Towards a Methodology to Design Provably Secure Cyber-physical Systems

Felipe Lisboa Malaquias, Georgios Giantamidis, Stylianos Basagiannis
Collins Aerospace, Applied Research and Technology Centre, Ireland; email: {firstname.lastname}@collins.com
Simone Fulvio Rollini
Collins Aerospace, Applied Research and Technology Centre, Italy; email: simonefulvio.rollini@collins.com
Isaac Amundson
Collins Aerospace, Applied Research and Technology Centre, United States; email: isaac.amundson@collins.com

Abstract

The inordinate financial cost of mitigating post-production cybersecurity vulnerabilities in cyber-physical systems (CPS) is forcing the industry to rethink systems design cycles: greater attention is being given to the design phase – with the goal of reducing the attack surface of systems at an early stage (i.e., before silicon tape out). Fortunately, formal methods have advanced to the point that they can address such needs and contribute towards achieving security certification. However, new methods and tools focusing on industrial scalability and usability for systems engineers are required. In this ongoing research paper, we describe a framework that will help systems engineers to: a) design cyber-assured CPS using a Model Based Engineering (MBE) approach; b) formally map security requirements to different hardware and software blocks in the model; and c) formally verify security requirements. Based on the nature of each requirement, our framework collects formal correctness evidence from different tools: while high-level architectural properties are suitable for a contract- or ontology-based reasoning, more complex properties with rich semantics require the use of model checking or theorem proving techniques.

Keywords: Formal Methods, Cybersecurity, Cyber-Physical Systems, Model Checking, Theorem Proving.

1 Introduction

In 2020, researchers estimated that there were 12 billion active *Internet of Things* (IoT) devices, and that number would at least double within five years [1]. The attack surface for IoT devices is large, and an attacker may use several attack vectors to compromise a device, ranging from physical side-channel attacks to programming bugs like buffer and arithmetic overflows [2].

This document contains no technical information subject to EAR or ITAR. Work is funded under the REWIRE Horizon Europe project (REWiring the composItional security veRification and assurancE of systems) with grant agreement No. 101070627.

Considering that many of these IoT devices are used in critical *Cyber-Physical Systems* (CPS), security breaches can lead to seriously hazardous outcomes, both in terms of human life and financial loss. A well-publicised example of an IoT security breach is the remote hijacking of a Jeep on a U.S. highway [3]: white hat hackers were not only able to manipulate non-critical systems (e.g. display, air conditioning), but they were also able to control the engine and brakes.

Given the high stakes, guaranteeing that these devices correctly implement the appropriate security countermeasures is crucial. Furthermore, in order to achieve a high degree of trust regarding security claims, engineering teams must go beyond dynamic testing of software and hardware components and tackle the problem with *formal methods* tools.

While in the past formal methods encountered a strenuous barrier regarding industry adoption, theoretical advances led by academia, software engineering best practices, and the exponential growth of computational power have all contributed to reaching the current state-of-the-art: tools are now sophisticated enough to abstract from mathematical theories and provide user-friendly interfaces for systems engineers. This is highly beneficial, given the fast-paced needs of the industry.

Although recent research has successfully proposed frameworks for formal reasoning about cybersecurity in CPS, no comprehensive framework exists for modelling and formally verifying general-purpose CPS, such as IoT devices. Therefore, we propose a framework that will allow engineers to write requirements (not only security-related but also safety, timing, and functional requirements), design system architectures, and gather, on the same model, formal evidence that the stated requirements are satisfied either by the architectural model or by specific component implementations.

2 Related Work & Background

2.1. Model-Based Engineering & AADL

Model-Based Engineering (MBE) has emerged as a key set of methodologies to design complex systems [4]. One widely-adopted MBE technology is the *Architecture Analysis & Design*

Language (AADL) [5]. Initially developed for avionics applications, AADL has since been used to design a wide range of embedded real-time system architectures, largely due to its language constructs for specifying both software and hardware configurations. Moreover, AADL has a reference implementation called OSATE [6], which is an open-source modelling environment that comes with a few built-in analysis tools such as flow control and schedulability. Because OSATE is based on the Eclipse framework, creating new analysis plugins is relatively straightforward.

AADL includes an *annex* mechanism for extending the base grammar, thereby supporting new language features and analyses. One such annex is the *Assume Guarantee REasoning Environment* (AGREE) [7], which is a compositional assume-guarantee-style formal analysis tool. AGREE attempts to prove properties about one layer of the architecture using properties allocated to subcomponents. The composition is performed in terms of formal assume-guarantee contracts that are provided for each component. Assumptions describe the expectations the component has on its inputs and the environment, while guarantees describe bounds on the component's behaviour. The model checker then attempts to find any model execution traces that violate these contracts using one of several Satisfiability Modulo Theories (SMT) solvers. If the model checker covers all reachable states in the model without finding a violation, the model is proven to satisfy its contracts.

Another important annex for reasoning over AADL models is Resolute [8], which includes a language for embedding assurance cases in AADL models and a tool for evaluating the validity of the associated evidence. An assurance case is a structured argument, supported by evidence that a system will operate as intended in a specified environment. Because high-assurance products generally undergo certification at the system level, there is a natural mapping between a system design and the corresponding assurance argument. Resolute takes advantage of this alignment by enabling the specification of the assurance argument directly in the AADL model. The assurance case can then be instantiated and evaluated with elements specified in the model. The resulting assurance case can be viewed in the modelling environment, or exported to graphical tools such as AdvoCATE [9]. Resolute assurance cases are at the core of our approach, and we describe them in greater detail in Section 3.

Our choice of using AADL over other MBE languages such as SysML [10] is informed by multiple factors. First, AADL was designed for specifying *hierarchical* system architectures, enabling the composition of systems from subsystems, and refinement from abstract to concrete types. It includes first class objects for representing components that comprise embedded systems such as memory, buses, processors, threads, subprograms, and data. SysML, on the other hand, is more abstract and thus better-suited for early stages of system engineering. Second, AADL has a sufficiently rigorous run-time semantics, enabling a wide range of analyses that would otherwise not be possible. And third, AADL's annex support cannot be overstated. The ability to extend the language in order to perform new types of analyses is critical in the rapidly evolving – and heavily regulated – CPS design space.

2.2. Cyber-Assured Systems Engineering Framework

Cofer et al. recently developed BriefCASE [11], an AADL-based framework for designing, building and assuring cyber-resilient systems. In that work, high-level security requirements are mapped to seL4 microkernel [12] features via a (very) trustworthy tool chain. Although they did succeed at creating a framework for crafting formally verified secure applications, their work did not focus heavily on hardware security, which plays a fundamental role in protecting a wide range of CPS including IoT devices. In contrast, we propose a framework that allows system engineers to specify a wide range of system requirements and map them to the appropriate software or hardware block.

For example, one might require a platform capable of performing *trusted boot* to verify the authenticity of an over-the-air firmware update, or a platform capable of executing hardware-implemented *crypto-primitives* (e.g., symmetric or asymmetric encryption, hash functions, etc.). While the former security goal could be achieved through the use of a hardware *Root of Trust* (RoT) acting as the *Trusted Platform Module* (TPM), the latter would require *Instruction Set Extensions* (ISEs) or memory-mapped crypto-accelerators. These solutions are outside the scope of what BriefCASE currently offers.

Another important work that introduces a tool aimed at formal reasoning about CPS is KeYmaera X [13], a theorem prover for differential dynamic logic (dL). KeYmaera X introduces important advances in formal verification of CPS – its logic is well suited for reasoning about discrete and continuous dynamics, which are useful to encode functional and safety properties of CPS. It has been used, for instance, to model and verify the safety of flight collision avoidance software [14] and train controls with air pressure brakes [15]. In comparison, our work focuses instead on characterising software/hardware running in CPS and its underlying cybersecurity properties.

Finally, VRASED [16] is a HW/SW co-design that implements a formally verified Remote Attestation protocol. To achive that ultimate goal, VRASED relies on different formal verification techniques: a custom hardware module (used to reset the internal state of the micro-controller studied in their paper in case the code to be attested is compromised) has its correctness specified with Linear Temporal Logic (LTL) and checked with NuSMV – a model checker; the overall soundness and correctness of the resulting system is modelled and proved in a theorem prover; and finally, they also make use of pre-verified cryptographic code [17]. While VRASED is an impressive effort, their architecture is bare-metal and specific to a family of micro-controllers, such as MSP430. The goal of our work, contrarily, is to leverage MBE tools in conjunction with formal verification techniques to formally reason about more generic, richer architectures (e.g. with RISC-V cores and Trusted Execution Environments (TEE)).

We thus propose building a framework that: a) provides the tools to formally reason about security solutions implemented by both software and hardware; and b) maps security requirements to evidence gathered not only from AADL and its plugins, but also external tools, such as Coq.

3 Framework Description

Figure 1 presents an abstract architectural overview of the toolchain, under development at the time of writing. Note that, while some of the tools are embedded within OSATE, some are called by Resolute as an external source of formal correctness evidence.

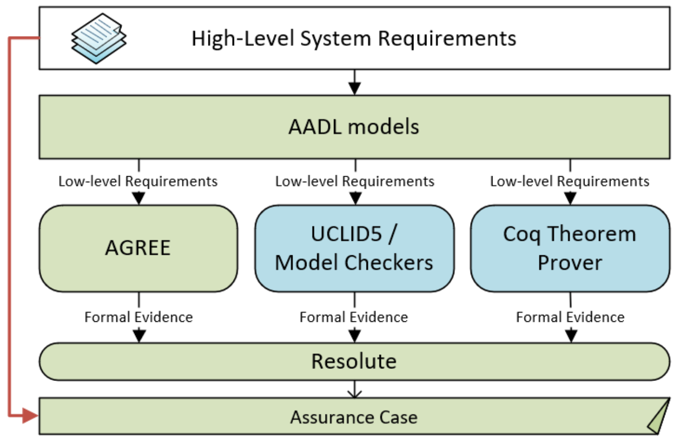

Figure 1: Toolchain. Legend: 🟩 OSATE tools 🟦 External tools

The framework we propose is intended to be used according to the following workflow:

1. Security requirements are specified;
2. System architecture is modelled in AADL;
3. Formal analysis of model is performed using AGREE to verify the design satisfies security properties;
4. Hardware and Software components are implemented manually, or through verified synthesis [18];
5. Where possible, formal analysis is performed on component implementations – this could be done by a variety methods (e.g., model checking and theorem proving);
6. Component implementations are integrated into a system build;
7. System testing is performed;
8. An assurance case is generated using Resolute, confirming that security goals are support by evidence (maintained by the framework).

In the following, we detail some of the steps presented above.

Requirement Specification (Step 1)

Consider three illustrative high-level security requirements, which are motivated by industrial use-case scenarios for embedded systems controlling safety-critical operations of CPS:

R1: "The TEE shall provide the necessary mechanisms to enable the isolated execution of sensitive functions in enclaves"

R2: "The crypto schemes that will be used for the secure communication of devices shall be provably secure, based on well-accepted underlying assumptions"

R3: "The system shall include mechanisms that detect replay attacks and can tell if newer messages or part of them are unauthorised repetitions of previously authorised exchanges"

Architecture Modelling (Step 2)

Based on the security requirements from Step 1 (as well as other high-level requirements), engineers use AADL to model a system architecture that captures the appropriate security solutions. Here, for illustrative purposes, consider the following architecture:

- *On the hardware layer*: a RISC-V core, such as the 2-stage pipeline 32-bit Ibex core[1] or the 6-state 64-bit Arianne core[2], and a hardware RoT (such as the OpenTitan[3]), which is assumed to have a crypto co-processor capable of enhancing the performance of functions such as AES and SHA-256.
- *On the firmware layer*: a custom-tailored Keystone TEE [19], configured to use not only the RISC-V Physical Memory Protection (PMP) Registers – primitives for ensuring memory isolation between secure enclaves – but also custom RISC-V instructions to access crypto co-processors and the RoT;
- *On the software layer*: a set of secure applications (Keystone enclave application), such as attestation agents, evidence collectors, and SW/FW update agents.

Figure 2 shows an abstraction of the proposed architecture.[4]

Modelling software and hardware components in AADL is a straightforward and well-documented process: software is described using components such as *processes* and *threads*, while hardware components include *processors*, *buses*, and *memories*. Modelling the TEE and its properties in AADL however is still an open problem and we consider it to be part of our research.

[1]https://ibex-core.readthedocs.io/en/latest/
[2]https://github.com/lowRISC/ariane
[3]https://opentitan.org/
[4]The architecture used as example here is similar, in some sense, to Keystone's architecture [19], although not entirely. Keystone, in this case, would have to be configured by the *Keystone Programmers* [19] to use the custom proposed hardware.

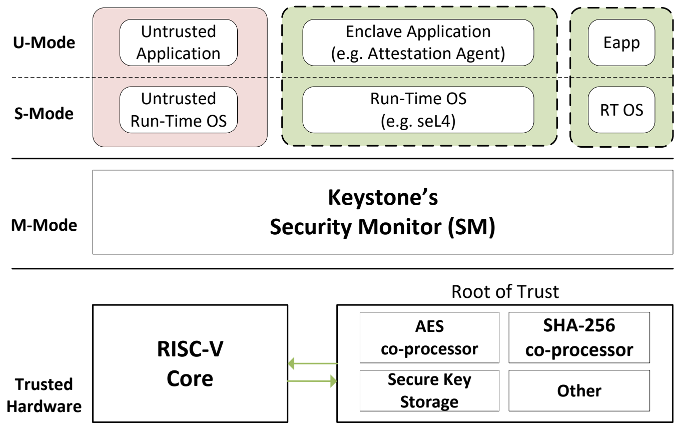

Figure 2: Abstraction of the example architecture.

Note that the architecture proposed above is merely illustrative and its security implications are not the focus of this paper. Rather, here our focus is on providing an overview of a methodology to formally verify a diverse range of security requirements.

Translation of Higher-Level Requirements into Low-Level Specifications (Step 5)

In order to determine whether individual component implementations satisfy their requirements, a necessary prerequisite is the translation of those requirements into a formal representation that facilitates (semi-)automated, computer-aided analysis/reasoning.

For low-level functional requirements characterising component behaviour, this representation can, for example, be a form of modal logic, such as *Linear Temporal Logic* (LTL) or *Computation Tree Logic* (CTL) [20], or *differential dynamic logic* (dL) [21].

For intermediate-level non-functional requirements, approaches for describing constraints on system architecture / component interconnections (or other desired properties of a non-functional nature) can vary. For example, in cases where properties are quite abstract and their satisfaction is difficult to define, so called soft-goal approaches are more appropriate [22], while in cases where properties are more concrete, a (constraint) logic-based approach is preferred [23]. Requirements *R1*, *R2*, and *R3* fall under the latter category. *R1*, for instance, can be formalised in first-order logic as follows:

$$\begin{aligned}&\forall e \in Enclaves,\\ &\forall f \in Functions, sensitive(f) \implies\\ &allocated\_on(f,e) \implies\\ &ensured\_isolated\_execution(f,e)\end{aligned} \tag{1}$$

In natural language, *R1* is said to be satisfied if any arbitrary sensitive function f is allocated inside of a secure enclave e.

Development of automated requirement formalisation tools is out of scope for this work. Furthermore, it will likely be the case that not all requirements can be formalised in a manner that permits automated reasoning. For now, we assume that translating high-level requirements into low-level specifications is a manual task to be performed by the framework user and verified through manual review.

Formal Verification of Requirements (Step 5) and System Assurance (Step 8)

Here, we detail how formal analysis can be performed (Step 5) considering requirements *R1*, *R2*, and *R3*; and how we can use Resolute to generate assurance cases, which can be used to provide confidence that cybersecurity requirements have been satisfied in both the design and implementation.

Requirement *R1* can be checked using Resolute, since it is structural in nature. We can use Resolute to "traverse" the model and check that a component implementing the desired functionality is present, cannot be bypassed, and has been implemented appropriately. In the proposed architecture, Keystone's Secure Monitor (SM) [19] implements enclave isolation by manipulating RISC-V *Physical Memory Protection* (PMP) registers, and thus, "*isolated execution of sensitive functions in enclaves*" is achieved.

Listing 1 is a first attempt at writing an assurance argument for requirement *R1* in Resolute. In the listing, `SW.Impl` is a rather simplified representation of a software process hosted in a system equipped with Keystone, where `Eapp` is a sensitive function. The Resolute goal `Iso_Exec` traverses the model to check that: 1) the enclave where `Eapp` executes is properly implemented and initialised, 2) `Eapp` executes on the enclave, and 3) Applications on U-Mode or S-mode cannot bypass the Security Monitor (see Figure 2). We omit the connections between threads for conciseness.

Listing 1: Verifying *R1* with Resolute.

```
process implementation SW.Impl
    subcomponents
        Eapp : thread Eapp.Impl;
        RT : thread RT.Impl;
        Enclave : thread Enclave.Impl;
        SM : thread SM.Impl;
    annex resolute {**
        argue Iso_Exec (this.Eapp, this.Enclave,
            this.SM)
    **};
end SW.Impl;

goal Iso_Exec (eapp : component, encl :
     component, sm : component) <=
    strategy: "Reason about architecture";
    enclave_exists(encl) and
    enclave_initialized(encl) and
    allocated_on(eapp, encl) and
    sm_not_bypassed(sm, encl)
```

The second requirement, *R2*, is more difficult. Informally, "provably secure" is naturally more complex than "shall provide". Here, for simplicity, let us first assume that the "*crypto-schemes that will be used for the secure communication of devices*" can be simply reduced to AES and SHA-256, as these are the co-processors specified in the architecture. Realistically, this is a strong assumption, since a real device would also require asymmetric encryption schemes, which are mostly implemented in software.

We can assume that the cryptographic algorithms themselves (AES and SHA-256) are secure by design and focus our efforts in proving that the actual hardware implementations are correct against a high-level formal specification of these algorithms. For that goal, considering that the architecture proposes the use of hardware co-processors, we could either: a) perform typical verification techniques, such as property checking with SystemVerilog Assertions (SVA) on existing HW IPs or b) produce correct-by-design *Register Transfer Level* (RTL) code using Coq *Domain Specific Languages* (DSLs), such as Kôika [24]. At this phase of our research, we explore the second option, as depicted in Listing 2. Notably, we prove that the hardware implementation of the RISC-V standardised crypto custom instructions [25] is correct against the instruction semantics, expressed in SAIL [26].

Listing 2: Verifying *R2* with Resolute.

```
system implementation HW.Impl
    subcomponents
        AES : processor AES.Impl;
        SHA_256 : processor SHA_256.Impl;
    annex resolute {**
        argue Correct_By_Design (this)
    **};
end HW.Impl;

goal Correct_By_Design (sys : system) <=
    ** "RTL code is provably correct" **
    forall(proc : processors(sys)) .
        analysis("coq", proc)
```

Finally, *R3* requires that the architecture includes mechanisms for detecting intruder operations such as message replay attacks. In such an attack, a malicious agent intercepts a message and/or controls its delivery to the intended target, thereby disrupting system operations or obtaining unauthorised information. If a protection mechanism is enabled (e.g., by including timestamps), then a model checker could explore whether or not a successful replay-attack state is reachable in which the protection would fail to detect the repetition of the message within a specific time threshold.

Devices that are part of communication networks are commonly modelled together with an intruder model (e.g., Dolev-Yao [27]) that can perform attack operations against eavesdropped messages. Model checking using SPIN [28] or OFMC [29] can provide evidence that proves the absence of a series of such attacks. Recent approaches also involve the usage of TAMARIN [30] in an attempt to verify cryptographic protocols using adversaries within the tool itself. In the case of *R3*, since the requirement can be directly modelled in the language of a model checker, the Resolute assurance argument can take the form of a simple predicate: $safe\_against\_replay\_attacks()$, supported by evidence generated by the model checking tool.

4 Conclusion & Next Steps

We propose a framework aimed at modelling and formally verifying cyber-assured CPS under the following design principles: a) security requirements are allocated to either system software or hardware, b) the system is modelled in a language (e.g., AADL) with sufficiently rich semantics that enable formal analysis, c) formal methods are applied at multiple points in the development workflow (compositional reasoning at the architecture level, verified synthesis for component generation, model checking and theorem proving of hardware and software component implementations, etc.), and c) an assurance case is generated that substantiates cybersecurity claims with evidence from formal analyses (and other workflow processes managed by the framework).

Currently, we focus our efforts on modelling Keystone and custom hardware in AADL – an open problem – since AADL has not been previously used to model TEEs. A subsequent challenge is how to scale the approach to model larger systems.

References

[1] K. L. Lueth, "State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time," *IoT Analytics*, Nov 2021.

[2] K. Keerthi, I. Roy, A. Hazra, and C. Rebeiro, "Formal verification for security in IoT devices," *Security and Fault Tolerance in Internet of Things*, pp. 179–200, 2019.

[3] A. Greenberg, "Hackers remotely kill a Jeep on the highway-with me in it," *Wired*, Jul 2015.

[4] P. de Saqui-Sannes and J. Hugues, "Combining SysML and AADL for the design, validation and implementation of critical systems," in *ERTS2 2012*, p. 117, 2012.

[5] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (AADL): An introduction," tech. rep., Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[6] P. Feiler, "The open source AADL tool environment (OSATE)," tech. rep., Carnegie Mellon University Software Engineering Institute, 2019.

[7] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods: 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings 4*, pp. 126–140, Springer, 2012.

[8] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 19–28, 2014.

[9] E. Denney and G. Pai, "Tool support for assurance case development," *Automated Software Engineering*, vol. 25, September 2018.

[10] M. Hause *et al.*, "The SysML modelling language," in *Fifteenth European Systems Engineering Conference*, vol. 9, pp. 1–12, 2006.

[11] D. Cofer, I. Amundson, J. Babar, D. Hardin, K. Slind, P. Alexander, J. Hatcliff, G. Klein, C. Lewis, E. Mercer, *et al.*, "Cyberassured systems engineering at scale," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 52–64, 2022.

[12] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: formal verification of an OS kernel," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009* (J. N. Matthews and T. E. Anderson, eds.), pp. 207–220, ACM, 2009.

[13] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE* (A. P. Felty and A. Middeldorp, eds.), vol. 9195 of *LNCS*, pp. 527–538, Springer, 2015.

[14] R. Cleaveland, S. Mitsch, and A. Platzer, "Formally verified next-generation airborne collision avoidance games in ACAS X," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 1, pp. 1–30, 2023.

[15] S. Mitsch, M. Gario, C. J. Budnik, M. Golm, and A. Platzer, "Formal verification of train control with air pressure brakes," in *RSSRail* (A. Fantechi, T. Lecomte, and A. Romanovsky, eds.), vol. 10598 of *LNCS*, pp. 173–191, Springer, 2017.

[16] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "Vrased: A verified hardware/-software co-design for remote attestation.," in *USENIX Security Symposium*, pp. 1429–1446, 2019.

[17] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "Hacl*: A verified modern cryptographic library," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1789–1806, 2017.

[18] E. Mercer, K. Slind, I. Amundson, D. Cofer, J. Babar, and D. Hardin, "Synthesizing verified components for cyber assured systems engineering," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 205–215, 2021.

[19] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.

[20] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pp. 1–30, 2012.

[21] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[22] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*, vol. 5. Springer Science & Business Media, 2012.

[23] J.-P. Katoen, T. Noll, H. Wu, T. Santen, and D. Seifert, "Model-based energy optimization of automotive control systems," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 761–766, IEEE, 2013.

[24] T. Bourgeat, C. Pit-Claudel, and A. Chlipala, "The essence of Bluespec: a core language for rule-based hardware design," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 243–257, 2020.

[25] B. Marshall, G. R. Newell, D. Page, M.-J. O. Saarinen, and C. Wolf, "The design of scalar AES instruction set extensions for RISC-V," *Cryptology ePrint Archive*, 2020.

[26] A. Armstrong, T. Bauereiss, B. Campbell, A. Reid, K. E. Gray, R. Norton-Wright, P. Mundkur, M. Wassell, J. French, C. Pulte, *et al.*, "ISA semantics for ARMv8-a, RISC-V, and CHERI-MIPS," 2019.

[27] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[28] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[29] A. A. et al, "The AVISS security protocol analysis tool," in *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, vol. 2404 of *Lecture Notes in Computer Science*, pp. 349–353, Springer, 2002.

[30] D. A. Basin, C. Cremers, J. Dreier, and R. Sasse, "Tamarin: Verification of large-scale, real-world, cryptographic protocols," *IEEE Secur. Priv.*, vol. 20, no. 3, pp. 24–32, 2022.