Securing Silicon: A Scalable, Platform-independent Hardware Security Verification Methodology

Muhammad Abdullah Al Faisal
Chipglobe GmbH
Munich, Germany
al.faisal@chipglobe.com

Jaimini Nagar
Infineon Technology GmbH & Co.
Dresden, Germany
Jaimini.Nagar@infineon.com

Thorsten Dworzak
Infineon Technologies AG
Munich, Germany
Thorsten.Dworzak@infineon.com

Sebastian Simon
Infineon Technology GmbH & Co.
Dresden, Germany
Sebastian.Simon@infineon.com

Ulrich Heinkel
Technical University of Chemnitz
Chemnitz, Germany
Ulrich.heinkel@etit.tu-chemnitz.de

Djones Lettnin
Infineon Technologies AG
Munich, Germany
Djones.Lettnin@infineon.com

***Abstract*— Modern System-on-Chips (SoCs) are vulnerable due to micro architectural weakness in Register Transfer Level (RTL) implementation, having significant security risk to the sensitive design asset. Various techniques like Formal-based verification, Fuzzing and Information Flow Tracking have been proposed to accomplish the hardware security verification. Unfortunately, these techniques are not yet sufficiently developed to address the full range of potential weaknesses present in digital SoC designs. In this paper, we propose a novel and scalable hardware security verification methodology that formalize the security requirements, create use case scenarios and a comprehensive set of meaningful tests using the portable test and stimulus standard (PSS). The result shows that our proposed methodology could detect hardware weakness of the SoC design and incorporates stimuli coverage closure for quantitative assessment of test intent.**

Keywords— Hardware Security Verification, Portable Test and Stimulus Standard, OpenTitan, Common Weakness Enumeration

I. Introduction

Nowadays SoC designs have become more complex due to their heterogeneous architecture that includes microprocessors, embedded software, on-chip memory hierarchies, hardware accelerators, I/O, and security controllers. Trustworthiness and hardware security are being very important of the modern SoCs that are designed for security-critical application, automotive, mobile, and service robots. Hardware vulnerabilities and weaknesses of the design can be exploited by attackers, e.g. stealing or manipulation of cryptographic key by observing control and status registers. This can result into harsh consequences such as damaging system's behavior completely and endanger human lives. Therefore, it is very crucial to verify the security-critical functionality of the hardware design and to detect weaknesses and security vulnerability. At present, various methods for functional verification are being used to tackle security concerns in the design involving manual review of design, formal verification, simulation, and emulation. Model checking uses a mathematical model of a design under verification and explores all possible behaviors of the design. Recently, a comprehensive use of model checking based formal verification method to verify the security-critical features of a processor was presented [1]. However, code reviews and formal verification methods involve significant manual effort and scalability is also challenging with increasing design size and complexity. Information flow of the security critical signals can be verified by formalizing security requirements as taint-propagation properties [2]. A security verification framework has been developed to verify security properties which are generated using an information flow tracking template [3], [4]. The information flow tracking approach is very promising to find information leakage, but it is overly conservative and could inaccurately report the existence of flow in certain cases (i.e. false positive). A security verification method must be efficient to find the hardware weakness and should be scalable to the SoC level. It is intriguing to verify the design against security properties like confidentiality, integrity, and availability. Common Weakness Enumeration (CWE) is a list of common hardware weakness that could have security ramification developed by MITRE [6]. It is very crucial to find such weaknesses of design at early stage during verification because hardware weaknesses could lead to vulnerability and cause security breach.

The proposed hardware security verification methodology outlines a strategy for establishing a comprehensive security verification plan and leverage Portable Test and Stimulus Standard (PSS) to enhance reusability and scalability. The prime purpose of the PSS is to define a domain-specific language (DSL) that is declarative in nature for specifying verification intent [7]. It enables engineers to create a unified representation of stimulus and test scenarios, usable across various integration levels and configurations. We have demonstrated the proposed methodology on an open source SoC design, where it successfully uncovered a security vulnerability. Furthermore, we have ensured the effectiveness of this methodology by detecting security flaws that were deliberately introduced in the design.

II. Related Work

Security verification of the hardware design has become an important phase of the current SoC design due to various security threats and hardware attacks at different stages of the development cycle. A detailed analysis of hardware security properties, hardware security threats and countermeasures is presented in [5]. It explains new challenges of design verification and opportunities of integrating an additional dimension of security into robust hardware design and verification.

Wang et al. discussed the application of the PSS to verify the security features of RISC-V based SoC, the granular control of permissions across multiple physical memory regions provided by the Physical Memory Protection (PMP) unit in [8]. Their study demonstrates the generation of a substantial volume of tests using PSS model, which encompass both positive and negative security test scenarios. While the study provides insights into the verification of security features, it does not specifically address the application for uncovering common hardware weaknesses or vulnerabilities in micro architectural implementations. Hardware weaknesses could lead to security risk, and it is very important to identify them early during verification to ensure the trustworthiness of SoC designs. In our proposed methodology, CWEs related to confidentiality and integrity are systematically analyzed to derive a list of security requirements, and the design under test is verified against them. Moreover, the paper does not discuss about the specific strategies for maximizing coverage or handling corner cases effectively. In our work, we have integrated stimuli coverage closure for measurable evaluation of test intent that boosts effectiveness of proposed methodology.

There has been formal verification work to verify RISC-V architecture [1] and OpenRISC-1200 based SoC architecture [9]. Security analysis of the SoC using an assertion based formal verification method is presented in these works. Security requirements of the hardware design can be formalized as taint-propagation properties prove them using formal verification tool [2], [14]. Security properties of SoC bus implementation have been derived from the bus protocol and prove them using formal

verification tool [13]. Nevertheless, it is challenging to scale up formal verification method for the complex SoC designs and formalized properties could not be ported or reused for other verification platforms. The security requirements have been formulated as a PSS model in our developed security verification methodology that can be reused across various verification platforms i.e., simulation, emulation, and FPGA prototyping. Moreover, a PSS model facilitates generation of meaningful and comprehensive tests for multiple test case scenarios that saves time spent for test writing.

A design and verification framework for SoC access control systems based on the information flow approach was developed in [3], [4]. An automation framework to generate the security properties based on information flow tracking approach has been presented in [12]. A security verification tool has generated a security model using a security property template based on the information flow tracking approach. The security model was integrated with a testbench for the simulation and to prepare a report for property pass/fail. Researchers have also referred to CWEs to find potential weakness in the design during creating security properties. In this method, a coverage model cannot be formulated for either functional coverage or code coverage. We have included stimuli coverage closure for quantitative assessment of test intents in our proposed security verification approach.

III. Methodology

A. Overview of Methodology

The traditional verification process for digital designs starts with developing a verification plan (vPlan) derived from the specifications and hardware requirements, encompassing all verification items essential for the verification. As depicted in Fig. 1, a security verification plan is additionally incorporated in the verification plan as the starting point for security verification. It entails outlining the security requirements and a security coverage plan to monitor the verification progress on the simulation platform. Once the security vPlan is well-defined, the generation of a PSS model starts that captures security requirements in the declarative domain-specific language. The PSS model encapsulates an abstract behavior of verification test intent of the design, relying on the requirements and target implementation for test realization. It also includes checkers and monitors that are essential for the test validation. The PSS model incorporates coverage definition, a feature that allows for the tracking of engaged stimuli and test scenario space coverage. This feature is a key in pre-analyzing the coverage of generated tests before executing them on verification platforms like simulation. It ensures the time-efficiency and gives confidence that the tests can address intended behaviors of the design under verification. In parallel of the RTL (Register Transfer Level) design implementation by the design team and ensuring the readiness of testbench environment, the generation of tests from the PSS model can commence, aimed at the target platform (simulation or emulation). When the tests are passed successfully, a coverage report is produced on the simulation platform. If the coverage objectives are met, the process advances to the hardware implementation phase, signifying the completion of the security verification process. On the other hand, if a test fails, the design requires alterations to rectify the security flaws or weaknesses. After these modifications, the generated test is run again to confirm that there are no security bugs, or any micro-architectural weaknesses present in the RTL implementation of digital design.

B. Security Verification Plan

The first and crucial step of the proposed methodology is to establish a robust and comprehensive security verification plan as shown in Fig. 1 and it is depicted in detail in Fig. 2. The security verification plan encompasses the meticulous formulation of security requirements, a scalable verification process to check them during the design development process, and a security coverage plan for monitoring the verification progress to ensure security signoff before tape-out. Fig. 2 illustrates a comprehensive flow to achieve the security requirements. It is explained in detail how we can derive a

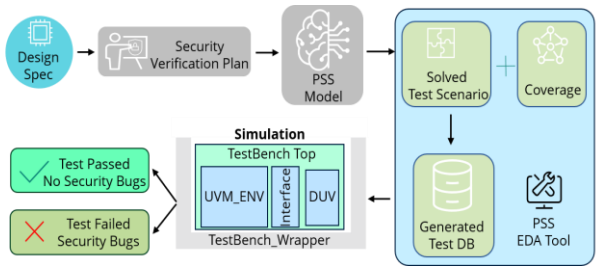

Fig. 1. Overview of Methodology

list of security requirements and the process of reasoning about these requirements in following text.

Design Asset Identification: A design asset can be defined as a resource with a security critical value that is worth protecting from an adversary. An asset may manifest as registers, modules, or ports within a design IP. For an instance, on-device key (Secret/Private key(s) of an encryption algorithm) or protected data (Sensitive user data, meter reading), Device configuration (Service/Resource access configuration) etc.

The choice of security assets varies design by design and for the different abstraction layers. Mainly, the declaration of security assets is heavily dependent on the security policies that will be defined at different levels of integration for different components in SoC.

Threat Model: The next step is to develop the threat model. It is crucial to articulate the relevant security concerns related to every security asset defined at the first step. Hardware threats are vast and must be assessed based on the asset and usage of the hardware under design. The threat model is composed of a set of realistic assumptions and definitions of what an adversary can and cannot do in the system. Threat model can be classified and established based on the following categories:

• Confidentiality violation: The flow of assets to an untrusted IP or observable points is considered as a confidentiality violation. Security assets associated with the cryptographic primitives can be retrieved indirectly by monitoring the responses of the components under scrutiny to a series of actions. This indicates that there is no need for direct access to retrieve these security assets. The existence of an insecure form of data or control flow causes confidentiality violations.

• Integrity violation: Illegal modification or corruption of an asset is regarded as an integrity violation. Unauthorized interaction between trusted and untrusted IPs, illegal accesses to protected memory addresses, protected states of a controller module, and out of bound memory access fall under this threat model.

• Denial of Service (DOS)/Availability violation: Service or connectivity disruption of the modules embedded in a SoC or in IP can be categorized as a denial of service (DoS) threat. Numeric exceptions (e.g., divide by zero) and deadlocks can make a resource unavailable for a particular time and are examples of DoS threats. Moreover, DoS attacks can introduce timing (latency) overhead for recovery which can be utilized for leaking information or violating access controls.

Potential Weaknesses Identification: After having identified assets and established a threat model, the subsequent step involves finding the points of weaknesses and vulnerabilities through which an attacker might gain access to the asset for an exploitation. At this stage, vulnerabilities may arise due to inadequate design practices, as

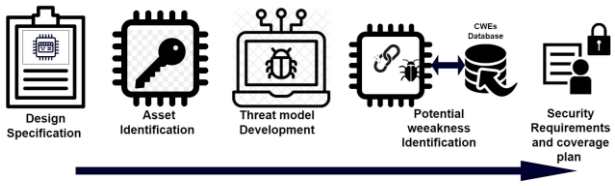

Fig. 2. Security Verification Plan

This work has been developed in the project VE-VIDES (project label 16ME0243K) which is partly funded within the Research Programme ICT 2020 by the German Federal Ministry of Education and Research (BMBF)

traditional design objectives often overlook security constraints and issues in micro-architecture implementation. Identifying these weaknesses is often challenging and time-consuming since it requires understanding the design's specification, the design's implementation, an understanding of the nuances in the correlation between these two aspects, and which parts of the design are most relevant to the threat model. To increase the chance of identifying security critical weaknesses, the MITRE CWE database [6] serves as a valuable resource. Once potential weaknesses related to the threat model are identified, it is essential to map them to CWE numbers to prepare security coverage. It is not always possible to incorporate the identified assets and threat models with CWE as there may be some weaknesses that are not yet listed in the database. However, identifying and stating potential weaknesses is still important to develop the security requirements.

Define security requirements and coverage plan: The last and significant step of the process is to define security requirements for the identified weaknesses in plain text. Once a mechanism is identified as a potential weakness, a security requirement can be expressed to address the weakness in the design. A coverage plan also needs to be defined for monitoring the stimulus that is going to cover by the test, as well as mechanism for assessing the extent to which the security requirements have been fulfilled.

IV. Methodology Application

The proposed methodology is applied on a commercial-grade System-on-Chip (SoC), an open-source Root of Trust (RoT) [10] design to demonstrate the real-world applicability and effectiveness. The implementation steps include formulating a security verification plan, creating a PSS model based on security requirements, and demonstrating the effectiveness of tests in the presence of intentionally inserted bugs in the design.

A. Design under verification

The design selected for implementing the proposed methodology is an Advanced Encryption Standard (AES) IP from Google's OpenTitan Root of Trust SoC. This SoC is utilized for Universal Second Factor (U2F) security key authentication and Trusted Platform Module (TPM) functionality, where the management of sensitive data is a primary aspect. It relies on security IPs such as AES and Keccak Message Authentication Code (KMAC). Among these security IPs, AES stands out due to its complex design and size, making it a suitable candidate for the application of the methodology. The AES IP functions as a peripheral on the chip interconnect bus, overseeing OpenTitan's symmetric encryption and decryption operations. It serves as a cryptographic accelerator, responding to requests from the processor to encrypt or decrypt 16-byte blocks of data. The AES unit supports AES-128/192/256 in ECB, CBC, CFB, OFB, and CTR modes through a single shared data path, allowing either encryption or decryption, but not both simultaneously [15].

The UVM was employed to create a testbench for the selected design. The RTL and testbench code are accessible in the OpenTitan Git repository [11]. To effectively apply this methodology, it was necessary to make fine-tuned adjustments to the open-source code.

B. Security Verification Plan

The first step in the proposed methodology is to develop a comprehensive security vPlan. The establishment of a security vPlan is a manual process, and it should be formulated by following the steps outlined in the Section III. Without a proper layout of these steps, this security verification plan may become skewed towards a functional verification plan.

The initial step in the process entails a thorough examination of the design specification, coupled with a prior understanding of the CWE database. This analysis is fundamental to grasp the potential vulnerabilities within the design's scope. After an in-depth analysis of the design specification for the OpenTitan AES IP, we identified five crucial assets within the design that need to be protected from the potential adversaries. Once these assets are defined, we then assume a threat model based on the attack surface. This could result in multiple threats being associated with a single asset. To ensure these assets have been thoroughly tested for specific weaknesses and are free from vulnerabilities, they are subsequently mapped to the CWEs. Therefore, the title for each weakness is maintained similar to those in the CWEs database for consistency and ease of understanding. After analyzing the considered assets and threat models, we have outlined ten security requirements. These requirements are focused on protecting confidentiality, integrity, availability, and preventing access control breaches of the crucial assets. Our security vPlan is depicted in Table I. It contains the identified assets, related threats and weaknesses, as well as the derived security requirement (SR) in plain language and the coverage plan.

C. PSS model

A PSS model of the intended test of the design under verification is prepared based on the security requirements listed in Table I. Atomic actions are modeled to represent a set of behavior where the constraining of stimuli is defined. Furthermore, coverage groups are devised to track and cover the target stimuli and actions. All actions are encapsulated within a component, which is part of a package. PSS model interacts with foreign languages typically HVL (SV or *e*) for simulation or C for embedded or emulation to initiate the behaviors that leaf-level actions represent in a test scenario. The PSS abstract model is expanded to define the intended implementation, where executable sections are used to call external target functions, and

TABLE I. Security Verification Plan

#	Attribute	Definition
	Asset	Control register "CTRL SHADOWED".
	Threat_1	Integrity violation, accessing the reserved bits could potentially compromise the state of the hardware.
1	Weakness	**CWE-1209**: Failure to disable reserved bits
	S_Requirement	The reserved bits inside the "CTRL SHADOWED" should be nonwritable.
	Coverage plan	Cover both write and read operations of "CTRL SHADOWED".
	Threat_2	Confidentiality violation, disclosure of configuration data.
2	Weakness	**CWE-200:** Exposure of sensitive information to an unauthorized Actor.
	S_Requirement	Control register should "CTRL SHADOWED" remain unobservable.
	Coverage plan	Cover all valid settings of "CTRL SHADOWED" register.
	Asset	Key registers "KEY SHARE0 and KEY SHARE1".
	Threat_1	Confidentiality and access control violation, software has the capability to read out key information.
3	Weakness	**CWE-1262:** Improper access control for register interface.
	S_Requirement	The key register should remain unreadable.
	Coverage plan	Cover write to all 16 key registers and also cover read actions for reading from all these registers.
	Threat_2	Confidentiality violation, disclosure of key.
4	Weakness	**CWE-200:** Exposure of sensitive information to an unauthorized actor.
	S_Requirement	key register should remain unobservable.
	Coverage plan	Cover write to all key register and actions.
	Asset	Data registers "DATA IN".
	Threat_1	Confidentiality and access control violation, Software has the capability to read out data information.
5	Weakness	**CWE-1262:** Improper access control for register interface.
	S_Requirement	The data register should remain unreadable.
	Coverage plan	Cover write to 4 data registers and cover read actions for reading from all these registers.
	Threat_2	Confidentiality violation, disclosure of data.
6	Weakness	**CWE-200:** Exposure of sensitive information to an unauthorized actor.
	S_Requirement	Data register should remain unobservable.
	Coverage plan	Cover data write and observe action.

	Asset	Lock bit for auxiliary control register “CTRL AUX REGWEN”.
	Threat	Integrity violation, write access to Auxiliary Control Register “CTRL AUX SHADOWED”.
7	Weakness	**CWE-1233:** Improper prevention of lock bit modification.
	S_Requirement	Control auxiliary register CTRL AUX SHADOW should not be modifiable after lock bit is enable.
	Coverage plan	Cover write to CTRL AUX SHADOW and CTRL AUX REGWEN action.
	Asset	Cipher core.
	Threat_1	Confidentiality violation, disclosure either a key, data, or both
8	Weakness	**CWE-203:** Observable discrepancy**.**
	S_Requirement	Intermediate result should not flow to output before round count value is complete.
	Coverage plan	Cover operation mode and key length with compound action.
	Threat_2	Confidentiality violation, disclosure of key length through timing side channel.
9	Weakness	**CWE-1254:** Incorrect Comparison Logic Granularity.
	S_Requirement	Time difference during a cipher operation involving different key lengths should not be observable.
	Coverage plan	Cover three key (128,192,256) length action.
	Threat_3	Integrity and availability violation, Disturbances or manipulation of control flow.
10	Weakness	**CWE 1245:** Improper FSM in hardware logic.
	S_Requirement	FSM States, Mux selection should not be able to enter undefined states.
	Coverage plan	Cover 26 target signal in three different FSM from cipher core.

checkers are set up to compare expected outcome with actual values. The next step involves creating test scenarios based on the security requirements utilizing the implemented atomic actions as building blocks. As shown in Fig. 5, a snippet of a solved test scenario and coverage data have been extracted from the EDA tools Perspec System Verifier, Incisive Metrics Center (IMC) used for compiling PSS model, and for coverage collection. In the following listing, we use the Security Requirement SR2 from our security vPlan (see Table I) to exhibit the implementation of workflow using PSS.

In Listing 1, a brief code snippet is provided that defines a package named “aes_test_pkg” (at line 1). This package is designed to encapsulate all the behaviors related to the AES IP that the test intends to cover. This package includes the component “aes_c” that specifies hardware IP AES (at line 6) and a struct “aes_seq_item_s” (at line 5) is defined to group all global stimuli within the package. These stimuli are transmitted to the target platform as part of a specific test.

Listing 2 includes a code snippet of all atomic actions required to create a test scenario for the security requirement SR2, all of which

```
 //aes_base.pss
1 package aes_test_pkg {
2 enum aes_op_e {AES_ENC = 2'b01, AES_DEC = 2'b10};
3 enum aes_mode_e {AES_ECB = 6'b00_0001,
                   AES_CBC = 6'b00_0010,
                   AES_CFB = 6'b00_0100,
                   AES_OFB = 6'b00_1000,
                   AES_CTR = 6'b01_0000,
                   AES_NONE = 6'b10_0000};
 .............
4 enum op_e {AUTOMATIC = 1'b0, MANUAL = 1'b1};
5 struct aes_seq_item_s {
   rand string name ;
   rand bit dut_init;
   //Control Shadow register
   rand aes_op_e aes_op ;
   rand aes_mode_e aes_mode;
   rand op_e op;
   ..............
  }
6 Component aes_c {
    ..............
  }//aes_c
 }//aes_test_pkg
```

Listing. 1. Test package definition

are grouped within “aes_c” (at line 1). Actions are unit behavior and defines system´s function and include the data object, resources and data attribute required for execution. For an example, action “aes_read_status_reg” (line 16) specifies a function of design to reading register value, Furthermore, a state data flow object “ctrl_shadow_state” (line 3) is defined to store updated values for later checks. This object is instantiated as output and as input, this ensures a smooth flow of data between actions. Abstract action “aes_base_a” (at line 9) serves as base actions for other actions and can only be inherited, it cannot be instantiated directly. Its primary purpose is to provide a foundation for other actions to build upon. An action “aes_write_ctrl_shadow_reg” (at line 19) is defined to write to the control register of the AES module. The actions may incorporate constraints to generate tests for specific behaviors of the DUV.

To execute all these actions, a binding is made with UVM testbench components, and necessary adjustments are made, such as establishing API and appropriate hooks to run the PSS test on the existing UVM testbench. In addition, an explicit cover group is defined in PSS to monitor the generated stimuli, and an inline cover group is established to track which actions and scenarios were exercised during the test generation.

Listing 3 contains a code snippet illustrating test scenario and its inline scenario covergroup, aimed at generating tests for SR2. A compound action “ctrl_shadow_reg_observability_sr” (at line 3) is defined to encapsulate the test scenario. Series of actions crafted to execute in one after one manner (at line 5), starting with “aes_init_a”

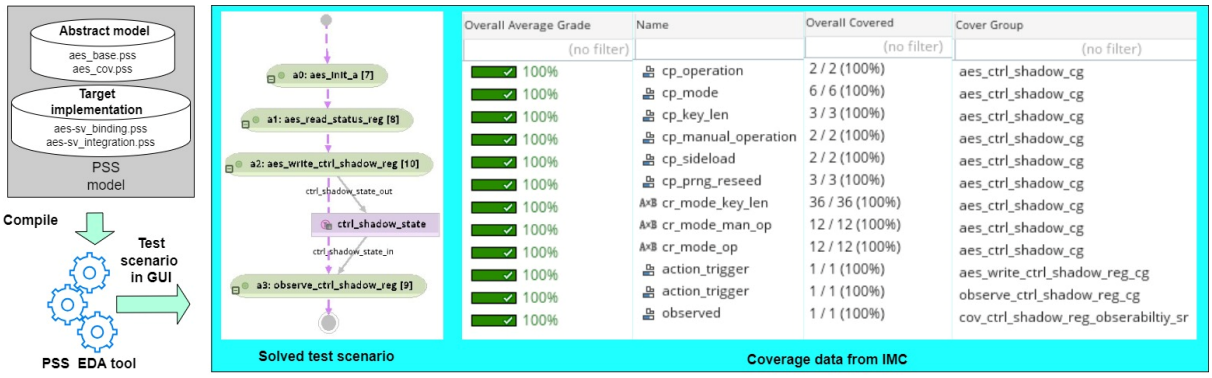

Fig. 5. Implementation workflow with PSS model

```
//aes_base.pss
component aes_c {
//Declare state to store data
 state ctrl_shadow_state {
   rand bit [32] updated_ctrl_shadow_reg;}
   pool ctrl_shadow_state ctrl_shadow_state_pool;
   bind ctrl_shadow_state_pool *;
   ..........
 abstract action aes_base_a {
   rand aes_seq_item_s seq_item;
   bool wait_for_completion = true;}

//To initialize DUV
 action aes_init_a : aes_base_a {
  constraint seq_c {seq_item.name ==
    "aes_init_vseq"; seq_item.dut_init == 1;}}

//To read status register
 action aes_read_status_reg : aes_base_a {
  constraint seq_c {seq_item.name ==
    "aes_read_status_reg";
    seq_item.dut_init == 0;}}

//To write control shadow register of aes
 action aes_write_ctrl_shadow_reg : aes_base_a {
  output ctrl_shadow_state ctrl_shadow_state_out;
  constraint seq_c {
   seq_item.name == "aes_write_ctrl_shadow_reg";
   seq_item.dut_init == 0; }}

//To monitor any potential leakage of
  ctrl_shadow_reg
 action observe_ctrl_shadow_reg : aes_base_a {
  input ctrl_shadow_state ctrl_shadow_state_in;
  constraint seq_c {seq_item.name ==
    "observe_ctrl_shadow_reg";
     seq_item.dut_init == 0;} }

}//aes_c
```

Listing 2. Top component with atomic actions

to initialize AES, arrange the clock and reset, and confirm that the AES is in idle mode followed by read and write operation of status and control register for the configuration. Finally, "observe_ctrl_ shadow_ reg" action is performed to confirm that the values in the output register do not correlate with the configured values in the control shadow register.

Following the successful compilation of the PSS model and the resolution of the specified PSS test scenarios for ten requirements, the tool generated tests tailored to the desired stimuli. These tests were executed in simulation environment with the help of a developed bash script that is meant to automate the process of running the test by invoking simulator and extracting the result from generated logfile after test completion.

```
//aes_scenarios.pss
extend component pss_top {
  import aes_test_pkg::*;
  action ctrl_shadow_reg_observabiltiy_sr {
    activity {
      sequence {
         a0: do aes_c::aes_init_a;
         a1: do aes_c::aes_read_status_reg;
         a2: do aes_c::aes_write_ctrl_shadow_reg;
         a3: do aes_c::observe_ctrl_shadow_reg;
      };};};};

//aes_scenario_coverage.pss
extend component pss_top {
  extend action ctrl_shadow_reg_obserabiltiy_sr {
      //scenario coverage
      covergroup {
          bit in [1] observed: coverpoint 1;
      }cov_ctrl_shadow_reg_obserabiltiy_sr
    };
  };
```

Listing 3. Compound action with inline coverage

V. Result

The implementation of the developed methodology on OpenTitan AES was marked as successful. A total of 287 tests were generated from the PSS model to verify ten security requirements. Out of these, 281 tests cleared successfully while six tests failed. The failure of these tests indicated that there is a weakness related to SR9 in the design. It was observed that the duration required to perform cryptographic operations varied based on the key length. The number of cycles required for operation are 10,12, and 14 corresponds to the key length of 128,192, and 256 bits, respectively. This suggests that the duration of the operation is influenced by the key length. Ideally, the operation should take the same amount of time regardless of the key length to ensure asset confidentiality. This unmasked RTL implementation of the AES IP is not an optimal choice for applications with high-security demands, as its weakness could be exploited to deduce the length of the AES key. Table II provides a summary of the test outcomes.

To validate the efficiency of autogenerated tests from the PSS model, deliberate security weaknesses were inserted into the RTL implementation of OpenTitan AES design. This intentional RTL alteration was aimed at implementing three specific weaknesses tied to SR2, SR6 and SR8. The tests successfully detected the intentionally inserted bugs and flagged errors. A brief description and simulation result are presented below for SR8 that shows the tests were able to detect the intentionally implemented security weakness in RTL implementation.

The scenario "intermediate_result_sr" was resolved for the ECB mode of AES operation, as this mode is considered the least secure compared to other modes. The action was constrained to generate tests for both encryption and decryption in ECB mode using three distinct key lengths: 128, 192, and 256 bits, resulting in a total of six tests. Fig. 6 shows a snapshot of the waveform observed during simulation, demonstrating that when the encryption operation is active, the 'crypt_busy' signal is high, and the 'data_out_we' signal remains low until the cipher text is prepared, the 'data_out' signal transitions to high, and data is transferred to the output register. The test is aimed to determine whether any intermediate values appear in the output register. Since no intermediate values were observed in the output register, the tests were deemed successful. Fig. 7 shows the waveform outcomes following the introduction of RTL implementational weakness into the design. The tests available on GitHub for OpenTitan AES functional verification were not able to identify this specific introduced weakness in the design, therefore, goes undetected as all tests are successfully passed. The generated test for SR8 (from Table 1) detected the weakness in the design and terminated the test as the checker in PSS flagged fatal. The tests for functional verification passed because the modification maintains the functionality of the design, it introduces a flaw that allows a leakage of intermediate results. However, the final output remains consistent.

TABLE II RESULT SUMMARY OF SECURITY REQUIREMENTS

#	PSS Test Scenario	Generated Test	Test Status
1	ctrl_shadow_reg_reserve_access_sr	108	Passed
2	ctrl_shadow_reg_observability_sr	108	Passed
3	key_reg_read_access_sr	10	Passed
4	key_reg_read_observability_sr	10	Passed
5	data_in_reg_read_access_sr	10	Passed
6	data_in_reg_observability_sr	10	Passed
7	ctrl_aux_regwen_protection_sr	4	Passed
8	intermediate_result_sr	6	Passed
9	timing_sca_sr	6	Failed
10	aes_cipher_core_fi_sr	15	Passed

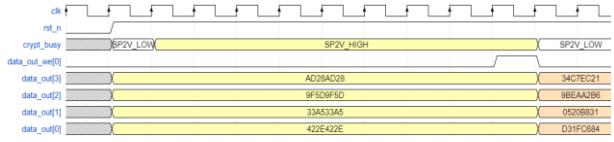

Fig. 6. SR8 simulation waveform

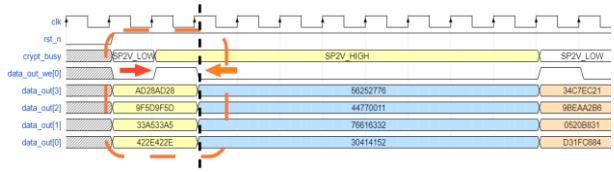

Fig. 7. SR8 simulation waveform after RTL weakness insertion

Formulating security requirements in PSS offers several benefits. Firstly, it allows for the establishment of scenarios that might be challenging to reach with other methodologies. Moreover, PSS facilitates automated test generation, capable of producing a comprehensive set of tests depending on the coverage objectives. Another significant advantage is the reusability of the model. New security requirement formulations can be derived on top of the previously established model, enhancing its adaptability and efficiency. Lastly, this methodology exhibits scalability, by enabling the use of developed model for IP (such as AES) in sub-System (AES with DMA and a processor) and System level, which is particularly advantageous at the System-on-a-chip (SoC) level, making it a robust and versatile approach for assorted security verification tasks.

VI. Summary

An effective security verification methodology is developed to identify and address security weaknesses and expose vulnerabilities in the digital design. The methodology outlines a strategy for creating a comprehensive security verification and coverage plan. To bridge the gap between different verification platforms and enable the reuse of test cases across various design verification stages of the SoC development cycle, the Portable Test and Stimulus Standard (PSS) is employed to formulate the security requirements as an abstract model. An abstract modelling of security requirements in PSS simplifies the process of creating and visualizing tests, bridge the communication gap between the verification engineers at different verification platforms by offering a common language through actions and security scenario diagrams. The proposed methodology is particularly well-suited for complex System-on-Chip (SoC) designs that are intended to execute security-critical applications. This is because of the capability of PSS to capture internal behaviors, resources, and their usage, so it can automate hard-to-achieve security-related scenarios and generate the significantly higher number of machine-produced variations.

The findings showed the potential vulnerabilities arising from security weaknesses in digital design and underscore the importance of robust security verification methodology for ensuring system security and trustworthiness. Simulation results are showcased to illustrate how a security breach can occur, even when functionality appears to be correct.

References

[1] C. S. Chuah, C. Appold, and T. Leinmueller, “Formal Verification of Security Properties on RISC-V Processors,” in *Proceedings of the 21st ACM-IEEE International Conference on Formal Methods and Models for System Design*, Hamburg Germany: ACM, Sep. 2023, pp. 159–168.

[2] P. Subramanyan and D. Arora, “Formal verification of taint-propagation security properties in a commercial SoC design,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, Dresden, Germany: IEEE Conference Publications, 2014, pp. 1–2.

[3] F. Restuccia, A. Meza, and R. Kastner, “AKER: A Design and Verification Framework for Safe and Secure SoC Access Control.” arXiv, Jun. 24, 2021. Accessed: Apr. 10, 2024. [Online]. Available: http://arxiv.org/abs/2106.13263

[4] C. Deutschbein, A. Meza, F. Restuccia, R. Kastner, and C. Sturton, “Isadora: automated information-flow property generation for hardware security verification,” *J. Cryptogr. Eng.*, vol. 13, no. 4, pp. 391–407, Nov. 2023.

[5] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, “An Overview of Hardware Security and Trust: Threats, Countermeasures, and Design Tools,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 6, pp. 1010–1038, Jun. 2021.

[6] Common Weakness Enumeration. [Online]. Available: https://cwe.mitre.org/data/definitions/1194.html

[7] Portable Test and Stimulus Standard Version 2.1. Accellera Systems Initiative. Accessed: Jan. 26, 2024. [Online]. Available: https://accellera.org/images/downloads/standards/pss/Portable_Test_Stimulus_Standard_v2.1.pdf

[8] J. Wang et al., “RISC-V Security Verification using Perspec/Portable Stimulus,” in proceeding of *Design amd. Verification Conference and Exhibition (DVCon) Europe 2024*.

[9] P. Bhamidipati, S. M. Achyutha, and R. Vemuri, “Security Analysis of a System-on-Chip Using Assertion-Based Verification,” in *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Lansing, MI, USA: IEEE, Aug. 2021, pp. 826–831.

[10] OpenTitan for silicon root of trust (RoT) chips. [Online]. Available: https://opentitan.org/

[11] OpenTitan git repository. [Online]. Available: https://github.com/lowRISC/opentitan

[12] C. Deutschbein, A. Meza, F. Restuccia, M. Gregoire, R. Kastner and C. Sturton, "Toward Hardware Security Property Generation at Scale," in *IEEE Security & Privacy*, vol. 20, no. 3, pp. 43-51, May-June 2022.

[13] J. He, X. Guo, T. Meade, D. Raj, Z. Yiqiang and J. Yier, “ SoC interconnection protection through formal verification,” in Integration, the VLSI journal, vol. 64, pp. 143-151, January 2019.

[14] A. Ardeshiricham, W. Hu, J. Marxen and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, Lausanne, Switzerland, 2017, pp. 1691-1696.

[15] Morris Dworkin. 2016. Recommendation for block cipher mode of operation. NIST Special Publication 800 (2016),38G.