Karafan

Quarterly Scientific Journal of National University of Skill

Spring 2026, Vol. 23, No. 1, p.410-427

Journal Homepage: https://karafan.tvu.ac.ir/?lang=en

:10.48301/kssa.2025.526604.3235

High-level modeling in SystemC using an aspect-oriented programming approach

Parviz Mohsenzadeh[1]

[1] Department of Faculty of Electrical and Computer Engineering, Technical and Vocational University (TVU), Tehran, Iran.

ARTICLE INFO

Article Type:
Original Research

Received: 2025.08.19
Revised: 2025.09.29
Accepted: 2025.11.25

Keyword:
SystemC library
aspect-oriented extension
time complexity
component systems
system description verification
cryptography
security simulation

***Corresponding Author:**
Parviz Mohsenzade
Email: *pmohsenzadeh@tvu.ac.ir*

ABSTRACT

An appropriate platform for modeling and simulation, with a steep and accelerated learning curve, is essential for analyzing the time complexity or runtime behavior of algorithms. Leveraging methods based on the SystemC library routines within the structure of an electronic system enables implementations that significantly enhance the performance of cryptographic models. This structure provides one of the critical security and performance attributes namely, algorithmic time complexity or execution time—which serves as a safeguard against potential attacks. The SystemC-based methodology is widely employed for system-level modeling, architectural exploration, performance evaluation, software development, performance verification, and high level synthesis. It is frequently integrated with Electronic System-Level (ESL) design and Transaction-Level Modeling (TLM). However, applying SystemC in security oriented simulations typically requires modifications to existing code, thereby increasing the complexity of the modeling process. One of the key advantages of this approach, without necessitating any code alterations, is the adoption of Aspect Oriented Programming (AOP), which can be effectively utilized for both security simulation and cryptographic modeling. Consequently, the model can be evaluated within a functional verification environment without introducing any changes to the original code. This model is implemented using an aspect-oriented extension of the C or C++ languages, which functions as an aspect-oriented programming language. Simulation results demonstrate that the integration of the aspect-oriented approach imposes negligible overhead on simulation runtime or executable file size. Nevertheless, this methodology substantially increases cryptographic flexibility in assessing diverse security conditions.

E-ISSN: 2538-4430
ISSN: 2382-9796

EXTENDED ABSTRACT

Introduction

A cryptographic system is considered to be the fundamental and primary principles in a computer system that implements the ability to perform cryptographic operations. Such systems include secure email systems that use methods such as digital signatures, hash functions, and key management techniques. Software engineers face challenges in their efforts to ensure the accurate implementation and integration of cryptographic algorithms into these systems while maintaining performance and constraints. Using the SystemC library methods and functions, which are based on a standard language for modeling and verifying complex systems, is expected. SystemC is a set of C++ classes and macros that provide a simulated event-driven interface. This environment, considering that security is considered in relation to both software and hardware domains, provides different methods for assessing the security posture at different stages of the design. Thus, it allows for comprehensive testing and validation, ultimately resulting in more reliable and secure cryptographic systems. However, most methods require developers to modify the SystemC library program code to create and detect errors. Therefore, aspect-oriented programming introduces a structure that no longer requires modifying the source code of cryptographic algorithms and allows for efficient separation of distinct issues through clear modularization. There are different methods for assessing the security situation at different design stages, of which simulation is one of the most common options. Through simulation, it is possible to understand the inherent resistance of the system against attack, as well as examine and compare different defense mechanisms.

Methodology

Aspect-oriented programming (AOP) is a paradigm that aims to increase the modularity of different parts of an application by separating them into sections. To do this, additional behavior is added to the existing code without changing the original code. An instance or phenomenon can contain multiple pieces of information code at the same time, where each piece of information is associated with a slice. There are three types of information code: before, after, and in the middle. These different types of information code allow developers to control the flow of execution before, after, or in the middle of a particular join point, providing flexibility in cross-management. The goal is to create a representation model in SystemC using the aspect-oriented programming method. To achieve this goal, the cryptographic processes implemented by the representation cryptosystem were divided into many modules. Figure 2 shows a set of proposed representation models and theories. The SystemC representation model diagram shows their module interconnections using AspectC++ and aspect-oriented programming. The SystemC representation model of aspect-oriented programming consists of 6 parts. The proposed functional verification environment for the PRESENT model uses the transaction-based verification environment in SystemC. The system has a reference model that provides a comprehensive design description and is run concurrently with the PRESENT model to evaluate the simulation output results. The reference model manages the module interactions at the transaction

level instead of simulating signals. A module that randomly generates encryption keys and texts for the reference and proposed models.

Results and discussion

This section focuses on evaluating a set of proposed SystemCAOP models and theories and analyzing the effects of using SystemC and AspectC++ in ESL on cryptographic architecture. The fault detection capabilities were evaluated using two scenarios, pure SystemC and SystemCAOP. In addition, a fault detection scheme was used for this evaluation [6]. To do this, the AspectC++ and SystemC simulation kernels were used to evaluate and compare the results of the PRESENT SystemCAOP environment. The findings indicate that incorporating AOP into the SystemC framework does not degrade the detection capabilities of the design models. The margin of error associated with the measurement method employed (Linux commands) is negligible and thus does not affect the accuracy of the simulation results. The results show that using the AOP approach to integrate the Input_Interface, Database, Controller_Unit, FDS, and Output_Interface modules into the SystemC model has no significant effect on the simulation execution times. This indicates that the additional complexity introduced by the AOP approach does not significantly affect the simulation performance. Therefore, it can be concluded that the integration of AOP into SystemC does not have a significant impact on kTime and uTime in this particular study. In an aspect-oriented programming, one thing that should be considered when evaluating the impact of AOP is the resulting executable file size. Table 3 shows the sizes of executable files produced by AspectC++ and the SystemC kernel in two scenario situations.

Conclusion

This study presents a SystemC model for the electronic system level based on aspect-oriented programming (AOP). A functional verification environment is recommended to verify the model performance, simulation time, and executable file size. The verification environment includes test benches, stimulus generators, and coverage monitoring to ensure the accuracy and completeness of the model. In addition, the verification environment facilitates the detection of potential design defects and helps in model troubleshooting. Furthermore, it provides a comprehensive framework for evaluating the performance and reliability of the PRESENT SystemC model under different conditions. Simulation results show that the efficiency of the AspectC++ model and the PRESENT module texture have minimal impact on the simulation time and executable file size. In addition, the proposed PRESENT SystemC aspect-oriented program model exhibits greater modularity and maintainability compared to conventional methods. Furthermore, the proposed techniques for error injection and detection are viable and efficient in evaluating the resilience of cryptographic schemes against error attacks. Simulation time is minimally affected by AOP, which proves its usefulness in evaluating security domains by minimizing efforts and errors. Furthermore, the AOP approach facilitates customization and modification of the error injection/detection mechanism to adapt to unique cryptographic systems. The flexibility of this method allows its use in various security situations, thereby increasing its efficiency in evaluating the strength of cryptographic schemes. In summary, the use of AOP in SystemC modeling has the potential to improve design consistency and productivity in electronic system-level development.

فصلنامه علمی دانشگاه ملی مهارت
بهار ۱۴۰۵، دوره ۲۳، شماره ۱، ۴۲۷-۴۱۰
آدرس نشریه: https://karafan.tvu.ac.ir
10.48301/kssa.2025.526604.3235

مدل‌سازی سطح بالا در SystemC با استفاده از رویکرد برنامه‌نویسی جنبه‌محور

پرویز محسن‌زاده [۱]

۱- گروه مهندسی برق و کامپیوتر، دانشگاه ملی مهارت، تهران، ایران.

اطلاعات مقاله

نوع مقاله: مقاله پژوهشی

دریافت مقاله: ۱۴۰۴/۰۵/۲۸

بازنگری مقاله: ۱۴۰۴/۰۷/۰۷

پذیرش مقاله: ۱۴۰۴/۰۸/۰۴

کلید واژگان:
کتابخانه‌ی SystemC
برنامه افزودنی جنبه‌گرا
پیچیدگی زمانی
سیستم‌های مؤلفه‌ای
توصیف و درستی‌یابی سیستم
رمزنگاری
شبیه‌سازی امنیتی

***نویسنده مسئول:**
پرویز محسن‌زاده
پست الکترونیکی:
pmohsenzadeh@tvu.ac.ir

چکیده

پیچیدگی زمانی یا زمان اجرای در الگوریتم‌ها نیازمند داشتن یک بستر مناسب مدل سازی و شبیه‌سازی با شیب تند و سریع می‌باشد بکارگیری روش‌های مرتبط با استفاده از روال کتابخانه‌ای SystemC در ساختار سیستم الکترونیکی امکان پیاده سازی را بوجود می‌آورد که منتج به افزایش سرعت مدل‌های رمزنگاری خواهد شد. این ساختار بیانگر یکی از ویژگی های مهّم امنیتی و عملکرد پیچیدگی زمانی یا زمان اجرای در الگوریتم ها را فراهم می‌کند که سدّی در برابر حملات احتمالی را تضمین می‌کند. در استفاده از روش کتابخانه‌ای SystemC غالباً که برای مدل‌سازی در سطح سیستم، کاوش معماری، مدل سازی عملکرد، توسعه نرم‌افزار، تأیید عملکرد و سنتز سطح بالا استفاده می‌شود. اغلب این روش با طراحی در سطح سیستم الکترونیکی [۱] و با مدل‌سازی در سطح تراکنش همراه است. با این حال، استفاده از SystemC در شبیه‌سازی‌های امنیتی مستلزم تصحیح کدها موجود می‌باشد و در نتیجه پیچیدگی فرآیند مدل سازی را افزایش می‌دهد. یکی از ویژگی‌ها در بکارگیری این روش بدون نیاز به هیچ گونه تغییر کد، استفاده از برنامه‌نویسی جنبه‌گرا است که می‌تواند برای شبیه سازی امنیتی و مدل سازی رمزنگاری ا ستفاده شود بنابراین مدل در یک محیط تایید عملکردی بدون تغییر در کد ارزیابی می‌شود. این مدل با استفاده از یک برنامه افزودنی جنبه‌گرا [۲] از زبان‌های C یا C++ ارائه می‌شود که به عنوان یک زبان جنبه‌گرا مورد استفاده قرار می‌گیرد. نتایج شبیه سازی نشان می‌دهد که اثربخشی مدل و ادغام روش جنبه‌گرا اثرات ناچیزی بر مدّت زمان شبیه‌سازی یا اندازه فایل اجرایی دارد امّا منتج به انعطاف‌پذیری رمزنگاری در شرایط مختلف امنیتی در ارزیابی را افزایش می‌دهد.

۱ electronic system level (ESL)

۲ AspectC++

شاپای الکترونیکی: ۲۵۳۸-۴۴۳۰
شاپای چاپی: ۲۳۸۲-۹۷۹۶

مقدمه

یک سیستم رمزنگاری، اصول مهّم و اوّلیه در یک سیستم رایانه‌ای قلمداد می‌شود که امکان عملیات رمزنگاری را پیاده‌سازی می‌کند. از این قبیل سیستم‌ها می‌توان به سیستم پست الکترونیکی امن اشاره کرد که در آن از روش‌هایی مانند امضای دیجیتال، تابع درهم‌سازی و تکنیک‌های مدیریّت کلید استفاده می‌شود. در واقع، اصول رمزنگاری دانش تغییر دادن متن پیام یا اطلاعات به کمک کلید رمز و با استفاده از یک الگوریتم رمزنگاری است. در سیستم رمزنگاری الکترونیکی متضمن امنیّت اطلاعات محرمانه و حفاظت از داده‌های حساس، بی شک نقشی ارزنده و حیاتی در سیستم‌های تعبیه شده ایفا می‌کنند. دستگاه داده‌ها را با استفاده از الگوریتم‌ها و پروتکل‌های پیچیده رمزگذاری و رمزگشایی می‌کنند تا اطمینان حاصل کنند که فقط کاربرانی که مجوز دارند می‌توانند به آن دسترسی داشته باشند [۱].

پیاده‌سازی دستگاه‌های رمزنگاری در سیستم‌های تعبیه شده به تقویت اقدامات امنیتی در برابر بی‌نظم، پراکندگی و بی‌نظمی داده‌ها کمک می‌کند. با این حال، آن‌ها مستعد حملات فیزیکی به زیر ساخت‌های سخت‌افزاری هستند که می‌توانند به عوامل نامطلوب در داده‌های محرمانه یا کلیدهای مخفی دسترسی داشته باشند [۲-۴]. حملات تزریق خطا، که نوعی حمله فیزیکی هستند، یک تکنیک بسیار موثر برای به دست آوردن این کلیدها و تضعیف امنیت تجهیزات رمزنگاری هستند [۵-۷]. بنابراین تکنیک به دست آوردن این کلیدها و تضعیف امنیت تجهیزات رمزنگاری را به ارمغان خواهد آورد امّا می‌توان به جرأت بیان کرد که در حال حاضر، توانایی توسعه‌دهندگان برای ایجاد و تأیید سیستم‌های تعبیه‌شده رمزنگاری برای پاسخگویی به پیچیدگی روزافزون آن‌ها کافی نخواهد بود [۸-۱۰]. به عنوان مثال و به طور خاصّ، نیاز به محافظت از داده‌های حساس در برنامه‌های کاربردی، مانند اینترنت اشیا، دستگاه‌های تلفن همراه و فضای ابری، باعث افزایش تقاضا برای الگوریتم‌های رمزنگاری امن می‌شود [۸-۹]. مهندسان علوم نرم‌افزاری در تلاش خود برای اطمینان از اجرای دقیق و یکپارچه سازی الگوریتم‌های رمزنگاری در این سیستم‌ها با حفظ عملکرد و محدودیّت‌ها با مشکلاتی مواجه هستند. استفاده از روش و توابع کتابخانه‌ای SystemC که مبتنی بر یک زبان استاندارد برای مدل‌سازی و تایید سیستم‌های پیچیده مورد انتظار می‌باشد. SystemC مجموعه‌ای از کلاس‌ها و ماکروهای زبان C++ محسوب می‌شوند که یک رابط شبیه‌سازی شده از رویداد محور را ارائه می‌کند. این امکانات طراح را قادر می‌سازد تا فرآیندهای همزمان را شبیه‌سازی کند که هر کدام با استفاده از روش ساده زبان C++ توصیف شده‌اند. فرآیندهای SystemC می‌توانند در یک محیط بلادرنگ شبیه سازی شده و با استفاده از سیگنال‌ها و با انواع داده‌های ارائه شده توسط C++ ارتباط برقرار کند. موارد اضافی ارائه شده توسط کتابخانه‌ی SystemC و با تعریف توسط کاربر انجام می‌شود که از جهات خاصّی SystemC عمداً از زبان‌های توصیف سخت افزار VHDL و Verilog تقلید می‌کند، امّا به درستی به عنوان یک زبان مدل‌سازی در سطح سیستم توصیف می‌شود. SystemC برای مدل سازی در سطح سیستم، کاوش معماری، مدل سازی عملکرد، توسعه نرم‌افزار، تأیید عملکرد و سنتز سطح بالا استفاده می‌شود. SystemC اغلب با طراحی در سطح سیستم الکترونیکی[۱] و با مدل‌سازی سطح تراکنش[۲] مرتبط است که می‌تواند به طور دقیق رفتار الگوریتم‌های رمزنگاری را نشان داده و تقلید نماید و آن را به انتخاب خوبی برای

۱ ۱ electronic system level

۲ transaction-level modeling

شبیه‌سازی با حداقل تزریق و الحاق خطا در طراحی‌های سیستم روی تراشه و سخت افزار تبدیل کند [۱۰-۱۲]. SystemC به توسعه دهندگان این امکان را می‌دهد تا امنیت و انعطاف‌پذیری پیاده‌سازی‌های رمزنگاری خود را با معرفی عمدی خطاها و مطالعه واکنش سیستم ارزیابی کنند. استفاده از کتابخانه‌ی SystemC که محیط شبیه‌سازی را با هدف ارزیابی امنیتی ارائه شده را به خوبی نمایش دهد. این محیط با توّجه به این مو ضوع که امنیّت در ارتباط با هر دو حوزه‌ی نرم‌افزار و سخت‌افزار بررسی می‌شود روش‌های متفاوتی برای ارزیابی وضعیّت امنیت در مراحل مختلف طراحی وجود دارند. بنابراین امکان آزمایش و اعتبارسنجی جامع را فراهم می‌کند و در نهایت منتج به سیستم‌های رمزنگاری قابل اعتمادتر و ایمن‌تر می‌شود. با این حال، اکثر روش‌ها به توسعه‌دهندگان نیاز دارند که کد برنامه‌ی کتابخانه‌ی SystemC را برای ایجاد و شناسایی خطاها تغییر دهند. بنابراین برنامه‌نویسی جنبه‌گرا یا جنبه‌محور ساختاری را معرفی می‌کند که دیگر نیازی به تغییر کد منبع الگوریتم‌های رمزنگاری را نداشته باشد و امکان جداسازی کارآمد مسائل متمایز را از طریق مدولارسازی واضح را فراهم می‌کند. برنامه‌نویسی جنبه‌محور این مهم را با جداسازی مشکلات مقطعی اعم از معرفی و شناسایی خطاها از عملیات اساسی روش‌های رمزنگاری انجام می‌دهد [۱۳]. این نه تنها در ادامه‌ی الگوریتم‌سازی و شبیه‌سازی فرآیند آزمایش را ساده می‌کند، بلکه ظرفیت استفاده مجدد و نگهداری کد را نیز بهبود می‌بخشد و توسعه‌دهندگان را در تجزیه و تحلیل و افزایش قدرت و امنیّت راه حل‌های رمزنگاری خود تسهیل می‌کند.

امروزه با گسترش بکارگیری سیستم‌های دیجیتال با کاربردهای حساس، تضمین امنیّت آن‌ها در برابر حمله‌های مختلف به یک موضوع حیاتی تبدیل شده است. در این میان سخت‌افزار به‌عنوان لایه‌ای با بیشترین سطح دسترسی در یک سیستم دیجیتال و یک مولفه‌ی اساسی در حفظ امنیّت به‌حساب می‌آید. به‌طور معمول آسیب‌پذیری‌های سخت‌افزاری با روش‌های سطح بالاتر مثل روش‌های نرم‌افزاری قابل برطرف‌کردن نیستند. با درک اهمیّت در مقوله‌ی امنیّت سخت‌افزارها لزوم درنظرگرفتن آن به‌عنوان ساختاری مهّم که باید از ابتدای مراحل طراحی با جدیّت به آن توّجه کرد، پدیدار می‌شود. برای درک عمیق‌تر از حمله‌های احتمالی و درنظرگرفتن تأثیر آن‌ها در مراحل طراحی، همچنین برای بررسی سازوکارهای دفاعی، لازم است امنیّت سیستم مورد ارزیابی قرار گیرد.

روش‌های متفاوتی برای ارزیابی وضعیّت امنیّت در مراحل مختلف طراحی وجود دارند که شبیه سازی از رایج‌ترین گزینه‌های انتخابی خواهد بود. از طریق شبیه‌سازی می‌توان ضمن درک مقاومت ذاتی سیستم در برابر حمله، سازوکارهای دفاعی متفاوت را بررسی و با یکدیگر مقایسه کرد. به‌علاوه شبیه سازی امکان ارزیابی سیستم در مراحل مختلف طراحی از سطوح بالای توصیف سیستم تا سطوح پایین پیاده سازی را ممکن می سازد. در این پژوهش، یک محیط شبیه‌سازی با هدف ارزیابی امنیّت ارائه شده است. در مسئله‌ی رمزنگاری در تمایز مدل الگوریتم رمزنگاری، سیستم‌های تشخیص خطا و روش‌های حمله خطا از درجه اهمیّت بالایی برخوردار است همچنین این محیط متأثر به نوع مشخّصی از حمله نیست و امکان مدل‌سازی حمله‌های جدید را در نظر گرفته می‌شود.

ماژول‌ها با در هم آمیختن در طول فرآیند کامپایل، به جای کدگذاری، یکپارچه می‌شوند تا یک سیستم رمزنگاری قوی را تشکیل دهند که در برابر حملات خطا مقاوم باشد. از طریق فرآیند تقسیم این اجزا، مهندسان می‌توانند بر تقویت هر ماژول به طور جداگانه تمرکز کنند و اطمینان حاصل کنند که سیستم رمزنگاری هم موثر و هم محافظت شده است [۱۴-۱۶]. این مقاله یک مدل جدید رمزنگاری SystemC را با استفاده از تکنیک برنامه‌نویسی جنبه‌محور را معرفی می‌کند. کارایی مدل تکنیک برنامه‌نویسی جنبه‌محور در SystemC و همچنین اثرات برنامه‌نویسی جنبه‌محور

بر زمان شـــبیه‌ســـازی و اندازه‌ی فایل اجرایی، با اســـتفاده از یک محیط تأیید عملکردی، مورد ارزیابی قرار گرفت. این فرآیند از افزونه‌های C++ [۱۳] برای برنامه‌نویســـی در ســـطح برنامه و SystemC برای طراحی ســـخت افزار اســـتفاده می‌شود.

بررسی رویه

۱) برنامه‌نویسی جنبه‌گرا

AOP برنامه‌نویســـی جنبه‌گرا یک ابرانگاره بوده و با هدف افزایش مدولار بودن بخش‌های مختلف یک برنامه کاربردی با جداسازی مقطعی قلمداد می‌شود. برای انجام این کار بدون تغییر کد اصلی رفتار اضافی به کد موجود اضافه می‌شود. این نگرش یک سبک برنامه‌نویسی است که به مفهوم تقسیم نگرانی‌های مختلف را مورد بررسی قرار می‌دهد [۱۰، ۱۳]. برنامه‌نویسی جنبه‌گرا ابرانگاره یک برنامه کاربردی از کلاس‌ها و ساختار مختلف تشکیل شده است. انتظار غیر کاربردی به عنوان ساختارهایی در داخل ماژول‌ها قرار داده می‌شوند که به صورت کد مقطعی بیان می‌شود و اجزاء در کد عملیاتی برای توسعه یک برنامه کامل گنجانده شده است.

ساختارها برای اجرای ویژگی‌های فنی در کد یک برنامه از دو جزء تشکیل خواهد شد، که شامل برش نقطه‌ای[۱] و کد برنامه‌ی فعال[۲] که مورد اســـتفاده قرار می‌گیرد برای مثال، وظیفه‌ی احراز هویّت در یک برنامه را می‌توان به عنوان یک ســـاختار در نظر گرفت و ماژول جداگانه‌ای برای آن تولید کرد تا از این پس در تمام پروژه‌های مشـــابه بتوان از آن استفاده کرد. بیشتر زبان‌های برنامه‌نویسی از برنامه‌نویسی جنبه‌گرا پشتیبانی خوبی به عمل می‌آورند. بدون استفاده از برنامه‌نویسی جنبه‌گرا بایستی در هر ماژول توابع جدید افزوده شوند و در تمام ماژول‌ها ممکن است بخشی از کد تکرار شـــود. امّا با اســـتفاده از برنامه‌نویســـی جنبه‌گرا می‌توانید بدون اینکه در ماژول‌های دیگر تغییری ایجاد کنید یک ماژول دیگر بنویسید تا به صورت خودکار در ماژول‌های دیگر فراخوانی و اجرا شود. در این راستا امکان تنظیم اینکه هر متد در ماژول جدید در کدام قسمت از هر ماژول دیگر اجرا شود وجود دارد.

Pointcut تکنیکی اســـت که با تعیین محل یک یا چند نقاط اتصـــال[۳] که نشـــان دهنده‌ی نقاطی اســـت که کد عملکرد عرضی در آن درج می‌شود امکان استفاده از کد عملکرد عرضی را در یک ساختار فراهم می‌کند. با جدا کردن این نگرانی‌ها و آشفتگی، توسعه‌دهندگان می‌توانند به راحتی تغییراتی را در ساختار خاص کد ایجاد کنند بدون اینکه بر ســـاختار کلی تأثیر بگذارند. با جدا کردن این آشـــفتگی توســـعه‌دهندگان می‌توانند به راحتی تغییراتی را در جنبه‌های خاص کد ایجاد کنند بدون اینکه بر ساختار کلی تأثیر بگذارند.

Code Advice بخشی از کد است که در نقاط اتصال درج می‌شوند، که نشان دهنده وجود عملکرد متقابل است. با استفاده ازCode Advice توسعه‌دهندگان می‌توانند به طور موثر نگرانی‌های بین‌بخشی، مانند ورود به سیستم امنیت و رسیدگی به خطا را بدون به هم ریختن منطق برنامه اصلی مدیریّت و اعمال کنند. این رویکرد مقیاس‌پذیری و قابلیّت استفاده مجدد پایگاه کد را بهبود می‌بخشد و در نهایت منجر به فرآیند توسعه نرم‌افزار قوی‌تر و کارآمدتر می‌شود.

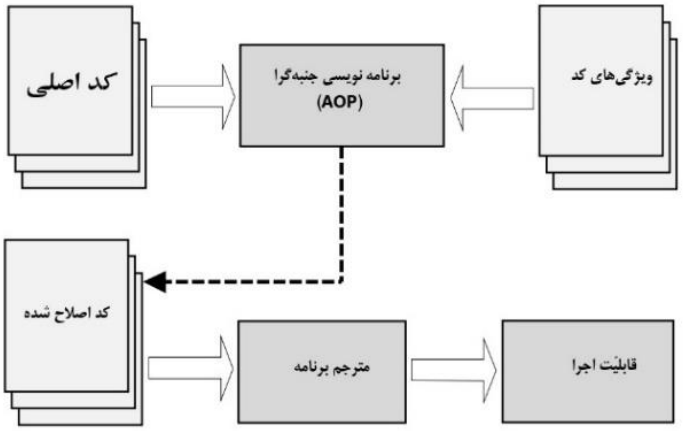
---

۱ Pointcut

۲ Code Advice

۳ Join Point

یک نمود یا پدیده می‌تواند شامل چندین کد اطلاعاتی به طور همزمان باشد که در آن هر توصیه با یک برش همراه است. سه نوع کد اطلاعاتی شامل قبل، بعد و میانه وجود دارد. این تنوع مختلف کد اطلاعاتی به توسعه‌دهندگان اجازه می‌دهد تا جریان اجرا را قبل، بعد یا در میانه توسط یک Join Point خاص کنترل کنند و انعطاف‌پذیری را در مدیریّت متقابل فراهم کنند. با به کارگیری استراتژیک این توصیه‌ها، توسعه‌دهندگان می‌توانند قابلیّت نگهداری و ماژولار بودن پایگاه کد خود را بهبود بخشند و سازگاری با نیازهای متغیر یا افزودن ویژگی‌های جدید در آینده را آسان‌تر کنند. برای گنجاندن یک ویژگی جدید در کد یک برنامه، کد اصلی باید مناطق خاصی را ایجاد کند که در آن ویژگی تعریف شده باید عمل کند. شکل۱ ادغام کد نمایشی در برنامه را نشان می‌دهد. با ادغام ویژگی‌ها در پایگاه کد توسعه‌دهندگان می‌توانند عملکرد کلی و قابلیّت نگهداری برنامه را افزایش دهند. این رویکرد به روشی کارآمدتر برای رسیدگی به نگرانی‌های مقطعی بدون درهم ریختن پایگاه کد اصلی را اجازه می‌دهد.



شکل۱. ویژگی‌های کد اصلی

رویکرد برنامه‌نوسی جنبه‌گرا اغلب برای آزمایش برنامه‌های جاسازی شده در C++ و جاوا استفاده می‌شود. این مطالعه یک مدل ارائه‌ی جدید را با استفاده از رویکرد AOP برای جلوگیری از هرگونه تغییر در کد منبع بوده و نیاز به تجزیه و تحلیل سیستم رمزنگاری مورد بررسی قرار می‌گیرد. این راه حل مسائلی را که بیش از یک ناحیه را تحت تأثیر قرار می‌دهند از هم جدا می‌کند و روش جدیدی برای توزیع مدل رمزنگاری با کلاس‌ها و جنبه‌هایی ایجاد می‌کند که از کد ماژول‌های اضافه شده محافظت می‌کند. این جداسازی با کاهش پیچیدگی کد و بهبود قابلیّت استفاده مجدد کد، امنیت و قابلیّت نگهداری برنامه را افزایش می‌دهد. علاوه بر این، امکان اشکال‌زدایی و بروزرسانی آسان‌تر سیستم رمزنگاری را بدون تأثیر بر پایه کد اصلی را فراهم می‌کند.

۲) ویژگی الگوریتم

ویژگی بلوکی یک رمز در حالت سبک وزنی متشکل از ۳۱ چرخه است که مبتنی بر یک شبکه‌ی SP که در محاسبات، یک بسته‌ی خدماتی یا سرویس که شامل مجموعه‌ای از بروزرسانی، تغییرات یا پیشرفت برنامه نرم‌افزاری است که به شکل یک بسته ارائه شده است. رمزگذاری بلوک‌های ۶۴ بیتی به یک کلید ۸۰ یا ۱۲۸ بیتی نیاز دارد [۱۷]. اجرای

کلید ۸۰ بیتی برای استقرارهای مبتنی بر برچسب که معمولاً به برنامه‌های با امنیّت پایین نیاز دارند کافی است. به ترتیب سه تابع مجزا در طول هر دور اجرا می‌شوند که مشتمل بر ()addRoundKey، ()sBoxLayer و ()pLayer خواهند بود. تابع ()addRoundKey شامل XOR بیتی با حالت کلید میانی است و تابع ()sBoxLayer یک جعبه جایگزین را برای هر بایت از وضعیّت اعمال می‌کند، در حالی که تابع ()pLayer بایت‌های داخل هر ستون را تغییر می‌دهد.

آثار مرتبط

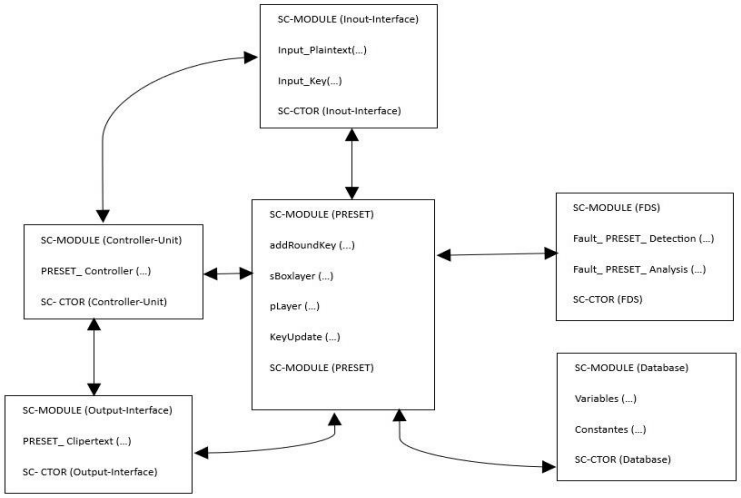
برنامه‌نویسی جنبه‌گرا به تدریج به عنوان یک چارچوب حیاتی برای ارزیابی برنامه‌های کاربردی سیستم در حال ظهور است [۱۳]. برنامه‌های جاوا که با AspectJ و در C++ با صورت AspectC++ [۱۰] یک پسوند برنامه‌نویسی جنبه‌گرا می‌باشد مجهز شده‌اند [۱۸]. یک بازو برای AOP مبتنی بر C++ به طور فزآینده‌ای برای اعتبارسنجی نرم‌افزار استفاده می‌شود [۱۹]. در روش جدیدی برای شناسایی خطاهای ناشی از عملکرد ناپایدار حافظه، اجرای نامناسب الگوریتم یا تداخل رشته با استفاده از ساختارهای C++ ارائه شد [۲۰]. ساختار به‌طور خودکار تولید و در برنامه‌های C++ ادغام می‌شوند تا به صورت پویا نقص‌های نرم‌افزار را آشکار کنند. در تحقیقات مشابهی در مورد آزمایش سیستم عامل (OS) تعبیه شده ارائه شد [۲۱]. جنبه‌های دستی برای آزمایش برنامه‌های C++ تعبیه شده در سیستم عامل به کار گرفته شد. این مطالعه بر چهار اصول کلیدی تست عملکردی، یعنی پوشش برنامه C++، حافظه، عملکرد و استحکام تأکید داشت. در [۱۰] سخت شدن امنیت نرم‌افزار بر اساس برنامه‌نویسی جنبه محور مورد بررسی قرار گرفت. برنامه‌های کاربردی امن الگوهای امنی را که در برنامه‌نویسی جنبه محور از طریق استفاده از رمزگذاری کد حافظه ابداع شده‌اند، گنجانده شده است. یک روش مشابه در [۱۳] شرح داده شده است. برنامه‌نویسی جنبه‌محور تکنیکی برای ادغام تحمل خطا در برنامه‌های کاربردی مبتنی بر سیستم جاسازی شده توزیع شده است. بسیاری از روش‌های تحمل‌پذیر خطا و پیکربندی‌های سخت‌افزارو نرم‌افزار اضافی مورد بررسی قرار گرفته‌اند. برنامه‌نویسی جنبه محور برای ارائه تحمل نقص در سطح نخ برنامه در سیستم استفاده می‌شود. این استراتژی مبتنی بر برنامه‌نویسی جنبه‌محور ماژولار بودن را افزایش می‌دهد، تلاش لازم برای مدرن‌سازی سیستم‌های قدیمی را کاهش می‌دهد و پیکربندی برای آزمایش و توسعه خط محصول را بهبود می‌بخشد. برنامه‌نویسی جنبه‌محور همچنین برای بررسی طرح‌های SoC برای سیستم‌های سخت‌افزارو نرم‌افزار هیبریدی استفاده می‌شود. افزونه‌ی AspectC++ [۲۲] روشی برای ایجاد تصویری فراگیر از نرم‌افزار و اجزای سخت‌افزاری است. برنامه‌های جنبه‌محوردارای ویژگی‌های سخت‌افزاری و نرم‌افزاری هستند که به توضیحات جامع SoCها پیوند داده می‌شوند. طراحی مورد نظر یک راه‌حل سخت‌افزار و نرم‌افزار ترکیبی مبتنی بر یک آرایه دروازه‌ای برنامه‌پذیر میدانی ترکیبی (FPGA) است. در [۲۳] یک بررسی SoC ارائه شده است. تأیید عملکردی عمدتاً از طریق استفاده از پیاده‌سازی‌های سخت‌افزاری خالص بر برنامه‌نویسی جنبه‌محور متمرکز است. برنامه‌نویسی جنبه‌محور در تعداد محدودی از روش‌ها برای طراحی یا مدل‌سازی اجزای سخت افزاری استفاده شده است [۱۳، ۲۴]. این مطالعات با استفاده از مفاهیم معماری صریح، مانند همزمانی و زمان، توصیفات قابل ترکیبی از SoC ها را ارائه می‌دهند. روش‌های برنامه‌نویسی جنبه‌محور امکان استخراج اجزای SoC را از طریق مدل‌سازی SystemC [۱۰] از جمله ارتباطات، قوانین و معیارهای عملکرد فراهم می‌کنند. مدل‌های SystemC تولید شده مشکلاتی را از نظر سنتز ارائه می‌کنند. در نتیجه، همانطور که در [۲۵] نشان داده شده است، این روش‌ها در حوزه‌ی شبیه‌سازی و تایید سود بیشتری دارند. مطالعات قبلی محدود بود زیرا ماژول‌های تأیید امنیتی فقط در اتصالات ماژول اضافه می‌شدند، فاقد تجزیه و تحلیل از محل تأیید امنیت داخلی بودند و کد اصلی SystemC باید تغییر زیادی می‌کرد. با ترکیب فرآیندهای تأیید امنیتی در ماژول‌ها و اتصالات بدون

تغییر بلوک‌های عملکردی، روش پیشنهادی یکپارچه‌سازی یکپارچه را امکان‌پذیر می‌کند که هیچ گونه اختلالی در کد اصلی ایجاد نمی‌کند. با رفع آسیب‌پذیری‌های احتمالی در تأیید امنیت داخلی مکان‌های درون ماژول‌ها، این روش امنیت سیستم را بدون نیاز به اصلاح کد بهبود می‌بخشد و فرآیند را ساده می‌کند.

مدل ارائه شده برای طراحی

هدف ایجاد یک مدل نمایش داده شده در زبان SystemC با استفاده از روش برنامه‌نویسی جنبه‌محور است. برای دستیابی به این هدف، فرآیندهای رمزنگاری اجرا شده توسط سیستم رمزنگاری نمایش داده شده به بسیاری از ماژول‌ها تقسیم شدند. شکل ۲ مجموعه‌ای از الگوها و نظریه‌ها نمایش داده شده پیشنهادی را نشان می دهد. نمودار مدل نمایش داده شده SystemC اتصالات متقابل ماژول آن‌ها را با استفاده از AspectC++ و برنامه‌نویسی جنبه‌محور نشان می‌دهد. مدل نمایش داده شده SystemC برنامه‌نویسی جنبه‌محور از ۶ قسمت تشکیل شده است.

شکل ۲. نمودار PRESENT برنامه‌نویسی جنبه‌گرا در SystemC.



ماژول نمایش داده شده (PRESENT module)

این ماژول فرآیندهای رمزگذاری و رمزگشایی PRESENT را انجام می‌دهد. همانطور که در الگوریتم ۱ نمایش داده شده است، این ماژول وظایفی را انجام می‌دهد، کلاس PRESENT عملیاتی را اعلام می‌کند که در نهایت رمزگذاری و رمزگشایی پیام ورودی را انجام می‌دهد.

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

واحد کنترل (Controller Unit)

این مؤلفه همگام سازی کل مدل رمزنگاری PRESENT را امکان‌پذیر می‌کند. این فرآیند وظایف مرتبط را انجام می‌دهد. کلاس Controller_Unit و تابع PRESENT_Controller(…) را که شامل یک ماشین حالت محدود برای اطمینان از همگام سازی بین همه‌ی ماژول‌ها است، اعلام می‌کند.

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

رابط ورودی (Input Interface)

این مؤلفه جریان داده را به دنبال مشخصّاتی که توسط پروتکل ارتباطی تعیین شده است، مجدداً مرتب می‌کند. همانطور که در الگوریتم ۳ نشان داده شده است، ماژول Input_Interface دو تابع را اعلام می‌کند که در آن فرآیند وظایف زیر را انجام می‌دهد: توابع Input_Plaintext(…) و Input_Key(…) را اعلام می‌کند و پیام‌های ورودی را به اندازه‌ی مورد نیاز تقسیم می‌کند (۶۴ بیت برای متن ساده - ۸۰ یا ۱۲۸ بیت برای کلید).

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

رابط خروجی (Output Interface)

این ماژول جریان رمزگذاری شده را به قالب پروتکل ارتباطی بازیابی می‌کند و عملکرد Output_Ciphertext(…) را اعلام می‌کند.

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

طرح تشخیص عیب (FDS)

هدف از توسعه ماژول FDS محافظت از ماژول PRESENT در برابر حملات خطا است. همانطور که در الگوریتم ۵ نشان داده شده است، ماژول FDS دو عملکرد را اعلام می‌کند: Fault_PRESENT_Detection(…) و Fault__PRESENT_Analysis(…). ماژول FDS وظایف زیر را انجام می‌دهد. تمام خطاها را در طول فرآیند رمزگذاری تأیید شده شناسایی می‌کند و نتایج تشخیص خطا را تجزیه و تحلیل می‌کند.

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

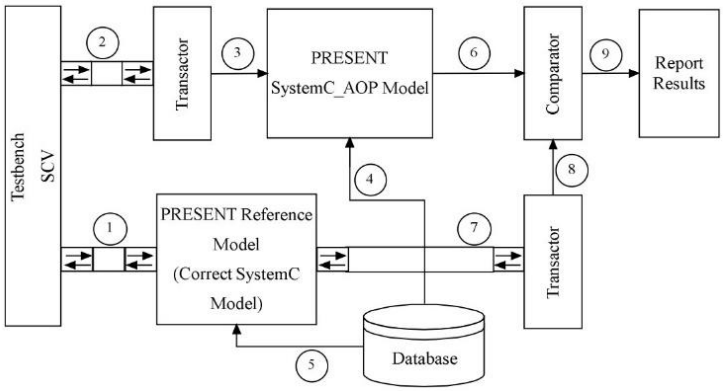
بانک اطلاعاتی (Database)

این ماژول تمام متغیرها و ثابت‌هایی را که در فرآیند رمزگذاری احراز هویت استفاده می‌شود را در خود نگه می‌دارد. کد شامل تعریف دو تابع است: Variables(…) و Constants (…).

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

محیط تأیید عملکردی

محیط تأیید عملکردی پیشنهادی برای مدل PRESENT از محیط تأیید مبتنی بر تراکنش در SystemC استفاده می‌کند. این سیستم دارای یک مدل مرجع است که یک توصیف طراحی جامع را ارائه می‌دهد و همزمان با مدل PRESENT برای ارزیابی نتایج خروجی شبیه سازی اجرا می‌شود. مدل مرجع به جای شبیه‌سازی سیگنال‌ها، تعاملات ماژول را در سطح تراکنشی مدیریّت می‌کند. شکل ۳ تست SCV را نشان می دهد، ماژولی که کلیدهای رمزگذاری و متون را برای مدل‌های مرجع و پیشنهادی به صورت تصادفی تولید می کند. تست SCV سیگنال‌هایی را به ترانسکتور ارسال می‌کند، که سپس آن سیگنال‌ها را به ورودی‌های مدل اعمال می‌کند. یک ماژول مقایسه کننده خروجی‌ها را پس از هر تراکنش مقایسه می‌کند و نتایج را تولید می‌کند.



شکل۳. محیط تأیید عملکردی موجود PRESENT.

- مسیر ۱ و ۲ : تست SCV محرک‌ها را به‌طور تصادفی برای دو مدل رمزنگاری، یعنی مدل‌های پیشنهادی و مدل‌های مرجع تولید می‌کند.
- مسیر ۳ : ماژول تراکنش تراکنش‌ها را به سیگنال‌هایی تبدیل می‌کند که سپس با ورودی های مدل پیشنهادی غیر TLM تطبیق داده می‌شود.
- مسیر ۴ و ۵ : هر دو مدل از پایگاه داده استفاده می‌کنند.

- مسیر ۶ و۸ : ورودی ماژول مقایسه شامل AOP مبتنی بر SystemC و خروجی های ترانسکتور است.
- مسیر ۷ : خروجی TLM مدل مرجع.
- مسیر ۸ : تراکنش‌کننده تراکنش‌ها را به سیگنال‌هایی تبدیل می‌کند که می‌توانند برای مطابقت با خروجی‌های مدل غیر TLM تنظیم شوند.
- مسیر ۹ : نتایج گزارش مقایسه تولید می‌شود.

نتایج و بحث

این بخش بر ارزیابی مجموعه‌ای از الگوها و نظریه‌ها پیشنهادی SystemCAOP و تجزیه و تحلیل اثرات استفاده از SystemC و AspectC++ در ESL بر روی معماری رمزنگاری متمرکز است. زمان شبیه‌سازی و اندازه فایل اجرایی مدل رمزنگاری برای تعیین تأثیر رویکرد AOP بر هر دوی این عوامل ارزیابی شد. فرآیند مدل‌سازی با استفاده از AspectC++ 2.3 و SystemC 2.3.4 انجام شد و مدل PRESENT SystemC AOP با استفاده از پردازنده Intel Core I5-3470 با فرکانس ۳/۲ گیگاهرتز، ۸ گیگابایت رم و کامپایلر gcc ۱۰/۳ تأیید شد.

تجزیه و تحلیل خطا AOP Impact

حملات خطا با استفاده از محیط پیشنهادی برای آزمایش قابلیّت‌های تشخیص محیط PRESENT SystemCAOP اجرا شدند. قابلیت‌های تشخیص خطا با استفاده از دو سناریو SystemC خالص و SystemCAOP ارزیابی شد. علاوه بر این، یک طرح تشخیص خطا برای این ارزیابی استفاده شد [۶]. برای انجام این کار، از هسته های شبیه سازی AspectC++ و SystemC برای ارزیابی و مقایسه نتایج محیط PRESENT SystemCAOP استفاده شد. نتایج شبیه‌سازی، نشان‌داده‌شده در جدول ۱ نشان می‌دهد که هر دو مدل SystemCAOP و SystemC PRESENT دارای قابلیّت‌های تشخیص عیب قابل مقایسه از نظر خطاهای شناسایی‌شده هستند، بنابراین اثربخشی تکنیک SystemCAOP پیشنهادی را تأیید می‌کند. یافته‌ها نشان می‌دهد که ترکیب AOP در چارچوب SystemC، قابلیّت‌های تشخیص مدل‌های طراحی را تضعیف نمی‌کند.

جدول۱. قابلیّت تشخیص SYSTEMC و SYSTEMCAOP

رمز بلوکی PRESENT	قابلیّت تشخیص عیب تصادفی	
	SystemC	SystemC_AOP
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری استاندارد)	٪ ۹۹/۹۹۷	٪ ۹۹/۹۹۷
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری ارائه شده در [۱۷])	٪ ۹۹/۹۹۹	٪ ۹۹/۹۹۹

تأثیر AOP بر شبیه سازی

برای تعیین تأثیر AOP بر طول مدت شبیه سازی، برخی شبیه سازی‌ها با محیط تو سعه یافته انجام شد. همچنین uTime (User Time) مدّت زمانی که شبیه‌سازی در سطح کاربر کد C/C++ و ماژول‌های SystemC مصرف می‌کند و kTime (Kernel Time) مدّت زمانی که در سطح هسته سیستم‌عامل فراخوانی‌های سیستمی، مدیریّت رویدادها، I/O صرف می شود، که بـ سیار کاربردی در تـ شخیص نقص و شنا سایی کُندی یا ناکارآمدی شبیه‌سازی ناشی از کد کاربر خواهد بود. دو طرح تشخیص نقص برای تعیین زمان هسته (kTime) و زمان کاربر

(uTime) در طول این روش استفاده شد. لازم به ذکر است که kTime و uTime مشروط به مقدار Join Point و مقدار ماژول‌هایی هستند که با استفاده از تکنیک AOP درج می شوند. جدول ۲ نتایج شبیه سازی‌های مورد استفاده برای تعیین kTime و uTime را در دو سناریو برای SystemC و SystemC_AOP مختلف نشان می‌دهد.

جدول۲. زمان شبیه سازی SYSTEMC و SYSTEMCAOP

رمز بلوکی PRESENT	kTime (s)		uTime (s)	
	SystemC	SystemC_AOP	SystemC	SystemC_AOP
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری استاندارد)	۰/۰۲۶	۰/۰۲۵	۱/۴۳۲	۱/۴۳۳
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری ارائه شده در [۱۷])	۰/۰۲۳	۰/۰۲۴	۱/۱۲۳	۱/۱۲۶

حاشیه‌ی خطای مرتبط با روش اندازه‌گیری به کار گرفته شده (دستورات لینوکس) غیرقابل انکار است و به این ترتیب، دقّت نتایج شبیه‌سازی را تحت تأثیر قرار نمی‌دهد. نتایج نشان می‌دهد که استفاده از رویکرد AOP برای ادغام ماژول‌های Input_Interface، Database، Controller_Unit، FDS و Output_Interface در مدل SystemC هیچ اثر قابل‌توجهی بر زمان‌های اجرای شبیه‌سازی ندارد. این نشان می دهد که پیچیدگی اضافی معرفی شده توسط رویکرد AOP به طور قابل توجهی بر عملکرد شبیه‌سازی تأثیر نمی‌گذارد. بنابراین، می‌توان نتیجه گرفت که ادغام AOP در SystemC تأثیر قابل توجهی بر kTime و uTime در این مطالعه خاص ندارد. در یک برنامه‌نویسی جنبه‌محوری که باید در هنگام ارزیابی تأثیر AOP در نظر گرفت، اندازه فایل اجرایی به دست آمده است. جدول ۳ اندازه فایل‌های اجرایی تولید شده توسط AspectC++ و هسته‌ی SystemC را در دو موقعیّت سناریو نشان می‌دهد.

جدول۳. تأثیر اندازه فایل اجرایی SYSTEMC و SYSTEMCAOP

رمز بلوکی PRESENT	SystemC	SystemC_AOP
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری استاندارد)	۰/۴۶۵MB	۰/۴۶۳MB
مدل رمز بلوکی موجود توسط FDS محافظت شده است. (معماری ارائه شده در [۱۷])	۰/۴۶۸MB	۰/۴۶۷MB

نتایج شبیه‌سازی نشان می‌دهد که با وجود استفاده از دو زبان برنامه‌نویسی (SystemC و AspectC++) برای مدل‌سازی مدل پیشنهادی و استفاده از تکنیک AOP برای ترکیب همه ماژول‌ها، اندازه‌ی فایل‌های اجرایی تقریباً بدون تغییر باقی می‌مانند. این نشان می دهد که تنظیم AOP به طور قابل توجهی بر اندازه فایل اجرایی تأثیر نمی‌گذارد. علاوه بر این، نتایج نشان می‌دهد که تکنیک AOP به طور موثر نگرانی‌های مقطعی را بدون تأثیر قابل توجهی بر عملکرد کلی سیستم مدیریت می‌کند. این نشان دهنده مزایای بالقوه استفاده از AOP در توسعه‌ی نرم‌افزار برای ماژولارسازی و بهبود

نگهداری کد است. علاوه بر این، نتایج نشان می‌دهد که AOP می‌تواند قابلیت استفاده مجدد و خوانایی کد را با جدا کردن نگرانی‌ها به طور مؤثرتر افزایش دهد. به طور کلی، یافته‌ها از ادغام تکنیک‌های AOP در توسعه نرم‌افزار برای ساده‌سازی فرآیند طراحی و بهبود عملکرد کلی سیستم پشتیبانی می‌کنند. برای سنجش اثر FDS بر یک رمز بلوکی باید هم بُعد امنیّت (تشخیص خطا/حمله) و هم بُعد عملکرد (زمان/توان‌گذردهی) را جداگانه و با آزمون‌های توان‌گذردهی، نرخ موفقیت حمله تزریق خطا، نرخ تشخیص خطا و کیفیّت بیت‌های خروجی قابل اندازه‌گیری خواهند بود کارایی که زمان اجرا، توان‌گذردهی (Mb/s)، سربار حافظه، اندازه فایل اجرای، امنیّت در برابر نقص/حمله، نرخ تشخیص خطا (TPR)، نرخ هشدار کاذب (FPR)، احتمال موفقیّت حمله تزریق خطا و کیفیّت

خروجی توزیع بیت‌های خروجی، یکنواختی و استقلال (برای تضمین عدم اثر منفی FDS بر صحیح‌بودن رمز) شاخص‌های قابل مقایسه خواهند بود.

کارایی (زمان اجرا):

```
H_0:\ \mu_{\text{time,FDS}} = \mu_{\text{time,NoFDS}}
H_1:\ \mu_{\text{time,FDS}} \neq \mu_{\text{time,NoFDS}}
```

توان‌گذردهی:

```
H_0:\ \mu_{\text{throughput,FDS}} =
\mu_{\text{throughput,NoFDS}}
H_1:\ \mu_{\text{throughput,FDS}} \neq
\mu_{\text{throughput,NoFDS}}
```

نرخ موفقیّت حمله تزریق خطا:

```
H_0:\ p_{\text{attack,FDS}} = p_{\text{attack,NoFDS}}
H_1:\ p_{\text{attack,FDS}} < p_{\text{attack,NoFDS}}
```

(FDS آزمون یک‌سویه به نفع کاهش موفقیت با)

نرخ تشخیص خطا:

```
H_0:\ p_{\text{detect,FDS}} \le p_{\text{detect,NoFDS}}
H_1:\ p_{\text{detect,FDS}} > p_{\text{detect,NoFDS}}
```

نتیجه‌گیری

این مطالعه یک مدل SystemC را برای سطح سیستم الکترونیکی ارائه می‌کند که بر اساس برنامه‌نویسی جنبه‌گرا (AOP) است. یک محیط تأیید عملکردی برای تأیید عملکرد مدل، زمان شبیه سازی و اندازه فایل‌های اجرایی توصیه می‌شود. محیط تأیید شامل میزهای آزمایش، مولدهای محرک و مانیتورینگ پوشش برای اطمینان از دقّت و جامعیت مدل است. علاوه بر این، محیط تأیید تشخیص عیوب طراحی احتمالی را تسهیل می‌کند و به عیب‌یابی مدل کمک می‌کند. علاوه بر این، یک چارچوب جامع برای ارزیابی عملکرد و قابلیت اطمینان مدل PRESENT SystemC در شرایط مختلف ارائه می‌دهد. نتایج شبیه‌سازی نشان می‌دهد که کارایی مدل AspectC++ و بافت ماژول PRESENT حداقل تأثیری بر زمان شبیه‌سازی و اندازه‌ی فایل اجرایی دارد.

علاوه بر این، مدل پیشنهادی برنامه‌ی جنبه‌گرا PRESENT SystemC در مقایسه با روش‌های مر سوم، مدولار بودن و قابلیّت نگهداری بیشتری را نشان می‌دهد. علاوه بر این، تکنیک‌های پیشنهادی برای تزریق و تشخیص خطاها در ارزیابی انعطاف‌پذیری طرح‌های رمزنگاری در برابر حملات خطا قابل دوام و کارآمد هستند. زمان شبیه‌سازی حداقل تحت تأثیر AOP قرار می‌گیرد، که سودمندی خود را در ارزیابی حوزه‌های امنیتی با به حداقل رساندن تلاش‌ها و

خطاها ثابت می‌کند. علاوه بر این، رویکرد AOP سفارشی‌سازی و اصلاح مکانیسم تزریق/تشخیص خطا را برای انطباق با سیستم‌های رمزنگاری منحصربه‌فرد تسهیل می‌کند. انعطاف‌پذیری این روش امکان استفاده از آن را در شرایط مختلف امنیتی فراهم می‌کند و در نتیجه کارایی آن را در ارزیابی قدرت طرح‌های رمزنگاری افزایش می‌دهد. به طور خلاصه، استفاده از AOP در مدل‌سازی SystemC پتانسیل بهبود سازگاری طراحی و بهره‌وری در توسعه سطح سیستم الکترونیکی را دارد. از محدودیت‌ها برنامه‌نویسی جنبه‌محور (AOP) در این حوزه می‌توان به عدم پشتیبانی بومی در SystemC، پیچیدگی در نگهداری کد، تداخل جنبه‌ها با سنتز سخت‌افزاری و ابزارهای محدود اشاره کرد، همچنین ترکیب SystemC، HLS و AOP در شبیه‌سازی، ابهام در سطح انتزاع، قابلیت اطمینان و درستی‌سنجی و پشتیبانی ابزارهای HLS چالش‌های پیش‌رو می‌توان متصوّر شد.

فهرست منابع

[1] H. Mestiri and I. Barraj, "High-Speed Hardware Architecture Based on Error Detection for KECCAK," *Micromachines*,14,6, Jun. 2023, Art.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 RecentAdvances and Opportunities," Micromachines,14, 6, jun, 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,70, 2, 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*, 22, Jul. 2023, Art.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*, 9,72568–72586, 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*,25,12, 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. 2019,1–6 https://doi.org/10.1109/ DTSS.2019.8914979

[8] V. A. Thakor, M. A. Razzaque, and M. R. A. Khandaker, "LightweightCryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities," *IEEE Access*, 9, 28177–28193, 2021, https://ieeexplore.ieee.org/document/9328432

[9] T. K. Goyal, V. Sahula, and D. Kumawat, "Energy Efficient Lightweight Cryptography Algorithms for IoT Devices," *IETE Journal of Research*, 68, 3, 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*, 73, 2, 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, ardware/Software Co-Design of Cryptographic SoC Based on RISCV Virtual Prototype," *IEEE Transactions on Circuits and Systems II: Express Briefs*,70, 9, 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*,1, 2, 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*, 16, 3, Mar. 2024, Art. 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*, 55, 4, Aug. 2022, Art. 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*, 7, 2, 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*, 7, 9, 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. https:/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. 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. 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. 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, 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*, 11, 1,6719–6723, 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*, 12, 18, Jan. 2023, Art. 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*, 12, 4, Dec. 2022, Art. 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-onchip design,"*Microprocessors and Microsystems*, 87, Nov.2021, Art.104363. https://doi.org/10.1016/j.micpro.2021.104363.