A PRESENT Lightweight Algorithm High-Level SystemC Modeling using AOP Approach

Hassen Mestiri
Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia | Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse, Tunisia | Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Tunisia
h.mestiri@psau.edu.sa (corresponding author)

Imen Barraj
Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia | Systems Integration & Emerging Energies (SI2E), Electrical Engineering Department, National Engineers School of Sfax, University of Sfax, Tunisia | Higher Institute of Computer Science and Multimedia of Gabes (ISIMG), University of Gabes, Tunisia
i.barraj@psau.edu.sa

Taoufik Saidani
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha 91911, Saudi Arabia
taoufik.Saidan@nbu.edu.sa

Mohsen Machhout
Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Tunisia
mohsen.machhout@fsm.rnu.tn

Received: 17 July 2024 | Revised: 31 July 2024 | Accepted: 11 August 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.8417

ABSTRACT

The increasing complexity of the PRESENT algorithm necessitates a fast modeling and simulation security environment, which is achieved using the SystemC language at the Electronic System Level (ESL), enhancing the speed of cryptographic models. This allows efficient verification of the security properties and performance of the PRESENT algorithm, ensuring robustness against potential attacks. Additionally, the use of SystemC in ESL facilitates easier integration with other hardware components for a more comprehensive security analysis. However, including SystemC in security simulations necessitates modifying the existing code, hence increasing the complexity of the modeling process. Without requiring any code modifications, Aspect Oriented Programming (AOP) can be used for security simulation and cryptographic modeling. This study presents a novel PRESENT SystemC model that incorporates the AOP approach. The model is evaluated in a functional verification environment. The model is constructed using AspectC++ as an AOP language. The simulation results indicate that the effectiveness of the model and the incorporation of the AOP method have negligible effects on the simulation duration or the size of the executable file. The model architecture is based on interlacing all the components.

Keywords-PRESENT block cipher; AOP; SystemC; high-level modeling; cryptography

I. INTRODUCTION

Electronic cryptography devices play a vital role in embedded systems by ensuring the security of confidential information and safeguarding sensitive data. Devices encrypt and decrypt data using sophisticated algorithms and protocols to ensure that only users with permission can access it [1]. The implementation of cryptographic devices in embedded systems serves to fortify security measures against data corruption. However, they are susceptible to physical attacks on hardware infrastructures, which could grant access to malicious actors on confidential data or secret keys [2-4]. Fault injection attacks, which are a type of physical attack, are a very effective technique to acquire these keys and undermine the security of cryptographic equipment [5-7].

Currently, the ability of developers to create and confirm cryptographic embedded systems is insufficient to meet their increasing complexity. Specifically, the need to protect sensitive data in applications, such as IoT, mobile devices, and cloud computing, is driving the increasing demand for secure cryptographic algorithms [8-9]. Software engineers face difficulties in their quest to ensure the accurate implementation and integration of cryptographic algorithms within these systems, all while preserving performance and power constraints. SystemC is a standard language for modeling and verifying complex systems. It can accurately represent and mimic the behavior of cryptographic algorithms, making it a good choice to simulate fault injection in System-on-Chip (SoC) and hardware designs [10-12]. SystemC allows developers to assess the security and resilience of their cryptographic implementations by deliberately introducing faults and studying the reaction of the system. This enables comprehensive testing and validation, eventually resulting in more reliable and secure cryptographic systems. However, most methods require developers to modify the SystemC code to create and detect faults. Aspect Oriented Programming (AOP) is a method that avoids the need to alter the source code of the cryptographic algorithms being tested, enabling efficient separation of distinct issues through clear modularization. AOP does this by segregating cross-cutting problems, such as the introduction and identification of faults, from the fundamental operation of cryptographic methods [13]. This not only streamlines the testing process but also improves the capacity to reuse and maintain code, facilitating developers in analyzing and enhancing the strength and security of their cryptographic solutions. The issue in cryptography lies in the differentiation of the cryptographic algorithm model, fault detection systems, and fault attack procedures. The modules are integrated by interlacing during the compilation process, rather than during coding, to form a robust cryptographic system that is resilient to fault attacks. Through the process of splitting these components, engineers can focus on enhancing each module separately, ensuring that the cryptographic system is both effective and protected [14-16].

This study introduces a novel cryptographic PRESENT SystemC model using the AOP technique. The efficacy of the PRESENT SystemC AOP model, as well as the effects of AOP on simulation time and executable file size, were evaluated using a functional verification environment. This process used AspectC++ [13] for application-level programming and SystemC for hardware design.

II. BACKGROUND

A. *Aspect Oriented Programming (AOP)*

AOP is a programming style that adheres to the concept of dividing various concerns [10, 13]. Under the AOP paradigm, an application is composed of classes and aspects. Non-functional concerns are wrapped as aspects inside modules in the cross-cutting code. The components are included in the operational code to develop a full application.

Aspects are used to execute technical features in the code of an application, comprising two components: Pointcut and Code Advice:

- Pointcut is a technique that allows for the application of transverse functionality code in an aspect by defining the location of one or more Join Points, which represent the points where the transverse functionality code will be inserted. By separating these concerns, developers can make changes to specific aspects of the code more easily, without affecting the overall structure. This can lead to cleaner, more organized code that is easier to understand and maintain over time.
- A Code Advice is a part of the code that is inserted at the Join Points, indicating the weaving of cross-functionality. By utilizing Code Advice, developers can effectively manage and apply cross-cutting concerns, such as logging, security, and error handling, without cluttering the main application logic. This approach improves the scalability and reusability of the codebase, ultimately leading to a more robust and efficient software development process.

An aspect can include several advice codes at the same time, where each advice is associated with a cut. There are three types of Code Advice: before, after, and around. These different types of code advice allow developers to control the execution flow before, after, or around a specific Join Point, providing flexibility in managing cross-cutting concerns. By strategically applying these advices, developers can improve the maintainability and modularity of their codebase, making it easier to adapt to changing requirements or add new features in the future.

To include a new feature, such as an aspect, in the code of an application, the main code has to establish specific areas where the aspect should operate. Figure 1 demonstrates the integration of the aspect code into the application code. By integrating aspects into the codebase, developers can enhance the overall functionality and maintainability of the application. This approach allows for a more efficient way to address cross-cutting concerns without cluttering the main codebase.

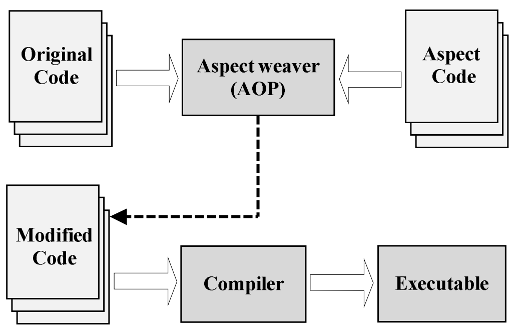

Fig. 1. Weaving aspect code into original code.

The AOP approach is often used for testing embedded applications in C++ and Java. This study presents a novel PRESENT model using the AOP approach to prevent any changes to the source code and the need to analyze the cryptographic system. This solution separates issues that affect more than one area, creating a new way to distribute the cryptographic model with classes and aspects that protect the code for added modules. This segregation enhances the security and maintainability of the application by reducing code complexity and improving code reusability. Additionally, it allows easier debugging and updating of the cryptographic system without affecting the main codebase.

B. PRESENT Algorithm

PRESENT is a lightweight block cipher consisting of 31 cycles that is based on an SP network. Encrypting 64-bit blocks requires an 80- or 128-bit key [17]. An 80-bit key implementation is adequate for tag-based deployments that typically necessitate low-security applications. In the following order, three distinct functions are executed during each round: addRoundKey(), sBoxLayer(), and pLayer(). The addRoundKey() function involves bitwise XOR of the round key with the state. The sBoxLayer() function applies a substitution box to each byte of the state, whereas the pLayer() function permutes the bytes within each column.

III. RELATED WORKS

AOP is progressively emerging as a crucial framework for evaluating system applications [13]. Java programs have been equipped with AspectJ weavers [18]. AspectC++ [10], an AOP weaver based on C++, is increasingly being used for software validation [19]. In [20], a new method was presented to identify faults caused by memory leakage, improper algorithm implementation, or thread interference using C++ aspects. Aspects are automatically generated and integrated into C++ programs to dynamically disclose software defects. In [21], analogous research on embedded Operating System (OS) testing was presented. Manual aspects were employed to test embedded C++ programs within the OS. This study emphasized four key aspects of functional testing, namely, C++ program coverage, memory, performance, and robustness.

In [10], software security hardening based on AOP was investigated. Secure applications have incorporated secure patterns devised in AOP through the utilization of memory code encryption. An analogous method was elaborated in [13]. AOP is a technique for integrating fault tolerance into distributed embedded system-based applications. Many fault-tolerant methods and redundant hardware/software configurations have been examined. AOP is utilized to provide application thread-level defect tolerance in the system. This AOP-based strategy increases modularity, reduces the effort required to modernize legacy systems, and enhances the configuration for testing and product line development.

AOP specialization is also applied to the investigation of SoC designs for hybrid hardware/software systems. AspectC++ [22] is a method to generate an all-encompassing depiction of software and hardware components. Aspect programs incorporate both hardware and software characteristics, which are then linked to the comprehensive description of SoCs. The intended design is a hybrid hardware/software solution based on a combined Field Programmable Gate Array (FPGA). In [23], a SoC investigation was presented. Functional verification focuses primarily on AOP through the use of pure hardware implementations. AOP has been employed in a restricted number of methods to design or model hardware components [13, 24]. By employing explicit architectural concepts, such as concurrency and time, these studies present synthesizable descriptions of SoCs. AOP methods allow the derivation of SoC components through SystemC modeling [10], including communication, cache rules, and performance measures. The generated SystemC models present difficulties in terms of synthesis. Consequently, as shown in [25], these methods are more advantageous in the domain of simulation and verification.

Previous studies were limited because security verification modules were only added at module interconnections, lacked an analysis of where internal security verification took place, and the original SystemC code had to be changed a lot. By incorporating security verification processes into modules and interconnections without modifying functional blocks, the proposed method enables seamless integration that does not cause any disruption to the original code. By addressing potential vulnerabilities at internal security verification locations within modules, this method improves system security without requiring code modifications, simplifying the process.

IV. SYSTEMC PRESENT MODELING

The objective was to create a PRESENT model in the SystemC language using the AOP method. To achieve this objective, the cryptographic processes executed by the PRESENT cryptosystem were partitioned into many modules. Figure 2 illustrates the proposed PRESENT paradigm. The PRESENT SystemC model diagram illustrates their module interconnections using AspectC++ and AOP. The PRESENT SystemC AOP model is composed of 6 parts.

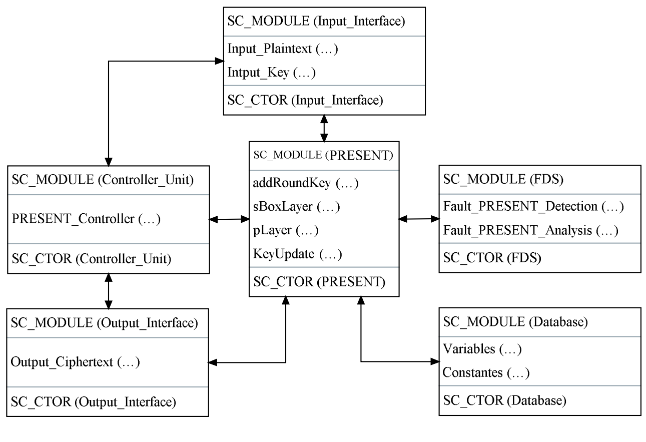

Fig. 2. PRESENT diagram SystemC AOP.

A. PRESENT module

This module carries out the PRESENT encryption and decryption processes. As shown in Algorithm 1, this module performs the following tasks: declares the PRESENT class, declares the PRESENT operations, and finally carries out the encryption and decryption of the input message.

```
Algorithm 1: The PRESENT module
SC_MODULE (PRESENT) {
  SC_CTOR (PRESENT)
  void PRESENT::addRoundKey(…)
  void PRESENT::sBoxLayer(…)
  void PRESENT::pLayer(…)
  void PRESENT::KeyUpdate(…)
}
```

B. *Controller Unit*

This component enables the synchronization of the entire PRESENT cryptographic model. The process performs the following tasks: declares the Controller_Unit class, and declares the PRESENT_Controller(…) function, which includes a finite-state machine to ensure synchronization between all modules.

```
Algorithm 2: The Controller Unit
SC_MODULE(Controller_Unit) {
  SC_CTOR(Controller_Unit)
  VoidController_Unit::PRESENT_Controller(…)
}
```

C. *Input Interface*

This component rearranges the data stream following the specifications laid out by the communication protocol. As shown in Algorithm 3, the Input_Interface module declares two functions where the process performs the following tasks: declares the functions Input_Plaintext(...) and Input_Key(...), and divides the input messages into the required size (64 bits for plaintext - 80 or 128 bits for the key).

```
Algorithm 3: The Input Interface
SC_MODULE(Input_Interface) {
  SC_CTOR(Input_Interface)
  voidInput_Interface::Input_Plaintext(…)
  void Input_Interface::Intput_Key(…)
}
```

D. *Output Interface*

This module restores the encrypted stream to the communications protocol's format and declares the function Output_Ciphertext (…).

```
Algorithm 4: The Output Interface
SC_MODULE(Output_Interface) {
  SC_CTOR(Output_Interface)
  VoidOutput_Interface::Output_Ciphertext(…)
}
```

E. *Fault Detection Scheme (FDS)*

The purpose of developing the FDS module is to safeguard the PRESENT module from fault attacks. As shown in Algorithm 5, the FDS module declares two functions: Fault_PRESENT_Detection (...) and Fault__PRESENT_ Analysis (...). The FDS module performs the following tasks: detects all the faults during the authenticated encryption process and analyzes the fault detection results.

```
Algorithm 5: The Fault Detection Scheme (FDS)
SC_MODULE(FDS) {
  SC_CTOR(FDS)
  void FDS::Fault_PRESENT_Detection(…)
  void FDS::Fault_PRESENT_Analysis(…)
}
```

F. *Database*

This module holds all the variables and constants that are used in the process of authenticated encryption. The code includes the definition of two functions: Variables(…) and Constants (…).

```
Algorithm 6: The Database
SC_MODULE(Database) {
  SC_CTOR(Database)
  void Database::Variables (…)
  void Database::Constantes(…)
}
```

V. FUNCTIONAL VERIFICATION ENVIRONMENT

The functional verification environment proposed for the PRESENT model utilizes the transaction-based verification environment in SystemC. The system incorporates a reference model that offers a comprehensive design description and runs concurrently with the PRESENT model to evaluate the simulation output results. Instead of simulating signals, the reference model handles module interactions at the transactional level. Figure 3 shows the testbench SCV, a module that produces the encryption keys and texts for the reference and proposed models at random. The testbench SCV sends signals to the transactor, which then applies those signals to the model inputs. A comparator module compares the outputs after each transaction and generates the results.

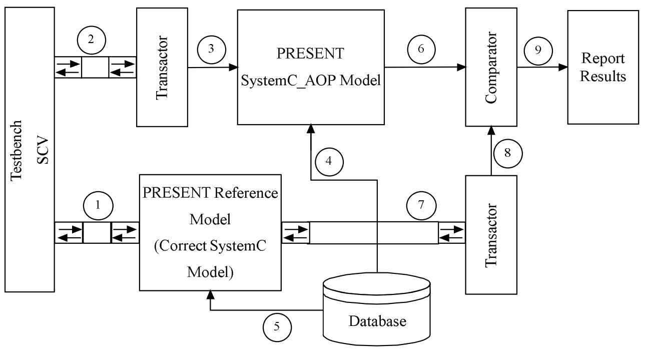

Fig. 3. PRESENT functional verification environment.

- (1)+(2): The testbench SCV generates stimuli randomly for the two cryptographic models, namely the proposed and the reference models.
- (3): The transactor module converts transactions into signals, which are then adapted to the inputs of the proposed non-TLM model.
- (4)+(5): Both models use the databases.
- (6)+(8): The comparator module's input consists of PRESENT SystemC-based AOP and transactor outputs.
- (7): The reference model's TLM output.
- (8): The transactor converts transactions into signals that can be adjusted to match the outputs of the non-TLM model.
- (9): The results of the comparison report are produced.

VI. RESULTS AND DISCUSSION

This section focuses on evaluating the proposed SystemC-AOP paradigm and analyzing the effects of employing SystemC and AspectC++ in ESL on the cryptography architecture. The simulation time and the size of the executable file of the cryptographic model were evaluated to determine the impact of the AOP approach on both of these factors. The modeling process was performed using AspectC++ 2.3 and SystemC 2.3.4, and the PRESENT SystemC AOP model was validated using an Intel Core I5-3470 3.2 GHz CPU, 8 GB RAM, and the gcc 10.3 compiler.

A. *Fault Analysis AOP Impact*

Fault attacks were executed using the proposed environment to test the detection capabilities of the PRESENT SystemC-AOP environment. Fault detection capabilities were assessed using two scenarios: pure SystemC and SystemC-AOP. In addition, a fault detection scheme was used for this evaluation [6]. To do this, the AspectC++ and SystemC simulation kernels were used to assess and compare the results of the PRESENT SystemC-AOP environment.

The simulation results, shown in Table I, demonstrate that both the SystemC-AOP and the SystemC PRESENT models had comparable fault detection capabilities in terms of identified faults, thus affirming the effectiveness of the proposed SystemC-AOP technique. The findings indicate that incorporating AOP into the SystemC framework does not undermine the detection capabilities of the design models.

TABLE I. DETECTION CAPABILITY: SYSTEMC-AOP/SYSTEMC

PRESENT block cipher	**Random fault detection capability**	
	SystemC	**SystemC_AOP**
PRESENT block cipher model protected by FDS [6] [a]	99.997%	99.997%
PRESENT block cipher model protected by FDS [6] [b]	99.999%	99.999%

a. PRESENT standard architecture, b. PRESENT architecture presented in [17]

B. *Impact of AOP on Simulation*

To ascertain the impact that AOP has on the duration of the simulation, some simulations were performed with the developed environment. Two defect detection schemes were utilized to determine the kernel time (kTime) and user time (uTime) during this procedure. It should be noted that kTime and uTime are contingent upon both the amount of Join Points and the amount of modules to be inserted using the AOP technique. Table II displays the results of the simulations used to determine kTime and uTime in two different scenarios: SystemC and SystemC_AOP.

TABLE II. SYSTEMC/AOP SIMULATION TIME

PRESENT block cipher	**kTime (s)**		**uTime (s)**	
	SystemC	**SystemC_AOP**	**SystemC**	**SystemC_AOP**
PRESENT protected by FDS [6] [a]	0.026	0.025	1.432	1.433
PRESENT protected by FDS [6] [b]	0.023	0.024	1.123	1.126

a. PRESENT standard architecture, b. PRESENT architecture presented in [17].

The margin of error associated with the measurement procedure employed (Linux commands) is undeniably negligible, and, as such, does not affect the accuracy of the simulation results.

The results indicate that the utilization of the AOP approach to integrate the Input_Interface, Database, Controller_Unit, FDS, and Output_Interface modules into the SystemC model does not result in any discernible effect on simulation execution times. This suggests that the additional complexity introduced by the AOP approach does not significantly affect simulation performance. Therefore, it can be concluded that the integration of AOP in SystemC does not have a noticeable effect on kTime and uTime in this particular study. An additional pivotal aspect to consider when evaluating the impact of AOP is the size of the resulting executable file. Table III shows the executable file sizes produced by AspectC++ and the SystemC kernel in the two scenario situations.

TABLE III. EXECUTABLE FILE SIZE IMPACT: SYSTEMC/AOP

PRESENT block cipher	**SystemC**	**SystemC_AOP**
PRESENT block cipher model protected by FDS [6] [a]	0.465 MB	0.463 MB
PRESENT block cipher model protected by FDS [6] [b]	0.468 MB	0.467 MB

a. PRESENT standard architecture, b. PRESENT architecture presented in [17].

The simulation results demonstrate that despite the utilization of two programming languages (SystemC and AspectC++) to model the proposed model and employing the AOP technique to weave together all modules, the executable file sizes remain almost unaltered. This indicates that adjusting the AOP does not substantially affect the file size of the executable. Additionally, the results suggest that the AOP technique effectively manages cross-cutting concerns without significantly affecting the overall performance of the system. This highlights the potential benefits of using AOP in software development for modularization and improved code maintenance. Furthermore, the results show that AOP can enhance code reusability and readability by separating concerns more efficiently. In general, the findings support the integration of AOP techniques in software development to streamline the design process and enhance overall system performance.

VII. CONCLUSION

This study presents a SystemC model for the Electronic System Level that is based on Aspect-Oriented Programming (AOP). A functional verification environment is recommended to validate model functionality, simulation time, and executable file sizes. The verification environment comprises testbenches, stimuli generators, and coverage monitors to ensure the precision and comprehensiveness of the model. Furthermore, the verification environment facilitates the detection of possible design defects and assists in troubleshooting the model. In addition, it offers a comprehensive framework for assessing the performance and dependability of the PRESENT SystemC model in various circumstances. The simulation results indicate that the efficiency of the AspectC++ model and the weaving of the PRESENT module have minimal impact on both the simulation time and the size of the executable file. In addition, the proposed PRESENT SystemC AOP model exhibits enhanced modularity and maintainability compared to conventional methods. Moreover, the proposed techniques for injecting and detecting faults have been proven to be viable and efficient in assessing the resilience of cryptographic schemes against fault attacks. The simulation time is minimally affected by the AOP, which proves its usefulness in evaluating security domains by minimizing both efforts and errors. Furthermore, the AOP approach facilitates convenient customization and modification of the fault injection/detection mechanism to adapt to unique cryptographic systems. The flexibility of this method allows it to be used in various security circumstances, thus increasing its effectiveness in assessing the strength of cryptographic schemes. In summary, the use of AOP in SystemC modeling has the potential to improve design adaptability and productivity in electronic system-level development.

ACKNOWLEDGMENT

This study is supported via funding from Prince Sattam bin Abdulaziz University project number (PSAU/2024/R/1445).

REFERENCES

[1] H. Mestiri and I. Barraj, "High-Speed Hardware Architecture Based on Error Detection for KECCAK," *Micromachines*, vol. 14, no. 6, Jun. 2023, Art. no. 1129, https://doi.org/10.3390/mi14061129.

[2] X. Yang, L. Shu, Y. Liu, G. P. Hancke, M. A. Ferrag, and K. Huang, "Physical Security and Safety of IoT Equipment: A Survey of Recent Advances and Opportunities," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4319–4330, Jul. 2022, https://doi.org/10.1109/TII.2022.3141408.

[3] H. Mestiri, I. Barraj, A. Alsir Mohamed, and M. Machhout, "An Efficient AES 32-Bit Architecture Resistant to Fault Attacks," *Computers, Materials & Continua*, vol. 70, no. 2, pp. 3667–3683, 2022, https://doi.org/10.32604/cmc.2022.020716.

[4] F. Thabit, O. Can, A. O. Aljahdali, G. H. Al-Gaphari, and H. A. Alkhzaimi, "Cryptography Algorithms for Enhancing IoT Security," *Internet of Things*, vol. 22, Jul. 2023, Art. no. 100759, https://doi.org/10.1016/j.iot.2023.100759.

[5] I. Salam, T. H. Ooi, L. Xue, W. C. Yau, J. Pieprzyk, and R. C. W. Phan, "Random Differential Fault Attacks on the Lightweight Authenticated Encryption Stream Cipher Grain-128AEAD," *IEEE Access*, vol. 9, pp. 72568–72586, 2021, https://doi.org/10.1109/ACCESS.2021.3078845.

[6] T. De Cnudde and S. Nikova, "Securing the PRESENT Block Cipher Against Combined Side-Channel Analysis and Fault Attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3291–3301, Sep. 2017, https://doi.org/10.1109/TVLSI.2017.2713483.

[7] H. Mestiri, N. Benhadjyoussef, and M. Machhout, "Fault Attacks Resistant AES Hardware Implementation," in *2019 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS)*, Gammarth, Tunisia, Apr. 2019, pp. 1–6, https://doi.org/10.1109/DTSS.2019.8914979.

[8] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," *IEEE Access*, vol. 9, pp. 28177–28193, 2021, https://doi.org/10.1109/ACCESS.2021.3052867.

[9] T. K. Goyal, V. Sahula, and D. Kumawat, "Energy Efficient Lightweight Cryptography Algorithms for IoT Devices," *IETE Journal of Research*, vol. 68, no. 3, pp. 1722–1735, May 2022, https://doi.org/10.1080/03772063.2019.1670103.

[10] H. Mestiri, I. Barraj, and M. Machhout, "An AOP-Based Security Verification Environment for KECCAK Hash Algorithm," *Computers, Materials & Continua*, vol. 73, no. 2, pp. 4051–4066, 2022, https://doi.org/10.32604/cmc.2022.029794.

[11] X. Zheng, J. Wu, X. Lin, H. Gao, S. Cai, and X. Xiong, "Hardware/Software Co-Design of Cryptographic SoC Based on RISC-V Virtual Prototype," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 9, pp. 3624–3628, Sep. 2023, https://doi.org/10.1109/TCSII.2023.3267186.

[12] N. Veeranna and B. C. Schafer, "S3CBench: Synthesizable Security SystemC Benchmarks for High-Level Synthesis," *Journal of Hardware and Systems Security*, vol. 1, no. 2, pp. 103–113, Jun. 2017, https://doi.org/10.1007/s41635-017-0014-1.

[13] H. Mestiri, I. Barraj, M. Bedoui, and M. Machhout, "An ASCON AOP-SystemC Environment for Security Fault Analysis," *Symmetry*, vol. 16, no. 3, Mar. 2024, Art. no. 348, https://doi.org/10.3390/sym16030348.

[14] A. Baksi, S. Bhasin, J. Breier, D. Jap, and D. Saha, "A Survey on Fault Attacks on Symmetric Key Cryptosystems," *ACM Computing Surveys*, vol. 55, no. 4, Aug. 2022, Art. no. 86, https://doi.org/10.1145/3530054.

[15] A. Chattopadhyay and U. Mitra, "Security Against False Data-Injection Attack in Cyber-Physical Systems," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 2, pp. 1015–1027, Jun. 2020, https://doi.org/10.1109/TCNS.2019.2927594.

[16] M. M. N. Aboelwafa, K. G. Seddik, M. H. Eldefrawy, Y. Gadallah, and M. Gidlund, "A Machine-Learning-Based Technique for False Data Injection Attacks Detection in Industrial IoT," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8462–8471, Sep. 2020, https://doi.org/10.1109/JIOT.2020.2991693.

[17] R. Chatterjee and R. Chakraborty, "A Modified Lightweight PRESENT Cipher For IoT Security," in *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, Gunupur, India, Mar. 2020, pp. 1–6, https://doi.org/10.1109/ICCSEA49143.2020.9132950.

[18] S. Mohite, A. Sarda, and S. D. Joshi, "Analysis of System Requirements by Aspects-J Methodology," in *2021 International Conference on Computing, Communication and Green Engineering (CCGE)*, Pune, India, Sep. 2021, pp. 1–6, https://doi.org/10.1109/CCGE50943.2021.9776384.

[19] M. Ramalingam, D. Saranya, R. ShankarRam, P. Chinnasamy, K. Ramprathap, and A. Kalaiarasi, "An Automated Framework For Dynamic Web Information Retrieval Using Deep Learning," in *2022 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, Jan. 2022, pp. 1–6, https://doi.org/10.1109/ICCCI54379.2022.9741044.

[20] R. Jain, R. Agrawal, R. Gupta, R. K. Jain, N. Kapil, and A. Saxena, "Detection of Memory Leaks in C/C++," in *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*, Bhopal, India, Feb. 2020, pp. 1–6, https://doi.org/10.1109/SCEECS48394.2020.32.

[21] E. Yoshiya, T. Nakanishi, and T. Isshiki, "RTL Design Framework for Embedded Processor by using C++ Description," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, Feb. 2021, pp. 1208–1211, https://doi.org/10.23919/DATE51398.2021.9473942.

[22] H. Mestiri, I. Barraj, and M. Machhout, "AES High-Level SystemC Modeling using Aspect Oriented Programming Approach," *Engineering, Technology & Applied Science Research*, vol. 11, no. 1, pp. 6719–6723, Feb. 2021, https://doi.org/10.48084/etasr.3971.

[23] G. Biagetti, L. Falaschetti, P. Crippa, M. Alessandrini, and C. Turchetti, "Open-Source HW/SW Co-Simulation Using QEMU and GHDL for VHDL-Based SoC Design," *Electronics*, vol. 12, no. 18, Jan. 2023, Art. no. 3986, https://doi.org/10.3390/electronics12183986.

[24] P. Pieper, V. Herdt, and R. Drechsler, "Advanced Embedded System Modeling and Simulation in an Open Source RISC-V Virtual Prototype," *Journal of Low Power Electronics and Applications*, vol. 12, no. 4, Dec. 2022, Art. no. 52, https://doi.org/10.3390/jlpea12040052.

[25] K. Bjerge, J. H. Schougaard, and D. E. Larsen, "A scalable and efficient convolutional neural network accelerator using HLS for a system-on-chip design," *Microprocessors and Microsystems*, vol. 87, Nov. 2021, Art. no. 104363, https://doi.org/10.1016/j.micpro.2021.104363.