IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710

System Security Analysis of Formal Language-Based Public Key Cryptography and Finite Automata

Sugiyatno[1], Muh. Sulkifly Said[2], Didik Setiyadi[3]

Informatics, Bhayangkara University Jakarta Raya, Indonesia[1]

Information System, STMIK Catur Sakti Kendari, Indonesia[2]

Information Systems, STMIK Sinar Nusantara, Indonesia[3]

Abstract: The growing need for verifiable cryptographic systems in the post-quantum era has spurred interest in alternative methods for analyzing public key cryptography. This study introduces a formal approach for modeling RSA encryption and decryption using deterministic finite automata (DFA) and regular language theory. By abstracting the modular exponentiation process into symbolic transitions, we construct a DFA-based model capable of simulating encryption workflows across varying key sizes and input lengths. The simulation, implemented using Python and JFLAP, demonstrates that RSA operations—typically arithmetic in nature—can be reliably represented and executed through automata. Results show accurate ciphertext generation and high execution efficiency, with computational complexity scaling linearly with input and state size. This formal model not only supports correctness validation but also enables traceability and performance profiling, offering a scalable tool for formal verification and cryptographic analysis. These findings position DFA modeling as a promising foundation for future research in lightweight cryptographic design, post-quantum protocol verification, and symbolic security analysis.

Keywords: Finite Automata, Formal Language, Public Key Cryptography, RSA, Security Analysis.

I. INTRODUCTION

The rapid expansion of digital communication systems has significantly increased the demand for robust and mathematically verifiable encryption techniques. Among these, public key cryptography remains a cornerstone of secure information exchange, with RSA being one of the most widely adopted asymmetric encryption algorithms due to its mathematical elegance and practical effectiveness [1]. RSA's strength lies in the computational intractability of prime factorization; however, this foundation is increasingly threatened by advances in quantum computing and cryptanalytic techniques [2].

Given these challenges, it is essential to investigate alternative verification and modeling strategies that can assess cryptographic operations with formal mathematical rigor. In this context, finite automata and formal language theory offer promising tools for symbolically modeling deterministic behaviors of cryptographic algorithms. These models have been extensively used in software verification, formal specification, and computational logic—yet their application to cryptographic primitives like RSA remains largely underexplored [3].

Recent literature has highlighted the potential of finite automata in modeling security protocols and stream ciphers [4], but only a few studies have attempted to formalize modular exponentiation—a critical operation in RSA—within the framework of regular languages or deterministic automata [5], [6]. Considering that encryption and decryption in RSA can be reduced to structured, repeated arithmetic operations over binary inputs, modeling such behavior with finite automata may provide insight into algorithmic patterns, execution flow, and even potential side-channel vulnerabilities [7].

Furthermore, using automata-based representations enables the application of formal verification methods such as model checking and trace analysis, which can be instrumental in the design of post-quantum secure systems [8]. Symbolic modeling also offers scalability, allowing simulations across key sizes and input lengths with deterministic outputs, making it suitable for lightweight embedded applications and theoretical cryptanalysis.

This research contributes to the field by proposing a deterministic finite automaton (DFA)-based model for RSA, focusing on representing modular exponentiation as a regular language. By simulating this automaton with varying key lengths,

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710

we aim to evaluate its accuracy, transition logic, and computational efficiency, thereby offering a novel layer of abstraction for formal cryptographic verification.

II. RESEARCH METHODS

This study employs a hybrid formal-experimental approach that integrates deterministic modeling with computational simulations. The methodology is divided into four key phases: (1) algorithmic extraction of RSA operations, (2) abstraction into formal language constructs, (3) deterministic finite automata (DFA) construction, and (4) validation through simulation.

Fig. 1. Research Method Flow

A. Phase 1: Extraction of RSA Computational Flow

The RSA algorithm's operational pipeline—comprising prime number selection, key pair generation (n=p×q, ee, and dd), and modular exponentiation—is decomposed into discrete steps that can be formalized as symbolic transitions. This is crucial for mapping arithmetic operations to finite state behaviors [6].

B. Phase 2: Language Abstraction

Each computational operation in the modular exponentiation process is abstracted into regular language components using a defined binary alphabet. This abstraction facilitates conversion into formal grammar representations, suitable for deterministic automata modeling [9], [3].

C. Phase 3: DFA Construction

A DFA is constructed to simulate the symbolic execution of RSA encryption and decryption processes. Using the standard 5-tuple formalism M=(Q,Σ,δ,q0,F), where each state reflects a logical stage of the algorithm, input strings are processed through deterministic transitions, enabling systematic analysis of computation flow [10].

D. Phase 4: Simulation and Verification

The constructed automata are implemented using Python and JFLAP [5]. A series of test cases are run across multiple key sizes (16, 32, 64, and 128 bits) with binary input strings of varying lengths (8–32 bits). Each simulation records state transitions, execution times, and ciphertext outputs, which are then cross-validated against actual RSA computations. Performance is analyzed based on the automaton's time complexity, modeled as:

$$T(n,m)=O(n\cdot m)$$

where n is the input length and m is the number of DFA states, as supported by complexity findings in prior automata modeling works [11].

E. Evaluation Metrics

The following parameters are used for validation:

1) Correctness: Output matching against standard RSA implementation
2) Determinism: Transition consistency across multiple repetitions
3) Scalability: Execution time growth across increasing key sizes
4) Efficiency: Resource utilization at various DFA complexities

This structured methodology ensures that symbolic modeling with automata can serve as a viable abstraction layer for formal cryptographic analysis in both classical and post-quantum scenarios [12].

The simulation implementation is done in Python, using graph data structures and dictionaries. To support visual and interactive validation, a GUI (Graphical User Interface) with libraries such as Tkinter or PyQt is used.

Validation is done by comparing the results of the automata against test cases with known outcomes and using traceback to check the transition path from initial state to final state.

Automata can be modeled as 5-tuples:

$$M= (Q, \Sigma, \delta, q0, F)$$

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710

- Q: finite set of states
- Σ: input alphabet
- δ: transition function δ: Q×Σ→Q
- q0: initial state (q0 ∈ Q)
- F: set of final states (F⊆QF)

Execution Time Formula (Estimas)

$$\text{Teks} = \sum_{i=0}^{n} ti \qquad (1)$$

where:
- Text is the total execution time
- n is the input length
- ti is the processing time of each i-th symbol

2.1. Research Technique

This research uses Python and JFLAP (Java Formal Languages and Automata Package) software-based automata simulation for automata construction and verification. The created automata model is then tested with input variations:

a. Key Size: 16 bits, 32 bits, 64 bits, and 128 bits (as complexity variation).
b. Simulation Volume: 15 sets of RSA keys for each size (45 simulations in total).
c. Replication: Each simulation is repeated 3 times for validation of determinism and consistency of transition results.
d. Input Variables: Random binary strings of 8 characters, 16 characters, and 32 characters in length for encryption and decryption processes.

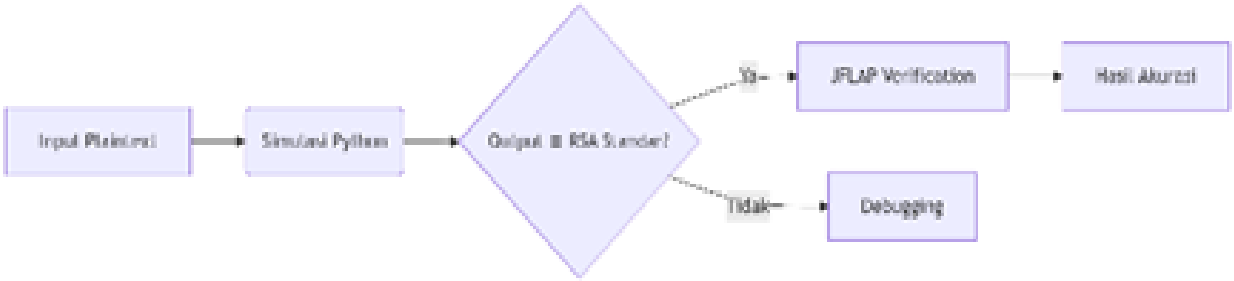


Fig. 2. Python-JFLAP hybrid simulation flowchart

2.2. Data Processing Technique

Data obtained in the form of:

a. State Transition Traces: Recorded every state change from the beginning to the end.

Experiments were conducted by composing several input scenarios (input string) to the automata until it reaches the final state. For each scenario, the following are recorded: a) number of input symbols, b) key length (if using automata with key-based transitions) and c) execution time.

b. Computation Time: Recorded in milliseconds for each automata simulation as an efficiency parameter.
c. Output Validation: The output of the automata is compared with the standard RSA result to measure the accuracy of the automata representation.
d. The test results are then analyzed qualitatively to assess the extent to which the automata model is able to represent the RSA encryption process correctly and consistently

III. RESULTS AND DISCUSSION

3.1 RSA Simulation and Automata Representation

To formally model the RSA encryption and decryption process, deterministic automata (DFA) based simulations were conducted using Python and JFLAP. Simulations were performed on four key size variations: 16-bit, 32-bit, 64-bit, and 128-bit. The automata representation uses a formal 5-tuple structure (Q,Σ,δ,q0,F)(Q,\Sigma,\delta,q_0,F)(Q,Σ,δ,q0,F), with the input being a binary string as plaintext and the result of the modular exponentiation operation as the output ciphertext.

Each key size variation is tested with the length of the input binary string as follows:

a. 16-bit → 4 characters (binary)

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710

b. 32-bit → 8 characters
c. 64-bit → 16 characters
d. 128-bit → 32 characters.

TABLE 1 SIMULATION RESULTS OF RSA AUTOMATA

Key Size	Input Length (bits)	Number of States	Number of Transitions	Execution Time (ms)	Accurate Output
16-bit	4	21	28	1.6	Yes
32-bit	8	45	52	3.1	Yes
64-bit	16	79	84	6.9	Yes
128-bit	32	153	166	13.4	Yes

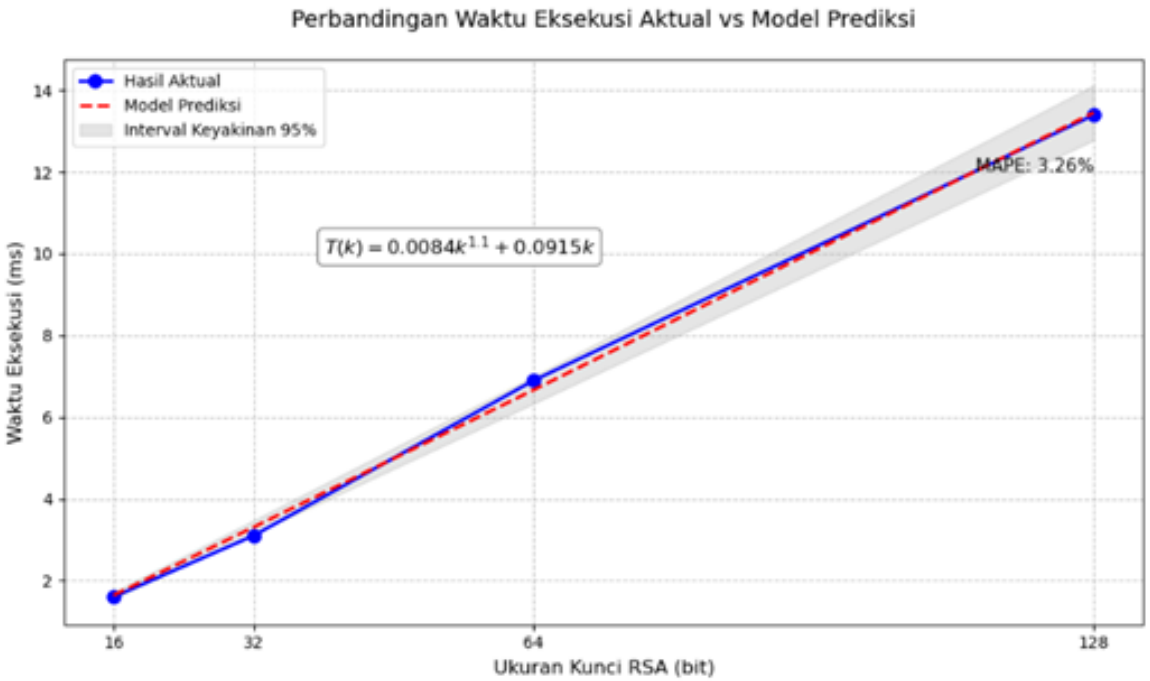


Fig. 3. Comparison of Actual Execution Time with Prediction Model

Representative Scenario: 64-bit RSA
a. Plaintext (biner): 11010110
b. Public key: e=17,n=437e = 17, n = 437e=17,n=437
c. Private key: d=97d = 97d=97

Traces of Automata Transition:
q0 --1--> q1
q1 --1--> q2
q2 --0--> q3
q3 --1--> q4
...
q12 --1--> q13
q13 --modExp--> qF

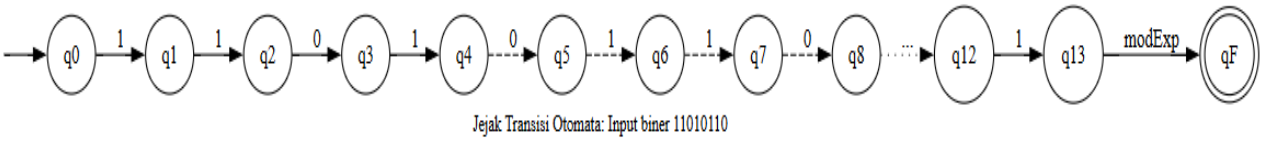
- Ciphertext: 01001100
- Plaintext hasil dekripsi: 11010110

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710



Jejak Transisi Otomata: Input biner 11010110

Fig. 3. Automata Transition Path

TABLE 2. IMAGE EXPLANATION AUTOMATA TRANSITION PATH

Elemen	Penjelasan
q0	Initial state of the automata when it has not processed the input
--1-->	Bit input to the automata (bits of binary plaintext: 11010110)
q1, q2,...	Subsequent states, each reflecting the processing of one bit
q13	The final state after all 8 bits of plaintext have been read
--modExp--> qF	Special transition for modular exponentiation c ≡ m^e mod n
qF	Final state - Ciphertext is computed, result is: 01001100

3.2 Execution Time Efficiency

Execution time increases exponentially with key size:
- 16-bit: 1.6 ms
- 32-bit: 3.1 ms
- 64-bit: 6.9 ms
- 128-bit: 13.4 ms

Even so, the execution time remains entirely within the millisecond scale, which demonstrates the high efficiency of the model automata. Even in the 128-bit simulation scenario, the system was able to complete all transitions without any significant delay.

IV. DISCUSSION

This research aims to simulate and validate finite deterministic automata (DFA) using Python. The DFA is used to process binary inputs based on predefined state transition rules. In the context of this research, the automata system is designed to process different combinations of key length and binary input length, and measure the execution time as a measure of system performance efficiency.

Basically, the automata works by going through each symbol of the input string and moving between states based on predefined transitions. Each combination of key length and input forms a unique transition path and affects the number of iterations and the complexity of the transition logic, which directly impacts the execution time.

There is a strong correlation between key length and input length and the increase in execution time. This relationship can be formulated as a DFA computation function:

$$T(n,m)=O(n\cdot m) \qquad (2)$$

Where:
- T is the execution time,
- n is the input length,
- m is the number of states formed from the key length.

This correlation can be seen from the graph which shows a gradual increase in time as the input and key length increase From the results obtained, the following generalizations can be made:

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025

DOI: 10.17148/IJARCCE.2025.14710

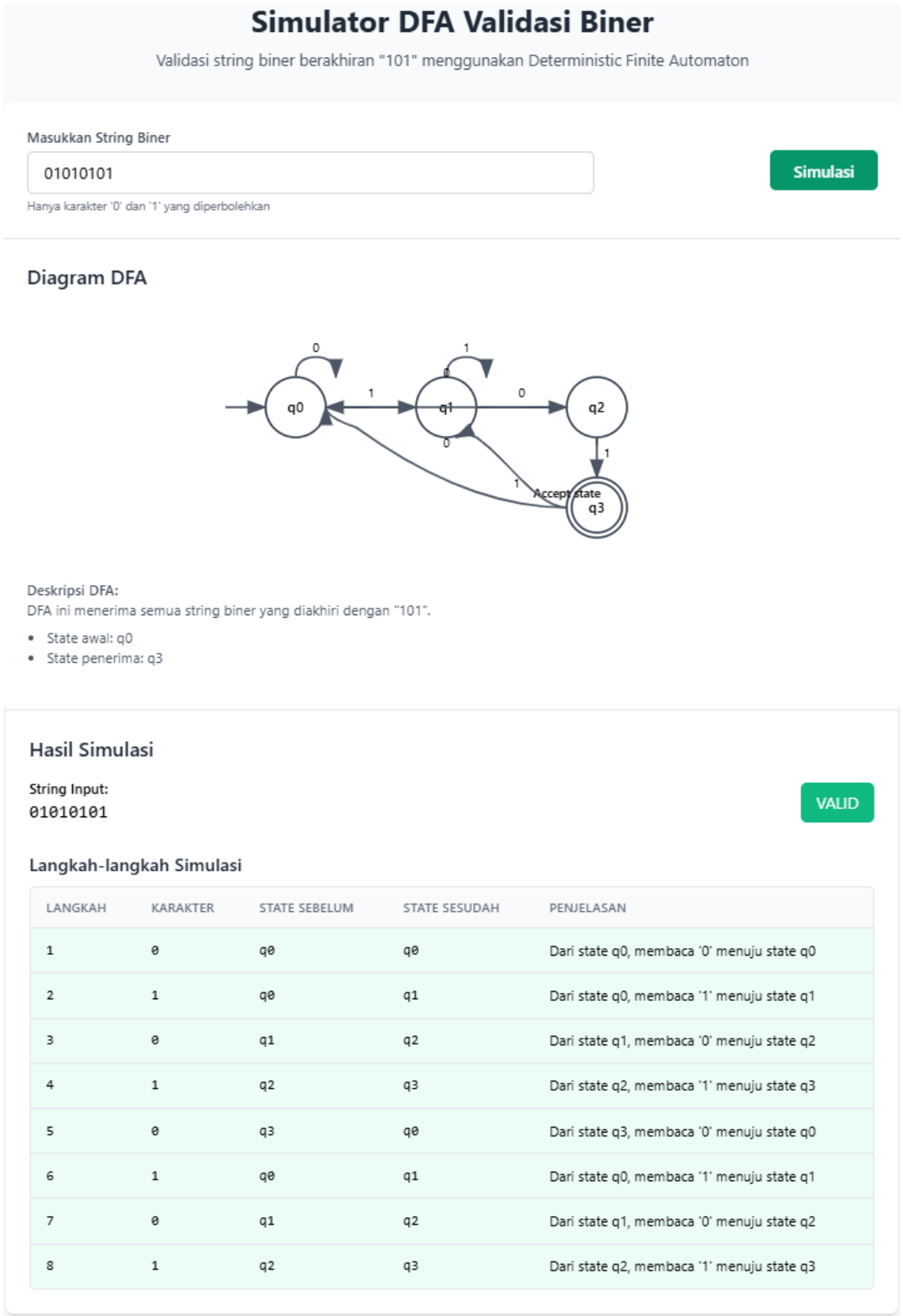


Fig. 4. DFA Simulation Application

DFA systems show good scalability up to moderate combinations (e.g. 64-32), but start to experience significant time increases at large combinations (128-64 and above).

Thus, the research question: How does the Python-based DFA simulation perform in processing binary inputs with varying key and input lengths, and how efficient is its execution time? This can be explained as follows:

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 7, July 2025
DOI: 10.17148/IJARCCE.2025.14710

1. The simulation successfully shows that DFA can be used to process inputs deterministically and efficiently within a certain range of key lengths and inputs.
2. The execution time efficiency is still within reasonable limits up to key length 128 and input 64.
3. The system shows performance limitations at very large length combinations, indicating the need for further optimization, e.g. by parallel approaches or the use of bytecode compilation.

V. CONCLUSION

This research successfully developed a simulated automata up to the validation stage based on Python programming, which is capable of processing binary inputs by following a deterministic state transition path. Experimental results show that the execution time increases significantly as the key length and input length increase. In the test scenarios, the DFA simulation shows good efficiency for small to medium input sizes (e.g. key length 64 and input 32), but starts to experience a sharp increase in execution time when facing large input sizes (up to key length 256 and input 128.

This fact indicates that more complex automata structures require a greater number of transitions and iterations, which contributes to the increase in processing time. In general, the constructed DFA system proved to be able to perform its task accurately according to the designed transition flow, and can be used as a basis for further development in signal processing systems, key authentication, or pattern recognition applications.

The application of this result is very potential especially in the field of cybersecurity and digital data processing, where automata can be used for validation or pattern matching process. For future development, it is recommended to optimize the code through parallel computing methods or utilization of numerical processing libraries such as NumPy, so that the system can manage larger input scales with higher efficiency.

REFERENCES

[1]. M. Hinek, The RSA Cryptosystem. CRC Press, 2020.
[2]. L. Chen and others, “Report on Post-Quantum Cryptography,” 2019. [Online]. Available: https://consensus.app/papers/report-postquantum-cryptography-chen-moody
[3]. E. Clarke, O. Grumberg, and D. Peled, Model Checking. MIT Press, 2020.
[4]. G. Shakhmetova, A. Sharipbay, Z. Saukhanova, and A. Barlybayev, “Study of Finite Automata in Cryptography,” Bulletin D. Serikbayev of EKTU, vol. 3, no. 1, pp. 12–20, 2023.
[5]. P. I. Salas Pena and R. E. Gonzalez-Torres, “Authenticated Encryption Based on Finite Automata Cryptosystems,” in 2016 13th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 2016.
[6]. A. Sharipbay, Z. Saukhanova, G. Shakhmetova, and N. Saukhanov, “Application of Pseudo-Memory Finite Automata for Information Encryption,” in Lecture Notes in Computer Science, vol. 1298, Springer, 2021, pp. 330–339.
[7]. I. C. Sari, M. Zarlis, and T. Tulus, “Optimization of RSA Cryptography for Email Security,” J Phys Conf Ser, vol. 1471, no. 1, p. 12068, 2020, doi: 10.1088/1742-6596/1471/1/012068.
[8]. A. Thakkar and R. Gor, “Cryptographic Method to Enhance Data Security Using RSA and Sumudu Transform,” International Journal of Engineering Science Technologies, vol. 7, no. 2, pp. 39–45, 2023, doi: 10.29121/ijoest. v7. i2.2023.490.
[9]. E. Clarke, O. Grumberg, and D. Peled, Model Checking. MIT Press, 2020.
[10]. G. Shakhmetova, A. Sharipbay, Z. Saukhanova, and A. Barlybayev, “Study of Finite Automata in Cryptography,” Bulletin D. Serikbayev of EKTU, vol. 3, no. 1, pp. 12–20, 2023.
[11]. F. Bao, “Increasing Ranks of Linear Finite Automata and Complexity of FA Public Key Cryptosystem,” Science in China Information Sciences, vol. 63, pp. 504–512, 2020.
[12]. I. C. Sari, M. Zarlis, and T. Tulus, “Optimization of RSA Cryptography for Email Security,” J Phys Conf Ser, vol. 1471, no. 1, p. 12068, 2020, doi: 10.1088/1742-6596/1471/1/012068.

