) = (\ddot{a}_1^V(0), \ddot{a}_2^V(0), \ddot{a}_3^V(0), \ddot{a}_4^V(0))$. According to the developed algorithm, the values of the check residues are determined using expression (35):

$$\begin{aligned} \tilde{a}_3(0) &= \ddot{a}_1^V(0) + z\ddot{a}_2^V(0) \bmod g_1(z), \\ \tilde{a}_4(0) &= \ddot{a}_1^V(0) + \ddot{a}_2^V(0). \end{aligned} \quad (61)$$

Using (18), the error syndrome is calculated. If the result is zero, the code combination is considered undistorted. In this case, no correction is performed. If the error syndrome satisfies condition (24), the first information residue is corrected according to (25). If condition (27) is satisfied, the second information residue is corrected. Expression (28) is used for this purpose. If the error syndrome satisfies condition (30), the first check residue is corrected according to (31). If condition (33) is satisfied, the second check residue is corrected. Expression (34) is used for this purpose.

6. The LFSR is shifted one byte to the left. In this case, the PMC combination of the fourteenth byte $a_i(14)$ is written to the least significant byte of the LFSR. At the same time, the code combination $(\ddot{a}_1^V(0), \ddot{a}_2^V(0), \ddot{a}_3^V(0), \ddot{a}_4^V(0))$ is written to the LFSR and is considered the most significant byte.
7. The process returns to step 2. The number of iterations is 16.

After 16 clock cycles, all bytes in the LFSR will have been replaced.

After this, the code combinations,

$$\ddot{a}(0) = (\ddot{a}_1^1(0), \ddot{a}_2^1(0)), \dots, \ddot{a}(15) = (\ddot{a}_1^{16}(0), \ddot{a}_2^{16}(0)), \quad (62)$$

are fed to the input of the fault-tolerant block implementing the S^{-1} transformation. In the SBCG, this converter replaces the input byte with another byte

$$\ddot{a}'(u) = \pi^{-1}(\ddot{a}(u)), \quad (63)$$

where $\ddot{a}(u)$ is the byte at the converter's output; $\pi^{-1}(\ddot{a}(u))$ is the replacement operation, the inverse of the replacement $\pi(a(u))$.

The transformation $\pi^{-1}(\ddot{a}(u))$ depends on the value of the input byte $\ddot{a}(u)$ and is determined by the rule

$$\pi^{-1} = (165, 45, 50, \dots, 215, 116). \quad (64)$$

In this case, if $\ddot{a}(u) = 0$, then $\ddot{a}'(u) = \pi^{-1}(0) = 165$. And if $\ddot{a}(u) = 255$, then $\ddot{a}'(u) = \pi^{-1}(255) = 116$.

The information residues of the input combination (61) will determine the row and column in the four tables of the fault-tolerant S^{-1} transformation block in the SBCG. In the first table, at the intersection of row $\ddot{a}_1(u)$ and column $\ddot{a}_2(u)$ the first information residue of the product by the feedback constant is located:

$$\ddot{a}'_1(u) = \pi^{-1}(\ddot{a}(u)) \bmod g_1(z). \quad (65)$$

In the second table, at the intersection of row $\ddot{a}_1(u)$ and column $\ddot{a}_2(u)$ the second information residue is located:

$$\ddot{a}'_2(u) = \pi^{-1}(\ddot{a}(u)) \bmod g_2(z). \quad (66)$$

In the third table, at the intersection of row $\ddot{a}_1(u)$ and column $\ddot{a}_2(u)$, the first check residue is located, calculated based on expression (35). Here, the information modulus $g_1(z) = z^4 + z + 1$ is used as the check base. In the fourth table, at the intersection of row $\ddot{a}_1(u)$ and column $\ddot{a}_2(u)$, the second check residue is located, calculated based on expression (35).

As a result, the output byte is represented as the code combination

$$\pi^{-1}(\ddot{a}(u)) = \ddot{a}'(u) = (\ddot{a}'_1(u), \ddot{a}'_2(u), \ddot{a}'_3(u), \ddot{a}'_4(u)). \quad (67)$$

Then, the mathematical model of the fault-tolerant block implementing the transformation $\pi^{-1}(\ddot{a}(u))$ into PMC consists of the following steps:

1. Code combinations are fed to the input of the fault-tolerant SBCG converter in accordance with Equation (62).
2. A nonlinear transformation is performed in PCM codes based on expression (67).

3. A procedure for searching for and correcting distorted residues in the calculated byte $(\ddot{a}_1(u), \ddot{a}_2(u), \ddot{a}_3(u), \ddot{a}_4(u))$. is performed. According to the developed algorithm, the values of the check residues are determined using expression (35):

$$\begin{aligned}\tilde{a}_3(15) &= \ddot{a}_1(15) + z \ddot{a}_2(15) \bmod g_i(z), \\ \tilde{a}_4(15) &= \ddot{a}_1(15) + \ddot{a}_2(15).\end{aligned}\quad (68)$$

4. The error syndrome is calculated. Expression (18) is used for this. If the result is zero, the code combination is considered undistorted. No correction is performed in this case. If the error syndrome satisfies condition (24), the first information residue is corrected according to (25). If condition (27) is satisfied, the second information residue is corrected. Expression (28) is used for this purpose. If the error syndrome satisfies condition (30), the first check residue is corrected according to (31). If condition (33) is satisfied, the second check residue is corrected. Expression (34) is used for this purpose.

Then, the results of the nonlinear transformation are mixed with the round key $K^s(u)$. This operation involves code combinations containing four residues each. The control residues of the round key are determined using expression (35). Then, the procedure X (modulo-two summation) can be represented as follows:

$$A_i(u) = \ddot{a}_i(u) + K_i^s(u). \quad (69)$$

This concludes one round of decryption in SBCG.

Let us consider an example of using the developed mathematical models of transformations in RMC when performing one round of encryption and decryption in SBCG.

4. Results

Let us analyze the performance of a fault-tolerant system implementing the SBCG encryption algorithm in PMCs. Let us consider a single round of encryption and decryption. To evaluate the fault tolerance of the SBCG cipher system using PMC, we will first perform the encryption and decryption process in a positional code. Suppose a 128-bit plaintext block is fed into the cipher. In Table 1, the plaintext block is represented as 16 bytes.

Table 1. (a) Plaintext block (first group of 8 bytes). (b) Plaintext block (second group of 8 bytes).

(a)							
$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$
73	82	5	11	210	21	48	154
(b)							
$X(8)$	$X(9)$	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$	$X(15)$
97	101	183	61	29	231	210	154

The bytes are represented in the decimal number system, that is, $X(0) = 73 = 01001001 = x^6 + x^3 + 1$.

Table 2 shows the secret key for the first round.

Table 2. (a) Secret key for the first round (first group of 8 bytes). (b) Secret key for the first round (second group of 8 bytes).

(a)							
$K^1(0)$	$K^1(1)$	$K^1(2)$	$K^1(3)$	$K^1(4)$	$K^1(5)$	$K^1(6)$	$K^1(7)$
143	174	200	8	157	101	32	74
(b)							
$K^1(8)$	$K^1(9)$	$K^1(10)$	$K^1(11)$	$K^1(12)$	$K^1(13)$	$K^1(14)$	$K^1(15)$
24	17	165	220	250	19	110	45

Table 3 presents the results of the transformation X.

Table 3. (a) Result of the transformation X (first group of 8 bytes). (b) Result of the transformation X (second group of 8 bytes).

(a)							
$a(0)$	$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$	$a(7)$
198	252	205	3	79	112	16	208
(b)							
$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$	$a(13)$	$a(14)$	$a(15)$
121	116	18	225	231	244	188	183

In this case, the first byte of the sum is next:

$$\begin{aligned}
 a(0) &= X(0) + K(0) = 73 + 143 = \\
 &= 01001001 + 10001111 = 11000110 = 198.
 \end{aligned}$$

The result of the modulo-2 summation is fed to the input of the block implementing the nonlinear transformation, as in (40). The results of this transformation are presented in Table 4.

Table 4. (a) Result of the S-transformation (first group of 8 bytes). (b) Result of the S-transformation (second group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
29	57	217	17	204	50	233	225
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
84	255	240	113	91	230	105	47

For the first byte, the following holds true: $\ddot{a}(0) = \pi(a(0)) = \pi(198) = 29$.

The results of the S-substitution are fed to the input of the linear transformer. Table 5 shows the results of the LSFR operation after 16 cycles.

Table 5. (a) Result of the LSFR operation after 16 cycles (first group of 8 bytes). (b) Result of the LSFR operation after 16 cycles (second group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
10	79	130	105	93	160	104	126
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
211	11	129	143	36	66	24	180

For the first byte, we obtain

$$\begin{aligned} \ddot{a}^1(15) = & L(29 + 57\Delta 148 + 217\Delta 32 + 17\Delta 133 + 204\Delta 16 + 50\Delta 194 + 233\Delta 192 + \\ & + 225 + 84\Delta 251 + 255 + 240\Delta 192 + 113\Delta 194 + 91\Delta 16 + 230\Delta 133 + 105\Delta 32 + \\ & + 47\Delta 148) \bmod g(x) = 10 = 00001010. \end{aligned}$$

This completes one round of encryption. The ciphertext in the form of bytes is shown in Table 6.

Table 6. (a) Ciphertext after one round of encryption (first group of 8 bytes). (b) Ciphertext after one round of encryption (second group of 8 bytes).

(a)							
$\tilde{C}(0)$	$\tilde{C}(1)$	$\tilde{C}(2)$	$\tilde{C}(3)$	$\tilde{C}(4)$	$\tilde{C}(5)$	$\tilde{C}(6)$	$\tilde{C}(7)$
10	79	130	105	93	160	104	126
(b)							
$\tilde{C}(8)$	$\tilde{C}(9)$	$\tilde{C}(10)$	$\tilde{C}(11)$	$\tilde{C}(12)$	$\tilde{C}(13)$	$\tilde{C}(14)$	$\tilde{C}(15)$
211	11	129	143	36	66	24	180

Let us perform one round of decryption. The ciphertext, in the form of bytes, is written to the shift register. In this case, the bytes are written in high to low order.

The results of performing the inverse linear transformation after 16 cycles of the decoder are presented in Table 7.

Table 7. (a) Result of the L^{-1} transformation (first group of 8 bytes). (b) Result of the L^{-1} transformation (second group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
29	57	217	17	204	50	233	225
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
84	255	240	113	91	230	105	47

The results of the L^{-1} transformation are fed to the block that implements S^{-1} . The results are presented in Table 8.

Table 8. (a) Results of the S^{-1} transformation (first group of 8 bytes). (b) Results of the S^{-1} transformation (second group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
198	252	205	3	79	112	16	208
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
121	116	18	225	231	244	188	183

The result is XORed with the round key. The result is the plaintext shown in Table 9.

Table 9. (a) Plaintext block (first group of 8 bytes). (b) Plaintext block (second group of 8 bytes).

(a)							
$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$
73	82	5	11	210	21	48	154
(b)							
$X(8)$	$X(9)$	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$	$X(15)$
97	101	183	61	29	231	210	154

Let us consider the execution of a single round of encryption using the PMC. In the first stage, the SBCG fault-tolerant encoder performs the conversion from the PC position code to the PMC. The code combinations obtained using algorithm (62) are presented in Table 10.

Table 10. (a) Plaintext block in PMC (first group of 8 bytes). (b) Plaintext block in PMC (second group of 8 bytes).

(a)							
$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$
(5, 6)	(13, 4)	(5, 5)	(2, 8)	(6, 3)	(6, 12)	(5, 2)	(2, 4)
(b)							
$X(8)$	$X(9)$	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$	$X(15)$
(11, 5)	(15, 1)	(9, 2)	(8, 15)	(14, 4)	(6, 4)	(6, 3)	(2, 4)

For the first byte $X(0) = 73$, according to (62), we have

$$X_1(0) = |73|_{g_1(x)} = \left| x^6 + x^3 + 1 \right|_{x^4+x+1} = \beta^6 + \beta^3 + 1 = 1100 + 1000 + 0001 = 0101 = 5$$

$$X_2(0) = |73|_{g_2(x)} = \left| x^6 + x^3 + 1 \right|_{x^4+x^3+1} = \beta^6 + \beta^3 + 1 = 1111 + 1000 + 0001 = 0110 = 6.$$

Table 11 presents the PMC combinations of the round key.

Table 11. (a) Secret key for the first round (first group of 8 bytes). (b) Secret key for the first round (second group of 8 bytes).

(a)							
$K^1(0)$	$K^1(1)$	$K^1(2)$	$K^1(3)$	$K^1(4)$	$K^1(5)$	$K^1(6)$	$K^1(7)$
(4, 8)	(3, 2)	(15, 0)	(8, 8)	(5, 3)	(15, 1)	(6, 11)	(6, 5)
(b)							
$K^1(8)$	$K^1(9)$	$K^1(10)$	$K^1(11)$	$K^1(12)$	$K^1(13)$	$K^1(14)$	$K^1(15)$
(11, 1)	(2, 8)	(8, 9)	(8, 13)	(8, 0)	(0, 10)	(4, 10)	(11, 6)

The result of the key overlay operation in the form of PMC combinations PMC is presented in Table 12.

Table 12. (a) Result of the transformation X (first group of 8 bytes). (b) Result of the transformation X (second group of 8 bytes).

(a)							
$a(0)$	$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$	$a(6)$	$a(7)$
(1, 14)	(14, 6)	(10, 5)	(3, 3)	(3, 0)	(9, 13)	(3, 9)	(4, 1)
(b)							
$a(8)$	$a(9)$	$a(10)$	$a(11)$	$a(12)$	$a(13)$	$a(14)$	$a(15)$
(0, 4)	(13, 9)	(1, 11)	(0, 2)	(6, 4)	(6, 14)	(2, 9)	(9, 2)

For byte $a(0) = (a_1(0), a_2(0))$ according to (39), we have:

$$a_1(0) = A_1(0) + K_1(0) = 5 + 4 = 0101 + 0100 = 0001 = 1,$$

$$a_2(0) = A_2(0) + K_2(0) = 6 + 8 = 0110 + 1000 = 1110 = 14.$$

The result of the modulo-two summation is fed to the input of the block implementing the nonlinear transformation. The results in the form of $\ddot{a}(u) = (\ddot{a}_1(u), \ddot{a}_2(u), \ddot{a}_3(u), \ddot{a}_4(u))$ are presented in Table 13.

Table 13. Result of the S-transformation implementation.

$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$
(14, 4, 6, 10)	(12, 11, 9, 7)	(13, 8, 14, 5)	(2, 8, 1, 10)
$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
(11, 4, 3, 15)	(7, 0, 7, 7)	(8, 10, 15, 2)	(0, 2, 4, 2)
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$
(11, 2, 15, 9)	(13, 5, 7, 8)	(2, 10, 5, 8)	(8, 12, 3, 4)
$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
(4, 13, 13, 9)	(7, 5, 13, 2)	(3, 13, 10, 14)	(9, 4, 1, 13)

To implement this transformation in PMC, four tables were used, which are presented in Figure 3. The tables used in the linear transformation block have a similar structure. A combination $a(0) = (1, 14)$ is fed into these tables. In the first table, at the intersection of the first row and the 14th column, is the first remainder $\ddot{a}_1(0) = 14$ of the number $\ddot{a}(0) = 29$ in the first PMC base. This situation is shown in Figure 3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	7	8	15	11	0	8	0	13	12	15	13	4	14	1	5
1	14	15	15	6	10	7	2	9	2	6	14	2	7	2	14	5

Figure 3. Obtaining the remainder using the first table.

In the second table, the second remainder $\ddot{a}_2(0) = 4$ of the number $\ddot{a}(0) = 29$ in the second PMC base is located at the intersection of the first row and the 14th column. This situation is shown in Figure 4.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	6	1	12	7	2	3	13	14	9	0	10	11	12	12	5	13
1	0	13	8	11	13	9	0	11	11	3	9	10	10	7	4	11

Figure 4. Obtaining the remainder using the second table.

In the third table, the third remainder $\ddot{a}_3(0) = 6$ of the number $\ddot{a}(0) = 29$ is found at the intersection of the first row and the 14th column. This situation is shown in Figure 5.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	5	3	2	15	6	1	15	12	12	8	8	15	5	11	12
1	14	6	12	3	3	6	2	12	7	0	15	5	0	12	6	0

Figure 5. Obtaining the remainder using the third table.

In the fourth table, the remainder $\ddot{a}_4(0) = 10$ of the number $\ddot{a}(0) = 29$ is located at the intersection of the first row and the 14th column. This situation is shown in Figure 6.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	8	6	4	8	9	3	5	14	4	12	5	6	8	2	4	8
1	14	2	7	13	7	14	2	2	9	5	7	8	13	5	10	14

Figure 6. Obtaining the remainder using the fourth table.

Thus, the code combination takes the form. Suppose a failure occurred during the nonlinear transformation, which led to a distortion of the second residue. As a result, the combination $\ddot{a}(0) = (14, 4, 6, 10) = (z^3 + z^2 + z, z^2, z^2 + z, z^3 + z)$ was removed from the block output instead of combination $\ddot{a}^*(0) = (14, 0^*, 6, 10) = (z^3 + z^2 + z, 0, z^2 + z, z^3 + z)$, output from the block and fed to the input of the error correction device. In this device, using expression (49), the redundant residues are calculated:

$$\begin{aligned}\tilde{a}_3(0) &= z^3 + z^2 + z = 1110 = 14, \\ \tilde{a}_4(0) &= z^3 + z^2 + z = 1110 = 14.\end{aligned}$$

To calculate the error syndrome, expression (18) is used.

$$\begin{aligned}\lambda_1(0) &= \ddot{a}_3(0) + \tilde{a}_3(0) = ((z^2 + z) + (z^3 + z^2 + z)) = z^3, \\ \lambda_2(0) &= \ddot{a}_4(0) + \tilde{a}_4(0) = ((z^3 + z) + (z^3 + z^2 + z)) = z^2.\end{aligned}$$

For this error syndrome, we have an error vector $\bar{e}(z) = (0, z^2, 0, 0, 0)$ n. We use expression (29) to correct the code combination:

$$\begin{aligned}\ddot{a}(0) &= \ddot{a}^*(0) + \bar{e} = (z^3 + z^2 + z, 0, z^2 + z, z^3 + z) + (0, z^2, 0, 0) = \\ &= (z^3 + z^2 + z, z^2, z^2 + z, z^3 + z).\end{aligned}$$

The corrected code combination $\ddot{a}(0) = (14, 4, 6, 10)$ is fed to the input of the LFSR, which performs a linear transformation.

Table 14 shows the process of calculating the first byte $\ddot{a}^1(15) = 10$ using the PMC. The result is obtained by adding, modulo two, the corresponding columns of remainders.

Table 14. Calculation of the first byte in PMC $a^1(15)$.

Remains	$\ddot{a}_1^1(15)$				$\ddot{a}_2^1(15)$				$\ddot{a}_3^1(15)$				$\ddot{a}_4^1(15)$			
Dedree	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
$C^1(0) = 29$	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	0
$C^1(1) = 66$	1	1	1	0	1	1	0	1	0	0	1	1	0	1	1	1
$C^1(2) = 28$	1	1	1	1	0	1	0	1	1	0	1	0	0	1	0	1
$C^1(3) = 2$	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0
$C^1(4) = 157$	0	1	0	1	0	0	1	1	0	1	1	0	0	0	1	1
$C^1(5) = 192$	0	1	1	1	1	0	0	0	1	1	1	1	0	1	0	0
$C^1(6) = 175$	0	0	1	0	0	1	1	1	0	1	0	1	1	1	0	0
$C^1(7) = 225$	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0
$C^1(8) = 110$	0	1	0	0	1	1	1	0	1	0	1	0	1	0	1	1
$C^1(9) = 255$	1	1	0	1	0	1	0	1	1	0	0	0	0	1	1	1
$C^1(10) = 253$	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	1
$C^1(11) = 114$	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0
$C^1(12) = 174$	0	0	1	1	0	0	1	0	0	0	0	1	0	1	1	1
$C^1(13) = 190$	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1
$C^1(14) = 190$	0	0	0	0	1	0	1	1	1	0	1	1	0	1	0	1
$C^1(15) = 232$	1	0	0	1	1	0	1	1	0	0	1	0	1	1	0	0
$a^1(15) = 10$	1	0	1	0	1	0	1	0	0	0	0	0	1	1	0	1
	10				10				0				13			

When performing the multiplication of the byte $\ddot{a}(1) = 57 = z^5 + z^4 + z^3 + 1 = (12, 11)$ and the constant $h(1) = 148 = z^7 + z^4 + z^2$, we obtain two information residues.

$$\begin{aligned}\ddot{a}_1^1(1) &= \left| \ddot{a}(1)h(1) \right|_{g(z)} \Big|_{g_1(z)} = \left| 57 \cdot 148 \right|_{p(z)} \Big|_{p_1(z)} = \left| z^6 + z \right|_{z^4+z+1} = z^3 + z^2 + z = 1110 = 14, \\ \ddot{a}_2^1(1) &= \left| \ddot{a}(1)h(1) \right|_{g(z)} \Big|_{g_2(z)} = \left| 57 \cdot 148 \right|_{p(z)} \Big|_{p_2(z)} = \left| z^6 + z \right|_{z^4+z^3+1} = z^3 + z^2 + 1 = 1101 = 13.\end{aligned}$$

These residues are located at the intersection of row 12 and column 11 of the first and second LFSR tables. Then, the check remainders according to expression (35) will be equal to:

$$\begin{aligned}\ddot{a}_3^1(1) &= \left| \ddot{a}_1^1(1) + z\ddot{a}_2^1(1) \right|_{g_1(z)} = \left| (z^3 + z^2 + z) + z(z^3 + z^2 + 1) \right|_{g_1(z)} = z^2 + z + 1 = 0011 = 7, \\ \ddot{a}_4^1(1) &= \ddot{a}_1^1(1) + \ddot{a}_2^1(1) = (z^3 + z^2 + z) + (z^3 + z^2 + 1) = z + 1 = 0011 = 3.\end{aligned}$$

These check remainders are located at the intersection of row 12 and column 11 of the third and fourth LFSR tables.

After 16 clock cycles, the LFSRs will contain the PMC ciphertext, as shown in Table 15. The ciphertext contains only information residues.

Table 15. (a) Ciphertext after one round of encryption in PMC (first group of 8 bytes). (b) Ciphertext after one round of encryption in PMC (second group of 8 bytes).

(a)							
$\tilde{C}(0)$	$\tilde{C}(1)$	$\tilde{C}(2)$	$\tilde{C}(3)$	$\tilde{C}(4)$	$\tilde{C}(5)$	$\tilde{C}(6)$	$\tilde{C}(7)$
(10, 10)	(3, 0)	(9, 5)	(3, 13)	(2, 11)	(13, 12)	(2, 12)	(7, 7)
(b)							
$\tilde{C}(8)$	$\tilde{C}(9)$	$\tilde{C}(10)$	$\tilde{C}(11)$	$\tilde{C}(12)$	$\tilde{C}(13)$	$\tilde{C}(14)$	$\tilde{C}(15)$
(7, 6)	(11, 15)	(10, 2)	(4, 12)	(2, 15)	(14, 9)	(11, 1)	(10, 5)

In the final stage of the developed method for improving the fault tolerance of SPN-SBCG ciphers, the PMC-PC inverse transformation is performed. Only information residues are involved in the computation. The orthogonal bases for this PMC are equal to:

$$B_1(z) = m_1(z)P_1(z) = (z^3 + z^2)(z^4 + z^3 + 1) = z^7 + z^5 + z^3 + z^2,$$

$$B_2(z) = m_2(z)P_2(z) = (z^3 + z + 1)(z^4 + z + 1) = z^7 + z^5 + z^3 + z^2 + 1.$$

Then, according to expression (14), we obtain the byte:

$$\tilde{C}(0) = \tilde{C}_1(0)B_1(x) + \tilde{C}_2(0)B_2(x) = (z^3 + z)(z^7 + z^5 + z^3 + z^2) + (z^3 + z)(z^7 + z^5 + z^3 + z^2 + 1) = z^3 + z = 1010 = 10.$$

All other ciphertext bytes are calculated in a similar manner. The result of the inverse transformation is presented in Table 16. The results obtained coincide with Table 6.

Table 16. (a) Ciphertext after one round of encryption (first group of 8 bytes). (b) Ciphertext after one round of encryption (second group of 8 bytes).

(a)							
$\tilde{C}(0)$	$\tilde{C}(1)$	$\tilde{C}(2)$	$\tilde{C}(3)$	$\tilde{C}(4)$	$\tilde{C}(5)$	$\tilde{C}(6)$	$\tilde{C}(7)$
10	79	130	105	93	160	104	126
(b)							
$\tilde{C}(8)$	$\tilde{C}(9)$	$\tilde{C}(10)$	$\tilde{C}(11)$	$\tilde{C}(12)$	$\tilde{C}(13)$	$\tilde{C}(14)$	$\tilde{C}(15)$
211	11	129	143	36	66	24	180

Let us perform decryption in PMC. In the first stage of the developed method for improving the fault tolerance of SBCG SPN cipher decryptors, the ciphertext presented in Table 16 is converted from PC to PMC. The PMC combinations obtained using algorithm (37) are presented in Table 17.

Table 17. (a) Ciphertext after one round of encryption (first group of 8 bytes). (b) Ciphertext after one round of encryption (second group of 8 bytes).

(a)							
$\tilde{C}(0)$	$\tilde{C}(1)$	$\tilde{C}(2)$	$\tilde{C}(3)$	$\tilde{C}(4)$	$\tilde{C}(5)$	$\tilde{C}(6)$	$\tilde{C}(7)$
(10, 10)	(3, 0)	(9, 5)	(3, 13)	(2, 11)	(13, 12)	(2, 12)	(7, 7)
(b)							
$\tilde{C}(8)$	$\tilde{C}(9)$	$\tilde{C}(10)$	$\tilde{C}(11)$	$\tilde{C}(12)$	$\tilde{C}(13)$	$\tilde{C}(14)$	$\tilde{C}(15)$
(7, 6)	(11, 15)	(10, 2)	(4, 12)	(2, 15)	(14, 9)	(11, 1)	(10, 5)

The ciphertext, in the form of bytes, is written to the shift register. The write operation is performed with the most significant bytes first. After each clock cycle, the next byte obtained, represented in PMC, is fed into the error correction unit. The corrected combination is written to the LFSR from the output of the error correction device. The results of applying the inverse linear transformation in the PMC after 16 decoder clock cycles are presented in Table 18.

The result of the L^{-1} transformation, represented in PMC, is fed to the block implementing the S^{-1} transformation. The resulting error correction codes are shown in Table 19.

Table 18. (a) Result of implementing the L^{-1} transformation in the PMC (first group of 8 bytes). (b) Result of implementing the L^{-1} transformation in PMC (second group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
(14, 4)	(12, 11)	(13, 8)	(2, 8)	(11, 4)	(7, 0)	(8, 10)	(0, 2)
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
(11, 2)	(13, 5)	(2, 10)	(8, 12)	(4, 13)	(7, 5)	(3, 13)	(9, 4)

Table 19. (a) Result of the S^{-1} transformation in PMC (first group of 8 bytes). (b) Result of the S^{-1} transformation in PMC (first group of 8 bytes).

(a)							
$\ddot{a}(0)$	$\ddot{a}(1)$	$\ddot{a}(2)$	$\ddot{a}(3)$	$\ddot{a}(4)$	$\ddot{a}(5)$	$\ddot{a}(6)$	$\ddot{a}(7)$
(1, 14)	(14, 6)	(10, 5)	(3, 3)	(3, 0)	(9, 13)	(3, 9)	(4, 1)
(b)							
$\ddot{a}(8)$	$\ddot{a}(9)$	$\ddot{a}(10)$	$\ddot{a}(11)$	$\ddot{a}(12)$	$\ddot{a}(13)$	$\ddot{a}(14)$	$\ddot{a}(15)$
(0, 4)	(13, 9)	(1, 11)	(0, 2)	(6, 4)	(6, 14)	(2, 9)	(9, 2)

The result is summed modulo two with the round key shown in Table 11. The result is plaintext in the form of PMC combinations, shown in Table 20.

Table 20. (a) Plaintext block in PMC (first group of 8 bytes). (b) Plaintext block in PMC (second group of 8 bytes).

(a)							
$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$
(5, 6)	(13, 4)	(5, 5)	(2, 8)	(6, 3)	(6, 12)	(5, 2)	(2, 4)
(b)							
$X(8)$	$X(9)$	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$	$X(15)$
(11, 5)	(15, 1)	(9, 2)	(8, 15)	(14, 4)	(6, 4)	(6, 3)	(2, 4)

In the final stage of the developed method to improve the fault tolerance of SBCG SPN cipher decryptors, the PMC-PC inverse transformation is applied. The plaintext bytes are computed using expression (14). Only the information residues are involved in the computation. The result of the inverse transformation is presented in Table 21.

Table 21. (a) Plaintext block (first group of 8 bytes). (b) Plaintext block (second group of 8 bytes).

(a)							
$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$X(5)$	$X(6)$	$X(7)$
73	82	5	11	210	21	48	154
(b)							
$X(8)$	$X(9)$	$X(10)$	$X(11)$	$X(12)$	$X(13)$	$X(14)$	$X(15)$
97	101	183	61	29	231	210	154

Thus, the example above demonstrated that using the PMC redundancy code yields results that match those of the classical SBCG encryption algorithm.

5. Discussion

In addition to block encryption systems, it is proposed to use homomorphic encryption (HE) in the IoV network. The main advantage of this type of encryption is the ability to perform mathematical operations on encrypted data. In this case, the result obtained after the decryption procedure will coincide with the result of applying these mathematical operations to the open data [56–60]. In [56], an analysis of modern methods using HE was carried out to ensure the security of basic operations, data aggregation, data queries, and other calculations with data in IoV. The paper also highlights the challenges faced by the application of homomorphic encryption in the IoV network and outlines development prospects. In [57], the authors analyzed decentralized VANETs (DVANETs). To ensure confidentiality, it was proposed to use HE to encrypt transport data at each edge computing node. In [58], it is proposed to switch from fully homomorphic encryption to pseudonymous homomorphic encryption to reduce the network load. The efficiency of the proposed one-way Hellman function method is compared with the Paillard cryptosystem. It is found that the proposed method provides less load on the communication channel when transmitting a message with a higher level of security than the Paillard cryptosystem. To ensure a higher level of confidentiality, [59] proposes the integration of several cryptographic technologies. These are homomorphic encryption, zero-knowledge proofs, blind signatures, and secure multiparty computation. This integration enables decentralized anomaly detection and cyberattack risk assessments without disclosing sensitive information. In [60], the authors presented the integration of homomorphic encryption with machine learning applications. The paper noted the potential to integrate HE with technologies such as differential privacy and secure multiparty computation. This approach allows for reduced computational costs while ensuring a high level of privacy in the IoV network.

However, homomorphic encryption methods have a significant drawback; they are computationally expensive, which negatively impacts the performance of encryptors and decryptors. To address this drawback, [61,62] proposed using a modular code for a residual class system. The paper in [61] describes an algorithm for base extension in a modular code for homomorphic ciphers. Simulation results show that the proposed algorithm optimization can increase the speed of the number-scaling algorithm in the residual class system by 46.15% on average. The paper in [62] presents an approximate homomorphic encryption method that is optimal for implementation on a standard computer system. Research has shown that the proposed solution improves the performance of constructing a logistic regression model from encrypted data by at least 1.9 times compared to previously known methods. Despite the proposed solutions improving the performance of homomorphic encryption algorithms, they remain inferior to block ciphers. Therefore, symmetric block cipher systems, including SBCG, have found wide application in IoV networks.

A comparative analysis of the developed method for ensuring fault tolerance for SBCG encryption systems based on PMCs will be conducted using two structural redundancy methods. The first method is duplication. In this case, the backup SBCG encryption system will be in cold mode. The second method is Triple Modular Redundancy (TMR). Furthermore, the comparative analysis includes the error-detection method for the SPN cipher presented in [53].

To estimate the encryption time in the SBCG fault-tolerant cryptographic system, we implemented it on a XILINX Artix7 (Speedgrade-1) FPGA using Xilinx Vivado-HLS 2018. The clock frequency was 28.5 MHz. We used 8810 LUTs and 2167 latches. The encryption time without using the error detection and correction system was 1531 ns. When using the developed error detection and correction algorithm in the PMC, the encryption time was 2523 ns. Thus, eliminating the consequences of failures and malfunctions in the encoder,

caused by both DFA-type attacks and natural-origin attacks, reduced encryption speed by a factor of 1.67.

As a criterion of efficiency, the safety factor proposed in [63] is used. This factor allows us to estimate the level of fault tolerance of the SBCG encryption system, under conditions when, during its operation, failures occur sequentially one after another. To calculate it, it is necessary to determine all possible states of the encoder (decoder) N_a , as well as the states in which the system remains operational H_a , with a different number of failures $a = 0, 1, 2, \dots$. It is calculated as follows:

$$\eta(a) = \frac{H_a}{N_a}, \quad (70)$$

This indicator was chosen because, using the developed decoding algorithm, error syndromes can be easily calculated for different values. By comparing the obtained values, it is possible to identify situations where the error syndromes will match. If the syndromes do not match across different numbers of failures, this indicates that the developed decoding algorithm can detect and correct the consequences of failures and malfunctions in the SBCG encryption system. However, when another failure occurs, an error syndrome may be obtained that matches that of a lower-order error syndrome. This situation indicates that the developed decoding algorithm will not be able to correct the consequences of this failure. Consequently, the encoder (decoder) is inoperative.

In the absence of failures or malfunctions in the equipment, all encoders are in working order. In this case, the safety factor is $\eta^D(0) = \eta^{TMR}(0) = \eta^{PMC}(0) = \eta^{[53]}(0) = 1$. In this case, the encryption time for structural redundancy methods, including duplication and TMR, will be 1531 ns. The encryption time using the developed decoding algorithm with error correction will be 2523 ns. The encryption time for the method [53] is 2974 ns. The increase in time costs is due to the need to perform the inverse transformation from the modular code to the positional code to compare the result with the working range. For this, the Chinese Remainder Theorem is used. If the degree of the resulting polynomial is less than the degree of the polynomial-working range calculated using expression (2), then it is concluded that the code combination does not contain an error. Thus, the use of the developed decoding algorithm reduces encryption speed by a factor of 1.67. Meanwhile, the [53] method reduces the encryption speed by a factor of 1.94.

Suppose the first failure occurs during the operation of the encoder (decoder). When using the unloaded duplication method, the faulty encoder (decoder) is disabled. The backup device begins operating in its place. In this case, the safety factor is $\eta^D(1) = 1$. When using the TMR method, the failure will be masked, since this structural redundancy method uses the “2 out of 3” principle. In this case, the safety factor is $\eta^{TMR}(1) = 1$. As a result of the first failure (glitch) during operation of the encoder (decoder) using the developed corrected decoding algorithm, one digit in the code combination will be corrupted. Since the GBCG encryption process uses only two information residuals, each four digits, the possible number of single errors in this CC is $N_1^{PMC} = 8$. The developed error-correction algorithm allows correction of all of them. This means that the number of operational states of the encoder (decoder) at the time of the first failure is $H_1^{PMC} = 8$. As a result, the safety factor will be $\eta^{PMC}(1) = 1$. When using the method in [53], the encoder will only be able to detect the failure but not correct the resulting error. In this case, the safety factor will be 16. In this case, the time spent on encryption with TMR methods will be 1531 ns. When using the duplication method, the following situation arises. In [64], the technical characteristics of the V2X on-board unit model TEDIX-V2 are presented, according to which the cold start is 26 s. During this time, a car traveling at 60 km/h will travel 433 m without communication with the RSU. When driving on the highway at 130 km/h, this

will be 939 m. Thus, the use of a cold start, which is used in duplication, reduces traffic safety and increases the likelihood of a traffic accident. Therefore, a cold start is performed within 26 s, and then the duplicating encoder begins operation. The encryption time will be 1531 ns. The encryption time using the developed decoding algorithm with error correction will be 2523 ns.

If a second failure occurs, the duplication method cannot mitigate it. In this case, the safety factor is $\eta^D(2) = 0$. Using the TMR method, the second failure will be mitigated if it occurs in the same encoder where the first failure occurred. In this case, the safety factor is $\eta^{TMR}(2) = 0.33$. A second failure in the encoder (decoder) operation results in a double error in the PMC codegram. In this case, the possible number of double errors in this CC is $N_2^{PMC} = C_8^2 = 28$. Research has shown that the developed algorithm can detect and correct all these double errors in the data residues. This means that the number of operational states of the encoder (decoder) when a second failure occurs, i.e., when $a = 2$, is $H_2^{PMC} = 28$. In this case, the safety factor is $\eta^{PMC}(2) = 1$. The encryption times are 1531 ns and 2523 ns, respectively.

When a third failure occurs, the duplication method is also unable to counter it. In this case, the safety factor is $\eta^D(3) = 0$. The safety factor using the TMR method is $\eta^{TMR}(3) = 0.09$. The third failure in the encoder (decoder) operation results in a triple error in the PMC codegram. In this case, the possible number of triple errors in this CC is $N_3^{PMC} = C_8^3 = 56$. Research has shown that the developed algorithm can correct only 41 errors. For the remaining CC, there is a coincidence between the triple-error syndrome calculated using (35) and the syndrome's lower-multiplicity errors. In this case, the safety factor is $\eta^{PMC}(3) = 0.73$. In this case, the encryption time will be 2523 ns. Figure 7 shows the results of a comparative analysis of the methods discussed above for increasing the fault tolerance of encryption systems using the SBCG algorithm.

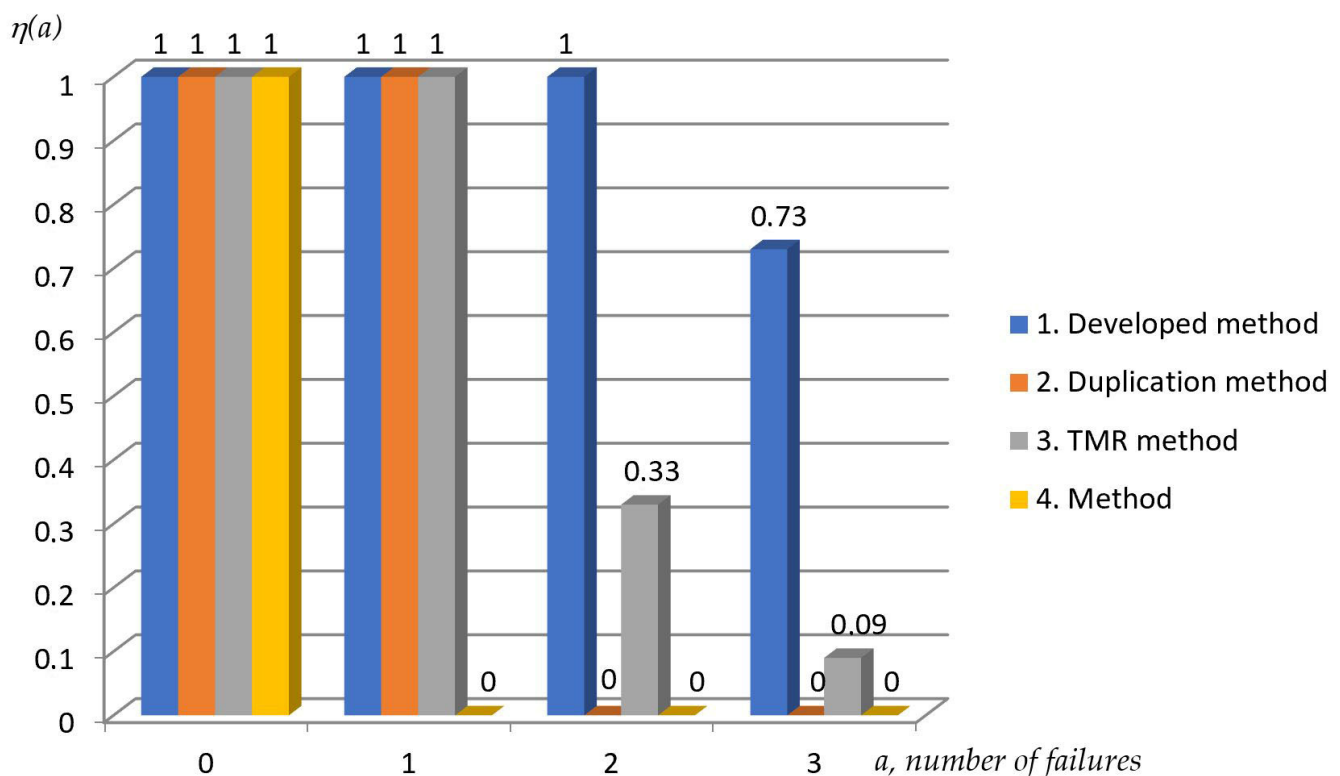


Figure 7. Fault tolerance margin of encoder (decoder) when using different methods for ensuring fault tolerance.

An analysis of the graph shows that implementing SBCG in polynomial modular codes effectively mitigates the consequences of failures and malfunctions during the operation of encryptors/decoders. Furthermore, the use of the developed decoding algorithm provides a higher level of fault tolerance for SBCG encryption systems than the hardware duplication method. As a result, even failures and malfunctions during their operation will not reduce the IoV network's resilience to cyberattacks.

6. Conclusions

The integration of Vanet and IoT technologies has led to the emergence and deployment of IoV networks. This is because the Internet of Vehicles (IoV) enables real-time data exchange, improving road safety, reducing accidents, and enhancing driver and passenger comfort. However, the wireless communication channel used for transmission has several vulnerabilities that are exploited in cyberattacks. An analysis of related work revealed a number of cyberattacks targeting IoV networks whose effectiveness can be mitigated using SBCG. A characteristic feature of SBCG is the ability to transition from encryption in a $GF(2^8)$ field to similar procedures in a ring of two $GF(2^4)$ fields. It is shown that SBCG can be implemented in a PMC whose bases are the generator polynomials of two $GF(2^4)$ fields. To ensure fault tolerance in encoders/decoders for the SBCG encryption standard, a decoding algorithm for a polynomial modular code with error correction was developed that, unlike previously known ones, corrects corrupted code residues at a lower circuit cost. Using this decoding algorithm, mathematical models of transformations for encryption and decryption were developed and implemented in modular code. To evaluate the efficiency of the developed method for increasing the fault tolerance of the PMC-based SBCG encryption system, a comparative analysis was conducted against the duplication method. The studies conducted have shown that, in the event of the first failure, the developed method, the duplication method, and the TMR method make it possible to eliminate its consequences; that is, the safety factor is 1. At the same time, the method [53] can only detect the error. Therefore, its safety factor is $\eta^{[53]}(1) = 0$. In the event of the second failure, the safety factor of the SBCG encryption system using the duplication method is $\eta^D(2) = 0$, and that of the TMR method is $\eta^{TMR}(2) = 0.33$. The developed method provides an operability safety factor of $\eta^{PCM}(2) = 1$ for the SBCG encryption system. When a third failure occurs, the operability safety factor for the duplication method is $\eta^D(3) = 0$, and for the TMR method, it is $\eta^{TMR}(3) = 0.09$. The developed method provides $\eta^{PMC}(3) = 0.73$. Thus, encryption in a polynomial modular code provides a higher level of resilience for the SBCG encryption system to failures caused by both DFA attacks and natural failures.

Limitations and Future Work

The goal of this study is to improve the resilience of SBCG encryptors/decoders to faults by using polynomial modular codes. Imparting fault-tolerant properties to SBCG encryption systems implemented in PMC will enable them to effectively mitigate real-time faults arising from both Differential Fault Analysis (DFA) attacks and natural faults.

The problem of ensuring fault tolerance for encryption systems used in IoV networks is quite pressing and has many solutions. Therefore, this paper has limitations that were not considered.

This paper analyzed cyberattacks targeting key IoV network objects (OBUs and RSUs) to obtain confidential information. Furthermore, the paper considered fault-based attacks, which target physical devices performing encryption. These attacks are implemented in two stages. The first stage involves introducing faults by physically affecting the device at a specific point during the encryption process. However, the article did not describe the methods used to introduce faults. However, exposure of the encoder to electromagnetic

pulses, lasers, or changes in supply voltage or clock frequency can lead to different fault effects on the cryptographic strength of the encryption algorithm. Secondly, the article does not describe a system for physically affecting the encoder or analyzing the consequences of a fault on the device's operation.

The second stage of a fault attack involves analyzing the fault-corrupted encrypted texts to obtain secret data (the original encryption keys). Only one fault analysis method was considered in the paper: Differential Fault Analysis (DFA). This method compares the correct and corrupted outputs from the encoder to compute the secret key. However, other methods are also used in analyzing the consequences of faults. First, the Collision Fault Attack (CFA) method was not considered. This method searches for plaintext blocks that yield identical intermediate results when encrypted. Such collisions allow an attacker to obtain information about the secret key. Second, the statistical fault attack (SFA) exploits the statistical properties of errors occurring during cryptographic operations to derive the secret key.

The article also did not specify areas for expansion in future research. The article examined the use of PMCs to improve the fault tolerance of SBCG encryption systems. This code was chosen because it is an arithmetic code. The article demonstrated that PMCs allow for the efficient implementation of modular operations (addition and multiplication) when performing a linear transformation in the SBCG cipher. Furthermore, the introduction of two check residues into the code combination allowed the PMC to correct errors caused by faults and failures. With such redundancy, the PMC can also correct a burst of errors within a single residue. Consequently, in terms of their error-correcting capabilities and the redundancy they introduce, RMS codes are similar to Reed–Solomon codes. Therefore, the future development of this topic lies in studying the error-correcting capabilities of RMS codes for burst errors that include multiple residues. This will allow us to formulate principles for constructing error-correcting polynomial modular codes.

Author Contributions: Conceptualization, I.K. and V.K.; methodology, I.K. and G.S.; validation, A.O. and T.P.; formal analysis, I.K. and V.K.; writing—original draft preparation, I.K.; writing—review and editing, I.K., M.L., A.O. and T.P.; supervision, V.K. and G.S.; project administration, M.L.; funding acquisition, I.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Russian Science Foundation, Grant No 25-71-30007, <https://rscf.ru/en/project/25-71-30007/> (accessed on 7 June 2026).

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lu, M. *Evaluation of Intelligent Road Transport Systems: Methods and Results*; IET: London, UK, 2016.
2. Paul, A.; Chilamkurti, N.; Daniel, A.; Rho, S. *Intelligent Vehicular Networks and Communications: Fundamentals, Architectures and Solutions*; Elsevier Inc.: Amsterdam, The Netherlands, 2016; p. 227.
3. da Cunha, F.D.; Villas, L.; Boukerche, A.; Maia, G.; Viana, A.C.; Mini, R.A.F.; Antonio, A.F.L. Data Communication in VANETs: Survey, Applications and Challenges. In *Ad Hoc Networks*; Elsevier: Amsterdam, The Netherlands, 2016.
4. A Assem, R.; Lyamine, G.; Abderrezak, B.; Zineb, S.; Abdelkrim, L. Data Processing from VANETs to IoV: Literature Review. In *Proceedings of the Future Technologies Conference (FTC), Vancouver, BC, Canada, 1 November 2023*; Springer Nature: Cham, Switzerland, 2023; pp. 472–494. [[CrossRef](#)]
5. Buinevich, M.; Vladyko, A. Forecasting Issues of Wireless Communication Networks' Cyber Resilience for An Intelligent Transportation System: An Overview of Cyber Attacks. *Information* **2019**, *10*, 27. [[CrossRef](#)]
6. Sylva, V.; Lala, A.; Biberaj, A. Vanet Security and Privacy—An Overview. In *the 19TH Edition of International Conference European Integration—Realities and Perspectives*; EIRP Proceedings: Galați, Romania, 2024; Volume 19, pp. 414–423.
7. Gayathri, M.; Gomathy, C. A Deep Survey on Types of Cyber Attacks in VANET. *J. Crit. Rev.* **2021**, *8*, 1029–1039.

8. Ahmad, F.; Adnane, A.; Franqueira, N.L. A Systematic Approach for Cyber Security in Vehicular Networks. *J. Comput. Commun.* **2016**, *4*, 38–62. [CrossRef]
9. GOST R 34.12-2015; Information Technology. Cryptographic Protection of Information. Block Ciphers: Russia, Moscow, 2015. Available online: <https://rst.gov.ru:8443/file-service/file/load/1699367634477> (accessed on 7 June 2026). (In Russian)
10. Kukartsev, A.M.; Kuznetsov, A.A.; Safonov, K.V. On the implementation of the Russian symmetric encryption algorithm “Kuznyechik” for resource-efficient platforms. *Prikl. Diskretn. Mat. Suppl.* **2025**, *18*, 247–250.
11. Maro, E.A.; Malyavina, V.O. Detection of statistical side-channel leaks in encryption algorithms based on the ELMO emulator. *Prikl. Diskretn. Mat. Suppl.* **2025**, *18*, 222–225.
12. Hashimoto, K.; Tsuchiya, T.; Kikuno, T. A new approach to fault-tolerant scheduling using task duplication in multiprocessor systems. *J. Syst. Softw.* **2000**, *53*, 159–171. [CrossRef]
13. Cojocar, L.; Papagiannopoulos, K.; Timmers, N. Instruction duplication: Leaky and not too fault-tolerant! In *Smart Card Research and Advanced Applications (CARDIS 2017)*; Springer: Cham, Switzerland, 2017; pp. 160–179.
14. Morgan, K.S.; McMurtrey, D.L.; Pratt, B.H.; Wirthlin, M.J. A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs. *IEEE Trans. Nucl. Sci.* **2007**, *54*, 2065–2072. [CrossRef]
15. Balasubramanian, P.; Prasad, K. A Fault Tolerance Improved Majority Voter for TMR System Architectures. *WSEAS Trans. Circuits Syst.* **2016**, *15*, 108–122.
16. Bertoa, T.G.; Gambardella, G.; Fraser, N.J.; Blott, M.; McAllister, J. Fault-Tolerant Neural Network Accelerators with Selective TMR. *IEEE Des. Test* **2023**, *40*, 67–74. [CrossRef]
17. Sakiz, F.; Sen, S. A survey of attacks and detection mechanisms on intelligent transportation systems: Vanets and iov. *Ad Hoc Netw.* **2017**, *61*, 33–50. [CrossRef]
18. Krishna, K.V.; Reddy, K.G. Classification of Distributed Denial of Service Attacks in VANET: A Survey. *Wirel. Pers. Commun.* **2023**, *132*, 933–964. [CrossRef]
19. Karim, S.M.; Habbal, A.; Chaudhry, S.A.; Irshad, A. Architecture, Protocols, and Security in IoV: Taxonomy, Analysis, Challenges, and Solutions. *Secur. Commun. Netw.* **2022**, *2022*, 1131479. [CrossRef]
20. Malla, A.M.; Sahu, R.K. Security Attacks with an Effective Solution for DOS Attacks in VANET. *Int. J. Comput. Appl.* **2013**, *66*, 45–49.
21. Ghorri, M.R.; Zamli, K.Z.; Quosthoni, N.; Hisyam, M.; Montaser, M. Vehicular ad-hoc network (VANET): Review. In Proceedings of the 2018 IEEE International Conference on Innovative Research and Development (ICIRD), Bangkok, Thailand, 11–12 May 2018. [CrossRef]
22. Zeadally, S.; Hunt, R.; Chen, Y.-S.; Irwin, A.; Hassan, A. Vehicular ad hoc networks (VANETS): Status, results, and challenges. *Telecommun. Syst.* **2010**, *50*, 217–241. [CrossRef]
23. Gaba, S.; Gupta, M.; Singh, H. A comprehensive survey on VANET security attacks. *AIP Conf. Proc.* **2023**, *2495*, 020029. [CrossRef]
24. Dinesh, D.; Deshmukh, M. Challenges in Vehicle Ad Hoc Network (VANET). *Int. J. Eng. Technol. Manag. Appl. Sci.* **2014**, *2*, 76–88.
25. Singh, K.; Sharma, S. Advanced Security Attacks on Vehicular AD HOC Network (VANET). *Int. J. Innov. Technol. Explor. Eng.* **2019**, *9*, 3057–3064. [CrossRef]
26. Gupta, S.; Singhal, A.; Kapoor, A. A literature survey on social engineering attacks: Phishing attack. In Proceedings of the 2016 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 29–30 April 2016; pp. 537–540. [CrossRef]
27. Zaidi, T.; Faisal, S. An Overview: Various Attacks in VANET. In Proceedings of the 2018 4th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 14–15 December 2018. [CrossRef]
28. El-Dalalmeh, A.; El-Dalalmeh, M.; Razzaque, M.A.; Li, J. Cryptographic methods for secured communication in SDN-based VANETs: A performance analysis. *Secur. Priv.* **2024**, *7*, e446. [CrossRef]
29. Islam, A.; Ranjan, S.; Rawat, A.P.; Maity, S. A Comprehensive Survey on Attacks and Security Protocols for VANETs. In *Innovations in Computer Science and Engineering. Lecture Notes in Networks and Systems*; Saini, H.S., Sayal, R., Govardhan, A., Buyya, R., Eds.; Springer: Singapore, 2021; Volume 171. [CrossRef]
30. Alimohammadi, M.; Pouyan, A.A. Performance Analysis of Cryptography Methods for Secure Message Exchanging in VANET. *Int. J. Sci. Eng. Res.* **2014**, *5*, 911–917.
31. Roy, A.; Madria, S. Secured Traffic Monitoring in VANET. *arXiv* **2020**, arXiv:1909.10057. [CrossRef]
32. Mustafa, A.S.; Hamdi, M.M. VANET: Towards Security Issues Review. In Proceedings of the 2020 IEEE 5th International Symposium on Telecommunication Technologies (ISTT), Shah Alam, Malaysia, 9–11 November 2020.
33. Isyanto, H.; Arifin, A.S.; Suryanegara, M. Performance of smart personal assistant applications based on speech recognition technology using IoT-based voice commands. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 21–23 October 2020; pp. 640–645.
34. Li, W.; Zhang, W.; Gu, D.; Cao, Y.; Tao, Z.; Zhou, Z.; Liu, Y.; Liu, Z. Impossible Differential Fault Analysis on the LED Lightweight Cryptosystem in the Vehicular Ad-Hoc Networks. *IEEE Trans. Dependable Secur. Comput.* **2015**, *13*, 84–92. [CrossRef]

35. Li, W.; Ge, C.; Gu, D.; Liao, L.; Gao, Z.; Shi, X. Security Analysis of the Khudra Lightweight Cryptosystem in the Vehicular Ad-hoc Networks. *KSII Trans. Internet Inf. Syst.* **2018**, *12*, 3421. [CrossRef]
36. Moradi, A.; Shalmani, M.T.M.; Salmasizadeh, M. A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In *Cryptographic Hardware and Embedded Systems—CHES 2006*. CHES 2006. *Lecture Notes in Computer Science*; Goubin, L., Matsui, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4249. [CrossRef]
37. Floissac, N.; L'Hyver, Y. From AES-128 to AES-192 and AES-256, How to Adapt Differential Fault Analysis Attacks on Key Expansion. In *Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Nara, Japan, 28 September 2011. [CrossRef]
38. Kim, C.H.; Quisquater, J.-J. New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough. Available online: <https://opendl.ifip-tc6.org/db/conf/cardis/cardis2008/KimQ08.pdf> (accessed on 7 June 2026).
39. Piret, G.; Quisquater, J.J. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In *CHES*; *Lecture Notes in Computer Science*; Walter, C.D., Koç, Ç.K., Paar, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2779, pp. 77–88.
40. Ali, S.S.; Mukhopadhyay, D.; Tunstall, M. Differential fault analysis of AES: Towards reaching its limits. *J. Cryptogr. Eng.* **2012**, *3*, 73–97. [CrossRef]
41. Krasovsky, A.V.; Maro, E.A. Side-channel attacks on the Kuznechik encryption algorithm. *Reshetnev Read.* **2019**, 434–435.
42. Altawy, R.; Duman, O.; Youssef, A.M. Fault Analysis of Kuznyechik. In *4rd Workshop on Current Trends in Cryptology*; CTCrypt: Kazan, Russia, 2015; pp. 302–316.
43. Fomin, D.B. A timing attack on CUDA implementations of an AES-type block cipher. *Math. Issues Cryptogr.* **2016**, *7*, 121–130. [CrossRef]
44. Chervyakov, N.I.; Shaposhnikov, A.V.; Sakhnyuk, P.A.; Makokha, A.N. *Neurocomputers in Residual Classes*; Radiotekhnika Publ.: Moscow, Russia, 2003; p. 272.
45. Chernyavsky, A.F.; Kolyada, A.A.; Protaseny, S.Y. Application of the neural network computing technology for calculating the interval-index characteristics of a minimally redundant modular code. *Dokl. Natl. Acad. Sci. Belarus* **2018**, *62*, 652–660. [CrossRef]
46. Chervyakov, N.I.; Babenko, M.G.; Deryabin, M.A.; Nazarov, A.S.; Shabalina, M.N. Computation of positional characteristics of numbers in RNS based on approximate method. In *Proceedings of the 2016 IEEE North West Russia Section Young Researchers in Electrical and Electronic Engineering Conference, EIConRusNW*, St. Petersburg, Russia, 2–3 February 2016; pp. 177–179.
47. Kalmykov, I.A.; Olenev, A.A.; Kononova, N.V.; Peleshenko, T.A.; Chistousov, N.K. A numerical method for parallel calculation of the positional characteristic for error correction in a polyalphabetic polynomial modular code. *Comput. Opt.* **2025**, *49*, 141–150. [CrossRef]
48. Chervyakov, N.I.; Lyakhov, P.A.; Babenko, M.G. The architecture of a fault-tolerant modular neurocomputer based on modular number projections. *Neurocomputing* **2018**, *272*, 96–107. [CrossRef]
49. Olenev, A.A.; Kalmykov, I.A.; Kononova, N.V.; Peleshenko, T.A.; Dukhovnyj, D.V.; Chistousov, N.K.; Kalmykova, N.I. Improvement of the Cybersecurity of the Satellite Internet of Vehicles through the Application of an Authentication Protocol Based on a Modular Error-Correction Code. *World Electr. Veh. J.* **2024**, *15*, 278. [CrossRef]
50. Mohan, P.V.A. Residue Number Systems. In *Theory and Applications*; Springer International Publishing: Cham, Switzerland, 2016. [CrossRef]
51. Kalmykov, I.A.; Pashintsev, V.P.; Tyncherov, K.T.; Olenev, A.A.; Chistousov, N.K. Error-Correction Coding Using Polynomial Residue Number System. *Appl. Sci.* **2022**, *12*, 3365. [CrossRef]
52. Chu, J.; Benaissa, M. Polynomial Residue Number System $GF(2^M)$ Multiplier Using Trinomials. In *Proceedings of the 17th European Signal Processing Conference (EUSIPCO 2009)*, Glasgow, UK, 24–28 August 2009; pp. 958–962.
53. Chu, J.; Benaissa, M. Error detecting AES using polynomial residue number systems. *Microprocess. Microsyst.* **2013**, *37*, 228–234. [CrossRef]
54. Bose, S.; Vijayakumar, P. *Cryptography and Network Security*; Pearson: London, UK, 2017; p. 545.
55. Patent for the Invention RU 2847242. Available online: https://www.fips.ru/registers-doc-view/fips_servlet?DB=RUPAT&Doc-Number=2847242&TypeFile=html (accessed on 7 June 2026).
56. Sun, X.; Yu, F.R.; Zhang, P.; Xie, W.; Peng, X. A Survey on Secure Computation Based on Homomorphic Encryption in Vehicular Ad Hoc Networks. *Sensors* **2020**, *20*, 4253. [CrossRef] [PubMed]
57. Chen, J.; Li, K.; Yu, P.S. Privacy-Preserving Deep Learning Model for Decentralized VANETs Using Fully Homomorphic Encryption and Blockchain. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 11633–11642. [CrossRef]
58. Prema, N.K. Efficient Secure Aggregation in VANETs Using Fully Homomorphic Encryption (FHE). *Mob. Netw. Appl.* **2018**, *24*, 434–442. [CrossRef]

59. Kasula, V.K.; Addula, S.R.; Ayyamgari, S.; Tumma, C. Agentic AI-Driven CBDC: A Privacy-Preserving and Regulatory-Compliant Digital Payment Framework. In *Applied Machine Learning and Data Analytics. AMLDA 2024. Communications in Computer and Information Science*; Tiwari, S., Bano, T., Jabbar, M.A., Ortiz-Rodriguez, F., Jiomekong, A., Eds.; Springer: Cham, Switzerland, 2026; Volume 2636. [CrossRef]
60. Ameer, Y.; Bouzeffrane, S. Enhancing privacy in VANETs through homomorphic encryption in machine learning applications. *Procedia Comput. Sci.* **2024**, *238*, 151–158. [CrossRef]
61. Babenko, M.G.; Sinitsyn, A.A.; Deryabin, M.A. Optimization of RNS Base Extension for Homomorphic Encryption Schemes. *Proc. Inst. Syst. Program. RAS* **2026**, *38*, 33–44.
62. Cheon, J.H.; Han, K.; Kim, A.; Kim, M.; Song, Y. A Full RNS Variant of Approximate Homomorphic Encryption. *Sel. Areas Cryptogr.* **2018**, *11349*, 347–368.
63. Avduevsky, V.S.; Melnikov, V.A.; Severtsev, N.A. Reliability and Efficiency in Engineering: Handbook. In *Methods of Similarity in Reliability*; Mashinostroenie: Moscow, Russia, 1987; p. 280.
64. V2x RSU OBU. Available online: <https://ripas.ru/v2x?ysclid=mpv7vlmw3x152731743> (accessed on 7 June 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.