TEM Journal. Volume 12, Issue 4, pages 2038-2049, ISSN 2217-8309, DOI: 10.18421/TEM124-14, November 2023.

Designing a Graphical Domain-Specific Modeling Language for Efficient Block Cipher Configuration: BCLang

Samar Amil Qassir [1], Methaq Talib Gaata [1], Ahmed T. Sadiq [2], Faiz Al Alawy [3]

[1] *Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq*
[2] *Computer Science Department University of Technology-Iraq, Baghdad, Iraq*
[3] *College of Engineering, Kent State University, Ohio, U.S.A.*

***Abstract* – Block cipher (BC) is a type of symmetric cipher used to encrypt data. Despite its advantages, it faces a substantial challenge. Writing the script code for the BC scheme accurately using General-Purpose Programming Languages (GPPLs) poses a significant challenge for programmers. The aim of this paper is to present the first graphical domain-specific modeling language (DSML) for designing and implementing BC algorithms, called BCLang. It is an extension to our previous DSML that was developed for Stream cipher. Programming efficiency and expressiveness were increased by reducing grammar and runtime errors and providing a high level of abstraction. BCLang provides the fundamental components of the BC three structures, which enable the programmer to design and implement BC algorithms in a graphical manner. Two keystream generation methods, performance analysis, and tests of the National Institute of Standards and Technology (NIST) for randomness analysis were provided. The presented language was evaluated based on five subjective metrics specific to graphical DSML evaluation. The design, evaluation details, and properties are explained in depth in this paper.**

DOI: 10.18421/TEM124-14
https://doi.org/10.18421/TEM124-14

Corresponding author: Samar A. Qassir,
Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq
Email: samarqassir@uomustansiriyah.edu.iq

Received: 01 June 2023.
Revised: 02 September 2023.
Accepted: 09 September 2023.
Published: 27 November 2023.

***Keywords* – Cryptography, block cipher, domain specific language, domain-specific modeling language, meta-model.**

1. Introduction

Cryptography involves using codes and ciphers to safeguard data and information from unauthorized access [1]. The encryption process transforms plain text into a form that only the intended recipient can understand. It is a crucial aspect of information security [2]. Block ciphers (BC) are the cornerstone of security applications. They encrypt blocks of plaintext bits (typically 64 or 128 bits) using a symmetric key method to create an equally sized block of cipher text bits [3].

The first type, substitution permutation networks (SPN), is a product cipher that uses a substitution layer for confusion and a permutation layer for diffusion at each round. To simulate non-linearity in confusion, the substitution function applies several substitution boxes (S-boxes such as S1,2, 3, ..., n) across the data. The permutation function can be produced using a fixed bit permutation or a matrix operation. In the permutation layer, the operation is applied throughout the entire block to disperse the relationship between different pieces of data. The round of an SPN is completed by applying the round keys (K[i]) to the block data [4]. Iterated block ciphers belong to the second category of classic feistel networks (CFN). The plain text is divided into two halves, (L[i]) and (R[i]), and the round transformation function (F) is applied to one half of the data. The other half is then processed with the result using an operation like XOR (⊕), and the two halves are then swapped. After multiple identical rounds, the cipher text is produced. The round keys (K[i]), which the key scheduling procedure derives from the seed key, are applied to the internal rounds. The structure facilitates decoding by using reverse-ordered round keys [5].

The third type, addition rotation XOR (ARX), is created using only three simple operations: modular addition, bitwise rotation, and ⊕. Due to its small size and simplicity, ARX has garnered significant attention from both academia and industry. ARX utilizes a combination of linear and non-linear processes, such as ⊕, bit shift, bit rotation, and modular addition [6], [7], [8].

To simplify the implementation of BC algorithms and make them faster and more efficient, we employ a domain-specific Modeling language (DSML) approach [9], [10], [11]. DSML offers a higher level of abstraction than general-purpose programming languages (GPPLs) by providing a DSML-based executable modeling editor, which is more flexible and precise toward a specific domain, thus reducing programming complexity. Rather than dealing with coding details, programmers work with a model of the program. DSML allows the development of 2D/3D object behavioral models in a particular modeling environment [14].

DSMLs are used to create models and are defined by two distinct parts. The first is the abstract syntax, which consists of a meta-model that describes the language concepts and their relationships. The second is the concrete syntax, or DSML notation, which is a set of graphical symbols used to draw diagrams that facilitate the understanding and visualization of the model [15], [16], [17]. This research introduces BCLang, a new graphical DSML that enhances the simplicity, flexibility, and expressiveness of BC design and implementation. BCLang automates the transformation of plain text to corresponding cipher text and caters to both novice and experienced programmers. The language comprises the main building blocks of CFN, SPN, and ARX, along with two keystream generation methods. Additionally, it offers fifteen NIST suite tests [18], [19] as graphical constituents and evaluation components for statistical and performance analysis of encrypted results.

The remaining sections of this paper are arranged as follows: Section 2 discusses related work on DSMLs, followed by the development of BCLang in Section 3. In Section 4, BCLang implementation and three case studies are presented. Section 5 reports the evaluation results, while Section 6 concludes the paper and offers suggestions for future research.

2. Related Work

A programmer who uses a GPPL is able to create a program in any field for a wide range of application domains. But each GPPL has its difficulties; some of them are sensitive to space, small, or capital letters [11]. A program's design is a genuine difficulty; even a small program can require the naming of many things like variables, procedures, functions, classes, objects, etc.

Thus, if the programmer is a beginner, he needs to first learn the syntax of that GPPL before trying to write the code, debug all bugs, and implement the program. In comparison to GPPLs, DSMLs provide various benefits for expressing a particular domain. One benefit is that it offers greater abstractions for the target domain, increasing output and improving the standard of the development process. The presented BCLang language alleviates the programming complexity of the GPPLs through the use of simplified interfaces; it provides interactive visualizations; and the user-friendly, common interactive GUI with drag-and-drop capability allows for fruitful and interactive use for creating and implementing a wide range of BC domain schemas. Our research on stream cipher and other projects across various application domains consistently demonstrate a common finding: that the development of programming languages using the DSML approach achieves more efficiency, simplicity, expressiveness, and better validation and verification for programs designed for specific application domains.

In our previous work [20], we defined a graphical DSML "SCLang" that significantly increases the flexibility, expressiveness, and ease for stream cipher schema design and implementation. SCLang is developed for both beginner and expert programmers. It provides the main components in a graphic manner to construct the stream cipher schema: six different keystream generation methods that can be used in a hybrid fashion (one or multi-levels), four logic gates, and fifteen tests of the NIST suite to facilitate statistical analysis of encrypted results. The abstract syntax of SCLang consists of five packages, along with its restrictions based on domain concepts. For the concrete syntax, meaningful icons for meta_elements were chosen in addition to the static type used to define the semantics. The proposed SCLang allows for reducing the complexities and testing of a generated random sequence by providing a higher abstraction level, generating the random sequence automatically, enhancing the performance of the cipher schema (in both design and implementation), and increasing efficiency by reducing the likelihood of mistakes.

Challenger *et al*. [21] developed a DSML for MASs known as SEA ML, which defines the language's syntax and semantics.

They illustrated how model-driven development could be used to build actual MASs using the language's graphical capabilities. The proposed DSML includes new perspectives that make it easier to develop software agents that operate in the Semantic Web environment. They also presented a recommended approach for building MASs based on SEA ML and a demonstration of how it can be used to create an agent-based stock market system.

Berendes *et al*. [22] proposed the High Street Journey Modeling Language (HSJML), a DSML that facilitates the analysis and planning of customer trips in a digital high street retail setting. The proposed language may be used to develop and implement online-offline customer journeys in digital high streets. Based on an empirical analysis of event logs from retail platforms, the proposed language provides the necessary elements for mapping, analyzing, and forecasting online-offline customer journeys in local high streets from a design perspective. Zweihoff *et al*. [23] have introduced Pyro, a framework that allows for DSML through the internet. Pyro transforms a web browser into a domain-specific graphical development environment with features similar to integrated development environments (IDEs), given an appropriate meta-model definition. Pyro provides high-level, simplicity-focused assistance for the required metamodeling, while the Meta Style Language (MSL) specifies the visual appearance of the modeling artifacts described in the Meta Graph Language (MGL). The MGL describes the possible types of nodes, edges, and syntactical constraints. The browser-based domain-specific development environment is constructed entirely automatically based on these requirements using architecture analysis and design language (AADL).

Alaca *et al*. [24] have developed a comprehensive evaluation framework and related tool, named AgentDSM-Eval, to systematically assess domain-specific modeling languages (DSMLs) for multi-agent systems (MASs). This framework enables the evaluation of various quantitative and qualitative aspects of agent software development, including the extent of domain coverage of MAS DSMLs, adoption of modeling aspects by agent developers, and the performance of MAS DSMLs in terms of development time and throughput. Furthermore, AgentDSM-Eval introduces new quality traits and metrics that are specifically developed for the MAS domain to evaluate the quality of MAS DSML features [12], [13].

This framework and tool can be used to improve the design and development of MAS DSMLs and enhance the quality of agent-based software systems.

In their research, Vjetica *et al.* [25] developed a system for formally defining and automatically executing production processes using model-driven (MD) principles and DSMLs.

The main management tools in this system are models of the production process. The study examined the production process modeling domain and proposed a DSML that can be used to generate models of the production process suitable for automatic code generation. The resulting code can be used to automate manufacturing operations in a shop floor or simulation and also identify potential faults that may occur during the process, as well as error handling and corrective actions. The DSML was evaluated by several user groups.

3. The BCLang

The abstract syntax of a graphical language consists of classes and relations that represent the concepts of the application domain and their relationships. This abstract syntax is represented by the meta-model, which defines the syntax and structure of the graphical language. On the other hand, the concrete syntax maps the domain concepts to their representations in model elements, as depicted in Figure 1. The concrete syntax can be either visual or textual. The proposed graphical language employs the visual type, which uses meaningful icons to make the language more intuitive and user-friendly in configuring BC models. Effective graphical modeling languages should provide specific visual representations that enable programmers to have a clear and accurate understanding of the models being expressed. The proposed language's abstract syntax is explained using a meta-model, which outlines how the model should be structured and describes the abstract syntax. This meta-model is made up of seven packages, which are linked together by either an inheritance or association relation, as illustrated in Figure 2. Each of these seven packages is described in detail as follows:

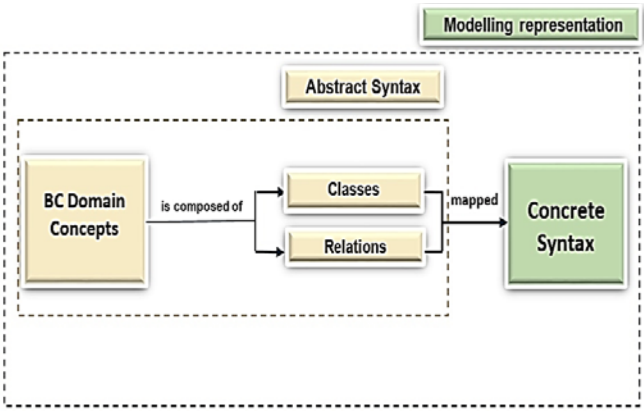


Figure 1. The architecture of proposed language

3.1. Package1

The first package of the proposed language defines all aspects of the graphical environment implementation. The graphical environment is built upon three different libraries, as illustrated in Figure 2. All of the other packages and their classes inherit their graphical details from Package1 through a generalized relation, as shown in Figure 2.

3.2. Package2

The second package of BCLang contains eleven classes, such as Plaintext, Ciphertext, and others, as explained in Figure 2 and Table 1. These classes are considered fundamental and are used in all models for the BC types, including CFN, SPN, and ARX, in the basic scenario of their encryption/decryption processes.

In BCLang, Plaintext is first encoded into a byte array, and then each byte is converted into a bit array. Encoding is carried out using the 8-bit Unicode Transformation Format (UTF-8).

3.3. Package3

The third package in the BCLang meta-model consists of four classes: ToState, SPNRounds, SPNDecrypted, and SPNKeyGene. This package utilizes the Performance class from Package7 and classes such as Plaintext, Ciphertext, Geffe, LFSR, ToBlock, and KeySize from Package2. The package details are explained in Table 2.

3.4. Package4

The fourth package in the BCLang meta-model consists of four classes: Into4Parts, ARXRounds, ARXDecrypted, and ARXKeyGeneration. This package utilizes the classes of Package7 and classes such as Plaintext, Ciphertext, Geffe, LFSR, ToBlock, and KeySize from Package2. The package details are explained in Table 3.

3.5. Package5

The fifth package in the BCLang meta-model consists of five classes: InitialPermutation, Finalpermutation, CFNRounds, CFNDecrypted, and CFN_KeyGen.

The TripleDESRounds and CFNDESRounds classes are subclasses of the CFNRounds superclass, while TripleDESDecrypted and DES_Decrypted classes are subclasses of the CFNDecrypted superclass. The CFNRounds class and CFNDecrypted utilize the CFNKeyGen based on a directed association relation The CFNDecrypted class also uses the Finalpermutation classes based on a directed association relation; this package utilizes the Performance class from Package 7 and all classes in Package2, as explained in Table 4.

3.6. Package6

The sixth package in the BCLang meta-model consists of fifteen classes that represent NIST tests: Frequency, Frequency within a Block, Runs, Longest-Run-of-Ones in a Block, Binary Matrix Rank, Discrete Fourier Transform, Non-Overlapping Template Matching, Overlapping Template Matching, Maurer's Universal Statistical, Linear Complexity, Serial, The Approximate Entropy, Cumulative Sums, Random Excursions, and Random Excursions Variant. These tests are implemented as reported in [19]. All the classes in this package are utilized through a directed association relation by the Ciphertext class in Package2. The package details are explained in Table 5.

3.7. Package7

The last package in the BCLang meta-model consists of two classes: Performance and Analyser, as explained in Figure 2 and Table 6. The Analyser class is utilized through directed association relation by the Geffe and LFSR classes in Package2. On the other hand, the Performance class is used through directed association relation by the Rounds and Decrypted classes in Packages 3, 4, and 5. The Analyser class computes each of the autocorrelation, periodicity, balance property, and run-length property of the seed key generated. The Performance class computes each of the encryption time, decryption time, memory used, and the throughput of the BC model.

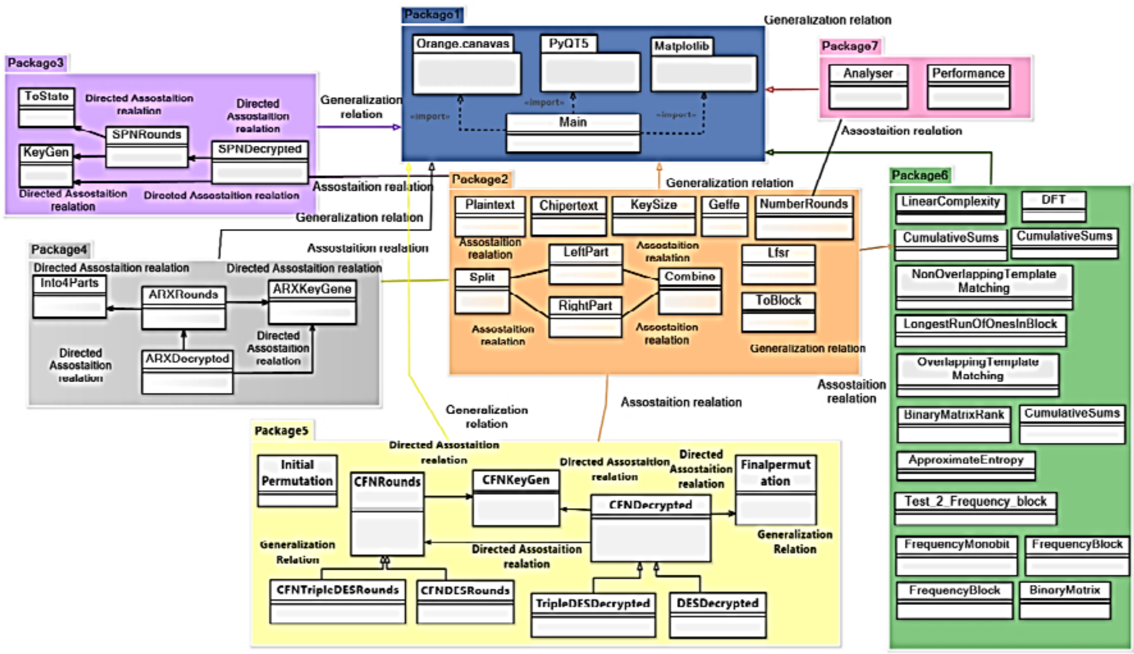


Figure 2. The Meta-model of the proposed language

Table 1. Package2

Class	Side link		No. of links	Accept	Produce
Plaintext	**Left**	-	-	-	bitList bitLength
	Right	Can be linked to ToBlock class.	Many		
	Details	It is the main class; the root of every BC model will be start by this component. It accepts plain text by manually or through load file of these types (*.doc, *.pdf, *.txt).			
Ciphertext	**Left**	Can be linked to Rounds class in packages 3, 4, and 5.	One	bitList	bitList bitLength
	Right	Can be linked to any class in package6.	Many		
	Details	It is the class used for every BC model as final component, it used to display and save the result of BC schema, as one of these file types (*.doc, *.pdf, *.txt).			
ToBlock	**Left**	Can be linked to the plaintext class in the package2.	One	bitList	blocks
	Right	Can be linked to the ToState in the package3, Split class in the package2, and Into4Parts in the package4.	One		
	Details	It is the second class that is used for the BC models; it is used by all packages 3, 4, and 5.			
Split	**Left**	Can be linked to ToBlock.	One	bitList	Two bitLists
	Right	Can be linked to LeftPart, RightPart in package2, and Rounds in package5.	Two		
	Details	This classis used by package5.			
Combined	**Left**	Can be linked to LeftPart, RightPart in package2.	Two	bitList	Two bitLists
	Right	Can be linked to Ciphertext.	One		
	Details	This classis used by packages5.			
LeftPart, RightPart	**Left**	Can be linked to split class.	One	bitList	bitList
	Right	Can be linked to Rounds in package5.	One		
	Details	This classis used by package5.			
KeySize	**Left**	-	-	-	no_k
	Right	Can be linked to Geffe and Lfsr classes in pakage2.	One		
	Details	This classis used by packages 3, 4, 5, it used to determine the key size of BC model.			
Geffe, LFSR	**Left**	Can be linked to the keysize for the BC model in the package2.	One	bitLen	keyList info
	Right	Connect to any class in the package6 and keyGen in packages 3, 4, and 5.	Many		
	Details	Two classes of the keystream generation methods used for the BC models to generate random sequences.			
NumberRounds	**Left**	-	-	-	no_r
	Right	Can be linked to Rounds, KeyGen, and Decrypted class in package5.	Three		
	Details	This classis used by package5, it used to determine the rounds number of BC model.			

Table 2. Package3

Class	Side link		No. of links	Accept	Produce
ToState 	**Left**	Can be linked to ToBlock class in in package2.	One	bitList	stateList
	Right	Can be linked to Rounds Class in package3.	One		
	Details	It is the first class used for SPN BC model to transfer bitList into matrix form.			
Rounds 	**Left**	Can be linked to ToState and KeyGene Class.	Two	stateList, subkey_r	Ciphstate List, Per_Infor
	Right	Can be linked to Decrypted class and Ciphertext class sin package2 , Performance class in package7.	Two		
	Details	This class is used by SPN BC model. It is performing the encryption process.			
Decrypted 	**Left**	Can be linked Rounds, KeyGene classes in package3 and Performance class in package7.	Three	Ciphstate List, subkey_r	stateList, Per_Infor
	Right	Can be linked to performance class in package7.	Two		
	Details	This classis used by SPN BC model. It is performing the decryption process.			
KeyGene 	**Left**	Can be linked to Geffe/Lfsr class in package2.	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds and Decrypted Classes in package3.	Two		
	Details	This classis used by SPN LWBC schema for generating the subkeys required for encryption and decryption processes.			

Table 3. Package4

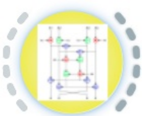
Class	Side link		No. of links	Accept	Produce
Into4Parts 	**Left**	Can be linked to the ToBlock class in the package2.	One	bitList	**4** bitList
	Right	Can be linked to the ARXRounds class in the package4.	One		
	Details	It is the first class used for the ARX BC model to split a bitList into four bitLists.			
Rounds 	**Left**	Can be linked to Left Part, RightPart Classes in package2 and KeyGen class in package4.	Many	Two bitLists, subkey_r	CiphbitL ist, Per_Infor
	Right	Can be linked to ARXDecrypted class and Ciphertext class in package2, Performance class.	Two		
	Details	This class is used by ARX BC model. It is performing the encryption process.			
Decrypted 	**Left**	Can be linked to Rounds and KeyGene Classes in package4	Two	CiphbitL ist, subkey_r ,	bitLists, Per_Infor
	Right	Can be linked to performance Class in package7.	Two		
	Details	This classis used by ARX BC model. It is performing the decryption process.			
KeyGene 	**Left**	Can be linked to Geffe/Lfsr class in package2.	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds and Decrypted Classes in package4.	Two		
	Details	This class is utilized by ARX BC model for generating the subkeys required for encryption and decryption processes.			

Table 4. Package5

Class	Side link		No. of links	Accept	Produce
InitialPermutation	**Left**	Can be linked to the ToBlock class in the Package2 package.	One	bitList	bitList
	Right	Can be linked to the Split class in the package2.	One		
	Details	It is the first class used for CFN BC models. It is performing an initial permutation on the input bitList.			
FinalPermutation	**Left**	Can be linked to the Combined class.	One	bitList	Byte bitList
	Right	Can be linked to the Decrypted class in package5, Ciphertext class package2.	Two		
	Details	It is the last class used for the CFN BC models. It is performing final permutation on the bitList.			
Rounds	**Left**	Can be linked to Left Part, RightPart, NumberRounds, classes in package2 and KeyGen class in package5.	Many	bitLists (Left and Right), no_r, subkey_r	bitLists (Left and Right), Per_Infor
	Right	Can be linked to LeftPart and RightPart classes in package2.	Two		
	Details	This class is used by CFN BC model. It is performing the encryption process.			
Decrypted	**Left**	Can be linked to KeyGen, Rounds classes in CFN and NumberRounds class in package2, Performance class in package7.	Many	bitList, no_r, subkey_r	bitList, Per_Infor
	Right	Can be linked to performance class in package7.	Two		
	Details	This class is used by CFN BC model. It is performing decryption process.			
KeyGen	**Left**	Can be linked to Geffe/Lfsr class in package2.	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds, Decrypted in package5, NumberRounds in package2.	Three		
	Details	This class is used by CFN BC models for generating the subkeys required for encryption/ decryption processes.			

Table 5. Package6

Class	Side link		No. of links	Accept	Produce
For all Tests	**Left**	Can be linked to Ciphertext class, Geffe/Lfsr class in the package2.	One	bitList	-
	Right	-	-		
	Details	Fifteen classes of the NIST tests were used for the BC models to measure the randomness of the results.			

Table 6. Package7

Class	Side link		No. of links	Accept	Produce
Analyzer 	**Left**	Can be linked to Geffe/Lfsr class in the package2.	One	bitLen bitList	Display the results only.
	Right	-	-		
	Details	This class of the package7 is used for the BC models to perform statistical analysis.			
Performance 	**Left**	Can be linked to Rounds class and Decrypted class in packages 3, 4, and 5.	One	2bitList	Display results only.
	Right	-	-		
	Details	This class is used to perform performance analysis.			

The definition of abstract syntax comprises both the concepts in the BC domain and the relations between those concepts. The BC concepts were defined by Packages (1-7).

Table 7 outlines the domain relationships and restrictions between these packages.

Table 7. The relations and restrictions of BCLang

Relation Type	Description	Example
1. Determine the classes number	Control the number of each constituent in BC model	Each BC model should have only one "plaintext" constituent, according to that an error message that appears whenever the user tries to use more than one.
2. The classes relations	Control the relation between constituents, each link between two constituent has a name	A notice stating "Link is missing" will be presented if a link between two constituents in the BC model is neglected to be linked.
3. Determine for number of classes relationship	One-to-one, many-to-one, and one-to-many connections in the meta-model are used to control the amount of relationships between the constituents	One ciphertext constituent is used for each BC model, although the same plaintext constituent can be used in several configurations of the BC model.
4. Determine the start point and end point	Control the relationship's direction to specify the beginning and ending of that BC model	Before creating the relation with the plaintext constituent, it is impossible to construct the link between the "ToBlock" and "Into4Parts" constituents. When something went wrong, BCLang responded by refreshing the workspace and going back one step.
5. The inheritance relation	Control the BCLang-defined inheritance connections. Naturally, a subclass in a model contains all of its superclass's characteristics and connections	Examples of this situation include "CFN DESRound" and "CFN TripleDESRound."
6. The association direction relation	Control the association direction relation defined in BCLang. Naturally, a class in meta-model used another class by one direction use	The “ToBlock” constituent use “plaintext” constituent by one direction. For wrong trying, the BCLang response by refreshing the workspace to one step back

4. Case Studies

The implementation details of BCLang and its examples are discussed in this section. BCLang is developed as an internal graphical Domain-Specific Modeling Language (DSML), based on Python as the host language and PyCharm as the Integrated Development Environment (IDE). The GUI templates for the graphical user interfaces and graphical elements were implemented using PyQt5, Matplotlib, and Orange Canvas libraries.

The BCLang meta-model was created using the Software Ideas Modeler tool. To demonstrate the capabilities of BCLang, we selected well-known BC algorithms such as DES, 3DES, AES, and IDEA as case studies.

Figure 3 illustrates model for encryption and decryption of AES, with two randomness analysis tests of cipher results and performance evaluation. Figure 4 shows model of DES, and Figure 5 presents the IDEA model.

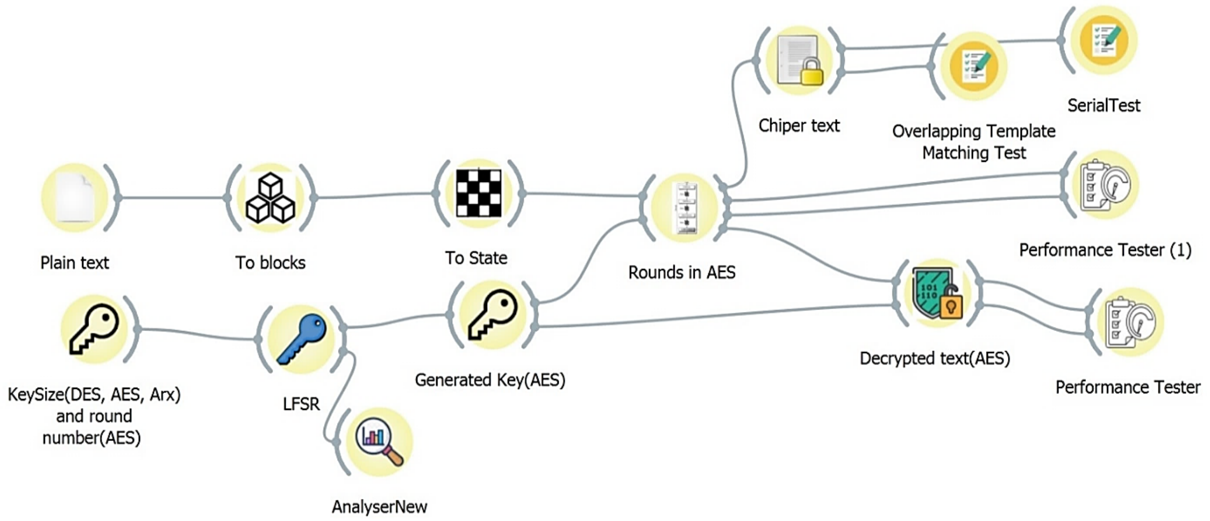


Figure 3. Encryption and decryption model of AES

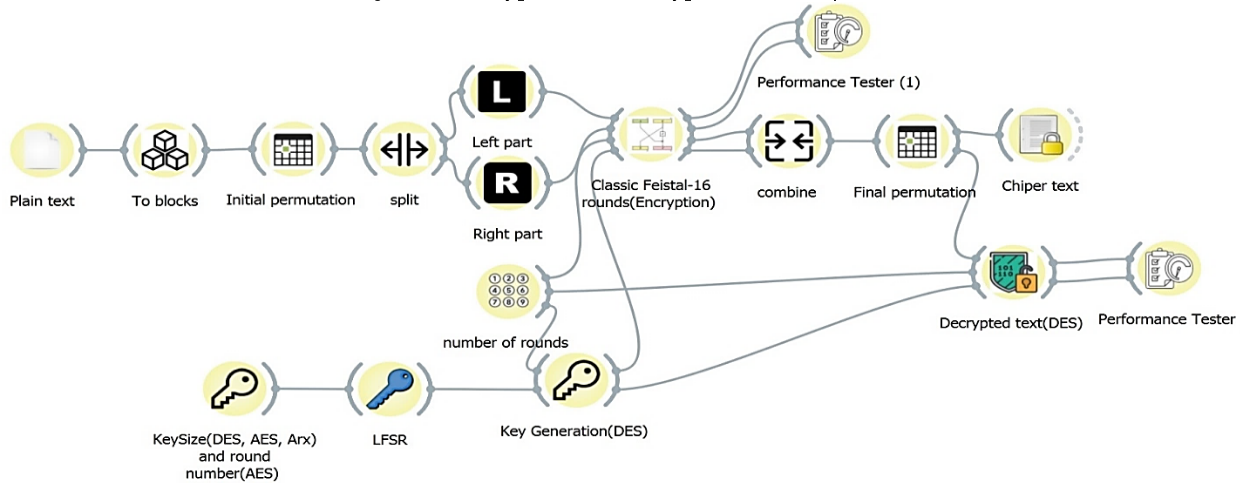


Figure 4. Encryption and decryption model of DES

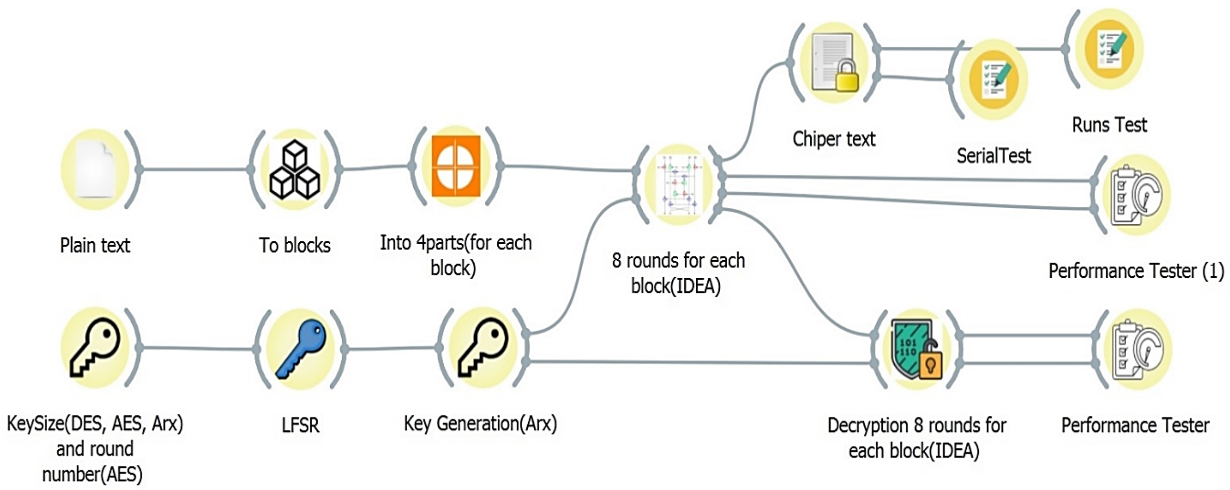


Figure 5. Encryption and decryption model of IDEA

5. Evaluation

In order to evaluate the quality and performance of the proposed BCLang language, a qualitative analysis was conducted based on five subjective criteria using the (goal and question) metric paradigm [26]. The five criteria used were: visual element, functionality, clarity, foundation support, and scalability, which are defined in Table 8 along with a list of more precise metrics for assessing each of these five requirements. The visual element criterion is concerned with the ability of the language to visually represent the concepts and their relationships. The metrics used for this criterion include graphics use, illustration type used, explicit usage thoroughness, using space efficiently, and effectiveness of using color. Functionality is another criterion that evaluates the capability of the language to support the necessary features required for modeling BC algorithms. Metrics for this criterion include support for different BC algorithms, support for different modes of operation, and support for randomness analysis tests. Clarity criterion is concerned with the ease of understanding the modeling elements and relationships for different levels of users.

The metrics used for this criterion include the comprehensibility of the modeling elements and relationships, user satisfaction, and the suitability for novices. Paradigm or (Foundation support) is another criterion that evaluates the ability of the language to support different modeling paradigms. Metrics for this criterion include the support for different abstraction levels, support for different modeling styles, and the ability to describe complex systems. Scalability criterion is concerned with the ability of the language to handle large and complex systems. Metrics for this criterion include the ability to handle large models, the ability to handle complex relationships, and the efficiency of the language. Overall, the analysis of these criteria indicates that BCLang is a promising graphical DSML for BC algorithms. The language supports various BC algorithms, and it provides a clear and consistent graphical representation of the concepts and their relationships. It is easy to comprehend and use for different levels of users, and it supports different modeling paradigms. Furthermore, it is efficient and scalable, making it suitable for handling large and complex systems. The metric table below, as in [26], was presented, and we shadowed in lighted blue to show what is achieved by BCLang as follows:

Table 8. Definitions and assessment for BCLang

Criteria	**highest score**			**lowest score**
Visual element	**How much data is graphically represented, such as through icons, diagrams, and graphs.**			
Graphics use	fully graphical	mainly visual	little graphic with text	fully textual
illustration type used	symbolic icons	fewer icons	Quite little	zero icons
Explicit usage Thoroughness	Applicable	Generally applicable	Generally applicable to half	zero icons
using space efficiently	Use effectively	useful in a variety of ways	a minimally effective	not enough
effectiveness of using color	Use effectively	Putting color to good use	a minimally effective	No use of color
Functionality	**The language's broad applicability as opposed to its focus on a specific field of application.**			
functioning flawlessly	general intent	some functions less	for several areas	specified purpose
Integrity of application	for all domains	for many domains	for few domains	for one domain
Clarity	**The ease with which programming in this language may be understood.**			
Programmers' ease of use	much simpler	Equitable	Equitable	a lot less
simplicity for programmers who are not technical	much simpler	Equitable	Equitable	a lot less
seasoned user	much simpler	Equitable	Equitable	a lot less
Foundation support	**The degree to which the suggested language is compatible with the programming model that it was intended for.**			
in favour of a paradigm	Robust	Equitable	Weak	very limited
assistance with a domain	All domains	Equitable	a lot less	One domain
Scalability	**A metric for this language's capacity to write complex programs.**			
assistance with modularity	Robust	Equitable	Weak	none
assistance with abstraction	Robust	Equitable	Weak	none
assistance with information concealing	Robust	Equitable	Weak	none
assistance with encapsulation	Robust	Equitable	Weak	none

6. Conclusion

The proposed graphical DSML (Domain-Specific Modeling Language) called BCLang offers several benefits for the development of BC (Block Cipher) algorithms. BCLang provides high-level abstraction and a flexible and efficient way of performing tasks in the BC domain. It offers a highly expressive graphical user interface with drag-and-drop capabilities, making it user-friendly and easy to use for both beginner and expert users. BCLang also includes the essential constituents of three basic inner structures of BC algorithms, namely CFN (Feistel network), SPN (Substitution-permutation network), and ARX (Add-Rotate-XOR) structures, in a graphical manner. This helps to hide the implementation details of these structures and make the modeling process more straightforward. Moreover, BCLang includes fifteen tests of NIST and performance evaluation, enabling users to compare different results for the same plain text. The ability to change and reconfigure the BC model during runtime is another key feature of BCLang. Overall, BCLang provides an efficient and flexible approach to modeling and analyzing BC algorithms, making it a valuable tool for researchers and practitioners in the field. For future research, BCLang can be extended to include constituents of other cipher types and randomness analysis tests.

References:

[1]. Abdullah, S. M., & Abduljaleel, I. Q. (2021). Speech Encryption Technique using S-box based on Multi Chaotic Maps. *TEM Journal*, *10*(3), 1429.

[2]. Ullah, S., Zheng, J., Din, N., Hussain, M. T., Ullah, F., & Yousaf, M. (2023). Elliptic Curve Cryptography; Applications, challenges, recent advances, and future trends: A comprehensive survey. *Computer Science Review*, *47*, 100530.

[3]. Halunen, K., & Latvala, O. M. (2021). Review of the use of human senses and capabilities in cryptography. *Computer Science Review*, *39*, 100340.

[4]. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., & Schofnegger, M. (2020). On a generalization of substitution-permutation networks: The HADES design strategy. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 674-704. Springer, Cham.

[5]. Shen, X., Cheng, L., Sun, B., & He, J. (2021). Revisiting Impossible Differential Distinguishers of Two Generalized Feistel Structures. *Security and Communication Networks*, *2021*.

[6]. Huang, M., & Wang, L. (2019). Automatic Search for the Linear (hull) Characteristics of ARX Ciphers: Applied to SPECK, SPARX, Chaskey and CHAM-64 (Full Version). *Cryptology ePrint Archive.*

[7]. Qassir, S. A., Gaata, M. T., & Sadiq, A. T. (2022). Modern and Lightweight Component-based Symmetric Cipher Algorithms. *ARO-THE SCIENTIFIC JOURNAL OF KOYA UNIVERSITY*, *10*(2), 152-168.

[8]. Daimi, K., Francia, G., Ertaul, L., Encinas, L. H., & El-sheikh, E. (Eds.). (2018). *Computer and network security essentials*. Springer.

[9]. Chodarev, S., Sulír, M., Porubän, J., & Kopčáková, M. (2022). Experimental Comparison of Editor Types for Domain-Specific Languages. *Applied Sciences*, *12*(19), 9893.

[10]. Myslín, J., & Kaiser, J. (2022). State Modeling Methodology for Business Processes. *TEM Journal*, *11*(4), 1824-1834.

[11]. Shen, L., Chen, X., Liu, R., Wang, H., & Ji, G. (2021). Domain-specific language techniques for visual computing: a comprehensive study. *Archives of Computational Methods in Engineering*, *28*(4), 3113-3134.

[12]. Bousse, E., Mayerhofer, T., Combemale, B., & Baudry, B. (2019). Advanced and efficient execution trace management for executable domain-specific modeling languages. *Software & Systems Modeling*, *18*(1), 385-421.

[13]. Hutchins, N. M., Biswas, G., Zhang, N., Snyder, C., Lédeczi, Á., & Maróti, M. (2020). Domain-specific modeling languages in computer-based learning environments: A systematic approach to support science learning through computational modeling. *International Journal of Artificial Intelligence in Education*, *30*, 537-580.

[14]. Kosar, T., Oliveira, N., Mernik, M., Pereira, V. J. M., Črepinšek, M., Da, C. D., & Henriques, R. P. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems*, *7*(2), 247-264.

[15]. Deshayes, R. (2013). A domain-specific modeling approach for gestural interaction. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing* ,181-182. IEEE.

[16]. Shim, H., & Fishwick, P. A. (2004, August). A customizable approach to visual programming using dynamic multimodeling. In *enabling technologies for simulation science VIII*, *5423*, 447-456. SPIE.

[17]. Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2015). ModeL4CEP: Graphical domain-specific modeling languages for CEP domains and event patterns. *Expert Systems with Applications*, *42*(21), 8095-8110.

[18]. Balajee, M. K., & Gnanasekar, J. M. (2016). Evaluation of key dependent S-box based data security algorithm using Hamming distance and balanced output. *Tem Journal*, *5*(1), 67.

[19]. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., ... & Vo, S. (2001). *A statistical test suite for random and pseudorandom number generators for cryptographic applications, 22*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

[20]. Qassir, S. A., Gaata, M. T., & Sadiq, A. T. (2023). SCLang: Graphical Domain-Specific Modeling Language for Stream Cipher. *Cybernetics and Information Technologies*, *23*(2), 54-71.

[21]. Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., & Kosar, T. (2014). On the use of a domain-specific modeling language in the development of multiagent systems. *Engineering Applications of Artificial Intelligence*, *28*, 111-141.

[22]. Berendes, C. I., Bartelheimer, C., Betzing, J. H., & Beverungen, D. (2018). Data-driven customer journey mapping in local high streets: a domain-specific modeling language. *A DSML for Customer Journeys in High Streets.*

[23]. Zweihoff, P., Naujokat, S., & Steffen, B. (2019). Pyro: generating domain-specific collaborative online modeling environments. In *International Conference on Fundamental Approaches to Software Engineering*, 101-115. Springer, Cham.

[24]. Alaca, O. F., Tezel, B. T., Challenger, M., Goulão, M., Amaral, V., & Kardas, G. (2021). AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems. *Computer Standards & Interfaces*, *76*, 103513.

[25]. Vještica, M., Dimitrieski, V., Pisarić, M., Kordić, S., Ristić, S., & Luković, I. (2021). Multi-level production process modeling language. *Journal of Computer Languages*, *66*, 101053.

[26]. Kiper, J. D., Howard, E., & Ames, C. (1997). Criteria for evaluation of visual programming languages. *Journal of Visual Languages & Computing*, *8*(2), 175-19.