Building a Graphical Modelling Language for Efficient Homomorphic Encryption Schema Configuration: HomoLang

Samar Amil Qassir [1]

[1] *Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq*

Abstract **– Homomorphic encryption (HE) is an emerging technology that enables computing on data while the data is encrypted. It has advantages, but it also has a significant difficulty. Programmers that use General-Purpose Programming Languages (GPPLs) may find it difficult to handwrite the script code for the HE correctly. This paper presents the front-end compiler design for the first graphical modelling language (DSML) to implement HE schemas, called HomoLang. It is providing a graphical environment with graphical building nodes that represent the HE concepts to enable the building of HE schemas. A high degree of abstraction and a decrease in grammatical and runtime errors improved the expressiveness and efficiency of implementation. Six security tests for security analysis were provided. The efficiency of the submitted language was evaluated using four subjective metrics. This paper provides a detailed explanation of the attributes, evaluation details, and design of the submitted HomoLang.**

Keywords **– Asymmetric cipher, homomorphic encryption, model-driven engineering, graphical modelling language, meta-model.**

DOI: 10.18421/TEM133-56
https://doi.org/10.18421/TEM133-56

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

Received: 25 April 2024.
Revised: 18 June 2024.
Accepted: 09 August 2024.
Published: 27 August 2024.

© 2024 Samar Amil Qassir; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 License.

The article is published with Open Access at https://www.temjournal.com/

1. Introduction

Over the previous several years, there has been a significant increase in the security requirements for algorithms and data. Technology has grown so rapidly that it has made it possible for a wide range of attacks against technological gadgets and digital commodities. There are several options for safely accessing and storing data, including secure data encryption. When one is asked to calculate (publicly) with encrypted data or to change functions so that they may still be executed while maintaining our privacy, the issue gets more complicated. Homomorphic encryption, or HE, can be applied in this situation [1]. In recent years, HE has demonstrated increased performance and practicality in several real-world applications, including as database search, cloud computing, and machine learning [4]. The creation, drafting, and implementation of effective software that safely incorporates HE, however, continues to be a significant practical problem. Creating programs that are useful and meaningful from atomic HE operations requires a high level of expertise. Unfortunately, most programmers are not privy to this information. It is also impractical to expect production programmers to pick up the sophisticated cryptographic expertise needed to implement HE [5]. Different HE schemes have different computing capacities; there are three different kinds of current HE constructions: partial HE, leveled HE, and fully HE. If a cryptosystem has multiplicative or additive homomorphism qualities, but not both, it is said to be PHE. On the other hand, addition and multiplication are permitted in the encrypted domain with LHE. To provide security, random noise is introduced to encrypted data in leveled homomorphic encryption methods, which rely on the learning with errors (LWE) or ring-learning with errors (RLWE) issue [6], [7]. The final version of HE, known as FHE, permits infinite addition and multiplication on encrypted data. Figure 1 provides an explanation of the typical HE structure [8].

In this paper, the focusing is on partial HE, which allow either additive or multiplicative homomorphic operations. Programmers using, HE face significant challenges that must be overcome for correct, efficient, and secure computation. One of the main problems with HE is that designing programs that are useful and meaningful are made from atomic HE processes, which requires a great deal of specialist knowledge. To overcome this obstacle, enhance the implementation efficiency in addition, improve the developer friendliness of HE, this paper present HomoLang. It is a graphical modelling language, provides a graphical environment for developing HE schemas utilizing each of Java programming language, and the Eclipse Modelling Framework (EMF) [9], [10]. These techniques are based on Domain-Specific Modelling Language (DSML) and Model-Driven Development (MDD) approaches [11]. The six contributions that HomoLang makes are as follows: Firstly, automating the PHE encryption and decryption processes by providing graphical building nodes for each concept in HE; it is managing the encoding of input and output data. It enables users to construct one or more PHE schemas in the same workspace. Second, the submitted DSML HomoLang offers an interactive visualization environment with a graphical user interface (GUI) that allows drag-and-drop functionality. It allows the users to select parameters for PHE schema, which is usually done by a HE specialist, and gives them various options for configuring and rearranging the PHE structure. Thirdly, there are no runtime issues, such as incorrect entries, in the HomoLang. Fourthly, the HomoLang language was developed to be free of the grammatical faults that happen in GPPL because of depending DSML approach and graphic concrete syntax. Fifthly, it is reducing the GPPLs' programming complexity by makes scaling and extension possible with little time and effort expended. Finally, it provides six security tests to assess the degree of security of the HE schema. As a result, HomoLang increases the flexibility and usability of HE for programmers.

This paper's remaining sections are organized as follows: Related work on DSMLs is covered in Section 2, while Section 3 presents the design of HomoLang. The language's implantation and semantics are given in Section 4. The evaluation findings are reported in Section 5, and the paper's conclusion and recommendations for more study are provided in Section 6.

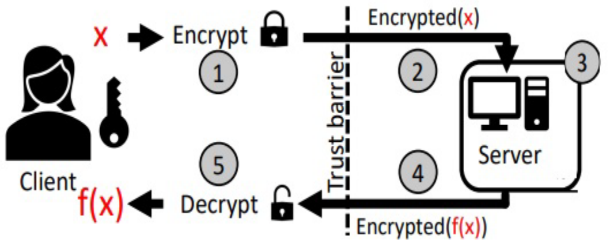


Figure 1. The typical HE structure [7]

2. Related Work

Programmers may build code for a wide range of application domains and in any field by using GPPL. However, every GPPL has unique challenges; some are sensitive to capital, tiny, or space characters [12]. The design of a program is a real challenge; even a little program may need to have several variables, procedures, functions, classes, objects, and so on named. Therefore, before attempting to create the code, find and fix all of the flaws, and put the program into action, a novice programmer must first become familiar with the syntax of that GPPL [13].

DSML approach provides a number of advantages over GPPL when it comes to expressing a certain domain. One benefit is that it offers more abstractions for the target domain, increasing production and raising the standard of the development process [14], [15]. Recently, several research disciplines have given attention to the design and use of DSML [16], [18], [19], [20]. These are some examples of study areas that have developed and used DSML to solve particular problems and other diverse application sectors.

Crockett *et al.* [21] in order to streamline and expedite the application of FHE, this work provide Alchemy, a textural DSML with internal implementation based on the Haskell language. Without requiring the programmer to have any particular understanding of FHE, Alchemy assembles "in-the-clear" calculations on plaintexts into matching homomorphic computations on cipher texts. The suggested language restriction had to be that the compiler generates keys and "key-switching hints," automatically selects the majority of the parameters by statistically inferring cipher text noise rates, and schedules necessary cipher text "maintenance" operations without allowing the user to test his selections.

Qassir *et al.* [22] developed a modelling language called BCLang, which is supported as a graphical modelling design for block cipher types. BCLang's higher abstraction level has the potential to reduce complexity. Software development performance may be enhanced by raising the level of abstraction and automatically producing artifacts. Reducing the possibility of runtime and language mistakes can boost efficiency.

The development of the abstract and concrete syntax, in addition to semantics of BCLang is based on the process of constructing a metamodel that permits the construction of valid block cipher models. Performance analysis, fifty tests of statistical analysis for randomness analysis, and two keystream-generating techniques were offered. Based on the crucial subjective indicators unique to graphical DSML evaluation, the offered language was assessed. They used three common block ciphers algorithms as case studies. The three primary inner structures of the block cipher type were created by the work detailed in this paper, which is helpful in enabling the production of various internal structure cipher types.

Marah *et al.* [23] proposed a DSML known as DSML4TinyOS for developing the tiny operating system that is used by wireless sensor networks. The proposed language is based on model-driven engineering MDE methodology, which defines the language's syntax and model transformations that apply between input and target models. There are two roles of RTE. The first is to provide a forward engineering process for Tiny OS applications. Reverse engineering (RE) technologies in the second part of the RTE enable software models to be retrieved from an existing Tiny OS application. To illustrate the usage of DSML4TinyOS and how it boosts productivity, a case study was covered. This case study shows how to utilize DSML4TinyOS to achieve synchronization between TinyOS models and the related code. The proposed language and the associated IDE may be used by both inexperienced and seasoned Tiny OS application developers.

Giovanni and Gerardo [24] for the purpose of creating both software and hardware cryptography systems, have put out a texturing DSML. It was developed as an internal language using the popular Python GPPL as a foundation. Its purpose is to provide compact support for native constructions and the representation of algebraic structures like fields, rings, and groups. The goal was to provide an easier transition between the phases of designing the cryptographic algorithm and implementing it on the target platform. The Blowfish block cipher technique was used as a case study to demonstrate the flexibility and increased abstraction of the suggested DSL. In comparison to the written C language, it had less code lines when it was resubmitted.

Ladeira *et al.* [3] have presented the RoBMEX technique, which consists of three complimentary DSMLs, in an attempt to minimize the work required in the present open source technologies for the design and implementation of drone behaviors of ROS and MAVLink. They have limits in addition to their degree of modularity and reusability, particularly in terms of the time and effort required for non-experts to acquire the skills required to utilize them for their intended purposes. ROSProML, a DSML created for algorithm experts, can model the process of transforming input signals into the desired outputs using basic operations, function calls, comparisons, and logic blocks. ROSModL, the framework that was submitted, is the foundation for being able to describe the structure of a general ROS system and each of its execution nodes. Using ROSMiLan, the DSML in charge of mission design, the link between execution nodes may be created by providing a non-expert end-user with a more straightforward language that does not require any knowledge of drone technology. A variation of ROSModL that is based on the current system MAVROS was created as an example. Additionally, two instances of ROSProML were created using fictitious jobs, and the majority of their C/C++ code was generated automatically. Python is one of the other targets for the code creation in addition to C/C++.

3. Design of HomoLang DSML

A thorough description of the submitted language's design is given in this section. 1) The abstract syntax defined by the meta-model serves as the foundational structure upon which the graphical representation of building nodes within the HomoLang language is built. Abstract syntax is based on HE fundamental concepts and the rules guiding their encryption and decryption operations. 2) Significant icons that specify the HE fundamentals are used to establish the concrete syntax [14], [15]. Figure 2 shows the front-end compiler design of the graphical DSML. The seventeen classes that comprise the meta-model of the HomoLang are connected via an association connection, directed association relation, or inheritance, as Figure 3 shows. A detailed description of each of these classes can be found below:

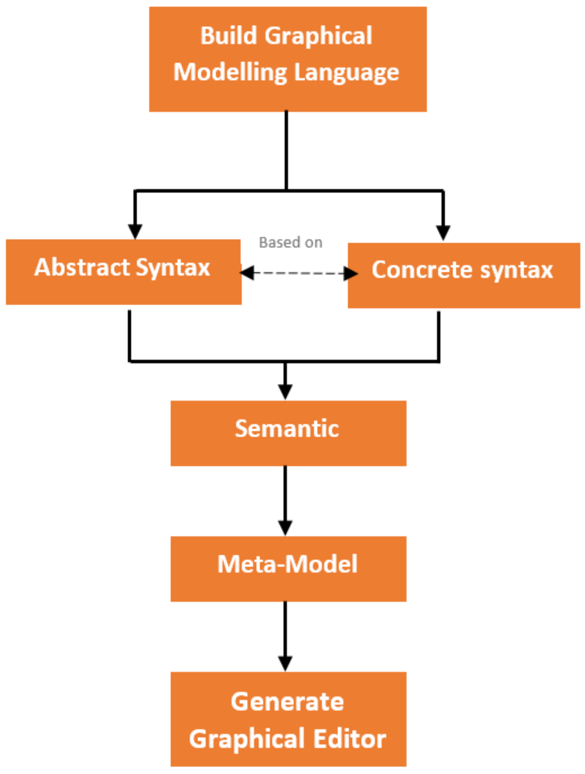


Figure 2. Front-end compiler design of DSML [14]

3.1. The Abstract Syntax

The abstract syntax of a DSML describes the fundamental concepts of the application domain and how they relate to one another without considering meaning. The abstract syntax for MDD Ecore, a graphical DSML implemented on the Eclipse platform, specifies how schemas are constructed in the HE domain and is constructed using a meta-model. As a result, the classes (meta-building nodes) that make up the language meta-model that was supplied are covered in this paragraph. In Figure 3, seventeen classes are explained, and their relationships are resubmitted by the connections between them, representing a single concept of the HE domain principles per class; the GUI of HomoLang is shown in Figure 4. The following describes each of these classes:

- Originaltext Class: The basic operation of the HE process will begin with Originaltext building node; The original input text is accepted by this class, which encodes it into a byte array before converting each byte into a bit array. After encoding in transformation format Unicode character encoding method-8 (UTF-8), the data is changed to an integer number in order to get it ready for HE process.
- EncDecPro Class: The second class is EncDecPro, which performs modular exponentiation operations for encryption and decryption HE processes. It is related to Originaltext, Decodedtext, PublicK, and PrivateK, classes with directed association relations.
- Server Class: it is the third class; this class received the result of the HE encryption process as an integer number and used it for the decryption process.
- Decodedtext Class: it is the fourth class, the final building node in the HE schema that receives the result of decryption as an integer number and converts it to clear text based on UTF-8 decoding. It is related to EncDecPro building node with directed association relation.
- PublicK Class: in this class, the key is computed based on the values of two primes P and Q. It is related to the EncDecPro class and Privatek classes based on directed association relations and to the SelectPQ and SelectP classes based on association relations.
- PrivateK Class: in this class, the key is computed based on the PublicK class. It is related to the EncDecPro class and the PublicK class based on the directed association relationship.
- SelectP Class: in this class, the ability to generate one large prime based on the given length of bits is provided. It is related to PublicK class with directed association relation.
- SelectPQ Class: it is used to generate two large prime numbers based on the given length of bits. It is related to PublicK class with directed association relation.
- Factorization Class: This class of security tests applies mathematical operations to determine whether the modulus number is sufficiently large; to withstand an attempt to factor it in an acceptable amount of time, hence preventing a factorization attack. There should be more than 300 decimals for the n that is calculated by (P * Q). It only has a directed association link with the Server class.
- EncryptionExponent Class: It is another type of security test, which determines whether or not encryption exponents are vulnerable to encryption exponent attacks. It is advised to use a prime that is somewhat near to this number, (65537). It only has a directed association link with the Server class.
- Plaintext Class: the homomorphic encryption and decryption operation carried out on an integer number in the same period. This test does two different kinds of checks. Firstly, it determines if the original text is short or not. To prevent this kind of attack, random bits must be inserted at the beginning and conclusion of the brief messages.

The second verification focuses on cycling: determining if the ciphertext is a variation of the original plaintext. It only has a directed association link with the server class.

- Modulus Class: The produced module n is compared to a common modulus in this security test. This kind of attack must be prevented by not sharing the modulus. Every entity must determine its own modulus. It only has a directed association link with the Server class.
- Timing Class: The submitted language for this security test is based on a fast-exponential method, which checks the relevant bit in the private exponent d to determine whether or not it satisfies the assumption. It only has a directed association link with the server class.
- DecryptionExponent Class: This type of security test determines whether or not there is a possibility of an attack on the decryption exponent. It verifies whether private exponent < $1/3n^{1/4}$. It only has a directed association link with the Server class.
- Mult Class: This class does a homomorphic multiplication operation between two ciphertexts. Based on association relations, it configures two EncDecPro Classes of two homo cipher schemas.
- Add Class: The addition homomorphic operation between two ciphertexts is carried out by this class. Based on association relations, it is configuring two EncDecPro classes of two homo cipher schemas.

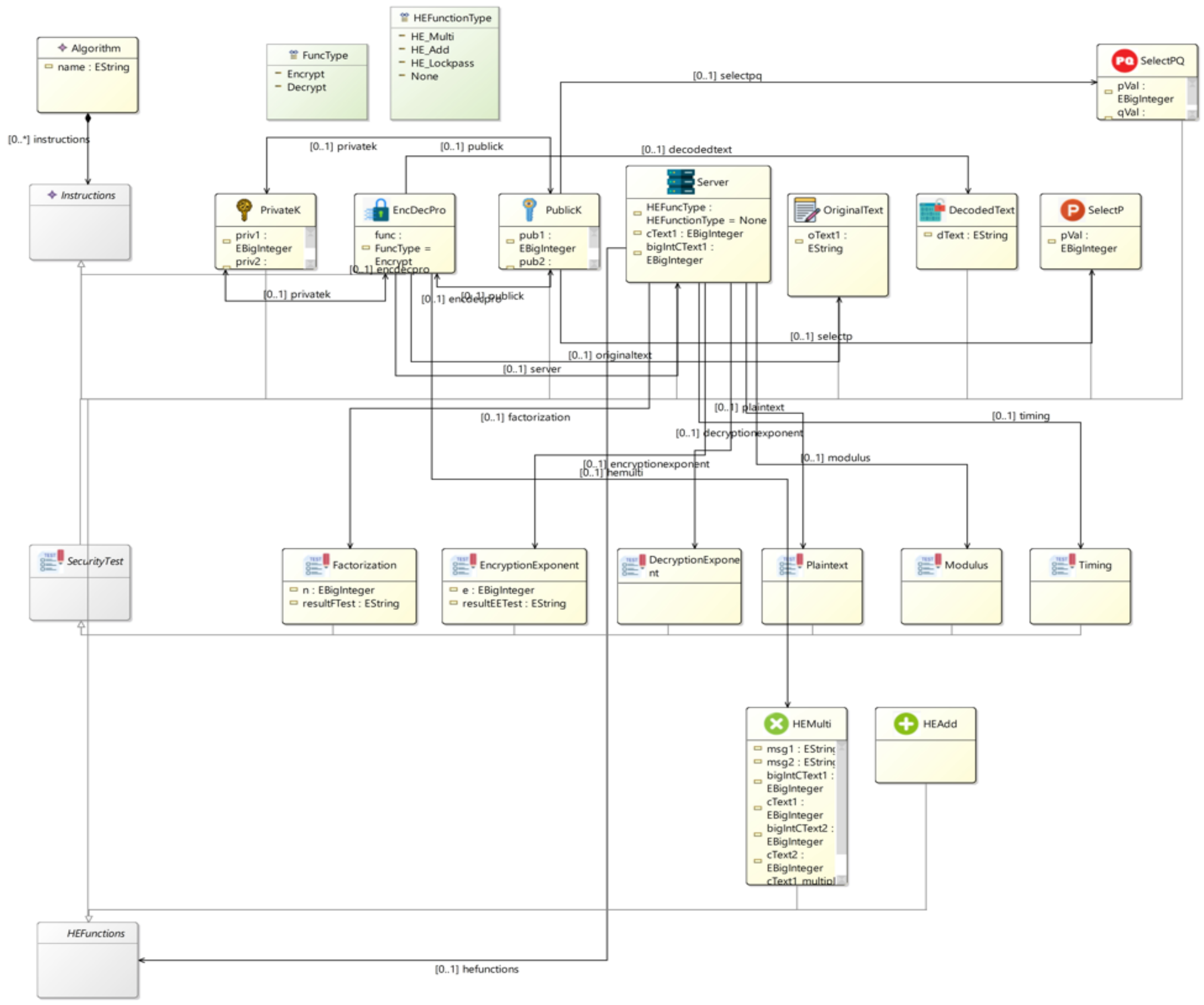
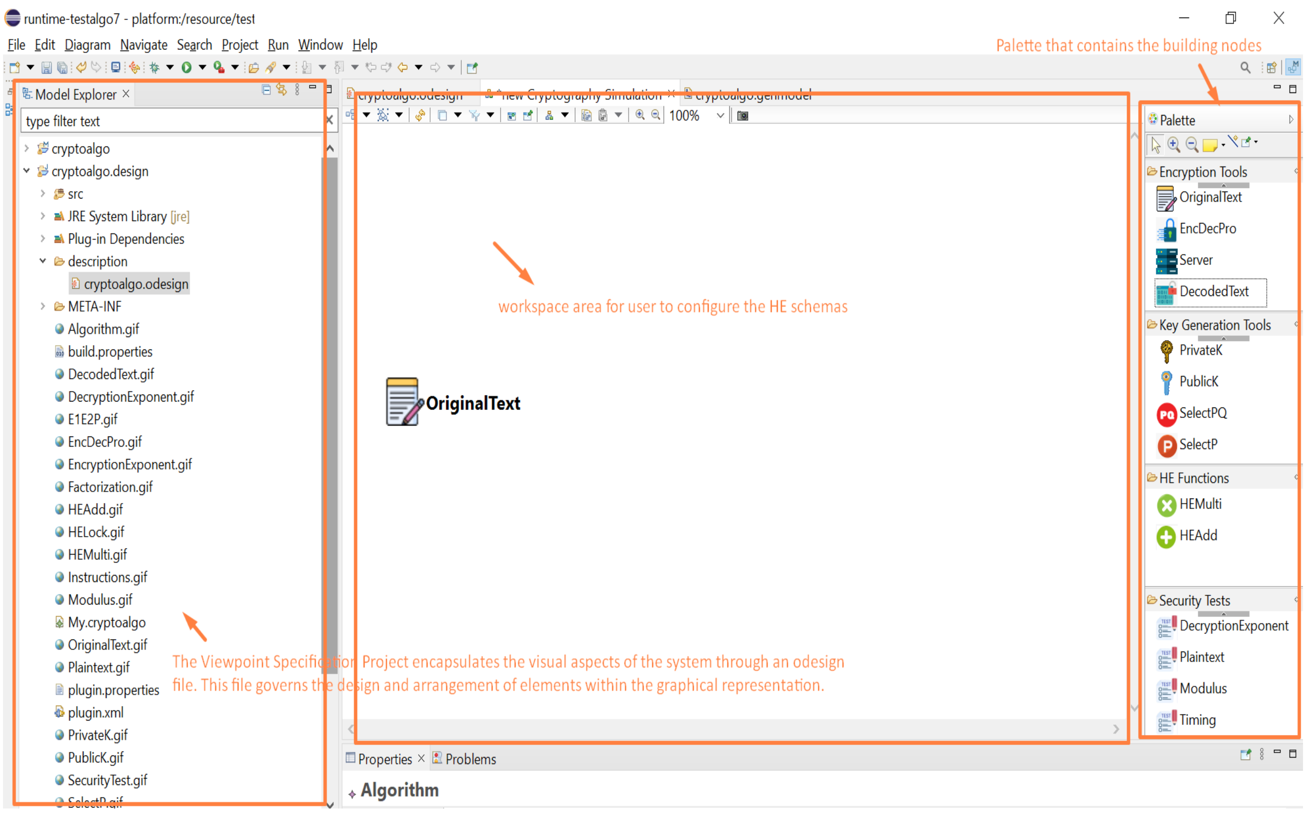


Figure 3. The Meta-model of the submitted DSML HomoLang language



The Viewpoint Specification Project encapsulates the visual aspects of the system through an odesign file. This file governs the design and arrangement of elements within the graphical representation.

Figure 4. The GUI of submitted DSML HomoLang language

3.2. The Concrete Syntax

While abstract syntax specification is established by meta-building nodes, concrete syntax (CS) definition provides a mapping between meta-building nodes and their representations for the HE schema. CS is the set of symbols used in the construction and presentation of the HE schema. CS comes in two primary kinds: graphical and textual [14].

First kind is used in the language that was given. For the meta-building nodes, a set of suitable iconography from the homomorphic domain was chosen. The graphical notations for the abstract syntax meta-building nodes were specified before the geometric characteristics and descriptions of the icons were established using the Eclipse platform. The artifacts achieved are the graphical editor within which cryptographer developers can configure HE schemas. Tables 1, 2, 3, and 4 list the CS used for the submitted HomoLang, respectively.

Table 1. Encryption tools section

Building Node	It Name	Its Input and output
	OriginlText	It receives the input data and is linked to the EncDecPro building node only.
	Server	It receives a link from an EncDecPro building node and is linked to the EncDecPro building node.
	DecodedText	It receives a link from the EncDecPro building node and displays the decoded results.
	EncDecPro	It receives a link from the OriginText building node and is linked to the Server building node in the encryption process; it is also linked to the DecodedText building node for the decryption process.

Table 2. Key Generation Tools

Building Node	It Name	Its Input and output
🔑	PublicK	It receives a link from the SelectPQ building node and is linked to each of the EncDecPro and PrivateK building nodes for the encryption process.
🔑	PrivateK	It receives a link from the publick building node and is linked to the EncDecPro building node for the decryption process.
PQ	SelectPQ	It receives the number of bits for two numbers and is linked to the PublicK building node.
P	SelectP	It receives the number of bits and is linked to the PublicK building node.

Table 3. HE Functions

Building Node	It Name	Its Input and output
✖	HEMulti	It receives a link from the EncDecPro building node of the first HE schema and is linked to the EncDecPro building node of the second HE schema in the workspace.
➕	HEAdd	It receives a link from the EncDecPro building node of the first HE schema and is linked to the EncDecPro building node of the second HE schema in the workspace.

Table 4. Security Tests

Building Node	It Name	Its Input and output
TEST	Plaintext	It receives a link from the Server building node and display the testing results.
TEST	Factorization	It receives a link from the Server building node and display the testing results.
TEST	Decryption Exponent	It receives a link from the Server building node and display the testing results.
TEST	Timing	It receives a link from the Server building node and display the testing results.
TEST	Modulus	It receives a link from the Server building node and display the testing results.
TEST	Encryption Exponent	It receives a link from the Server building node and display the testing results.

4. Semantics and Implantation of HomoLang

The provision of concepts is one of the main issues with language development. Just as crucial as the syntax of concepts is their meaning. HomoLang is built on static semantics based on constraint checks. When a connection is made incorrectly or an unapproved component is attempted to be hooked, the language response either displays an error message or takes the user back to the previous workspace. The semantic is formed by determining how to manage and alert users when there are erroneous or missing links. Together, these requirements comprise a valid meta-model that is implemented with Ecore and a graphical HomoLang DSML that is implemented using the Eclipse platform.

These constraint checks that pertain to the meta-model are explained as follows:

4.1. Condition Check on Class Number

Control the number of building nodes in the HE schema. There should be just one "OriginalText" for every schema of encryption.

4.2. Condition Check on the Number of Class Relationships

There can only be a single connection between the "OriginalText" and "EncDecPro" building nodes since the meta-model has a one-to-one link that controls the interactions between the building nodes.

4.3. Condition Check for Start and Finish Points

Handle the direction of the relationship to define the start and finish building nodes of that HE schema. Linking the "EncDecPro" and "Server" building nodes before forming a connection with the "OriginalText" building node is not practical.

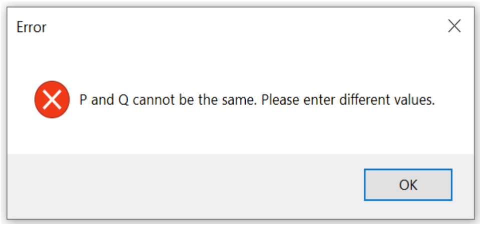
4.4. Condition Check Inheritance Relation

Manage the inheritance links that have been specified. In a HE schema, of course, every property and relationship of a superclass is also present in its subclass.

4.5. Condition Check Relation Direction

Manage the association direction connection as specified in the language that was submitted. Naturally, a class in the meta-model only ever makes use of another class in one direction.

As seen in Figure 5, an incorrect connection or invalid input results in either a step-backward workspace refresh or the display of an error message.

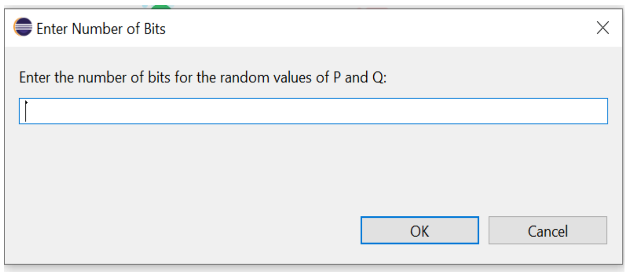

Error

P and Q cannot be the same. Please enter different values.

Figure 5. The response for an invalid input

5. Case Study

This section will provide a case study that demonstrates how the submitted language executes homomorphic features based on the RSA asymmetric cipher method [2]. The RSA algorithm definition uses two exponents: e and d, which stand for the public and private keys, respectively. As seen in Figure 6, the SelectPQ building node generates these two large primes by figuring out how many bits they require.


Enter Number of Bits

Enter the number of bits for the random values of P and Q:

Figure 6. Dialog box of SelectPQ building node

The encryption method computes the equivalent ciphertext (ci), ci = msg^e mod n, provided as input a message msg from the clear text Zn. Without knowing p and q, it is impossible to link this number, ci ∈ Zn, to the original message. The OriginalText, PublicK, and Server building nodes are connected to by the EncDecPro building node using the submitted HomoLang. In order to calculate the set of keys based on the two large primes that may be obtained by utilizing the SelectPQ building node for the encryption process, the PublicK also establishes a connection with the SelectPQ building node. The user enters original content that conveys a clear message. The EncDecPro building node, which uses the public key value to execute fast_exponentiation on a clear message and return the ciphertext that the Server building node had got, is configured by the user to perform encryption operations. The formula for decryption is msg = ci ^{d} mod n, whereby the private key (d, n) and the text of the cipher (ci) are sent into it as inputs. This is the original message in the language that was sent; the EncDecPro building node connects to the Server, PrivateK, and Decodedtext building nodes. This is because d is the inverse of e in Zn. In order to get the original input for the decryption procedure, fast_exponentiation is being applied on the ciphertext and PrivateK.

Without ever having to decode it, the homomorphic characteristic of RSA demonstrates the following Enc (msg1 ∗ msg2) immediately. RSA algorithm supports multiplicative over homomorphic encryption. Assume two messages msg1 and msg2, $ci_1 = msg_1^e \bmod n$, $ci_2 = msg_2^e \bmod n$. The RSA algorithm's HE characteristic is $(ci_1 * ci_2) = (msg_1^e * msg_2^e \bmod n)$. By the submitted language, the HEMulti building node that perform this homomorphic property is connected to EncDecPro building nodes of the two HE schemas; Figure 7 provides an explanation of the PHE encryption and decryption schemas implemented by HomoLang.

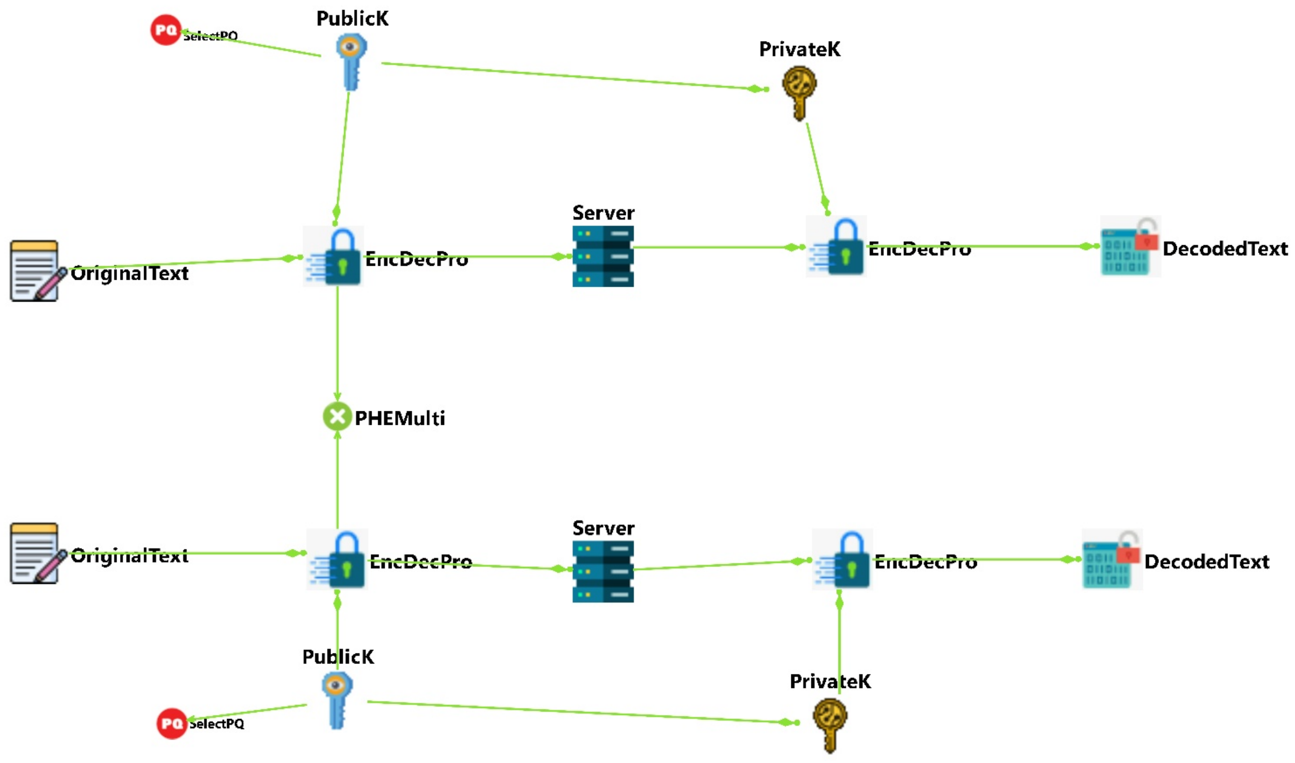


Figure 7. The PHE encryption and decryption schemas implemented by HomoLang

6. Evaluation

Based on objective question metric paradigm (OQMP) [17], a qualitative analysis of subjective criteria; four OQMP was carried out to assess the the front-end compiler design of the submitted HomoLang. The four criteria that were employed were: graphical nature; paradigm assistance and functionality; scalability; and, lastly, usability and ease of understanding. These criteria are delineated in Figures 8, 9, 10, and 11, respectively, and each of these needs is evaluated using sub-metrics of more specific metrics. The graphical nature requirements concentrate on the language's ability to represent concepts and relations visually. This criterion uses the following metrics: clear usage completeness, illustration kind, graphics utilization, efficient use of space, and color effectiveness. Another criterion that assesses the language's capacity to handle the characteristics required for modelling HE schemas is paradigm assistance and functionality. Support for a paradigm, the domain of support, flawless functioning, and naturalness of implementation are among the metrics for this criteria. The language's capacity to manage big, intricate systems is the focus of the scalability requirement. This criterion's metrics include the capacity to describe complicated systems, support for various modelling methods, and support for various abstraction levels. The degree to which software can be used by particular users to achieve quantifiable goals in a quantifiable context of usage with efficacy, efficiency, and satisfaction, as well as the usability and easy-to-understand criteria takes user enjoyment, the beginner friendliness, and the clarity of the building nodes and their relations into account. It evaluates how simple it is to comprehend and utilize HomoLang.

All things considered, the examination of these parameters suggests that HomoLang is a graphical DSML that shows promise for HE schemas. HomoLang supports the addition and multiplication operations characteristics of HE schemas and provides an accurate and consistent graphical representation of concepts and their relationships. All users, regardless of experience level, may easily understand and utilize it, and it solely supports the graphical modelling paradigm.

It can also handle huge and complicated systems since it is scalable and efficient. The experiment was carried out by students in the science college of Mustansiriyah University's fourth cyber security computer science course, who also assessed the usefulness and effectiveness of the provided language. 39 of the 46 were able to reliably and efficiently configure HE schemas. Positive post-study ratings were given by participants to HomoLang, particularly regarding its use as a graphical modelling language for the design and analysis of HE schemas. Additionally, using and learning HomoLang was easy. The submitted language received a score of about 90% on the graphic nature scale since it fully utilized icons to convey HE principles, made suitable use of color, and made the best use of the available workspace. Furthermore, it was discovered that the usability, ease of understanding, and ease of understanding ratios were all the same. A total of about 10% were acquired for each of the capabilities and paradigm aid measures since the OOP paradigm was the only one employed and offered, and because the capabilities evaluation only required the representation of HE principles. The evaluation and rates of these four metrics are displayed in Figures 8–11, respectively:

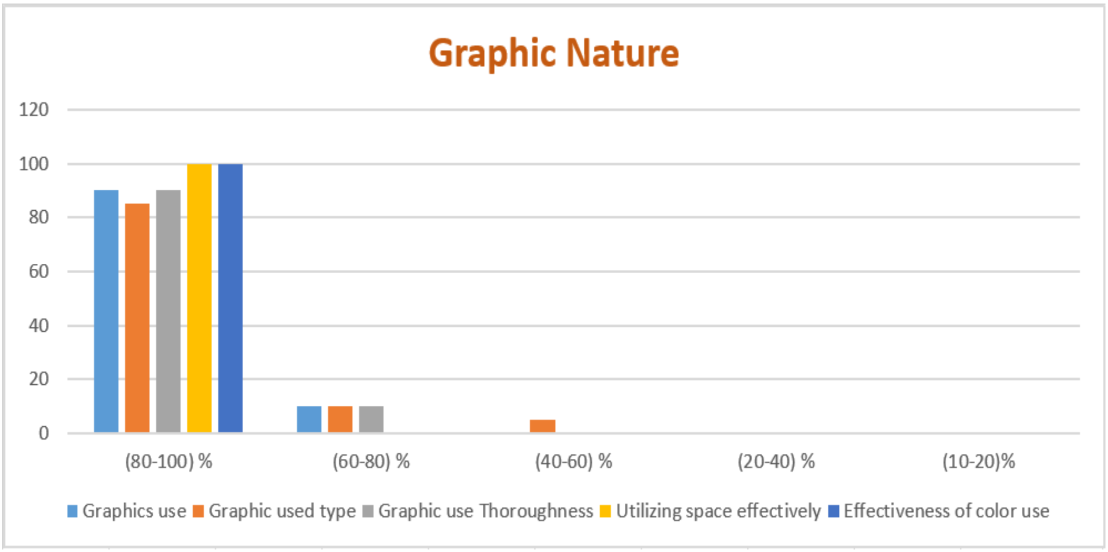


Figure 8. The OQMP for graphical properties

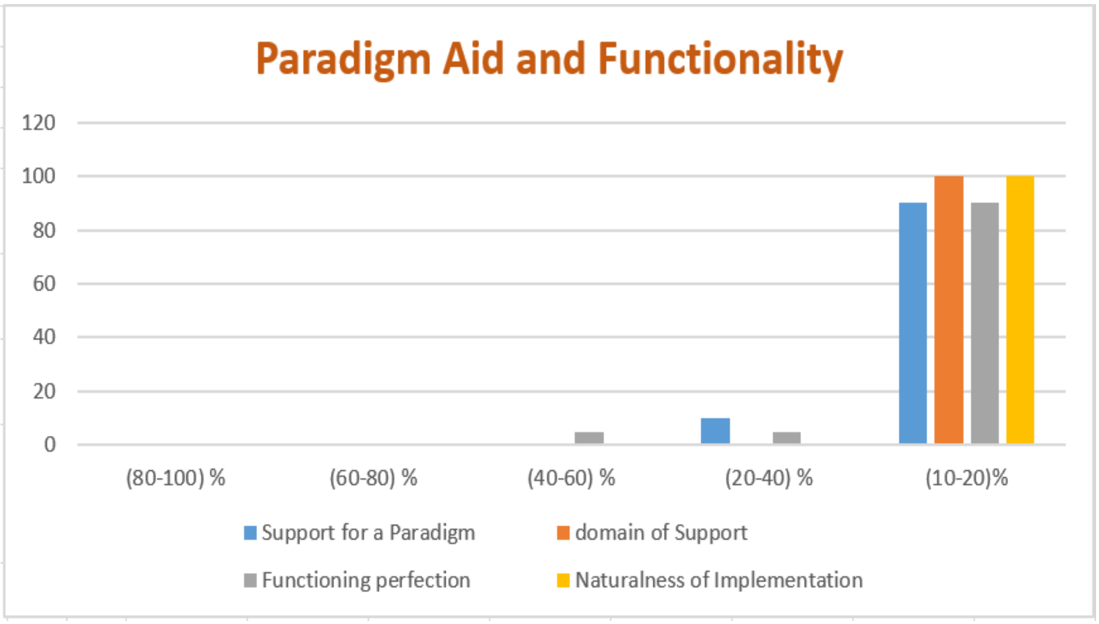


Figure 9. The OQMP for paradigm assistance and functionality

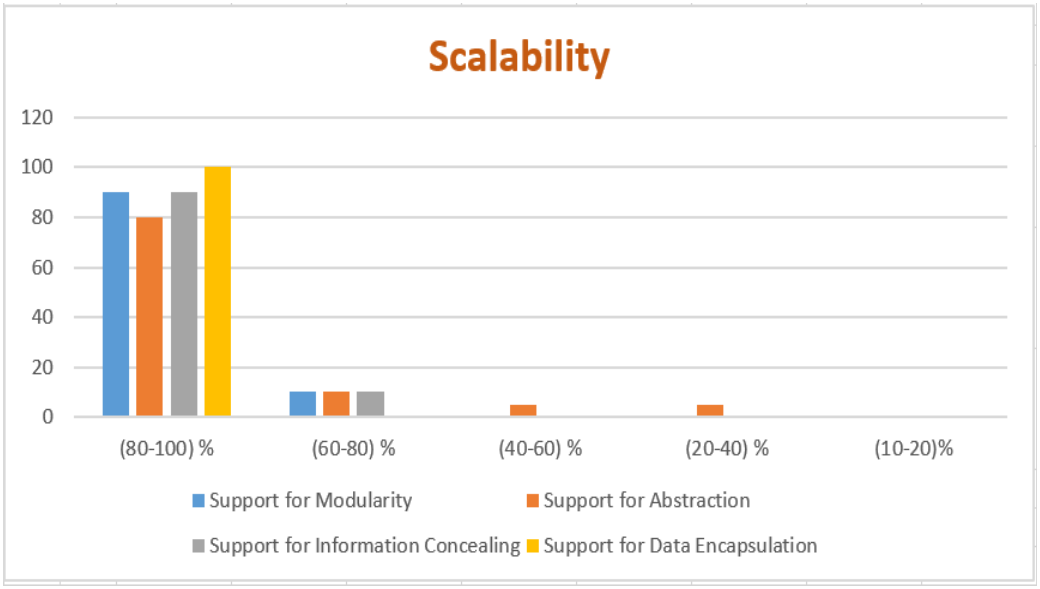


Figure 10. The OQMP for scalability

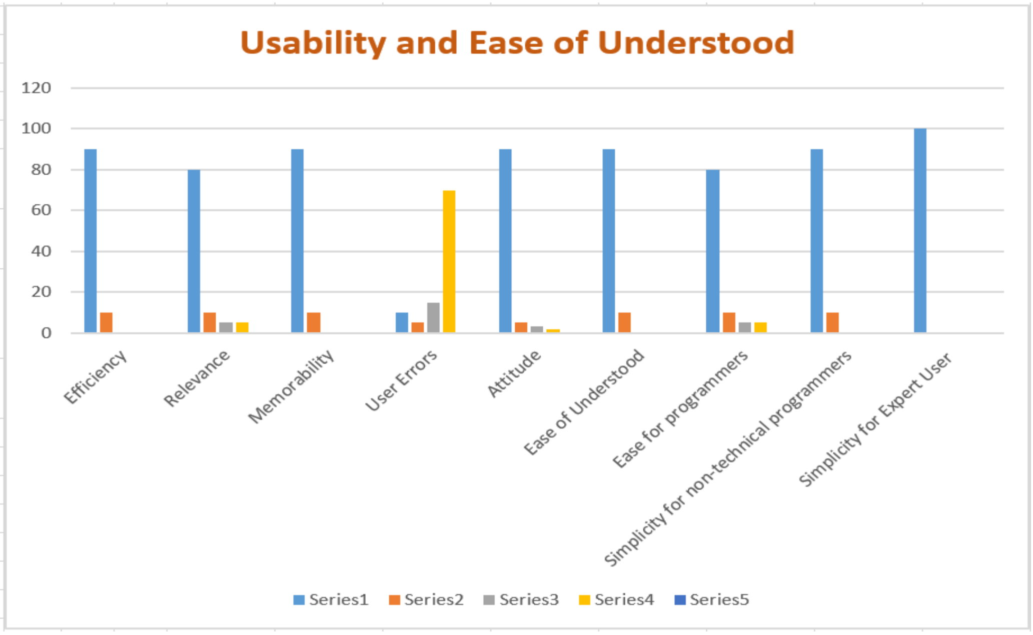


Figure 11. The OQMP for usability and ease of understood

7. Conclusion

HomoLang, the graphical DSML language that was submitted, has many advantages for the creation of HE schemas. HomoLang provides a high degree of abstraction and a flexible and efficient way to do tasks in the HE domain. It is easy for a variety of users to use and comprehend because of its highly expressive GUI and drag-and-drop features. Along with security tests: factorization, encryption exponent, decryption exponent, plaintext, modulus, and timing, all in a graphical manner, HomoLang also includes the essential building blocks of encryption and decryption processes: OriginalText, DecodedText, EncDecPro, and Server.

The key generation building nodes are: Publick, PrivateK, SelectPQ, and selectp. The HE has two main operations: addition and multiplication. This facilitates the modelling process and helps hide the implementation specifics. The submitted language's capability to alter and reconfigure the HE schema as it is being executed is another essential attribute. All things considered, HomoLang offers a productive and adaptable method for representing and evaluating HE schemas, making it an important resource for scholars and industry professionals. HomoLang can be expanded for use in security testing and the construction of nodes for alternative homomorphic types in future research.

Acknowledgements

The author would like to thank Mustansiriyah University, Baghdad-Iraq for its support in the present work.

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]. Geetha, V., Singh, P., Patil, N. S., & Reddy, S. S. S. (2023). *Introduction to Cryptography and Network Security*. AG PUBLISHING HOUSE (AGPH Books).

[3]. Ladeira, M., Ouhammou, Y., & Grolleau, E. (2021). RoBMEX: ROS-based modelling framework for end-users and experts. *Journal of Systems Architecture*, *117*, 102089.

[4]. Alloghani, M., Alani, M. M., Al-Jumeily, D., Baker, T., Mustafina, J., Hussain, A., & Aljaaf, A. J. (2019). A systematic review on the status and progress of homomorphic encryption technologies. *Journal of Information Security and Applications*, *48*, 102362.

[5]. Archer, D. W., Calderón Trilla, J. M., Dagit, J., Malozemoff, A., Polyakov, Y., Rohloff, K., & Ryan, G. (2019, November). Ramparts: A programmer-friendly system for building homomorphic encryption applications. In *Proceedings of the 7th acm workshop on encrypted computing & applied homomorphic cryptography*, 57-68.

[6].Sharma, D. K., Singh, N. C., Noola, D. A., Doss, A. N., & Sivakumar, J. (2022). A review on various cryptographic techniques & algorithms. *Materials Today: Proceedings*, *51*, 104-109.

[7].Alaya, B., Laouamer, L., & Msilini, N. (2020). Homomorphic encryption systems statement: Trends and challenges. *Computer Science Review*, *36*, 100235.

[8].Aslett, L. J., Esperança, P. M., & Holmes, C. C. (2015). A review of homomorphic encryption and software tools for encrypted statistical machine learning. *arXiv preprint arXiv:1508.06574*.

[9].Martínez-Lasaca, F., Díez, P., Guerra, E., & de Lara, J. (2023). Dandelion: A scalable, cloud-based graphical language workbench for industrial low-code development. *Journal of Computer Languages*, *76*, 101217.

[10]. Jäger, S., Maschotta, R., Jungebloud, T., Wichmann, A., & Zimmermann, A. (2016, April). Creation of domain-specific languages for executable system models with the eclipse modeling project. In *2016 Annual IEEE Systems Conference (SysCon)*,1-8. IEEE

[11]. Arslan, S., & Kardas, G. (2020). DSML4DT: A domain-specific modeling language for device tree software. *Computers in Industry*, *115*, 103179.

[12]. Steffen, B., Gossen, F., Naujokat, S., & Margaria, T. (2019). Language-driven engineering: from general-purpose to purpose-specific languages. *Computing and Software Science: State of the Art and Perspectives*, 311-344.

[13]. Kosar, T., Gaberc, S., Carver, J. C., & Mernik, M. (2018). Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. *Empirical Software Engineering*, *23*, 2734-2763.

[14]. 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*, 3113-3134.

[15].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.

[16]. Méndez-Acuña, D., Galindo, J. A., Degueule, T., Combemale, B., & Baudry, B. (2016). Leveraging software product lines engineering in the development of external dsls: A systematic literature review. *Computer Languages, Systems & Structures*, *46*, 206-235

[17]. Kiper, J. D., Howard, E., & Ames, C. (1997). Criteria for evaluation of visual programming languages. *Journal of Visual Languages & Computing*, *8*(2), 175-192.

[18]. 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

[19]. 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.

[20]. 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

[21]. Crockett, E., Peikert, C., & Sharp, C. (2018). Alchemy: A language and compiler for homomorphic encryption made easy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 1020-1037.

[22]. Qassir, S. A., Gaata, M. T., Sadiq, A. T., & Al Alawy, F. (2023). Designing a Graphical Domain-Specific Modeling Language for Efficient Block Cipher Configuration: BCLang. *TEM Journal*, *12*(4), 2038.

[23]. Marah, H., Kardas, G., & Challenger, M. (2021). Model-driven round-trip engineering for TinyOS-based WSN applications. *Journal of Computer Languages*, *65*, 101051.

[24]. Agosta, G., & Pelosi, G. (2007). A Domain Specific Language for Cryptography. In *FDL*, 159-164.