symmetry

Article

An ASCON AOP-SystemC Environment for Security Fault Analysis

Hassen Mestiri [1,2,3,*], Imen Barraj [1,4,5], Mouna Bedoui [3] and Mohsen Machhout [3]

1. Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
2. Higher Institute of Applied Sciences and Technology of Sousse, University of Sousse, Sousse 4002, Tunisia
3. Electronics and Micro-Electronics Laboratory, Faculty of Sciences of Monastir, University of Monastir, Monastir 5000, Tunisia
4. Systems Integration & Emerging Energies (SI2E), Electrical Engineering Department, National Engineers School of Sfax, University of Sfax, Sfax 3029, Tunisia
5. Higher Institute of Computer Science and Multimedia of Gabes (ISIMG), University of Gabes, Gabes 6029, Tunisia

* Correspondence: h.mestiri@psau.edu.sa

Abstract: Cryptographic devices' complexity necessitates fast security simulation environments against fault attacks. SystemC, a promising candidate in Electronic System Levels (ESLs), can achieve higher simulation speeds while maintaining accuracy and reliability, and its modular and hierarchical design allows for efficient modeling of complex cryptographic algorithms and protocols. However, code modification is required for fault injection and detection. Aspect-Oriented Programming (AOP) can test cryptographic models' robustness without modifications, potentially replacing real cryptanalysis schemes and reducing the time and effort required for fault injection and detection. Through the utilization of a fault injection/detection environment, this paper presents a novel approach to simulating the security fault attacks of ASCON cryptographic systems at the ESL. The purpose of this methodology is to evaluate the resistance of ASCON SystemC models against fault attacks. The proposed methodology leverages the advantages of AOP to enhance the fault injection and detection process. By applying AOP techniques, we inject faults into the SystemC models without making any changes to the main codebase. This approach not only improves the efficiency of testing cryptographic systems but also ensures that the main functionality remains intact during the fault injection process. The methodology was validated using three scenarios and SystemC ASCON as a case study. The first simulation involved evaluating fault detection capabilities, the second focused on the impact of AOP on executable file size and simulation time, and the third focused on the ESL impact on the ASCON design process. Simulation results show that this methodology can perfectly evaluate the robustness of the ASCON design against fault injection attacks with no significant impact on simulation time and file executable size. Additionally, the simulation results prove that the ASCON development life cycle at the ESL reduces the amount of time devoted to the design procedure by 83.34%, and the ASCON security attack simulations at the ESL decrease the simulation time by 40% compared to the register transfer level (RTL).

Keywords: aspect-oriented programming; SystemC; AspectC++; lightweight cryptography; ASCON; fault attacks

check for updates

Citation: Mestiri, H.; Barraj, I.; Bedoui, M.; Machhout, M. An ASCON AOP-SystemC Environment for Security Fault Analysis. *Symmetry* **2024**, *16*, 348. https://doi.org/10.3390/sym16030348

Academic Editors: Fajiang Yu, Kuo-Hui Yeh and Zhangyi Wang

Received: 5 February 2024
Revised: 8 March 2024
Accepted: 11 March 2024
Published: 14 March 2024

1. Introduction

Electronic cryptographic devices are crucial in embedded systems for securing secret information and protecting sensitive data. These devices use advanced algorithms and protocols to encrypt and decrypt information, ensuring that only authorized individuals can access it. Additionally, cryptographic devices play a vital role in preventing unauthorized tampering or modification of data, providing an extra layer of security for embedded systems. They store the secret key and cryptographic algorithm, which are designed to protect against mathematical attacks; however, when implemented on hardware systems, they are vulnerable to physical attacks. Attackers aim to gain knowledge of the secret key and confidential information, or to disrupt normal execution behavior. Fault injection attacks are an efficient method for obtaining these keys and compromising the security of cryptographic devices.

Currently, developers' capacity to design and verify cryptographic embedded systems is inadequate in regard to the escalating complexity of such systems. This is due to the growing demand for secure and efficient cryptographic algorithms, as well as the need to protect sensitive data in various applications, such as IoT devices, mobile devices, and cloud computing. As a result, developers are facing challenges in ensuring the correct implementation and integration of cryptographic algorithms within these systems while also meeting performance and power constraints. SystemC [1], a standard language for complex system modeling and verification, has been deemed suitable for simulated fault injection of System on Chip (SoC) and hardware designs due to its ability to accurately model and simulate the behavior of cryptographic algorithms. By using SystemC, developers can evaluate the security and robustness of their cryptographic implementations by injecting faults and analyzing the system's response [2–4]. This allows for thorough testing and validation, ultimately leading to more reliable and secure cryptographic systems. Nevertheless, majoritively developers need to make changes to the SystemC code in order to introduce and identify errors, and the AOP is an approach that circumvents the need to make modifications to the cryptographic algorithm source code under examination, allowing for the clean modularization of separate concerns. AOP achieves this by separating cross-cutting concerns, such as fault injection and detection, from the core functionality of the cryptographic algorithms [5]. This not only simplifies the testing process but also enhances code reusability and maintainability, making it easier for developers to analyze and improve the robustness and security of their cryptographic implementations. The challenge in cryptography involves separating the cryptographic algorithm model, fault detection schemes and fault attack process. These modules are combined through weaving throughout the process of compilation (not the coding one) to create a comprehensive cryptographic system that is resistant to fault attacks. By separating these components, developers can focus on optimizing each module individually, ensuring that the cryptographic system is efficient and secure.

This paper proposes a new technique for testing the resilience of the ASCON systemC model against fault injection attacks at the electronic system level. The methodology makes use of SystemC for hardware design while using AspectC++ as application-level programming [6]. The proposed methodology aims to provide a comprehensive evaluation of the cryptographic system's resilience by simulating various security fault attacks. By utilizing SystemC as the hardware design language and AspectC++ as an AOP language, the methodology enables the analysis of fault injection attacks at the ESL, offering valuable insights into the robustness of cryptographic designs. Furthermore, this approach allows for efficient and accurate assessment of potential vulnerabilities, aiding in the development of more secure cryptographic systems.

We summarize our contributions as follows:

- We proposed a new methodology to simulate the security fault attacks of ASCON cryptographic systems at the ESL.
- We proposed a fault injection/detection environment to test the resistance of ASCON cryptographic SystemC models against fault injection attacks.
- We used SystemC as a system level modeling language for hardware design and AspectC++ as an AOP programming language to test the robustness of the ASCON cryptographic models without any code modifications.
- We proved that the proposed environment perfectly evaluates the robustness of the ASCON cryptographic design against fault injection attacks without any code modifications.

- We proved that the impact of AOP on simulation time and file executable size is not significant.
- Finally, we proved that using SystemC and AOP for ASCON security verification in ESL enables a speedier design process and simulation execution in comparison to RTL.

This paper is organized as follows: Sections 2 and 3 discuss the background knowledge and the related work, respectively. Section 4 discusses the proposed ASCON security SystemC-AOP methodology, which will be evaluated in Section 5 through simulation security analysis. Section 6 concludes the paper.

2. Backgrounds

This section provides a comprehensive review of the necessary background information and definitions for the following paper.

2.1. Aspect-Oriented Programming

AOP is a new paradigm that enables modularity in object-oriented languages, allowing crosscutting concerns to be encapsulated in a single aspect code module. This allows for cleaner and more maintainable code by separating the core functionality of an object from its crosscutting concerns, such as logging or security. Additionally, AOP promotes code reusability and reduces code duplication by providing a centralized location for managing these concerns across multiple objects or modules. This aspect code, inserted during runtime or compile-time, helps to separate concerns from the system, reducing system tangles and improving overall system flexibility. By encapsulating crosscutting concerns in separate aspects, developers can easily modify or update these concerns without impacting the core functionality of the objects. This separation also allows for easier testing and debugging, as each aspect can be independently tested and maintained. Overall, AOP provides a powerful tool for managing complex systems and promoting code organization and reusability. Figure 1 presents the combining aspect module with original code.

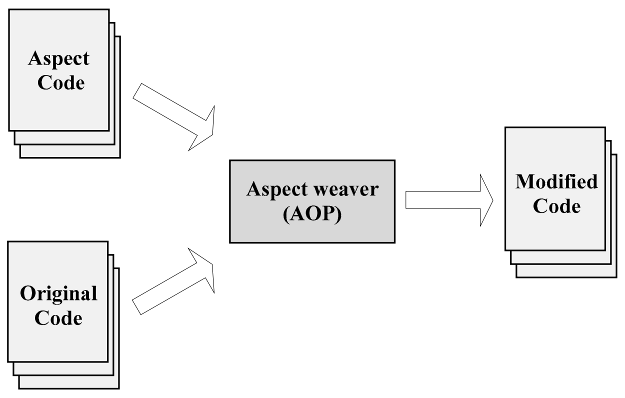

Figure 1. Combining aspect module with original code.

An aspect-oriented language involves many important concepts, as follows: [5]

- Advice is used to define a specific behavior that should be executed at a specific join point in the program, such as around, before or after a method is called.
- Aspect: A modular unit of cross-cutting concern that encapsulates advice and join points. Aspects allow for the separation of concerns in a program, making it easier to maintain and modify specific functionalities without affecting the entire codebase.
- Join point: A specific point in the execution of a program where an aspect can be applied. This can include method invocations, field accesses, or even exception handling. Aspects can target multiple join points within a program to address various concerns simultaneously.
- Pointcut: A specification that defines which join points should be affected by a particular aspect. It allows developers to selectively apply aspects to specific parts of

the codebase based on predefined conditions or patterns, enhancing flexibility and modularity in aspect-oriented programming.

To weave in a new functionality as an aspect to a system, join points are used to identify specific locations in the primary code where the functionality should be added. Join points can include method invocations, field accesses, and object instantiations. The aspect weaver weaves the new functionality into the system by inserting the aspect code at these identified join points. This allows the aspect to modify or enhance the behavior of the system without directly modifying its primary code.

2.2. ASCON Algorithm

The ASCON algorithm has been selected as the Lightweight Cryptography (LWC) standard by the National Institute of Standards and Technology (NIST) [7]. It is a symmetric key block cipher that provides high security with low computational requirements [8] based on sponge construction. It uses a permutation-based design, which contributes to its lightweight nature, and the algorithm also supports authenticated encryption, ensuring both confidentiality and integrity of the transmitted data. The ASCON algorithm is designed to be efficient and suitable for resource-constrained devices, such as IoT devices, embedded systems, and lightweight applications. It offers strong resistance against various cryptographic attacks while maintaining a small code size and low power consumption. ASCON features a 128-bit key, nonce, and tag that can have 64 or 128 bits of message length. It implements AEAD in a sponge duplex mode and employs two 320-bit permutations for its internal state. These permutations are bit-sliced into five 64-bit register words, which repeatedly apply a round transformation (ρ) based on substitution−permutation networks. A complete ASCON permutation has twelve rounds. A non-linear 5-bit S-Box, which is small and light for both Field Programmable Gate Arrays (FPGA) and Application-Specific Integrated Circuit (ASIC) implementations, is used in the substitution layer. The S-Box uses 64 simultaneous applications to update the internal state.

2.3. Fault Attacks

Fault injection attacks are a type of security attack where intentional errors or faults are injected into a system to compromise its integrity or availability. These attacks involve deliberately introducing unexpected inputs, such as invalid data or malformed commands, to exploit vulnerabilities in the system's design or implementation. By simulating various fault scenarios, attackers can gain unauthorized access, manipulate data, or disrupt the normal functioning of the targeted system. It is crucial for organizations to implement robust security measures and regularly test their systems against fault injection attacks to mitigate potential risks and protect sensitive information [9–11].

3. Related Work

3.1. AOP Software Application

AOP is being increasingly recognized as an indispensable paradigm in the realm of system application testing [12,13], with Java programs using AspectJ weavers [14] and AspectC++, a C++ based AOP weaver, becoming increasingly utilized for software validation [15]. Jain et al. utilized C++ aspects to detect vulnerabilities caused by memory leakage, improper algorithm implementation, or thread interference [16]. For dynamic software bug reporting, aspects are generated automatically and woven into C++ programs. This approach allows for the detection of bugs at runtime, providing valuable information for debugging and improving software quality. Additionally, the use of aspects in C++ programs can help identify performance bottlenecks and optimize system resources. This study discusses the manual testing of embedded C++ programs in OS, focusing on memory utilization, program robustness, coverage, and program performance in C++, similar to previous research on OS testing [17]. Aspects are implemented in the context of kernel testing. Interrupt synchronization for operating systems has been exhaustively described. A real-time operating system underwent Ada Aspects modification in order to optimize its performance during time-sensitive operations and accommodate real-time constraints [18]. By using Ada Aspects, the real-time operating system was able to prioritize and schedule tasks based on their deadlines, ensuring that time-sensitive operations were executed in a timely manner. The limitations were determined, and strategies were provided for adaptation. To rectify the deficiencies of the previous compiler/weaver architecture, a novel design was proposed. AOP-based software security hardening is the subject of [19]. Secure patterns that were developed in AOP have been implemented in secure applications via memory code encryption. A comparable methodology is described in reference [20]. AOP is a methodology that integrates fault tolerance into applications for distributed embedded systems, thereby increasing configuration for product line development, as well as testing and modernizing legacy systems.

Research on AOP-based software shows interest in functional testing, but few approaches emphasize AOP in fault-tolerant systems and security software encapsulation, despite significant interest in certain aspects of program testing. This research gap highlights the need for further exploration of AOP's potential in fault-tolerant systems and security software encapsulation. By incorporating AOP principles into these areas, researchers can enhance the overall effectiveness and robustness of program testing methodologies, ultimately leading to more reliable and secure software systems.

3.2. AOP for Security

The literature also employs AOP specialization to examine SoC architectures on hybrid hardware/software platforms. Notably, [21] proposes the utilization of an Electronic Design Automation (EDA) tool to generate an integrated description of software and hardware components employing AspectC++ programming language. This approach allows for a more comprehensive analysis of the system's behavior and performance, enabling better optimization and debugging capabilities. Additionally, [21] highlights the potential benefits of using AOP specialization in improving the reusability and maintainability of hybrid hardware/software systems, ultimately leading to more efficient development processes. The intended hardware and software implementation is a hybrid architecture based on the FPGA platform. Furthermore, SoC exploration was recommended in reference [22]. The development of a novel language called LARA, as well as the associated infrastructure, for the purpose of equipping application programs with monitoring, logging, and debugging functionalities, is detailed in an article published in [23]. Components of C++ incorporate hardware/software specializations, including non-functional and functional requirements, to increase the modularity of the code. The utilization of the AOP layer in LARA aids designers in the execution of optimized hardware and software applications based on FPGA technology.

The study reveals that AOP is utilized to differentiate hardware and software concerns in SoC architecture, but it is not considered for modeling or implementing extra specialization of encrypted data.

3.3. SystemC Modeling Using AOP

The principal focus in hardware-only implementations is AOP functional verification. The scope of AOP applications pertaining to the design or modeling of hardware components has been constrained [6,24]. In these endeavors, explicit architectural concepts, such as time and concurrency, are provided by Aspects; these concepts are then synthesized into descriptions of an SoC. AOP methodologies extract SoC-related components, including communication, cache, and performance metrics, via SystemC modeling [25]. As illustrated in [26], as synthesizing SystemC models is difficult, these techniques are more suitable for verification and simulation.

A perspective on organizing dependencies for model execution is presented in [27] when AOP is utilized in conjunction with the SystemC synthesizable subset. This approach allows for the separation of concerns in the design and the implementation of complex models. By using AOP, different aspects of the model can be modularized and managed independently, making it easier to understand and maintain the overall system. Additionally, integrating AOP with SystemC synthesizable subset enables efficient hardware synthesis, resulting in improved performance and resource utilization.

In summary, the related work proves that the limitations in these works are related to the fact that security verification modules are introduced only on module interconnections. No analysis has been carried out regarding the available internal security verification location and the possibility of injecting faults inside a SystemC module. Moreover, the original SystemC code that is modeling the system needs major modifications during security verification. The merit of our approach is that there is no need to modify the functional blocks in order to inject the security verification process into modules and interconnections. This allows for a more seamless integration of security verification processes within the SystemC modules without disrupting the original code. Our approach also addresses the potential vulnerabilities that may arise from internal security verification locations within the modules. By implementing security verification without code modifications, our method streamlines the process and enhances the overall security of the system.

4. ASCON SystemC-AOP Environment

4.1. Proposed Methodology

We utilized AspectC++ as an AOP programming language and SystemC as a modeling language at the ESL for ASCON hardware design. By integrating SystemC and AspectC++ at the system level, we have developed an ASCON AOP environment for the purpose of evaluating the fault attack detection capability and robustness of hardware designs. This environment allows us to easily apply aspect-oriented programming techniques to enhance the security of our hardware designs and analyze their resilience against fault attacks. Additionally, the integration of AspectC++ and SystemC provides a seamless workflow for designers, enabling them to efficiently implement and evaluate security measures in their hardware designs. In designing our proposed ASCON SystemC-AOP environment, we aimed to accomplish the subsequent goals:

1. Evaluate the effectiveness of fault attack detection techniques: Our proposed ASCON SystemC-AOP environment allows us to thoroughly assess the fault attack detection capability of hardware designs. By simulating various fault injection scenarios, we can identify vulnerabilities and improve the robustness of our designs.
2. Enhance hardware security through aspect-oriented programming: By leveraging aspect-oriented programming techniques, we can easily incorporate security measures into our hardware designs. This approach enables us to modularly add security aspects such as encryption, authentication, and access control, thereby enhancing the overall security of the system.
3. Analyzing robustness against fault attacks: Another goal of our ASCON SystemC-AOP environment is to analyze the robustness of hardware designs against fault attacks. By simulating various attack scenarios, we can assess the effectiveness of security measures and identify any vulnerabilities that need to be addressed.
4. Overall, our aim is to provide designers with a comprehensive toolset that not only enhances the security of their hardware designs but also simplifies the implementation process and ensures their resilience against fault attacks.

In Figure 2, a comprehensive overview of our methodology for assessing the fault attack robustness of the ASCON model is illustrated. The flow of the ASCON SystemC-AOP comprises four primary phases, as illustrated in Figure 2. During the ASCON Modeling phase, the ASCON SystemC and aspect modules are implemented. ASCON Analysis Aspect (AAA), ASCON Injector Aspect (AIA), and ASCON Controller Aspect (ACA) are the three components that comprise the aspect module. The AIA injects errors at the precise location and time. The ACA state controller is then employed to regulate the synchronization among the ASCON SystemC/AOP modules. A fault attack report is produced by the AAA, which encompasses details pertaining to fault classification, identification, injected faults, and the resulting impacts on functional designs.

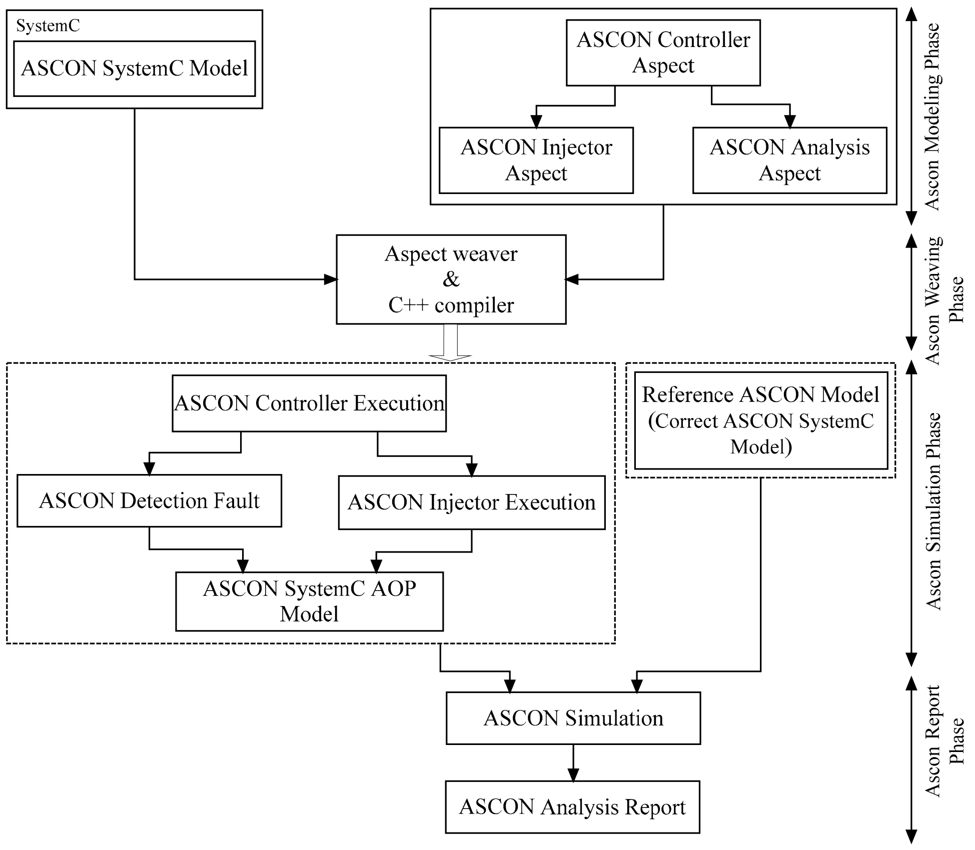

Figure 2. General view of proposed security analysis environment.

Model compilation and aspect weaving are the two discrete segments that compose the second phase, ASCON weaving. The incorporation of the three aspect codes that were previously introduced into the ASCON SystemC design does not necessitate any adjustments to the SystemC code, given that these codes are defined in AspectC++. The compiled source code and the generation of an executable file occur simultaneously with the weaving operation.

Aspect simulation constitutes the third stage of the ASCON SystemC-AOP environment. The system consists of two distinct modules, namely the reference ASCON model and the executable file system. The ASCON SystemC architecture incorporates the output of the aspect modules into the executable file system. The reference ASCON model is written in SystemC, and it presents the correct functionalities of the ASCON model without injecting faults. This allows for thorough testing and verification of the cryptographic model's performance and security features before implementation in a real-world application. Additionally, using SystemC for the reference model ensures compatibility with various hardware platforms and facilitates easier integration into existing systems. The outputs of the two modules are examined, as illustrated in Figure 2, to identify any errors that may have occurred in the executable file system.

The final environment phase of the ASCON SystemC-AOP is the ASCON analysis report, which details the impact of the fault on the ASCON design. The aspect report provides a comprehensive analysis of the fault classifications and their implications for the ASCON design, helping identify any potential vulnerabilities or weaknesses in the system and allowing for targeted improvements and optimizations to enhance its overall reliability and performance. The aspect report phase also provides recommendations for improving fault detection and minimizing false positives.

According to the outputs of the ASCON executable file system, the proposed environment is able to partition the simulation results into the following four classes:

- Silent fault: the ASCON result consists of the expected data and no errors are discovered.

- False positive: The ASCON result represents the expected data, although an error has been identified at an unspecified location.
- Undetected error: The ASCON output data does not correspond to the expected value, and no problem is identified.
- Detected error: The ASCON output values do not match the intended value, and an error has been found.

4.2. ASCON Injection/Detection via AOP

4.2.1. ASCON Controller Aspect

By means of an interface between the ACA, AIA, and AAA modules, all injection and detection duties are determined. The ACA process utilizes a Finite State Machine (FSM) to guarantee fault controllability during injection and detection while causing minimal disruption to the designated system. Figure 3 depicts the FSM proposal for ACA. The ACA controller includes various states and transitions that facilitate the seamless integration of injection and detection duties. It ensures that the process is efficient and effective in controlling defects while minimizing any potential impact on the target system.

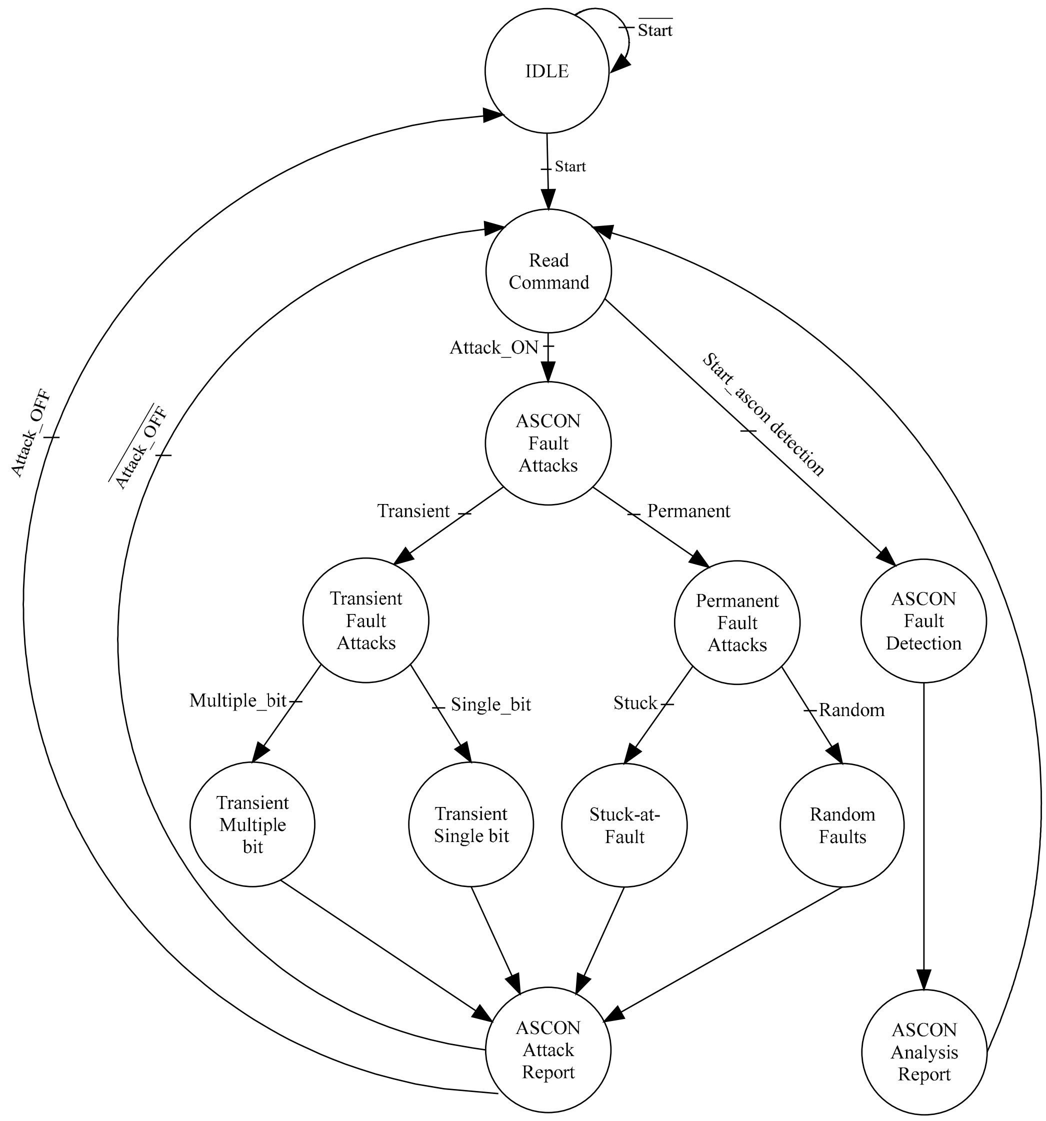

Figure 3. ACA controller.

An FSM description of the various ACA phases is provided, establishing the Attack_on transition triggers ASCON fault attack states. The subsequent state, which differs from the fault type selected, is then transmitted from the ACA module to the AIA module, Transient or Permanent. In that case, four states are produced (stuck at faults, random faults, and transient multiple bit and transient single bit, when the permanent and transient fault attack state is executed, respectively). The ACA module furnishes the AAA module with the necessary data to produce the ASCON analysis report, irrespective of the executed state. Configuring the Start_ASCON_detection transition triggers the ASCON fault detection state. Fault detection information is subsequently incorporated into the ASCON analysis report.

The pseudocode for the aspect utilized by the KFC module is detailed in Listing 1. The Ascon_command, Ascon_attack, and Ascon_detection pointcuts are described in lines 2, 3, and 4, respectively. In the ASCON class, each executable process is designated task_1, task_2, and task_3. Line 11 of the code contains the declaration advice (execution (Ascon_attack ())): after (). This indicates that the code contained in this advice is executed after the code specified by the Ascon_attack() function in the SystemC module when the Ascon_attack() function is invoked. Line 12 of the code declaration is the attack_on transition, the default value of which is zero. This transition is set after the execution of advice (Ascon_attack ()).

Listing 1. Pseudo code of aspect ascon_controller_aspect.

```
1   Aspect Ascon_Controller_Aspect {
2       pointcut Ascon_command()    = "%Ascon::task_1(...)";
3       pointcut Ascon_attack()      = "%Ascon::task_2(...)";
4       pointcut Ascon_detection()   = "%Ascon::task_3(...)";
5
6       advice (execution(Ascon_command())): after () {
7           if start
8             read_command();
9             ...
10      }
11      advice (execution(Ascon_attack())): after () {
12          if attack_on{
13            ascon_fault_attack ();
14            if permanent
15                permanent_fault_attacks();
16                if stuck_bit
17                    stuck_at_fault();
18                else if random
19                    random faults();
20            else if transient
21                transient_fault_attacks
22                if single_bit
23                    transient_single_bit();
24                else if multiple_bit
25                    transient_multiple_bit();
26            ascon_attack_report();
27          }
28          ...
25      }
29      advice (execution(Ascon_detection())): after () {
30          if start_ascon_detection {
31            ascon_fault_detection ();
32            ascon_analysis_report ();
33          }
34          ...
35      }
36      ...
37  }
```

The SystemC special characters in Listing 1 mean the following:

- "::" is used to access a member of a module.
- "%" is interpreted as a wildcard for names or parts of a signature.
- ". . ." matches any number of parameters in a function signature or any number of scopes in a qualified name.

4.2.2. ASCON Injector Aspect

The second module that establishes an interface with the ACA module is the AIA module, which injects database faults. Four distinct categories of faults are generated: transient single faults, transient multiple faults and permanent fault (stuck-at-0-fault and stuck-at-1-fault), as well as single faults and random faults. These faults are injected into simulate various real-world scenarios and test the robustness of the ASCON modules. The AIA module carefully generates these faults to ensure that they accurately mimic the types of faults that may occur in a production environment. The AIA module offers interfaces through which the values of each variable in the possible locations can be read and modified. By providing the ability to read and modify the values of each variable in the possible locations, the AIA module allows for thorough testing and analysis of the ASCON modules' robustness. This capability ensures that the faults generated accurately reflect potential faults that may occur in a real-world production environment, enabling developers to identify and address any vulnerabilities or weaknesses in the system. The quantity of terminals and modified variables is contingent upon the specific application.

The flowchart in Figure 4 illustrates the functionality principle of the AIA.

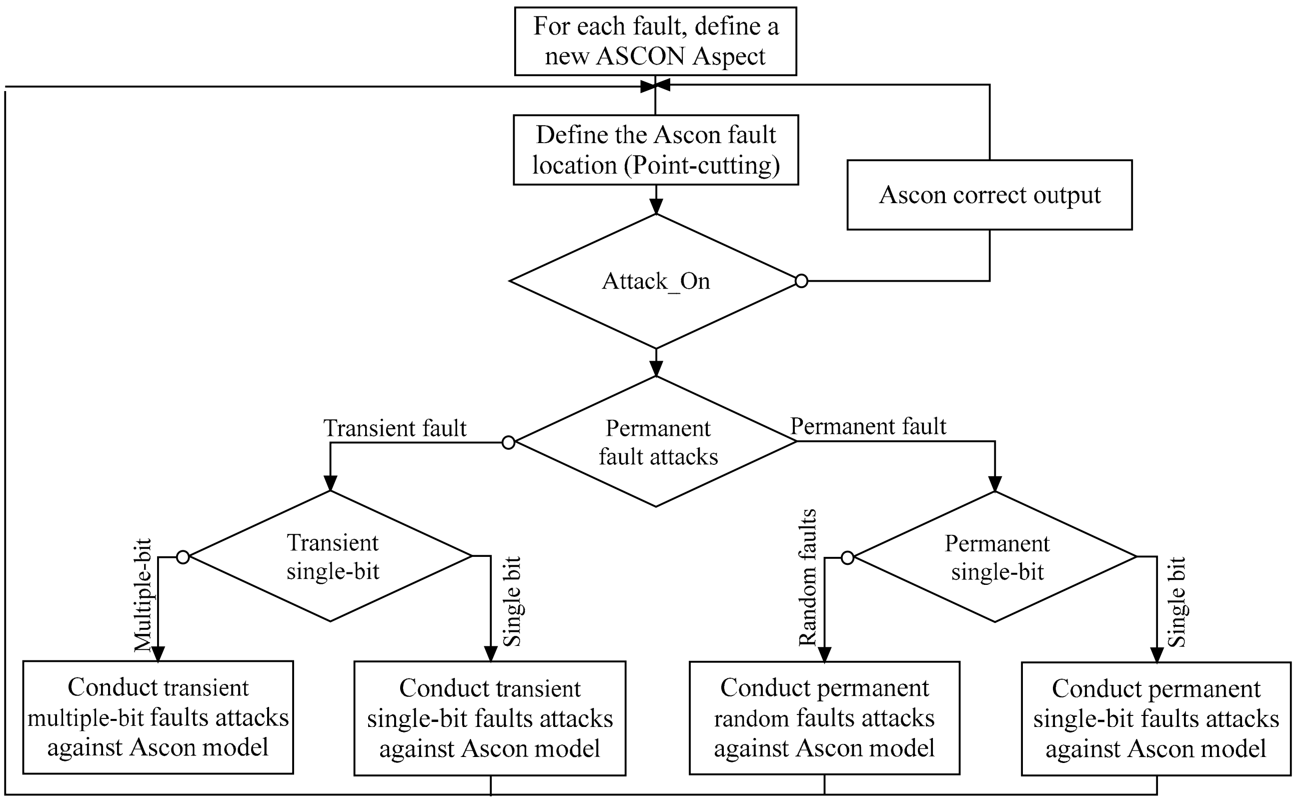
For each fault, define a new ASCON Aspect
Define the Ascon fault location (Point-cutting)
Ascon correct output
Attack_On
Transient fault
Permanent fault attacks
Permanent fault
Transient single-bit
Multiple-bit
Single bit
Random faults
Permanent single-bit
Single bit
Conduct transient multiple-bit faults attacks against Ascon model
Conduct transient single-bit faults attacks against Ascon model
Conduct permanent random faults attacks against Ascon model
Conduct permanent single-bit faults attacks against Ascon model

Figure 4. Flowchart of ASCON injector aspect.

As seen in Figure 5, the AIA module uses ports to modify variables at possible fault times and locations, using the ASCON cryptographic model to assess the vulnerability detection and injection methodology of SystemC-AOP. The possible injection locations are Fault Ascon Interconnections (FAI), Fault Ascon Registers (FAR) and Fault Ascon Bus (FAB).

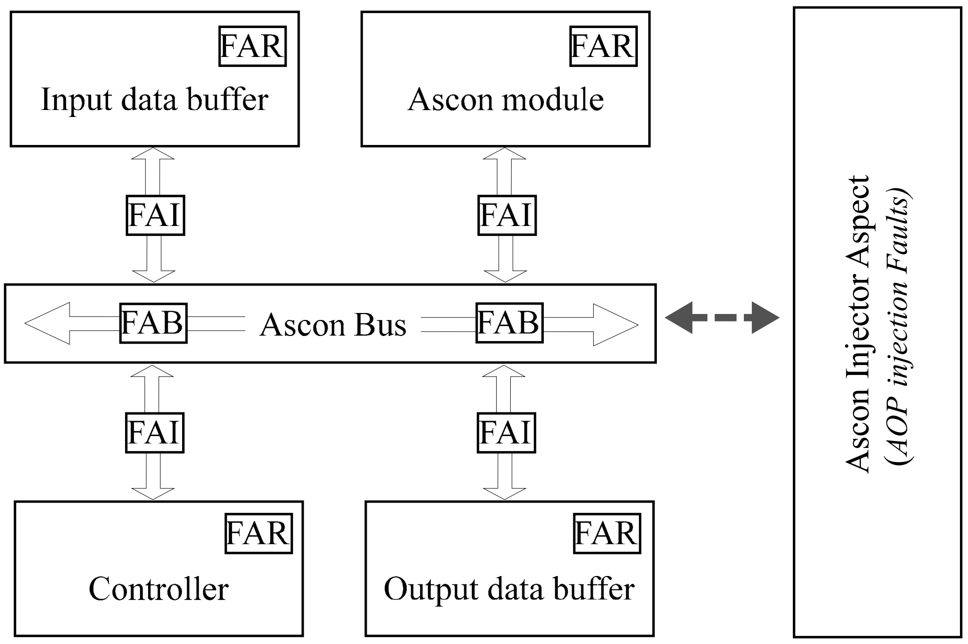

Figure 5. ASCON AOP fault location.

The standard configuration of the cryptographic model consists of the following five modules:

- Input and output buffer data.
- ASCON module for processing input messages.
- Control module for synchronization.
- ASCON bus for communication between modules.

Figure 5 shows faults can be introduced into peripherals, memories, registers, and functional modules, allowing for the modeling of all possible security attack faults. Fault injection includes transition elements like data buses and is performed using the AOP technique, eliminating the need for modification in the ASCON SystemC code. The FAB is integrated into the data FIFO, decoder, and data bus design in order to inject faults. Faults will be applied to the data that is being transmitted on the bus within the FAB. In order to introduce faults into operational modules, an FAI is inserted, which modifies data processing to inject faults, employing a saboteur to conceal the transmitted data subsequent to the activation of the fault.

The pseudocode for the aspect utilized by the FAI is detailed in Listing 2.

Listing 2. Pseudo code of aspect ascon_saboteur.

```
Aspect ASCON_Saboteur {
    pointcut FAI() = "%ASCON_AIA::Ascon(...)";

    advice (execution (FAI())): after () {
        if attack_on {
            faulty_ascon_output = apply_mask(ascon_in.read());
            ascon_out.write(ascon_output);
        }
        else
         ascon_out.write(ascon_in.read());
         ...
  }
  ...
}
```

The FAI is described in Line 2: in the ASCON_AIA class, with an executable process designated Ascon. Line 4 of the code contains the declaration advise advice (execution (FAI())): after (). This indicates that the code contained in this advice is executed after the code specified by the FAI() function in the SystemC module. The return value of execution (FAI ()) is ascon_out. This value depends on the attack_on transition value. If this transition is true, the ascon_out is equal to the faulty_ascon_output, modified by the apply_mask function; otherwise, ascon_out is equal to ascon_in.

The FAR can be used to generate faults in functional registers without altering the design of functional blocks, allowing for modification of signal values and variable value accessibility. The FAB, FAI, and FAR are linked to the fault injection environment, regulating the types, locations, and injection times of faults in the ASCON SystemC model.

The fault injection environment provides control over the fault injection process, allowing for precise manipulation of the faults introduced into the cryptographic model. This level of control ensures that specific aspects of the system can be targeted for testing and evaluation, improving the overall reliability and security of the system.

4.2.3. ASCON Analysis Aspect

The AAA module, based on fault detection schemes, guarantees that functional designs will continue to operate correctly in the presence of errors. The AAA module's aspect flowchart is shown in Figure 6. The presented environment utilizes fault detection approaches to safeguard the ASCON cryptographic algorithm against hostile attackers who may introduce faults in order to uncover secret keys. The fault analysis procedure is initiated when start_ascon_detection is set to true. The SystemC model is checked and compared using data from correct and faulty models. Once the fault detection process is complete, an analysis ASCON report is generated, detailing the impact of faults on operational designs. The report also includes details on the fault detection schemes utilized, highlighting any vulnerabilities that were successfully detected and mitigated. Additionally, the analysis report provides insights into the performance of the cryptographic algorithm under different fault injection scenarios. This information is crucial to identifying potential weaknesses and enhancing the overall security of the system.

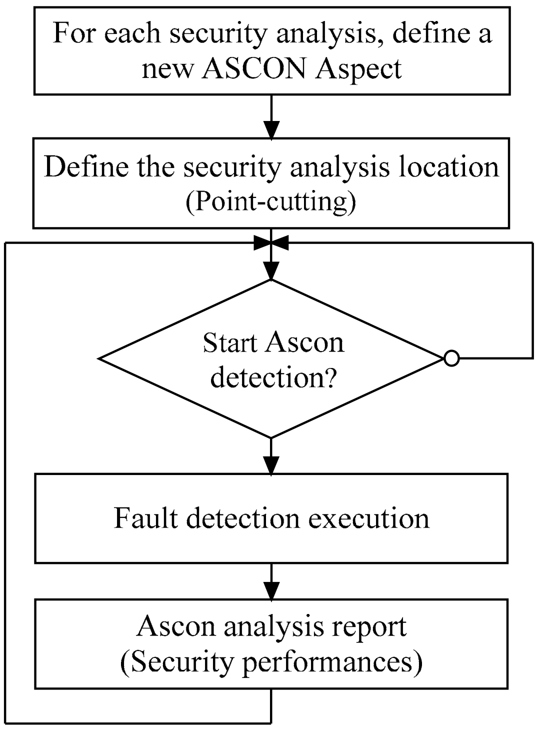
For each security analysis, define a new ASCON Aspect
Define the security analysis location (Point-cutting)
Start Ascon detection?
Fault detection execution
Ascon analysis report (Security performances)

Figure 6. Flowchart of ASCON analysis aspect.

The pseudocode for the aspect utilized by the AAA module is detailed in Listing 3.

Listing 3. Pseudo code of aspect ascon_analysis_aspect.

```
Aspect ASCON_Analysis_Aspect {
  pointcut AAA() = "%ASCON_AAA::Ascon(...)";
  advice (execution(AAA())): after () {
    if start_ascon_detection {
      result= ASCON_fault_detection(ascon_in.read());
      Ascon_Analysis_report(analysis.read());
    }
    ...
  }
  ...
}
```

5. ASCON Methodology Validation

This section evaluates the SystemC-AOP cryptographic verification security methodology by utilizing the ASCON model for fault injection/detection in the cryptographic verification process. We conducted a series of fault injection simulations to evaluate the fault detection capabilities of the SystemC-AOP verification environment. Next, we analyzed the impact of the AOP on both executable file size and simulation time. In part 3, we looked at how the AOP impacts ASCON design development. All designs were explained in terms of AspectC++ 2.3 and SystemC 2.3.4. All simulations were performed using gcc version 10.3 and an Intel Core I5-3470 3.2 GHz CPU with 8 GB of RAM.

5.1. AOP Impact on Fault Analysis

In order to assess the effectiveness of the ASCON SystemC-AOP environment in assessing the ASCON system's ability to withstand fault attacks, we conduct fault attacks using the suggested environment to evaluate the ASCON SystemC-AOP's detection capabilities. Two scenarios, namely pure SystemC and SystemC-AOP, are used to assess the fault detection capabilities. Additionally, fault detection methodologies are utilized for this evaluation [8]. In order to achieve this, we use the AspectC++ and SystemC simulation kernels to evaluate and contrast the outcomes of the ASCON SystemC-AOP environment.

The SystemC scenario employs SystemC as a language for modeling at the system level, necessitating modifications for the purpose of incorporating fault attacks and detection into the ASCON process. The SystemC-AOP scenario combines AspectC++ with SystemC without the need for any modifications to the model's ASCON SystemC code. The SystemC-AOP scenario allows for the integration of AspectC++ and SystemC, enabling the implementation of fault attacks and detection in the ASCON process without any alterations to the existing SystemC ASCON code. This approach provides a seamless and efficient way to enhance the security of the system-level modeling language.

5.1.1. AOP Impact on Transient Fault Analysis

We started by analyzing the fault detection capabilities of the ASCON cryptographic model, specifically searching for transient errors that affect single and multiple bits. We conducted various experiments to inject faults into the ASCON cryptographic model and observed its ability to detect and mitigate these faults. The ASCON SystemC model's detection capabilities for transient multiple-bit affecting at least two bits were tested through eight tests with varying fault multiplicity. Our analysis focused on the model's robustness against transient faults and its effectiveness in maintaining data integrity. The simulation security uses 1,500,000 faults, ensuring meticulous insertion of transient faults into every message, operation, and ASCON round.

The simulation results, seen in Table 1, show that both SystemC-AOP and SystemC ASCON models have equivalent fault detection capabilities in terms of undetected faults, indicating the validity of our SystemC-AOP technique, as both scenarios identified all inserted random and transient faults. These findings suggest that integrating AOP into the SystemC framework does not compromise the ASCON detection capabilities of the design models. Furthermore, the successful identification of all inserted transient faults highlights the effectiveness and reliability of our SystemC-AOP technique in ensuring robustness against potential security vulnerabilities.

Table 1. SystemC-AOP/SystemC detection capability.

Fault Types	**Fault Multiplicity**		**Secured ASCON Model** [a]		**Secured ASCON Model** [b]		**Secured ASCON Model** [c]	
			SystemC-AOP	**SystemC**	**SystemC-AOP**	**SystemC**	**SystemC-AOP**	**SystemC**
Transient faults	Single-bit		0.004	0.004	0.0044	0.0044	0.0046	0.0046
	Multiple-bit	2-bit	0.00327	0.00327	0.00367	0.00367	0.00387	0.00387
		3-bit	0.0022	0.0022	0.0028	0.0028	0.00293	0.00293
		4-bit	0.00167	0.00167	0.002	0.002	0.00233	0.00233
		5-bit	0.00113	0.00113	0.00167	0.00167	0.0018	0.0018
		6-bit	0.0008	0.0008	0.00107	0.00107	0.00147	0.00147
		7-bit	0.0006	0.0006	0.00087	0.00087	0.0012	0.0012
		Random	0.00127	0.00127	0.0016	0.0016	0.00187	0.00187
Permanent faults	Single-bit		0.0036	0.0036	0.0036	0.0036	0.00433	0.00433
	Random		0.00107	0.00107	0.0014	0.0014	0.0016	0.0016

[a] 1st fault detection scheme in [8]. [b] 2nd fault detection scheme in [8]. [c] 3rd fault detection scheme in [8].

5.1.2. AOP Impact on Permanent Fault Analysis

The second experiment focused on evaluating the detection capabilities of the ASCON SystemC model for both permanent single-bit and random faults. The permanent faults stuck-at-0 and stuck-at-1 have been considered in this evaluation. The ASCON model has been thoroughly tested with 1,500,000 permanent faults being meticulously inserted into every round, operation, and byte.

Table 1 displays the proportion of undetected permanent single-bit and random faults in the protected cryptographic model using two scenarios. In both scenarios, the fault detection functionalities of the ASCON SystemC model demonstrate consistent equivalence with regard to permanent faults, indicating the effectiveness of our SystemC-AOP technique. This finding suggests that our SystemC-AOP technique is reliable and robust in detecting permanent faults in the ASCON cryptographic model. These results further validate the potential of our approach for enhancing the security and reliability of cryptographic systems.

5.2. Simulation AOP Impact

An evaluation of the effect that AOP has on simulation time was carried out by injecting a total of 1500 random faults into the cryptographic model's possible fault locations and then determining the average amount of time required for simulation.

It is essential to note, prior to discussing the results, that the number of join point data inserted by the advice code into the ASCON SystemC code determines the effect of the AOP approach on simulation time.

Table 2 displays the results of simulation time, including user time (uTime) and kernel time (kTime), for both scenarios. Note that the difference is less than the Linux command time measurement tool's expected error.

The study concludes that the AOP's impact on simulation time is not significant and should not hinder its use in cryptographic verification security. Additionally, the study found that the AOP approach did not introduce any noticeable overhead in terms of simulation time. This suggests that incorporating AOP techniques into cryptographic verification security can be undertaken without compromising performance. Therefore, it is recommended to utilize AOP in this context for its potential benefits without concerns about its impact on simulation time.

Table 2. SystemC-AOP/SystemC simulation time.

ASCON Model	Scenarios			
	SystemC-AOP		SystemC	
	uTime (s)	kTime (s)	uTime (s)	kTime (s)
Secured ASCON Model [a]	1.762	0.031	1.763	0.030
Secured ASCON Model [b]	1.769	0.037	1.767	0.038
Secured ASCON Model [c]	1.779	0.050	1.777	0.049

[a] 1st fault detection scheme in [8]. [b] 2nd fault detection scheme in [8]. [c] 3rd fault detection scheme in [8].

Additionally, the research examined the size of SystemC and SystemC-AOP executable files produced by the SystemC kernel and AspectC++. As seen in Table 3, it was found that AOP does not significantly affect the size of the executable file. This indicates that incorporating AOP techniques into cryptographic verification security does not result in larger executable files. Hence, developers can confidently utilize AOP in this context without worrying about the impact on the size of the generated executable file.

Table 3. SystemC-AOP/SystemC file executable size.

ASCON Model	Scenarios	
	SystemC-AOP	SystemC
Secured ASCON Model [a]	1.124 MB	1.132 MB
Secured ASCON Model [b]	1.131 MB	1.140 MB
Secured ASCON Model [c]	1.156 MB	1.163 MB

[a] 1st fault detection scheme in [8]. [b] 2nd fault detection scheme in [8]. [c] 3rd fault detection scheme in [8].

5.3. ASCON Design Process: ESL Impact

We conducted a series of experiments to evaluate the impact of the ESL on the design process. The ESL modeling level uses TLM Programmer View Timed (TLMPVT) and SystemC as the modeling languages, whereas VHDL is used to describe the identical cryptographic model at RTL, with both being simulated under the same setting conditions. The purpose of this experimentation was to compare the impact of ESL on the simulation time and ASCON design process in comparison to RTL. By implementing the ESL modeling level TLMPVT and the modeling language SystemC, we were able to accurately assess the effects. Additionally, by describing the cryptographic model in VHDL at the RTL level, we were able to directly compare the simulation results between SystemC and VHDL.

The study revealed that the RTL ASCON code is five times more than the ESL ASCON code, as per the counted lines in both description levels. Additionally, the development life cycle was evaluated in weeks. While 18 weeks were dedicated to RTL coding, only 3 weeks were enough for ESL modeling. This significantly reduces the amount of time devoted to the design procedure by 83.34%. This reduction in time allows for faster iterations and more efficient design improvements. Additionally, the study found that the ESL code was more easily readable and maintainable compared to the RTL code, leading to increased productivity and reduced debugging time. Additionally, we compared the simulation times of security attacks in both scenarios involving the injection of 1,500,000 random faults in various locations. In both instances, 99.99% of the potential injection sites were covered. The ESL enhances the speed of security attack simulations and decreases the simulation time by 40%, according to the simulation results. Additionally, we found that the ESL approach also improves the accuracy of the security attack simulation, with a higher percentage of faults being detected compared to RTL coding. This suggests that adopting ESL modeling can not only save time in the ASCON design process but can also enhance system security.

It is less complicated and simpler to implement the proposed ASCON ESL environment compared to RTL due to its ability to abstract required RTL design details and provide templates for almost all HW/SW coding styles through ESL libraries. These APIs allow developers to quickly and efficiently integrate cryptographic algorithms into their designs without having to worry about low-level implementation details. Additionally, the ESL environment provides a higher level of abstraction, making it easier for designers to understand and modify the code as needed.

The AOP combination with ESL prevents modifications of the ASCON original code, focusing on security concerns in regard to the injection/detection aspects. After simulation, all security concerns are disconnected, including the detection aspect, injection faults aspect, and ASCON ESL model. This separation ensures that the original code remains intact and unaffected by security concerns. The injection faults and detection modules are designed to operate independently, allowing for easy customization and modification without altering the core functionality of the original ESL model.

The ESL literature predicts a reduction in simulation speed, but the novelty lies in speeding up security attack simulations with the same effectiveness in injection location reporting. The distinction between modeling levels is in the electrical intricacies rather than the cryptographic techniques. The ESL level provides a faster environment for estimating the resistance of cryptographic algorithms despite the fact that it performs the same operations at both levels. At the ESL level, the simulation speed reduction is achieved by abstracting away the electronic details and focusing solely on the behavior of the cryptographic algorithm. This allows for faster execution of the simulation while still maintaining an equal degree of coverage with regard to the locations of injections. Therefore, despite the difference in modeling levels, the underlying security attack simulation at ESL remains efficient and effective in evaluating the robustness of the ASCON cryptographic model.

6. Conclusions

This work introduces a mechanism for injecting and detecting faults at the ESL level to simulate the security fault attacks of ASCON cryptographic systems. It provides an overview of fault injection attacks on cryptographic systems and introduces the SystemC-AOP methodology for evaluating the ASCON cryptographic designs' robustness against these attacks. The AOP methodology is also presented for injecting faults at the ESL in SystemC designs. The fault injection/detection methodology we proposed involves injecting faults at the ESL using the AOP methodology in SystemC designs. Our approach allows for the evaluation of the fault injection resistance of cryptographic designs, providing valuable insights into their security vulnerabilities. Additionally, our methodology can be used to simulate and detect security fault attacks in a controlled environment, aiding in the development of more secure cryptographic systems.

The suggested methods for injecting and detecting faults have been demonstrated to be feasible and effective in evaluating cryptographic designs' robustness in resistance to fault attacks. A negligible effect of the AOP is observed in the simulation time, making it useful in testing security domains by reducing efforts and errors. Additionally, the AOP approach allows for easy customization and modification of the fault injection/detection methodology to suit specific cryptographic systems. This flexibility ensures that the methodology can be applied to a wide range of security scenarios, further enhancing its usefulness in evaluating the robustness of cryptographic designs.

Author Contributions: Conceptualization, designing, writing—original draft preparation, H.M. and I.B.; experimentation, simulation, and analysis, H.M., I.B. and M.B.; investigation and resources, H.M. and I.B.; writing review and editing, H.M. and M.M.; supervision, H.M.; project administration, H.M. and I.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research work was funded by Prince Sattam bin Abdulaziz University under project number (PSAU/2023/01/26665).

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors extend their appreciation to Prince Sattam bin Abdulaziz University for funding this research work through the project number (PSAU/2023/01/26665).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pomante, L.; Muttillo, V.; Santic, M.; Serri, P. SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems. *Microprocess. Microsyst.* **2020**, *72*, 102898. [CrossRef]
2. Lohmann, D.; Huf, A.; Lettnin, D.; Siqueira, F.; Güntzel, J.L. A Domain-specific Language for Automated Fault Injection in SystemC Models. In Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, 9–12 December 2018. [CrossRef]
3. Roux, J.; Beroulle, V.; Morin-Allory, K.; Leveugle, R.; Bossuet, L.; Cézilly, F.; Berthoz, F.; Genévrier, G.; Cerisier, F. High-level fault injection to assess FMEA on critical systems. *Microelectron. Reliab.* **2021**, *122*, 114135. [CrossRef]
4. Mestiri, H.; Lahbib, L.; Machhout, M.; Tourki, R. An AOP-Based Fault Injection Environment for Cryptographic SystemC Designs. *J. Circuits Syst. Comput.* **2015**, *24*, 1550008. [CrossRef]
5. Mestiri, H.; Barraj, I.; Machhout, M. An AOP-Based Security Verification Environment for KECCAK Hash Algorithm. *Comput. Mater. Contin.* **2022**, *73*, 4051–4066. [CrossRef]
6. Mestiri, H.; Barraj, I.; Machhout, M. AES High-Level SystemC Modeling using Aspect Oriented Programming Approach. *Eng. Technol. Appl. Sci. Res.* **2021**, *11*, 6719–6723. [CrossRef]
7. Turan, M.S.; McKay, K.; Chang, D.; Bassham, L.E.; Kang, J.; Waller, N.D.; Kelsey, J.M.; Hong, D. *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*; NIST.IR.8454. Available online: https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8454.pdf (accessed on 16 June 2023).
8. Kaur, J.; Kermani, M.M.; Azarderakhsh, A. Hardware Constructions for Error Detection in Lightweight Authenticated Cipher ASCON Benchmarked on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 2276–2280. [CrossRef]
9. Mestiri, H.; Barraj, I. High-Speed Hardware Architecture Based on Error Detection for KECCAK. *Micromachines* **2023**, *14*, 1129. [CrossRef] [PubMed]
10. Salam, I.; Yau, W.C.; Phan, R.C.W.; Pieprzyk, J. Differential fault attacks on the lightweight authenticated encryption algorithm CLX-128. *J. Cryptogr. Eng.* **2023**, *13*, 265–281. [CrossRef]
11. Rajkumar, S.; Sheeba, S.L.; Sivakami, R.; Prabu, S.; Selvarani, A. An IoT-Based Deep Learning Approach for Online Fault Detection Against Cyber-Attacks. *SN Comput. Sci.* **2023**, *4*, 393. [CrossRef]
12. Patel, S.; Katiyar, S.K.; Sharma, N. Metric Analysis for AOP and OOP Programming Paradigm. *J. Inst. Eng. (India) B* **2023**, *104*, 215–220. [CrossRef]
13. Mohite, S.; Sarda, A.; Joshi, S.D. Analysis of System Requirements by Aspects-J Methodology. In Proceedings of the IEEE International Conference on Computing, Communication and Green Engineering (CCGE), Pune, India, 23–25 September 2021. [CrossRef]
14. Bentrad, S.; Neslati, D. PAN4AJ: A Programming AssistaNt for Introductory AspectJ Programming. *Turk. J. Comput. Math. Educ.* **2022**, *13*, 565–578. [CrossRef]
15. Ramalingam, M.; Saranya, D.; ShankarRam, R.; Chinnasamy, P.; Ramprathap, K.; Kalaiarasi, K. An Automated Framework Dynamic Web Information Retrieval Using Deep Learning. In Proceedings of the IEEE International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 25–27 January 2022. [CrossRef]
16. Jain, R.; Agrawal, R.; Gupta, R.; Jain, R.K.; Kapil, N.K.; Saxena, A. Detection of Memory Leaks in C/C++. In Proceedings of the IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS), Bhopal, India, 22–23 February 2020. [CrossRef]
17. Tanigawa, I.; Hisazumi, K.; Ogura, N.; Sugaya, M.; Watanabe, H.; Fukuda, A. RTCOP: Context-Oriented Programming Framework based on C++ for Application in Embedded Software. In Proceedings of the 2nd International Conference on Information Science and Systems, Tokyo, Japan, 16–19 March 2019. [CrossRef]
18. Gabsi, W.; Zalila, B.; Jmaiel, M. Extension and adaptation of an aspect oriented programming language for real-time systems. *Int. J. Bus. Syst. Res.* **2020**, *14*, 139–161. [CrossRef]
19. AlSobeh, A.M.R.; Magableh, A.A. BlockASP: A Framework for AOP-Based Model Checking Blockchain System. *IEEE Access* **2023**, *11*, 115062–115075. [CrossRef]
20. Gabor, U.T.; Egidy, C.C.; Spinczyk, O. Interface Injection with AspectC++ in Embedded Systems. In Proceedings of the IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), Hangzhou, China, 3–5 January 2019. [CrossRef]
21. Yoshiya, E.; Nakanishi, T.; Isshiki, T. RTL Design Framework for Embedded Processor by using C++ Description. In Proceedings of the IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021. [CrossRef]

22. Elhariti, Z.; Alali, A.; Sadik, M.; Aamali, K. Cosimulation of Power and Temperature Models at the SystemC/TLM for a Soft-Core Processor. *Adv. Mater. Sci. Eng.* **2020**, *2020*, 2567915. [CrossRef]
23. Pinto, P.; Carvalho, T.; Bispo, J.; Cardoso, J. LARA as a language-independent aspect-oriented programming approach. In Proceedings of the Symposium on Applied Computing, Marrakech, Morocco, 3–4 April 2017. [CrossRef]
24. Silva, C.V.; Villarroel, R.; Johnson, F.; Madariaga, E.; Urz, A.; Carter, L.; Campos-Vald, C. Aspect-Combining Functions for Modular MapReduce Solutions. *Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 565–574. [CrossRef]
25. Lin, B.; Xie, F. A Systematic Investigation of State-of-the-Art SystemC Verification. *J. Circuits Syst. Comput.* **2020**, *29*, 2030013. [CrossRef]
26. Biagetti, G.; Falaschetti, L.; Crippa, P.; Alessandrini, M.; Turchetti, C. Open-Source HW/SW Co-Simulation Using QEMU and GHDL for VHDL-Based SoC Design. *Electronics* **2023**, *12*, 3986. [CrossRef]
27. Pieper, P.; Herdt, V.; Drechsler, R. Advanced Embedded System Modeling and Simulation in an Open Source RISC-V Virtual Prototype. *J. Low Power Electron. Appl.* **2022**, *12*, 52. [CrossRef]