

## SCLang: Graphical Domain-Specific Modeling Language for Stream Cipher

Samar Amil Qassir<sup>1</sup>, Methaq Talib Gaata<sup>1</sup>, Ahmed T. Sadiq<sup>2</sup>

<sup>1</sup>Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq

<sup>2</sup>Computer Science Department University of Technology Iraq, Baghdad, Iraq

E-mails: samarqassir@uomustansiriyah.edu.iq dr.methaq@uomustansiriyah.edu.iq

Ahmed.T.Sadiq@uotechnology.edu.iq

**Abstract:** A Stream Cipher (SC) is a symmetric-key encryption type that scrambles each piece of data in clear text to conceal it from hackers. Despite its advantages, it has a substantial challenge. Correct handwriting of the script code for the cipher scheme is a challenge for programmers. In this paper, we propose a graphical Domain-Specific Modeling Language (DSML) to make it easier for non-technical users and domain specialists to implement an SC domain. The proposed language, SCLang, offers great expressiveness and flexibility. Six different methods of keystream generation are provided to obtain a random sequence. In addition, fifteen tests in the NIST suite are provided for random statistical analysis. The concepts of the SC domain and their relationships are presented in a meta-model. The evaluation of SCLang is based on qualitative analysis and is presented to demonstrate its effectiveness and efficiency.

**Keywords:** Stream cipher; keystream generation methods; graphical domain-specific modelling language; meta-model.

### 1. Introduction

A Stream Cipher (SC) is a cipher type that scrambles data in the origin message encrypting one bit/byte at a time to protect it from outsiders, resulting in low complexity, a serial nature, and ease of encryption [1]. Through bit-by-bit processing, SC allows companies that manage messages written in a trickle to send information as soon as it is ready rather than waiting for completion [2]. Most web browser engines, real-time applications, and IoT technology use this cipher type [3, 4]. Fig. 1 shows other modern applications that use SC.

The SC Algorithm is based on three essential components: the plain text, a symmetric key, and an eXclusive-OR (XOR) logic gate. Cryptographers refer to this symmetric key as a keystream. The keystream is generated by a random key generator method, which is considered the most challenging process in SC schema, as many tests and statistical analyses must be passed to determine its suitability for the encryption process [5].

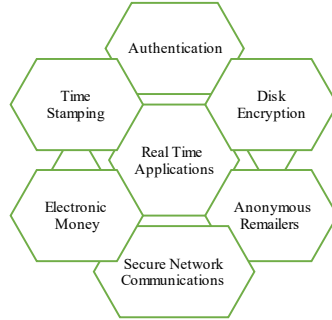


Fig. 1. Modern applications of SC

To make keystream generation in SC schema easier, faster, and more efficient, we use the Model-Driven Development (MDD) strategy known as Domain-Specific Modeling Language (DSML) [6]. DSMLs can reduce the complexities of a generated and tested random sequence by providing a higher abstraction level, which in turn, together with generating the random sequence automatically, enhances the performance of the cipher schema (in both design and implementation) and increases efficiency by reducing the likelihood of mistakes.

The contribution of this research is SCLang, a new graphical DSML that significantly increases the flexibility, expressiveness, and ease of SC schema design and implementation. In short, SCLang provides automatic and safe transformation of plain text computations into the corresponding cipher text. This language serves beginner and expert programmers. It provides the main components needed to construct the SC schema, and six different keystream generation methods that can be used in a hybrid fashion. In addition, it provides fifteen tests of the NIST suite as graphical components to facilitate the statistical analysis of encrypted results.

The remainder of this paper is structured as follows. The fundamental terminology used in SCLang is presented in Section 2. The development of SCLang and its implementation are discussed in Sections 3 and 4. The evaluation of SCLang development is presented in Section 5. Conclusions and future research are presented in Section 6.

## 2. Basic terminology

To understand better the proposed SCLang, an analysis of the SC operation mechanism is presented. Randomness with its test methods is explained. DSML and some related research are presented.

### 2.1. Stream ciphering

The operation mechanism of SC is based on using the same keystream in the encryption and decryption processes. SC encrypts the plain text using this keystream, a pseudorandom sequence with bits of plain text to generate cipher text, usually using an XOR gate. To encrypt uniquely each character depending on the matching digit in the keystream, the string of pseudorandom digits is created from a seed (real key) that spans the entire length of the plain text [5]. Fig. 2 explains the SC operation mechanism (basic schema).

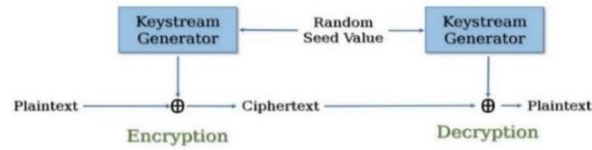


Fig. 2. SC operation mechanism

There are two types of stream ciphers: synchronous and asynchronous [5]. A synchronous keystream is created based on internal states that are unrelated to the plain text or cipher text. This is the type that SCLang uses. An excellent cipher example to comprehend SC is the one-time pad [8]. It is the only theoretically secure cipher that can generate a truly random key sequence as long as the message.

The randomness term is the probability-based uncertainty of each event in the event collection. The random sequence should meet three requirements in terms of statistics and cryptanalysis [9]: it follows a uniform distribution, each element is independent of the others, and the remaining elements cannot be predicted from any previous sequences. The wide use of random numbers in areas other than cryptography, such as statistical sampling, completely randomized designs, computer simulation, numerical analysis, decision-making, and others, highlights their significance [10-12].

There are two categories of random numbers, according to how they are produced: True Random Numbers Generator (TRNGs) and Pseudo-Random Numbers Generator (PRNGs) [13]. TRNG creation must be based on an unpredictably changing physical external variable. PRNGs use a seed as input and functions to produce an output sequence. They are used extensively for their efficiency and speed. There are many methods of generating PRNGs, such as a Feedback Shift Register (FSR), of which there are three types: Linear Feedback Shift Register (LFSR), Non-Linear Feedback Shift Register (NLFSR), and Feedback Carry Shift Register (FCSR), respectively [14], PANAMA-Like based, Geffe method, Random shuffled, Block cipher based and Blok cipher work mode [15].

To guarantee the randomness of PRNGs, many tests are required. Use of a non-random sequence results in weak cryptographic algorithms and vulnerabilities in applications. A sequence is statistically examined to determine if the output sequence randomness test exhibits the features of a true random number sequence. In general, there is more than one battery of tests [16, 17]; the most common is the NIST test suite used with SCLang.

## 2.2. Domain-Specific Modeling Language (DSML)

DSML is a type of Domain-Specific Language (DSL), a programming language approach that focuses on a particular domain but has restricted expressiveness [18]. This type of programming language is developed for a particular domain and specific problems. DSL development can be classified in three categories based on specific characteristics. Internal/external DSL determines whether the generated language depends on the host language or was created entirely from scratch [19]. A textual/graphical DSL is based on a compiler approach (textual) and a projection approach (graphical) [20]. Domain-Specific Visual Language (DSVL), Domain-

Specific Model Language (DSML), and Domain-Specific Embedded Language (DSEL) represent the three categories of DSL. These classifications are interconnected [19].

Models seem to be the current orientation of language development rather than traditional technologies [20, 21]. Model definition, and conversions from one model to another, and from a model to text are essential components in creating automated systems. In this context, a model is a condensed portrayal of a certain reality with the intention of better comprehending it. Whether the model is expressed graphically or textually, this abstraction requires the removal of irrelevant features from the model [19]. In this study, a graphical approach has been used, as it is simple and easily learned. Models are created using DSMLs and consist of three distinct parts [22]: the abstract syntax, which consists of fundamental concepts of a specific domain and relationships between them; the concrete syntax, or DSML notation, which is a collection of helpful graphical symbols for drawing diagrams; and the semantic definition, which is a validation rule to determine whether a model is well-formed according to the domain rules. The advantages of creating and using such a particular language type include reduced size of potential errors, increased productivity, improved communication with subject matter experts, easier adaptation to changes, allowing users to specify what the system should do but not how it is carried out [22]. The field of modeling and meta-modeling is too wide to be discussed in detail here [23]. The design and implementation of DSML have recently been considered in different research areas [24]. Some research areas that have designed and implemented DSML as a solution for domain difficulties are presented.

A DSML known as DSML4CP, created and developed by El a h e h, El h a m and Mo h a r r a m [25] is backed by a graphical modeling tool for concurrent programming. The greater abstraction level offered by DSML4CP can lessen the complexity. By increasing the degree of abstraction and automatically creating artifacts, it is possible to improve software development performance. Efficiency can be increased by reducing the likelihood of mistakes. A background for developing the abstract syntax, concrete syntax, and semantics of DSML4CP is produced by creating a meta-model that allows concurrent programs and interaction. Code generation is completed using the Xpand language transformation rules, instance model, and abstract syntax. Only 21% of the final code and 14% of the functions are contributed manually in the two case studies used to test DSML4CP; 79% of the final code and 86% of the functions have been created automatically.

Two graphical DSMLs are proposed by Juan, Guadalupe and In m a c u l a d a [26], the first – to assist domain specialists in defining Complex Event Processing (CEP) domains, and the second – to assist non-technical users in defining event pattern definitions. The suggested languages offer high expressiveness and flexibility and are not dependent on the implementation code for event patterns and actions. Experts without specialized knowledge of Eltron Programming Language (EPL) can specify pertinent event types and patterns in the initial DSML. In the second DSML users have an intuitive and user-friendly means of describing circumstances that must be found in a specific domain through defining of event

patterns, and actions that must be sent to interested users by email, social networking services, or other methods.

Campos and Grangel [27] have unveiled a brand-new DSML for Corporate Social Responsibility (CSR) allowing the creation of corporate models that are centered on the CSR dimension using this language as an Enterprise Modeling Language (EML). To specify the artifacts of the CSR meta-model, several firms have produced worldwide standards, manuals with laws, and CSR plans. To distinguish between the components directly connected to the CSR domain and those required for modeling (CoreElements), the CSR meta-model has been developed using two packages (CSRElements). A UML profile is used to construct the modeling tool and provide icons that resembled actual things. A useful case study using this DSML is presented as a last step to enhance and validate the proposed CSR meta-model and to demonstrate its application in actual use.

Chunlin et al. [28] have provided an executable DSML known as xSHS to capture both stochastic and hybrid behaviors. Eclipse Studio uses xSHS as its implementation. Two layers (the parent layer and child layer) in xSHS are modified to construct a meta-model of the dynamic behavior of CPS. Contrary to the abstract syntax used for internal representations, the concrete syntax of xSHS is primarily intended for users. Sirius is used to define the graphical concrete syntax of this language. Based on formal rules, operational semantics are defined. The SCILab jar files are incorporated into Eclipse Studio to run the language. An xSHS model for a temperature control system has been built to show the usefulness of xSHS.

Graphical Invariant Language (GIRL) is introduced by Marzina, Massoni and Ramalho [29]. GIRL is a DSL based on set theory used to express the structural invariants of software requirements. The Meta Object Facility (MOF) meta-model, consisting of items such as an integer, an operation, and their connections, provides the GIRL abstract syntax. Alloy Analyzer can assess the consistency of the structural requirement using translational semantics without user input. This translational technique offers the advantages of formal analysis by enabling early discovery of discrepancies in the requirement definitions. Ten software engineers participated in mixed empirical research for a qualitative survey to assess the provided GIRL.

Sadik and Geylani [30] have proposed a DSML4DT, or DSML for Device Trees (DTs). DTs offer descriptions of devices and peripherals found inside an embedded system and node specs. The DSML4DT language has been used to create a Model-Driven Development (MDD) of DT development technique, allowing developers to visually design and construct DT-based embedded systems. The suggested model-driven technique comprises automated code generation for precise DT implementations and system modeling. The meta-model for DSML4DT has been made up of more than 70 meta-entities and their connections. Constraint checks and static semantics controls are conducted automatically inside the environment in accordance with DSML4DT model validation rules created using Acceleo Query Language (AQL) on the Sirius platform. The next phase is to generate and finish the DT software code. DT models are transformed into DT code for a specified embedded

system. The output of the automated code generation process is DT source files, which may be used later in the OS as needed.

Ana et al. [31] suggest a graphical DSML to help domain specialists retrieve event logs from Enterprise Resource Planning (ERP) systems. In particular, domain experts are able to locate conceptually where instances and events are stored inside a database. Following automated validation, these conceptual models are converted into SQL code. This modeling language is designed to address complicated conditions when using ERP systems. The modeling language applicability has been demonstrated via a case study with actual data to show that the language contained the necessary components. These constructs make it possible for domain specialists to focus on modeling data during the log extraction stage without learning how to program, which simplifies the querying of process data.

### 3. SCLang development

In this section, the SCLang design is explained in detail. The meta-model representing the structure of the SCLang language (the valid model) is defined by:

- 1) The abstract syntax based on the SC concepts (components) and the rules that govern them in the encryption/decryption processes;
- 2) The concrete syntax is defined by selecting meaningful symbols (icons) to describe the components;
- 3) The semantic is defined by determining the correct connection and defining the method of notifying and handling if there are invalid or missing connections.

A valid SCLang meta-model is the result of all of these definitions and is implemented as an internal graphical DSML using the advantages of the infrastructure already existing in Python (the host language), and is run as a graphical editor. Fig. 3 shows the architecture of the SCLang language.

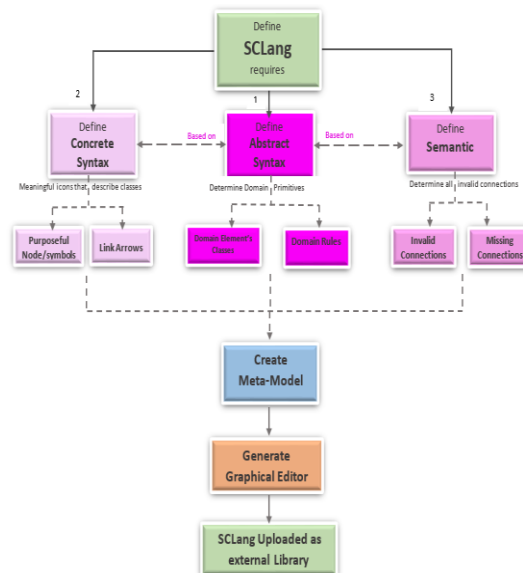
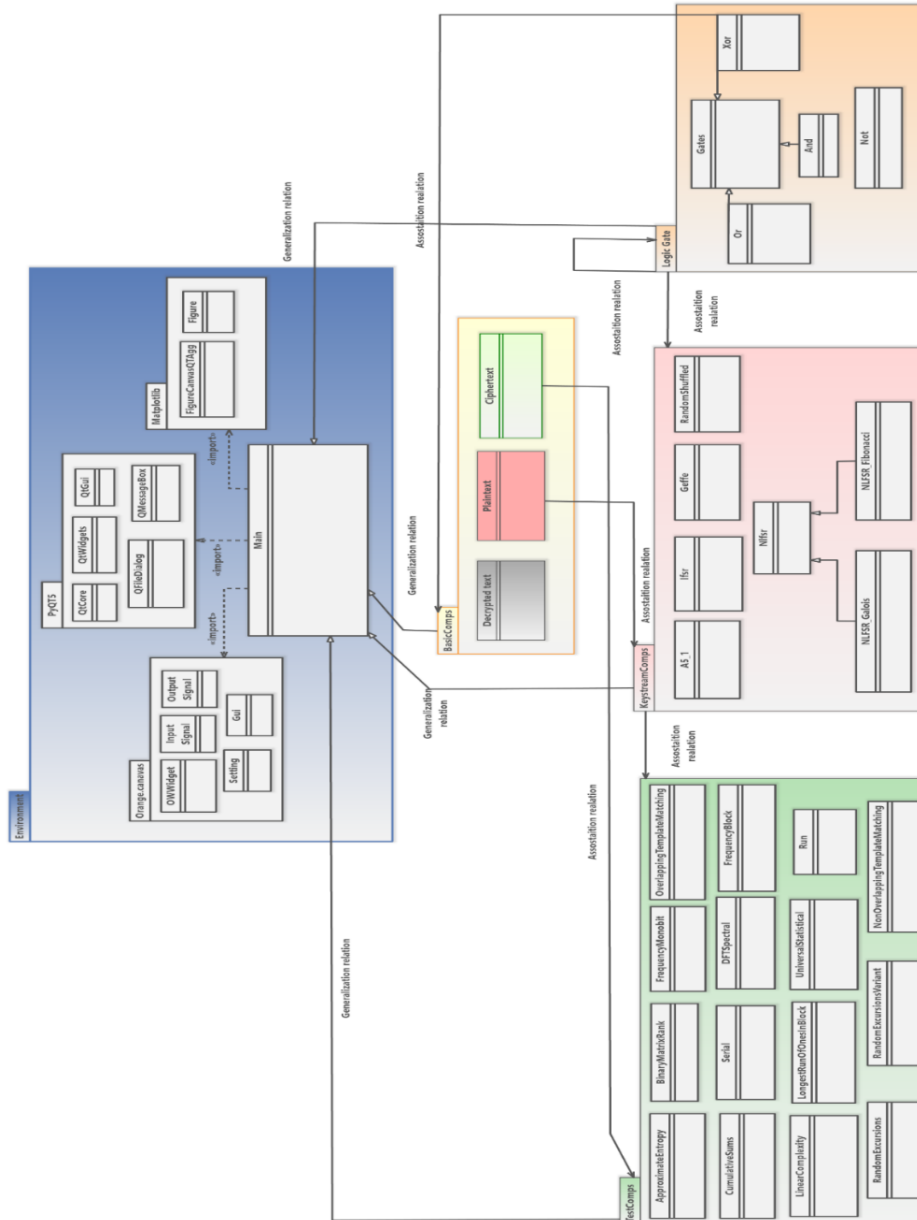


Fig. 3. The architecture of SCLang language

The principles of the SC domain are used to construct the SCLang meta-model. The model is divided into five packages, as shown in Fig. 4. Each package consists of a number of related classes to represent all SC concepts and their relationships.



The first package (Environment) is for defining all aspects of the graphical environment implementation. The presented graphical environment is based on

PyQt5, Matplotlib, and Orange canvas libraries. The four packages in the SCLang meta-model are described as follows.

- **Basic Comps** is the main package of the SCLang meta-model. It contains three classes (components): Plaintext, Ciphertext, and Decryptedtext. The basic scenario of the encryption process involves using Plaintext and Ciphertext classes respectively. On the other hand, Decryptedtext and Ciphertext are used for decryption process. Plain text is encoded to a byte array, and each byte is converted into a bit array. UCS Transformation Format 8 (UTF-8) is used for encoding. The bit length is computed in both Plaintext and Ciphertext.

- **Logic Gates** is the second package in the SCLang meta-model, consisting of two main classes: the NOT class and the Gates class. OR, XOR, and AND are derivative from the Gate class. The classes in this package use all classes in the Basic Comps and Keystream Comps packages based on directed association relation. The OR, XOR, and AND subclasses use all attributes and methods defined in the superclass Gate class in addition to their own attributes and methods.

- **Keystream Comps** is a package with five main classes for six different keystream generation methods. These methods are (Section 2.1): LFSR, Combine LFSRs, NLFSR (Fibonacci and Galois types), randomly shuffled, and the Geffe method. Each method of keystream generation is implemented as a separate component. Each method has fundamental parameters such as length of the shift register, initial state, number and value of tap locations, and the equation form in some of these methods. The ability to set these parameter values and reconfigure them during the run time is provided by SCLang. In the encryption process, the cipher schema is configured by connecting a plaintext component with a keystream generation method; each one connects to the same XOR gate component, and the XOR component connects to a ciphertext component. The difference in decryption is that the ciphertext component is connected with a keystream generation method and the XOR component connects with the decryptedtext component. The length of plaintext/ciphertext in the encryption/decryption process is passed to the keystream generation component. Keystream bits equal in length are generated based on the configured parameter values of the method.

The parameters in the LFSR shift register are the initial state, shift register length, and the number and value of tap locations. The ability to set these values and reconfigure them during the run time is provided by SCLang. The LFSR component form is explained in Fig. 5 (1). Combining LFSRs is the second keystream generator method provided by SCLang. This method is based on the A5/1 cipher algorithm for keystream generation. It consists of three LFSRs (R1, R2, R3). The initial states of these LFSRs and the tap location values are determined by the user. Combining LFSRs is explained in Fig. 5 (2). In the Geffe keystream generator method, the user can set the number of LFSRs according to the equation  $(2*n+1)$  and determine the length, initial state, and tap locations (number and value) for each LFSR. This method is explained in Fig. 5 (3). In the Fibonacci NLFSR keystream generator method user can set the length of the shift register, its initial state, and configure the nonlinear equation, as shown in Fig. 5 (4). In the Galois NLFSR keystream generator method user can set the length of the shift register, its initial state, and configure the two

nonlinear equations, as shown in Fig. 5 (5). In the random shuffled keystream generator method user can set the length of the shift register, its initial state, and the value of  $(n)$ . According to the value of  $n$ , the values of each T and S are initiated. The random shuffled method is explained in Fig. 5 (6).

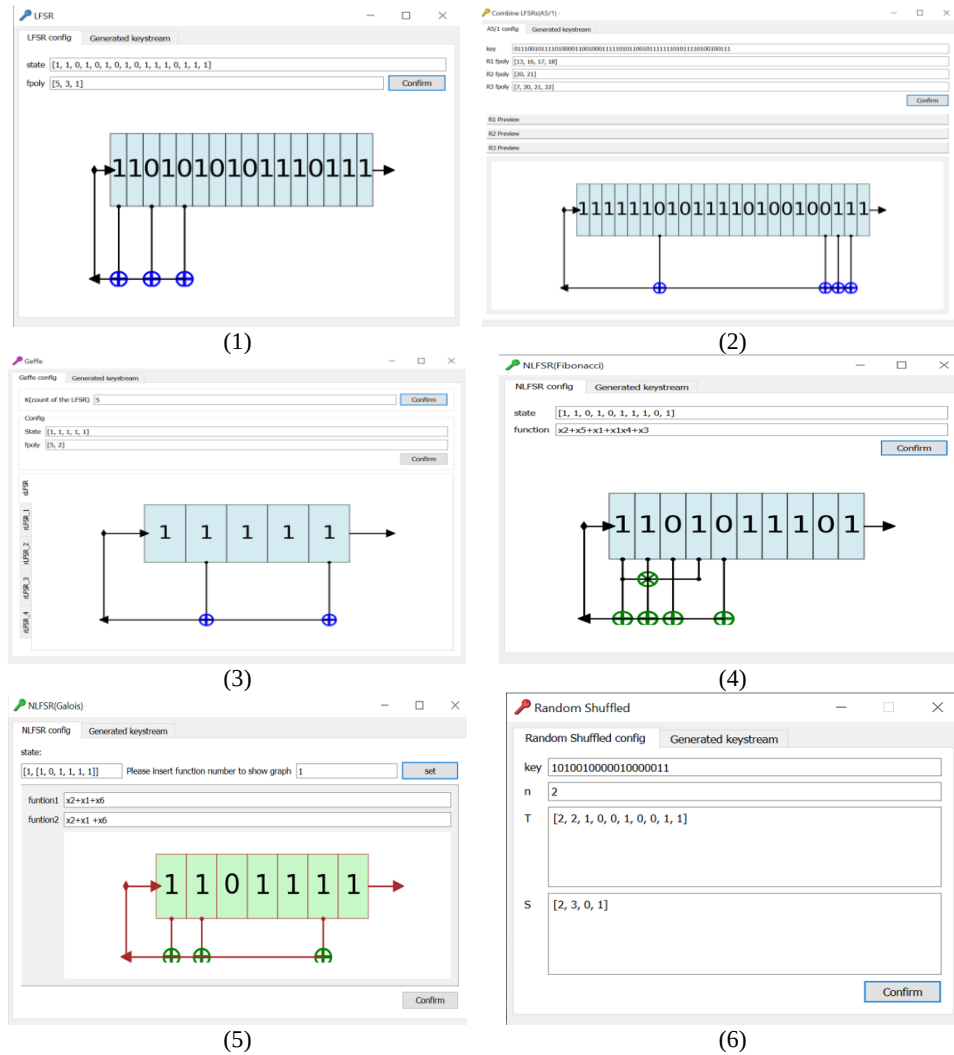


Fig. 5. Keystream generator methods; (1) LFSR form; (2) Combine LFSRs form; (3) Geffe form; (4) Fibonacci NLFSR form; (5) Galois NLFSR form; (6) Random Shuffled form

- **Test Comps** is the last package in the SCLang meta-model, it consists of fifteen classes that represent NIST tests. These tests are implemented as reported in [32]. Three of the tests are shown in Fig. 6. This package is used through directed association relation by the ciphertext class in the Basic Comps package and by any class in the Keystream Comps package.

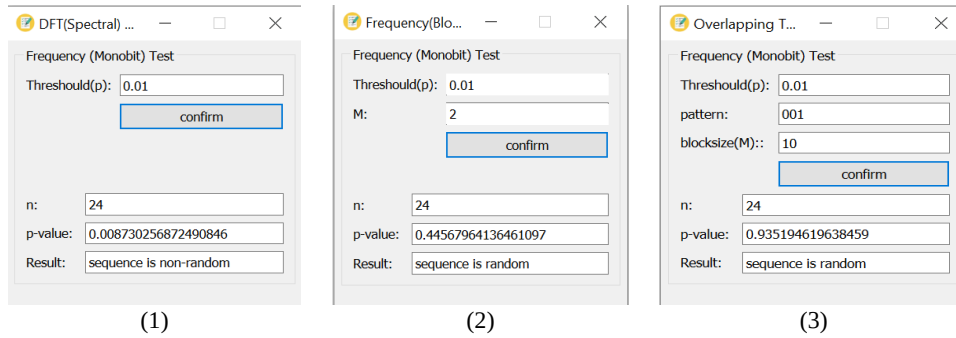


Fig. 6. NIST tests; (1) DFT(spectral) test form; (2) frequency(Block) Test form; (3) overlapping template matching test form

### 3.1.2. Domain rules

The validation rules of the SC domain are represented by the relations between classes in the meta-model. Table 1 shows the validation rules and their descriptions; any SC schema (SC instance model) confirmed in the meta-model must satisfy these rules.

Table 1. Description of SCLang domain rules

| Package Name    | Rules description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Basic Comps     | <p>Ciphertext class in this package uses (by association direction relation) all classes in the Test Comps package</p> <p>Decryptedtext class in this package uses (by directed association relation) all classes in the Logic Gates package</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Logic Gates     | <p>In the Logic Gates package, the three logic gates classes XOR, OR, AND are based on the Gate class</p> <p>Each class in this package uses the plaintext class in Basic Comps</p> <p>Each class in this package uses all classes in the Keystream Comps package</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Keystream Comps | <p>In this package, there are two classes (Fibonacci and Galois classes), each of which is a type (subclass) of NLFSR</p> <p>Each class in this package can use (by directed association relation) all classes in the Test Comps package</p> <p>The generated keystream length should be equal to the plaintext length in the encryption process. Each class in the Keystream Comps package should use (by directed association relation) the plaintext class in the Basic Comps package</p> <p>The generated keystream length should be equal to the ciphertext length in the decryption process. Each class in the Keystream Comps package should use (by directed association relation) the Decryptedtext class in the Basic Comps package</p> <p>Each keystream generation method should have an initial value</p> |

### 3.2. Concrete syntax of SCLang

There are two primary types of concrete syntax: Graphical Concrete Syntax (GCS) and Textual Concrete Syntax (TCS). Both are used in various languages. The concrete syntax type used by SCLang is GCS. A collection of appropriate icons for the meta-elements in the meta-model have been chosen with this objective and the ideas and relations of the SC domain in mind. The next subsections provide a detailed explanation of the concrete syntax for meta-elements and their connections.

### 3.2.1. Concrete syntax of components

After defining all concepts (components) and the relations (connections) between them, a meaningful icon that describes the concept is selected for each component, and the description for each connection is determined.

Table 2. GCS of SCLang components

| Package name    | As component                                                                                                    | Its connections |                                                                                                                                   |                      |
|-----------------|-----------------------------------------------------------------------------------------------------------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------|----------------------|
|                 |                                                                                                                 | I/O             | Details                                                                                                                           | Number of connection |
| Basic Comps     | <br>Plain text                 | Its Input       | -                                                                                                                                 | -                    |
|                 |                                                                                                                 | Its Output      | Connect to any class in Keystream Comps package and any class in Logic Gates package                                              | many                 |
|                 | <br>Chiper text                | Its Input       | Accept connect from any class in Logic Gates package                                                                              | one                  |
|                 |                                                                                                                 | Its Output      | Connect to any class in Test Comps package and any class in Logic Gates package                                                   | many                 |
|                 | <br>Decrypted text             | Its Input       | Accept connect from xor class in Logic Gates package                                                                              | one                  |
|                 |                                                                                                                 | Its Output      | -                                                                                                                                 | -                    |
| Keystream Comps | <br>Combine LFSRs(A5/1)       | Its Input       | Accept connect from plaintext or /and ciphertext class in Basic Comps package                                                     | one                  |
|                 |                                                                                                                 | Its Output      | Connect to any class in Logic Gates package and any class in Test Comps package                                                   | many                 |
|                 | <br>Geffe                    | Its Input       | Accept connect from plaintext or /and ciphertext class in Basic Comps package                                                     | one                  |
|                 |                                                                                                                 | Its Output      | Connect to any class in Logic Gates package and any class in Test Comps package                                                   | many                 |
| Logic Gates     | <br>and                      | Its Input       | Accept connect from plaintext class in Basic Comps package and any class in Keystream Comps package                               | two                  |
|                 |                                                                                                                 | Its Output      | Connect to ciphertext class in Basic Comps package and any class in Logic Gates                                                   | many                 |
|                 | <br>xor                      | Its Input       | Accept connect from plaintext, ciphertext class in Basic Comps package, and any class in Keystream Comps package                  | two                  |
|                 |                                                                                                                 | Its Output      | Connect to ciphertext and Decrypted class in Basic Comps package and any class in Logic Gates package and Keystream Comps package | many                 |
| Test Comps      | <br>Approximate Entropy Test | Its Input       | Accept connect from ciphertext class in Basic Comps package and any class in Keystream Comps package                              | one                  |
|                 |                                                                                                                 | Its Output      | -                                                                                                                                 | -                    |
|                 | <br>Binary Matrix Rank Test  | Its Input       | Accept connect from ciphertext class in Basic Comps package and any class in Keystream Comps package                              | one                  |
|                 |                                                                                                                 | Its Output      | -                                                                                                                                 | -                    |

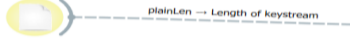
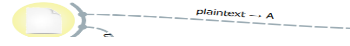
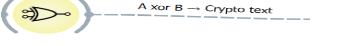
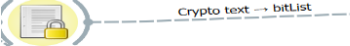
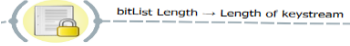
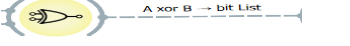
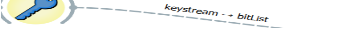
The number of connections in the meta-model can be one-to-one, one-to-many, and many-to-many relationships. For example, a ciphertext component can have more than one connection with NIST tests. The direction of the association relation

determines the source and destination of that relation. Inheritance and directed association are the two types of relations that the SCLang meta-model defines. The established generalized relation in the meta-model syntax naturally imposes some limitations in building the instance model. In an instance model, a subclass has all of the attributes and methods of its super-class, in addition to its own properties and methods. The gate class in the Logic Gates package is the best example. Some GCS of SCLang components are shown in Table 2.

### 3.2.2. Concrete syntax of connections

The relations between the components are represented by edges. Each edge has a description used to explain the type of connection and represent the concrete syntax of the SCLang language, as shown in Table 3.

Table 3. GCS of connections (edges) in SCLang

| Package name    | Edge                                                                                | Description                                   |
|-----------------|-------------------------------------------------------------------------------------|-----------------------------------------------|
| Basic Comps     |    | Connect plaintext with a keystream method     |
| Basic Comps     |    | Connect plaintext with one of the logic gates |
| Logic Gates     |  | Connect xor gate with ciphertext              |
| Keystream Comps |  | Connect keystream method with xor gate        |
| Basic Comps     |  | Connect ciphertext with one test              |
| Basic Comps     |  | Connect ciphertext with xor gate              |
| Basic Comps     |  | Connect ciphertext with a keystream method    |
| Logic Gates     |  | Connect xor with decrypted text               |
| Keystream Comps |  | Connect keystream method with a test          |

### 3.3. Semantics of SCLang

One of the key problems with language development is the provision of concepts. The semantics of the concepts are as important as their syntax. Static semantics based on constraint checks form the foundation of SCLang. The language response for an incorrect connection or attempting to hook a component with one that is not allowed is either an error message or refreshing the previous workspace one step back. Figs 7 (1) and 7 (2) depict valid object diagram models for the encryption and decryption processes, respectively.

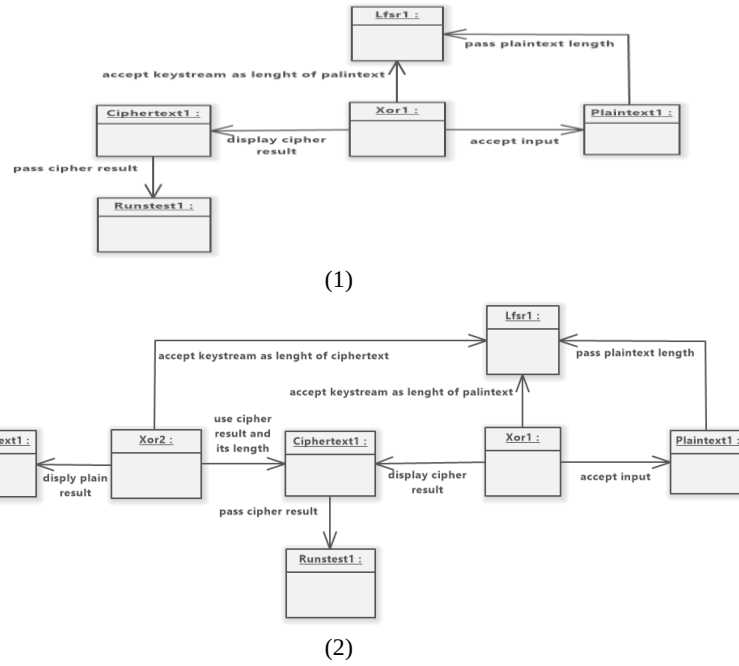


Fig. 7. Object diagram: (1) encryption process; (2) decryption process

#### 4. Implementation

To complete successfully the challenging process of designing SCLang, one needs expertise in language engineering and the SC domain. SCLang is built as a new library of five packages. The host language Python is used to implement it as an internal graphical DSML, with PyCharm serving as the Integrated Development Environment (IDE) using an Intel Core i7CPU, 8 GB of RAM, and the Windows 10 operating system. The graphical user interface is based on PyQt5, Matplotlib, and Orange. canvas library GUI templates. The meta-model has been created using the Software Ideas Modeler tool.

The main user interface of the SCLang consists of two parts. The first part is the dashboard (main control panel), containing the four packages and their components (building blocks such as plaintext, XOR). The second part is the workspace (modeling area); the components are used to configure the cipher schema with drag-and-drop functionality. The user is able to develop and implement programs by constructing SC schemas. After the SC schema is configured, plain text is input and the keystream method parameters are set. The encryption process runs directly, and the encrypted information is presented in the decryptedtext component. SCLang has a capability for randomness analysis, provided by NIST tests. In addition to the other advantages, the SCLang can accept any change or update in plaintext input or keystream parameters while the program (cipher schema) is running.

Five examples of SC schemas that can be constructed and implemented by SCLang are demonstrated in Figs 8-12, respectively.

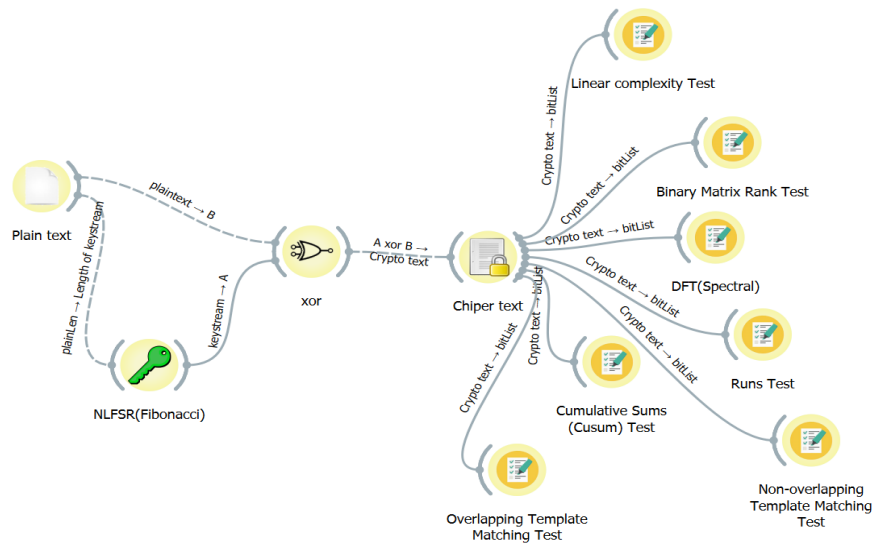


Fig. 8. Example\_1. This example shows that all NIST tests can be connected to the ciphertext constituent to compute the randomness of the result

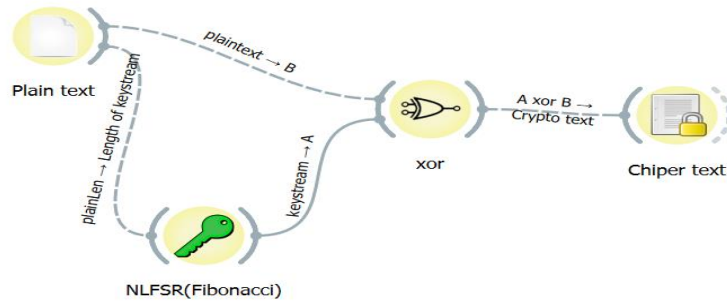


Fig. 9. Example\_2. This is the simplest SC schema that can be created. It is the traditional cipher in SC, where the plaintext is xored with the keystream to obtain the ciphertext and present it using the ciphertext constituent, as explained in this example

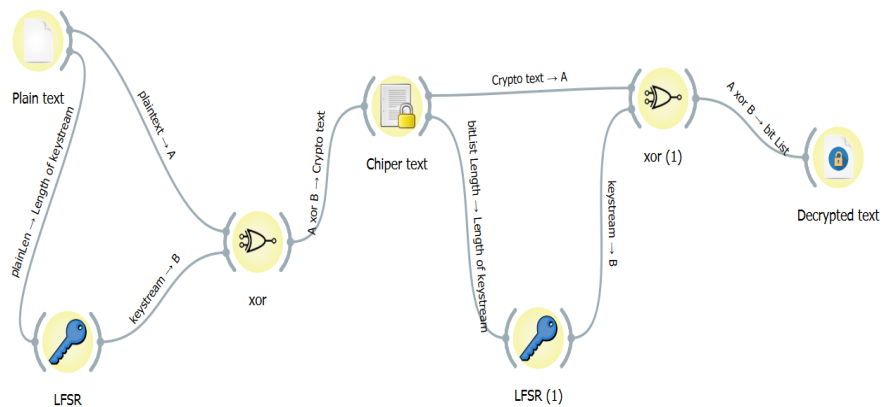


Fig. 10. Example\_3. The encryption and decryption processes are shown in this example

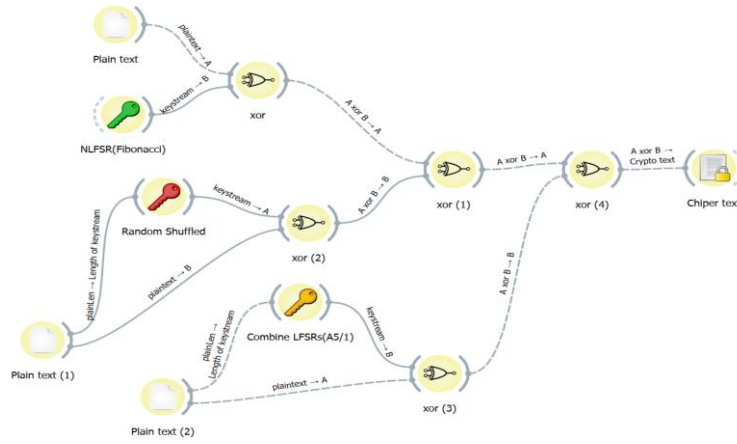


Fig. 11. Example\_4. This example shows that more than one encryption subsystem can be connected to produce one result

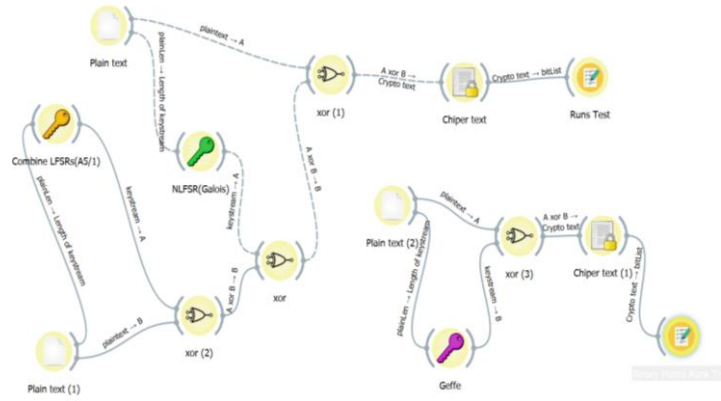


Fig. 12. Example\_5. Parallelism and the ability to configure more than one SC schema in the same workspace are also possible, as explained in this example.

## 5. Evaluation

According to the quality and performance requirements of graphical modeling languages, the assessment used in this study aims to estimate and determine the scope of the proposed language. Five subjective criteria are used for a qualitative analysis based on the “goal-question-metric” paradigm [33]. According to the definition, these five criteria are visual nature, functionality, simplicity of comprehension, paradigm support, and scalability. Table 4 presents a set of more specific metrics for evaluating each of these five criteria. The evaluation of the SCLang based on visual feature criteria achieved “mostly visual” for its initial metric, “meaningful icons” for its second metric, “Applicable throughout the language” for the third, and “Effective use” for all the rest of the metrics. The functional criteria that have been achieved for the first metric are, “specific intent”, and for its second metric, “To one domain”. For the third criterion, it was “much easier” for all the metrics in this category. The evaluation of the presented language based on paradigm support criteria has achieved “very limited” for its first metric and “one domain” for the second metric. For

“Scalability” the last criterion, has achieved “strong” for all its metrics as explained in Table 4.

Table 4. Definitions and assessment standards for graphical language

| Assess                                        | Highest rating                                                                              |                           |                                 | Lowest rating      |
|-----------------------------------------------|---------------------------------------------------------------------------------------------|---------------------------|---------------------------------|--------------------|
| 1. Visual feature                             | How much information is displayed graphically, such as through icons, diagrams, and graphs? |                           |                                 |                    |
| 1.1. Graphics use                             | completely graphic, such as iconic                                                          | mostly visual             | only a little graphic with text | completely textual |
| 1.2. Graphic used type                        | Meaningful icons                                                                            | Less icons                | Very little                     | No icons           |
| 1.3. Graphic use Thoroughness                 | Applicable throughout language                                                              | Less icons                | only a few meanings             | No icons           |
| 1.4. Utilizing space effectively              | Effective use                                                                               | Useful in several aspects | a minimally effective           | insufficient       |
| 1.5. Effectiveness of color use               | Effective use                                                                               | Using color effectively   | minimal use of color            | No use of color    |
| 2. Functionality                              | The language's wide applicability as opposed to its confinement to a single sector of use.  |                           |                                 |                    |
| 2.1. Functioning perfection                   | General purpose                                                                             | lacking some Functions    | Applicable to several areas     | specific intent    |
| 2.2. Naturalness of implementation            | To all domains                                                                              | To many domains           | To very few domains             | To one domain      |
| 3. Ease of comprehension                      | The simplicity with which this language's programming may be comprehended.                  |                           |                                 |                    |
| 3.1. Ease for programmers                     | Much easier                                                                                 | Moderate                  | Moderate                        | Much less          |
| 3.2. simplicity for non-technical programmers | Much easier                                                                                 | Moderate                  | Moderate                        | Much less          |
| 3.3. Expert user                              | Much easier                                                                                 | Moderate                  | Moderate                        | Much less          |
| 4. Paradigm support                           | How well the proposed language supports the programming paradigm it has been designed for?  |                           |                                 |                    |
| 4.1. Support for a paradigm                   | Strong                                                                                      | Moderate                  | Weak                            | very limited       |
| 4.2. Support for a domain                     | All domains                                                                                 | Moderate                  | Very few                        | One domain         |
| 5. Scalability                                | A metric for this language's capacity to write complex programs.                            |                           |                                 |                    |
| 5.1. support for modularity                   | Strong                                                                                      | Moderate                  | Weak                            | Nothing at all     |
| 5.2. support for abstraction                  | Strong                                                                                      | Moderate                  | Weak                            | Nothing at all     |
| 5.3. Support for information concealing       | Strong                                                                                      | Moderate                  | Weak                            | Nothing at all     |
| 5.4. support for data encapsulation           | Strong                                                                                      | Moderate                  | Weak                            | Nothing at all     |

## 6. Conclusion

A graphical DSML for the SC domain is proposed in this paper. For the SC domain, this language offers high-level abstraction. DSMLs have changed the way software is developed in several important ways. They are appropriate in particular areas as they accelerate development, reduce structural complexity, and improve productivity. Instead of creating a general language, a particular language is focused on a single field, allowing it to perform many tasks in a flexible manner. When there are errors, they are specific to the domain; thus, developing a specific language allows for easy understanding and use. SCLang has been proposed for both beginner and expert users working in the cipher domain. SCLang provides the essential components of SC graphically. This allows the implementation of many SC schemas using models. In addition to hiding, the implementation details, SCLang provides a highly expressive graphical user interface with drag-and-drop capability. Users have a versatile, straightforward, and user-friendly method for creating and implementing a wide range of schemas in the SC domain. The main advantage of SCLang is that expert

users can easily generate hybrid keystreams using provided keystream generation methods. Users with no programming language expertise can learn this domain and quickly configure cipher schemas. The fifteen NIST tests allow for analysis of the results. The graphical editor of SCLang enables users to design and configure single-level or multi-level cipher schema in the same workspace. This enables comparison of different analysis results for the same plain text. A key feature of SCLang is the ability to change and reconfigure the cipher schema during the run time. The ability to use hybrid schemas based on the six keystream generation methods allows greater productivity. In future research, SCLang can be extended to include components of other cipher types and other types of randomness analysis tests.

## References

1. Hasan, M. K., S. Muhammad, S. Islam, B. Pandey, Y. A. Baker El-Ebiary, N. S. Nafi, R. C. Rodriguez, D. E. Vargas. Lightweight Cryptographic Algorithms for Guessing Attack Protection in Complex Internet of Things Applications. – Complexity, 2021.
2. Wu, L., H. Cai. Novel Stream Ciphering Algorithm for Big Data Images Using Zeckendorf Representation. – Wireless Communications and Mobile Computing, 2021.
3. Megala, G., P. Swarnalatha. Efficient High-End Video Data Privacy Preservation with Integrity Verification in Cloud Storage. – Computers and Electrical Engineering, Vol. **102**, 2022, 108226.
4. Subramanian, A. K., A. Samanta, S. Manickam, A. Kumar, S. Shiaeles, A. Mahendran. Linear Regression Trust Management System for IoT Systems. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 4, pp.15-27.
5. Poonam, J., B. Singh. RC4 Encryption-A Literature Survey. – Procedia Computer Science, Vol. **46**, 2015, pp. 697-705.
6. Zhang, S., S. Wang, G. Bai, M. Zhang, P. Chen, C. Zhao, S. Li, J. Zhou. Design of Threat Response Modeling Language for Attacker Profile Based on Probability Distribution. – Wireless Communications and Mobile Computing, 2022.
7. Nastov, B., F. Pfister. Experimentation of a Graphical Concrete Syntax Generator for Domain Specific Modeling Languages. – In: Proc. of Conference: INFORSID, 2014, France.
8. Sudeepa, K. B., G. Aithal, V. Rajinikanth, S. C. Satapathy. Genetic Algorithm Based Key Sequence Generation for Cipher System. – Pattern Recognition Letters, Vol. **133**, 2020, pp. 341-348.
9. Mengdi, Z., Z. Xiaojuan, Z. Yayun, M. Siwei. Overview of Randomness Test on Cryptographic Algorithms. – In: Journal of Physics: Conference Series, Vol. **1861**, No 1, 012009. IOP Publishing, 2021.
10. Abd, E.-L., A. Ahmed, A.-E.-A. Bassem, S. E. Venegas-Andraca. Controlled Alternate Quantum Walk-Based Pseudo-Random Number Generator and Its Application to Quantum Color Image Encryption. – Physica A: Statistical Mechanics and Its Applications, Vol. **547**, 2020, 123869.
11. Karakaya, B., A. Gültén, M. Frasca. A True Random Bit Generator Based on a Memristive Chaotic Circuit: Analysis, Design and FPGA Implementation. – Chaos, Solitons & Fractals, Vol. **119**, 2019, pp. 143-149.
12. Lozanovski, B., D. Downing, P. Tran, D. Shidid, M. Qian, P. Choong, M. Brandt, M. Leary. A Monte Carlo Simulation-Based Approach to Realistic Modelling of Additively Manufactured Lattice Structures. – Additive Manufacturing, Vol. **32**, 2020, 101092.
13. Kordov, K., G. Dimitrov. A New Symmetric Digital Video Encryption Model. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 1, pp. 50-61.
14. Zhong, J., D. Lin. Decomposition of Nonlinear Feedback Shift Registers Based on Boolean Networks. – Science China Information Sciences, Vol. **62**, 2019, No 3, pp. 1-3.

15. Jiao, L., Y. Hao, D. Feng. Stream Cipher Designs: A Review. – Science China Information Sciences, Vol. **63**, 2020, No 3, pp. 1-25.
16. Deb, S., B. Bhuyan. Performance Analysis of Current Lightweight Stream Ciphers for Constrained Environments. – Sādhanā, Vol. **45**, 2020, No 1, pp. 1-12.
17. Qasaimeh, M., R. S. Al-Qassas, S. Tedmori. Software Randomness Analysis and Evaluation of Lightweight Ciphers: The Prospective for IoT Security. – Multimedia Tools and Applications, Vol. **77**, 2018, No 14, pp. 18415-18449.
18. Lazarov, A. D., P. Petrova. Modelling Activity of a Malicious User in Computer Networks. – Cybernetics and Information Technologies, Vol. **22**, 2022, No 2, pp. 86-95.
19. Shen, L., X. Chen, R. Liu, H. Wang, G. Ji. Domain-Specific Language Techniques for Visual Computing: A Comprehensive Study. – Archives of Computational Methods in Engineering, Vol. **28**, 2021, No 4, pp. 3113-3134.
20. Handzhyski, N., E. Somova. Tunnel Parsing with the Token's Lexeme. – Cybernetics and Information Technologies, Vol. **22**, 2022, No 2, pp. 125-144.
21. Boytcheva, S. Overview of Inductive Logic Programming (ILP) Systems. – Cybernetics and Information Technologies, Vol. **2**, 2002, No 1, pp. 27-36.
22. Challenger, M., B. T. Tezel, V. Amaral, M. Goulao, G. Kardas. Agent-Based Cyber-Physical System Development with Sea\_ml++. – In: Multi-Paradigm Modelling Approaches for Cyber-Physical Systems. Elsevier, 2021, pp. 195-219.
23. Mosteller, D., M. Haustermann, D. Moldt, D. Schmitz. Integrated Simulation of Domain-Specific Modeling Languages with Petri Net-Based Transformational Semantics. – In: Transactions on Petri Nets and Other Models of Concurrency XIV. Berlin, Heidelberg, Springer, 2019, pp. 101-125.
24. Shekhovtsov, V. A., S. Ranasinghe, H. C. Mayr, J. Michael. Domain Specific Models as System Links. – In: Proc. of International Conference on Conceptual Modeling, Cham, Springer, 2018, pp. 330-340.
25. Elaheh, A. M., A. M. Elham, C. Moharram. DSML4CP: A Domain-Specific Modeling Language for Concurrent Programming. – Computer Languages, Systems & Structures, Vol. **44**, 2015, pp. 319-341.
26. Juan, B.-P., O. Guadalupe, M.-B. Inmaculada. Model4CEP: Graphical Domain-Specific Modeling Languages for CEP Domains and Event Pattern. – Expert Systems with Applications, Vol. **42**, 30 November 2015, Issue 21, pp. 8095-8110.
27. Campos, C., R. Grangel. A Domain-Specific Modelling Language for Corporate Social Responsibility (CSR). – Computers in Industry, Vol. **97**, 2018, pp. 97-110.
28. Chunlin, G., Y. Ao, D. Du, F. Mallet. XSHS: An Executable Domain-Specific Modeling Language for Modeling Stochastic and Hybrid Behaviors of Cyber-Physical Systems. – In: Proc. of 25th Asia-Pacific Software Engineering Conference (APSEC'18), IEEE, 2018, pp. 683-687.
29. Marzina, V., T. Massoni, F. Ramalho. A Domain-Specific Language for Verifying Software Requirement Constraints. – Science of Computer Programming, Vol. **197**, 1 October 2020, 102509.
30. Sadik, A., K. Geylani. DSML4DT: A Domain-Specific Modeling Language for Device Tree Software. – Computers in Industry, Vol. **115**, 2020, 103179.
31. Ana, P. S., S. Babarović, O. Pantelić, S. Krstović. Towards a Domain-Specific Modeling Language for Extracting Event Logs from ERP Systems. – Applied Sciences, Vol. **11**, 2021, No 12, 5476.
32. Rukhin, A., J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks et al. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. – NIST Special Publication, 800-22 (revised 15 May 2002).
33. Kiper, J. D., E. Howard, C. Ames. Criteria for Evaluation of Visual Programming Languages. – Journal of Visual Languages & Computing, Vol. **8**, 1997, No 2, pp. 175-192.

*Received: 27.10.2022; Second Version: 26.04.2023; Accepted: 05.05.2023*