Special Session: CAD for Hardware Security - Automation is Key to Adoption of Solutions

Sohrab Aftabjahani
Dataplatform Engineering and Architecture
Intel Corporation
Hillsboro, USA
sohrab.aftabjahani@intel.com

Ryan Kastner
Department of Computer Science and Engineering
UC San Diego
San Diego, USA
kastner@ucsd.edu

Mark Tehranipoor
Department of Electrical and Computer Engineering
University of Florida
Gainesville USA
tehranipoor@ece.ufl.edu

Farimah Farahmandi
Department of Electrical and Computer Engineering
University of Florida
Gainesville USA
farimah@ece.ufl.edu

Jason Oberg
Tortuga Logic
San Jose, USA
jason@tortugalogic.com

Anders Nordstrom
Tortuga Logic
San Jose, USA
andersn@tortugalogic.com

Nicole Fern
Tortuga Logic
San Jose, USA
nicole@tortugalogic.com

Alric Althoff
Tortuga Logic
San Jose, USA
alric@tortugalogic.com

***Abstract*— Although hardware security has received significant attention in the past decade or so, security design and validation engineers and researchers in industry, academia, and government have not still been equipped with a mature security-aware toolset to automatically and effectively analyze designs for various types of security vulnerabilities at different to detect and fix the security issues or build security in designs efficiently and easily. Despite such a demand, currently, there is not an ecosystem of security-aware Electronic Design Automation (EDA) or Computer-Aided Design (CAD) tools whereas the commercial design for security and validation tools are still in their infancy. However, there exist many research works that try to come up with security analysis engines and provide solutions to address different classes of security issues such as data leakage, access control violation, side-channel leakage, hardware Trojans and malicious changes, and vulnerabilities to physical attacks, fault-injection attacks, reverse engineering attacks, and chip counterfeiting or overproduction attacks. This paper presents the foundation established by several academic and industry researchers who have been supporting the realization of an ecosystem of security-aware CAD tools with their focus on hardware security coverage and fault-injection assessment for SoC designs, and security assurance standardization for electronic design integration.**

Keywords—CAD for Security, Hardware Security Coverage, Fault-Injection Attacks, Security Assurance, Electronic Design Integration.

I. INTRODUCTION

In the last decade, many hardware security vulnerabilities have been identified by security researchers from academia and industry [1,2,4]. As fixing the hardware security issues are usually expensive and even in some cases not possible, many design and manufacturing companies, nowadays, invest a lot on hiring and retaining hardware security experts to rely on them for security assurance to avoid such vulnerabilities. Considering the lack of enough experts to meet these demands, the job market of hardware security is still hot and many investments have been made by the government and industry to establish academic programs to create the workforce required to meet this need. We see the increasing number of companies that have been actively working on building their hardware security teams to incorporate security into their Product Development Lifecycle (PLC) [3], i.e. Security Development Lifecycle (SDL) [3], which typically includes planning, design, development, validation, manufacturing, testing, and support steps. To minimize residual security risks before product shipment, this is an attempt to address their product security requirements using (a) systematic approaches by security architects, product security experts (blue teams) to defend and (b) ad-hoc approaches by security researchers (red teams) to attack their own products.

While the above trend is promising for strengthening the hardware security posture of products around us in the market, SDL processes still lack the ideally required scalable, systematic, comprehensive security analysis engines as well as security modeling standards for creating and managing portable security-related design collaterals. Such collaterals generally include security claims of designs, security integration guidelines, and ideally threat models (security objectives, adversary profiles, assets, attack surfaces, possible attacks scenarios exploiting vulnerabilities, and the mitigations of possible vulnerabilities, etc.). A new generation of security-aware EDA tools incorporating novel scalable approaches and methods are necessary to provide the level of security needed to be built into products as well as the required level of security assurance. The tools should take advantage of the speed and accuracy possible by automation and modern computation power while having the minimum impact on time-to-market, cost, and efficiency of products.

In the early years of the last decade, the semiconductor industry was not even ready for CAD for security as many design houses considered spending on design for security and security assurance like overhead that the end consumers were not really willing to pay for, and hence not much investment was made by Electronic Design Automation (EDA) companies in CAD for security. In 2015, for the first time, some like-minded security experts from industry and academia (some of them are among the authors of this paper) decided to plan how to close this gap by creating a movement to realize an ecosystem of security-aware CAD tools for Design for Security and Security Validation and Assurance. This required finding ways to fund

This work was supported by NSF Award #1718586 and SRC Tasks 2770.001 and 2993.001.

research and development to create various components (plugins) of this general tool as each security analysis plugin is usually specialized to address only a few aspects of design for security and security assurance. Hence, we tried to bring CAD for Security to the prioritized set of national and commercial research investments [4-6]. Many CAD for security-related research projects were later funded by government and private companies. Concurrently, we also tried to bring Security and CAD for Security to the attention of the HW security community and EDA community from academia, industry, and government by creating panels and tutorial sessions in DAC, IVSW [7], MTV [8], HOST, VTS and GLSVLSI (the titles of the panels and tutorials are given below). Many of the authors of this paper were among the panel organizers, panelists, and tutorial instructors, invited and keynote speakers.

We also utilized Trust-Hub [9] lead by Tehranipoor, sponsored by National Science Foundation (NSF), as the venue to bring CAD for Security solutions [10], first built around Taxonomy of Physical Attacks [11] to create plugins capable of vulnerability analysis for each type of attack. The solutions from academia and industry cataloged on Trust-Hub were also intended to promote collaboration and information sharing among researchers and attract government and industry (especially semiconductor and EDA companies) to invest in them. Currently, a catalog of more than 100 solutions [8] from our many partners from academia and industry to help us bring solutions to various aspects of design for security and security verification to realize building blocks of our planned "General Security Design, Analysis, and Validation Framework".

We believe that we have succeeded because, after several years, now, we are eyewitnesses of realization of our vision as several major and small EDA companies have been getting involved in creating an ecosystem of security-aware tools, numerous academic security researchers (UoF, UCSD, UT-Austin, UT-Dallas, GaTech, …) have been joining us or have started parallel efforts like CAD for Assurance [12], and semiconductor industry (Intel, AMD, IBM, TI, NXP, Analog Devices, …) EDA industry (Synopsys, Siemens EDA, Business, Cadence, ANSYS, Tortuga Logic, …) and government (DARPA, AFRL, Navy, NSF, …) are willing to support us in this mission by funding research and development projects directly or indirectly (e.g. through Semiconductor Research Corporation) to create more specialized security analysis engines and/or to commercialize some of the solutions.

With this background, the rest of the paper, sections II, III, and IV focus on the invited work of a few contributors to make our audience more familiar with their work in the area of CAD for security. Section II gives an overview of hardware security coverage. Section III discusses fault-injection assessment for SoC design. Section IV describes the security assurance standardization for electronic design integration work done under the Accellera IP Security Assurance workgroup. Section V concludes the paper with a summary.

II. Hardware Security Coverage

Unlike functional verification that checks if the specified requirements can happen, security verification takes requirements and ensures that they cannot happen. In other words, functional verification is largely focused on verifying the known whereas security verification targets the unknown. Thus, functional verification and security verification are fundamentally different.

Functional verification focuses on matching the hardware to a known and precise specification. Functional verification involves generating a set of cover properties that match the functional requirements, developing directed tests that activate those cover properties, evaluating the coverage, and if everything passes, signing-off on the functional correctness. Once all specified cover properties are sufficiently activated, there is assurance that the hardware is functionally correct, and the functional verification sign-off is complete. Functional verification is a significant undertaking that consumes over 70% of the hardware design process [13]. As such, there exists a rich set of mature commercial tools that rely on techniques like equivalence checking, assertion-based verification, property checking, and theorem proving.

The security verification process involves developing a threat model, analyzing the design under verification, attempting to point out potential vulnerabilities, and articulating those flaws to the design team. It is almost always possible to find additional vulnerabilities given more time and resources. Yet, the security sign-off also has strict time-to-market constraints. Thus, the security verification engineer struggles with the question "When is the hardware secure enough?" -- a challenging endeavor especially considering the general lack of hardware security verification tools.

Unfortunately, the hardware security verification process is not well defined. Threat models are wide ranging and often challenging to precisely specify. New vulnerabilities pop up that require disabling performance features or costly redesign of the hardware [14,15]. With the growing understanding that hardware vulnerabilities are readily exploitable and that hardware plays a foundational role in overall system security, there is an increasing amount of time devoted to hardware security verification. Yet, hardware security verification still relies heavily on designer intuition and experience. Powerful hardware security verification tools exist, but work remains on how to effectively use these tools to achieve the desired security coverage. Defining metrics for security coverage is crucial to using hardware security verification tools to achieve security sign-off.

Coverage metrics are key to verification [16]; without them, it is impossible to state anything about the completeness of the verification process. Functional coverage metrics are well-understood and include code coverage, circuit coverage, FSM coverage, assertion coverage, mutation coverage, and vacuity coverage [17]. Unfortunately, defining security coverage metrics is not as straightforward -- looking for unknowns is a never-ending process and the concept of being done does not exist. Yet, it is crucial to articulate what should and has been considered in the security verification process. We desperately need security coverage metrics to provide concise methods for enabling a security sign-off.

To address this, we introduce the CWE-IFT security coverage metrics. CWE-IFT uses information flow tracking (IFT) property templates based on common weakness enumerations (CWEs) to define the threat model and integrate

into a property driven hardware security verification flow [18]. CWE-IFT provides a quantitative way to measure hardware security coverage while using existing functional verification tools (including formal solvers, simulation, and emulation) to verify the security of the hardware.

A. *Security Coverage Trends*

An important trend in security coverage is the development of known and exploitable vulnerabilities. Common Weakness Enumerations (CWEs) [19] play a key role in security verification. The CWE taxonomy started in 2006 and quickly evolved into the de facto catalogue of common software weaknesses. Hardware CWEs were added in 2020 and there are over 100 listed hardware CWEs [20]. While the CWE database does not directly inform the security engineer that they have sufficiently "verified security" or achieved high coverage, it does provide an enumeration of relevant weaknesses for security engineers to focus on and refine their security requirements into what is most important for their hardware. This will continue to be an important driver for research and security tool development moving forward.

Another important trend of security coverage is information flow tracking (IFT) tools, which provide the capability to symbolically track information rather than values. IFT can verify hyperproperties [21] related to confidentiality, integrity, and timing-based flows. In doing so, security weaknesses can be more efficiently detected without the enormous effort of value-based assertion or directed test creation as is required when only considering trace properties. IFT has a long history of research [22] and is used at the core of Tortuga Logic's Radix software products.

A final important trend of security coverage is to leverage, as much as possible, the existing hardware design and test infrastructure. Modern hardware design and verification uses a rich set of tools from formal verification, simulation, hardware emulation, and FPGA prototyping. Due to strict time-to-market requirements and security's often perceived orthogonal goal for that requirement, it's critically important that security verification, and security coverage, be enabled across that same design life cycle. The big question is, how does one enable hardware security coverage in a manner that uses a conventional functional verification environment?

B. *CWE-IFT*

IFT tools provide a unique approach to capture previously unknown vulnerabilities. Pairing IFT and CWEs provides a methodology that can specify security properties, formally define the threat model, and provide coverage metrics that assess the verification process. CWE-IFT allows the designer to create security properties that adequately capture the CWE, executes them using commercially available verification tools and ultimately provides confidence that all relevant common weaknesses were covered and the threat model has been properly assessed.

To better understand the CWE-IFT methodology, consider for example CWE-1243: Sensitive Non-Volatile Information Not Protected During Debug and the example design shown below:

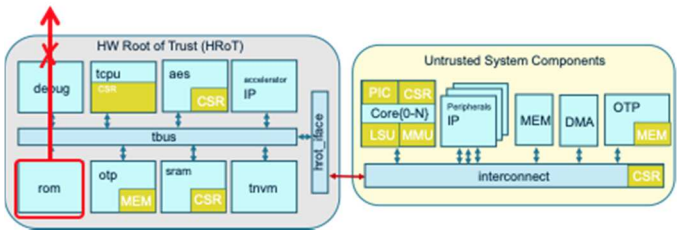

Fig. 1 Example design for CWE-IFT methodology.

Here the weakness is specifying that any information related to an asset stored in ROM should not leak out of the debug interface. From the requirement, one can easily identify three critical components:

1. **Asset**: ROM
2. **Security Boundary**: Debug
3. **Security Objective**: Confidentiality

From here, it is easy to create a simple information flow rule that captures the security requirement and CWE:

ROM.mem[FUSE_END:FUSE_START] when (tcpu.csr.debug_mode) =/=> debug.$all_outputs

This security property can then be passed to a IFT tool, e.g., Tortuga Logic's Radix, to then execute this in a conventional simulation or emulation environment.

CWE-IFT defines a systematic approach to formalize security requirements to CWEs using security properties and perform verification using hardware IFT all in a conventional functional verification environment. This process is outlined in the Radix Coverage for Hardware Common Weakness Enumeration (CWE) Guide [23]. Here we have outlined a five-step process for creating a verifiable security property that covers the relevant CWEs:

1. Identify CWE(s) relevant to the threat model.
2. State plain-language security requirement identified in the CWE(s).
3. List the assets (in the form of data or design signals), objectives (confidentiality, integrity, availability), and security boundaries of the design as they correspond to Step 2.
4. Use the Radix security rule template for the corresponding CWE in this document. Add design signals from Step 3 to create security rules that can be validated in Radix alongside standard verification environments from Cadence, Siemens EDA, and Synopsys. Perform sign-off that each CWE has been successfully checked.

Here is one such example using the requirement specified above:

1. Identify CWE relevant for the threat model: *CWE 1243: Sensitive Non-Volatile Information Not Protected During Debug*
2. State plain-language security requirement: *"ROM (eFuse) should not be readable by debug."*

3. Identify asset, objective (confidentiality, integrity, availability), and security boundary

Asset: "ROM (eFuse)

Security Objective: "must not be readable" is Confidentiality

Boundary: Debug

4. Use security rule template for CWE-1243:

```
{{Security-sensitive Fuse Values}} when (debug
mode enabled) =/=> {{User-accessible signals}}
```

5. Add corresponding signals in design to rule template

```
rom.mem[FUSED_END:FUSED_START]
when(tcpu.csr.debug_mode)=/=>
    debug.$all_outputs
```

Doing so across all relevant CWEs provides a powerful mechanism to effectively cover one's security requirements and enable a more comprehensive security verification program.

CWE-IFT leverages information flow tracking verification tools to cover common weakness enumerations to integrate into a property driven hardware security design flow. We described an example of the CWE-IFT methodology, that provides a quantitative way to measure hardware security coverage while using existing functional verification tools (including formal solvers, simulation, and emulation) to verify the security of the hardware. CWE-IFT is just a start in the important research of hardware security coverage.

III. PRE-SILICON FAULT-INJECTION ATTACK ASSESMENT

A. Introduction

System-on-chips (SoCs) are prevalent in modern computing devices deployed to military and space applications, mobile applications, financial systems, transportations, even household appliances. SoCs are subject to an array of attacks, namely information leakage, side-channel leakage, fault injection, physical attacks, rowhammer, and more. Among them, fault-injection attacks, though immensely powerful, have unfortunately received the least attention from the community.

Fault-injection attacks are based on intentionally injecting faults into a system to cause confidentiality and/or integrity violations of security assets of the design or denial of service. Fault-injection attacks are capable of tampering with vulnerable locations in a device and enable attackers to access the secret assets of the design. Various attacks have been successfully demonstrated on encryption and digest algorithms such as AES, DES, RSA, and SHA [24-29]. More attacks can be applied to many other security-critical applications including security controllers, artificial neural network accelerators, homeomorphic algorithms, and post-quantum algorithms.

The fault-injection vulnerabilities may be the result of poor design choices or automatically generated code by electronic design automation (EDA) tools (e.g., synthesis tools) when they try to optimize the design overheads without consideration of security as one of the decision factors. For example, some state encodings in controller designs make the design more susceptible to fault-injection attacks [30]. At the same time, synthesis tools may create don't-care states that are connected to the protected states (that are responsible for controlling security-critical operations) of the designs. It has been shown that these don't-care states can be accessed by changing the clock frequency and creating clock glitching. As a result, the protected states can be accessed in the next clock cycle and the security mechanism of the design will be bypassed.

Many different countermeasures such as tamper-proof packaging, error correcting codes, and triple modular redundant structures have been proposed so far to help with protecting against fault-injection attacks [31-32]. However, they would be expensive with the significant design effort and large area or performance overhead, making their application limited in practice. Moreover, the current EDA tools are not capable of effectively assessing the vulnerability of hardware designs against fault-injection attacks. As the current evaluations are limited and are done manually, it is difficult to ensure their effectiveness. Therefore, it is very important to be able to evaluate the susceptibility of designs to fault-injection attacks at pre-silicon while there are still opportunities to address the possible vulnerabilities efficiently.

To successfully evaluate the susceptibility of the hardware designs against fault-injection attacks, we need to perform the following steps: (i) classification of fault-injection methods; (ii) building comprehensive fault models to measure the success rate of different fault-injection techniques; (iii) identifying security-critical assets of the designs that are required to be protected against fault-injection attacks; (iv) creating fault lists based on the identified fault model and performing fault simulation to assess the resiliency of the design toward a specific fault-injection technique that aims to bypass the confidentiality, integrity, and availability properties of a design. Finally, a set of countermeasures with a cost-performance-security trade-off in consideration should be implemented to mitigate the identified vulnerabilities of a design against fault attacks.

B. Fault-Injection Attack Techniques

Several fault-injection techniques have been developed to alter the correct functionality and security features of an integrated circuit. Fault-injection techniques can be classified into two major categories: non-invasive and invasive attacks. Focusing on non-invasive fault-injection attacks, faults can be injected in vulnerable locations of the design by tampering with the working conditions of the design like performing clock or voltage glitching [33-34].

Clock Glitching: In this technique, faults can be injected by violating the setup- or hold-time requirements of design flip-flops. This may cause shortening the length of a clock signal temporarily and ultimately capturing the wrong value in memory elements of the design or skipping an instruction.

Voltage Glitching: These faults can be injected by tampering power supplies of the device. For example, running the chip with lower supply voltage may result in leaving high-threshold transistors in the design open, and ultimately flipping bits in the design, latching wrong values, or skipping some instructions.

Electromagnetic Radiation: Electromagnetic (EM) fault occurs when any IC surface faces a sudden variation of a magnetic field. The variation of magnetic field creates an

electromotive force in the IC surface which in turn gives rise to a parasitic current in the wire loops in the IC. The amplitude of the parasitic current is proportional to the variation rate of the magnetic field. Electromagnetic fault injections mainly modify the behavior of power and ground networks which has a direct effect on the operation of D-flip-flops (DFF's). When the fault occurs, the inputs of DFFs are disrupted and as a result, the DFFs may sample wrong values during their operation [35].

Faults can be injected in vulnerable locations of the design based on the following analysis:

- Timing analysis of the designs and measuring hold-time and setup-time of the flip-flops - shorter paths are more likely to have a hold-time violation and longer and critical paths are susceptible to setup-time violation. Hence, the effect of EM and power faults (in case of slow down) on the critical paths are measured for security-critical applications. Such faults can result in a processor skipping an instruction or storing incorrect data in the memory.
- Switching activity analysis to detect transitions that can be skipped using changing the clock frequency, reducing the voltage, or slowing down the design using EM-faults.
- Distance measurement analysis from power supplies to critical gates (especially in the control flow of the design) to evaluate the effect of depleted power resources to create bit flips.
- Fan-out and fan-in analysis as well as topology analysis of signals in security properties to measure the effect of power spikes to skip instructions or rounds of computations. Fan-out cone analysis provides valuable information about the length of interconnects. Longer interconnects can be attractive locations for EM and power faults since they propagate the potential slow down effects of these faults and can cause setup and hold time violations and bit flips.

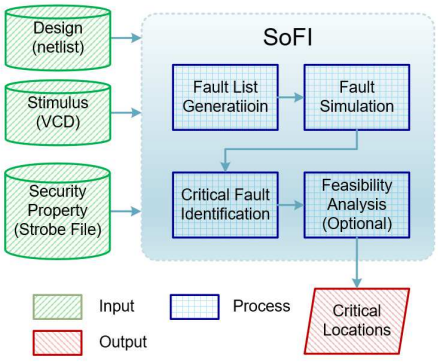

Fig. 2 A high-level overview of the SoFI framework.

Non-invasive fault-injections techniques are usually inexpensive compared to invasive approaches and do not need detailed knowledge about the design. For invasive attacks, such as FIB and probing [36], they require higher cost, de-packaging, and more knowledge of the design. The chip may be fully or partially damaged under the attack. These faults are local with better resolution.

C. *SoFI: A Security Property-Based Approach to Fault-Injection Attack Assessment*

SoFI is a security property-driven vulnerability assessment framework for SoCs against fault-injection attacks recently proposed in [37]. A security property defines operations that must be present or absent thereof in a design to maintain the integrity, confidentiality, and availability of the design. The critical locations to a fault-injection attack are identified by checking whether any security properties can be violated by the fault. The more critical locations identified, the more vulnerable the design is to fault-injection attacks. Also, by identifying critical locations, local countermeasures can be developed making protection overhead reduced significantly.

The overall flow of the SoFI framework is shown in Fig. 2. The SoFI framework takes the gate-level design, stimulus vectors, and security properties as the inputs. First, to map specific fault-injection techniques (e.g., clock/voltage glitch or laser) in the assessment, a fault list will be generated based on the fault models and their characterization. The characterized fault models enable us to simulate the fault injection in digital circuits for a specific fault injection technique, such as an external EM field or voltage glitching. Therefore, the fault simulation is performed as the next step to identify critical faults that can violate security properties without redundant fault locations.

To preserve security properties, SOFI identifies the most critical design locations that injecting faults in them will result in violating the given security properties. The identification of these locations depends on the fault model, fault-injection technique, and completeness of provided security properties. After detecting the most vulnerable locations in the design, local countermeasures can be applied for an effective and low-cost protection mechanism against fault-injection attacks. SOFI has been applied on AES, RSA, and SHA implementations, and the results show that the threat from fault-injection attacks can be significantly reduced by only protecting less than 0.6% of the design.

D. *COUNTERMEASURES*

After detecting vulnerable components of the design to fault-injection attacks, a set of countermeasures, like localized redundancy, should be applied to protect the design toward various fault-injection techniques/attacks. Some countermeasures focus on using sensors to detect the act of fault injection. However, the most common fault detection and mitigation method is based on creating redundancy, which tends to require more resources than the minimal necessary ones to complete the task. When a fault strikes, redundancy is utilized to mask the faults, thus maintaining the correct functionality of the system [24]. Typically, there are three types of redundancy in terms of available resources: *hardware*, *information*, and *time*. Hardware redundancy indicates adding extra hardware into the device to either detect or correct the impacts of the faults injected. One example is an M-of-N system, which consists of N modules and requires at least M to

function correctly. The system fails when fewer than M modules are working properly. Dynamic redundancy is another type of hardware redundancy, where unused resources are activated when faults are injected in the currently active resource. However, hardware redundancy incurs high overheads (at least 2X), which may be too expensive to be used in practice. The most common information redundancy countermeasure is an error detection code (EDC). In EDC, check bits are incorporated into the original bits so that errors can be detected by comparing the predicted and the received check bits, which is widely used in memory units. Time redundancy can also be utilized to detect faults by re-running the same process on the same hardware. Below, we describe countermeasure for a number of fault-injection attacks.

Clock/voltage glitching: When applying clock/voltage glitching attack, faults will propagate uniformly across the device, which can give an advantage to the attacker in terms of accessibility to rarely activated nodes and registers in the circuits. Countermeasures include internal oscillators, asynchronous logic, different threshold voltages and applying redundancies. One could use a temporal redundancy approach. Since the clock and voltage glitching are causing global faults, it is not feasible to protect the whole circuits with spatial redundancy. Temporal redundancy can be applied non-tiing critical parts of the circuits because it is not guaranteed that the global faults occur exactly the same due to their randomness when the same function is executed.

Local heating: Local heating influences the whole design, it requires minimal technical knowledge, and the equipment is readily available. One drawback of this technique is that it tends to cause invasive faults in sensitive devices. Another downside is that the circuit may be destroyed through excessive heating. Causing multiple bits to flip using local heating may be highly possible when there is an excessive heating.

EM pulses: EMFI (EM fault injection) causes transient voltage drops which may cause a significant amount of delay variation in the circuits. Sensors that can measure the change in circuit timing can be used to detect EMFI. For instance, time-to-delay converter (TDC) sensors and glitch detectors respond to delay variation in integrated circuits, so they can be used to detect EMFI. However, EMFI may impact the circuit locally, hence only one detector in a large SoC may not be sufficient to effectively detect injected faults in the entire circuit [38]. So, several detectors may be needed for the uninterrupted operation of an IC. We acknowledge that power supply noises, temperature, and process variation can also cause delay variation in the circuits [39]. As a result, it would be challenging to differentiate between EM-based fault injection and the above-mentioned process and environmental variations. This may require detectors to be calibrated precisely so as to efficiently detect EMFI.

Light radiation: Different from the light beam, light radiation exploits the light such as UV light to have an influence on the whole circuit instead of some certain parts. Thus, countermeasures against light radiation could be similar to that of EM pulses.

Light beam: Although light beams can be focused on a specific area of the circuit, it could be considered a weak version of laser beam attack, because the wavelength of the visible light used by light beam attacks covers more than ten times of the transistors' feature size. Different from light radiation, which is only applied to attack memories in common cases, a light beam can also flip the bits in FFs. One could apply redundancy or embed sensors into the sensitive functions in the design.

Laser beam: A laser beam attack is an improvement of a light beam attack as it can focus on one or part of a cell. Countermeasures include protective metal layers, metal shield, tamper sensors, redundancies, and sensors. Note, most of the recent approaches use backside attacks, hence one can use tamper-detection sensors and redundancies. Some prior research has demonstrated an approach to inject a one-bit fault in the AES module with a laser beam. However, such approaches are based on one critical assumption: different faults injected at a certain round of encryption will surely have different final outputs. Hence developing protection against such attacks is necessary.

Focused Ion Beam: Focused Ion Beam (FIB) is probably the most accurate and powerful fault-injection technique. FIB is able to inject multiple-bit faults at any location of interest in a circuit. When multiple-bit faults are injected, the conventional redundancy-based mitigations are no longer effective since the voter (typically TMR's voter) can also become corrupt. Countermeasures for this attack include spatial/temporal randomization where one would distribute the function execution and resources randomly in the circuit, thus, even with guaranteed multiple-bit faults, much more effort and analysis will be needed for attackers to break the circuits.

IV. An Overview of Security Assurance for Electronic Design Integration (SA-EDI) and its Applications

The importance of security in the electronic systems many of us rely on has become obvious to semiconductor design and manufacturing companies but most hardware security assurance practices in industry are still performed manually using proprietary methods, which can be expensive, time consuming, and error prone due to the ever-increasing complexity of systems. The Accellera IP Security Assurance (IPSA) Working Group [40], consisting of security and EDA experts is proposing a general and portable IP security specification standard describing the IP security concerns (threat model) and guidance to EDA vendors on how to produce security assurance collateral. This will enable tools for automation of security assurance of systems/subsystems/IPs with the main focus on security concerns with regards to IP integration. As IPSA will be releasing the Security Assurance for Electrical Design Integration (SA-EDI) standard in 2021, this paper introduces the collateral, methodology and a case study of application of the standard.

A. Introduction

Today's Systems on Chip (SoC) are very complex and consists of multiple reused IP blocks and software running on one or many embedded processors. This enables fast development and flexibility in applications by loading different software. Although Hardware Security Design Lifecyle (HSDL)

[40] methodologies for design for security and security assurance have been around for a decade, they are not widely used. If security is considered, it is often manual e.g., review of RTL files or penetration testing after fabrication. These methods are incomplete and may lead to security weaknesses in the final product. To be effective HW security needs to be addressed during the design and verification phase. However, a lack of standards hampers this effort.

Using third party IP saves lots of design and verification effort and is done in a majority of SoCs. However, the security implications of integrating IP can be huge and they are generally not understood. The IP creator doesn't know the application where the IP will be used and the IP integrator don't know what security risks are associated with the IP. Are there features or limitations in the IP that will compromise security in the application if not mitigated?

The Security Assurance for Electrical Design Integration (SA-EDI) work addresses this issue by defining a procedure and a format to capture security concerns in an executable specification to be delivered along with the IP. The IP provider documents security concerns in the IP that the user should be aware of. The IP integrator now knows the security risks in the IP and can mitigate them if applicable in his system. The standard also allows for the IP integrator to validate the integrity of the data delivered and to verify mitigations in the SoC.

For additional background, see the IP Security Assurance Standard whitepaper [41]. This section is organized as follows. Subsection IV.A gives an overview of SA-EDI followed by a case study in Subsection IV.B. Subsection IV.C concludes the paper with a short summary.

B. SA-EDI Overview

The objectives of the standard are, to improve trustworthiness of IP and IP providers by including a verifiable executable specification with the IP, to assist IP integrators in understanding the security concerns in the IP and to reduce security risk, and to accelerate tool development to enable security assurance. The methodology includes references to a knowledgebase that lists potential IP security concerns. One such database is MITRE's Common Weakness Enumeration (CWE) **Error! Reference source not found.**. It provides a common reference for identifying and describing weaknesses between IP provider and IP integrator.

As shown in Fig. 3, the standard defines four JavaScript Object Notation (JSON) [42] data objects that are included in the IP Bundle delivered to the IP integrator. The IP Bundle or collection of files also contains the IP design, verification code and documentation. The four objects are:

- Asset Definition
- Database
- Element
- Attack Point Security Objective (APSO)

The Asset Definition lists ports or storage locations in the design that may violate security objectives if they are modified or are readable outside the IP. The Database object specify which Security Weakness Knowledge Base (SWKB) is used by the IP provider. The Element object identifies which inputs can affect the asset and to which outputs the asset can leak information. It also contains references to related weaknesses e.g., CWE entries. The APSO object is created from the Element and includes security objective for the asset i.e., Confidentiality, Integrity or Availability. The Attack Points are the relevant inputs or outputs identified in the Element object with an optional condition specifying when the security objective is violated.

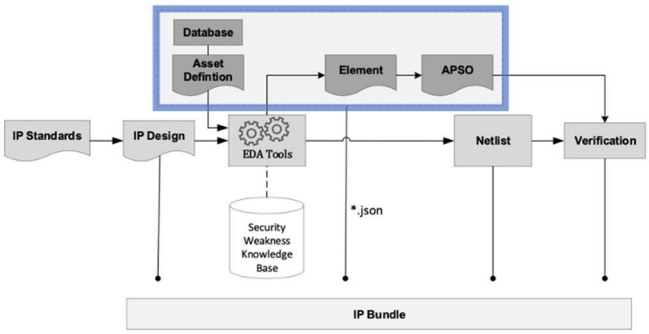

Fig. 3. Integrating the proposed security assurance flow with design and verification flow.

C. Case Study: SA-EDI Methodology Steps

There are several steps in the process of creating an SA-EDI compliant IP Bundle and additional steps for the IP Integrator to utilize the information. We will use a typical HW Root of Trust (HRoT) module (Fig. 5) as a case study to illustrate the steps and the data created.

The first step is identifying assets i.e., design elements that may have security implications for the IP integrator. The full hierarchical name of each asset is captured in Asset Definition objects.

The One Time Programmable (OTP) memory contains encryption keys, programmed at device manufacturing. Unauthorized access to these may have security implications for the product that integrate the IP. To make the HRoT IP more flexible and easier to debug, the OTP is readable through the debug interface and the hrot_iface interface. The IP provider captures this information in an Asset Definition object (Fig. 4) to make the IP integrator aware of the risks.

```
{
 "Name" : "hrot.otp.mem.mem_out",
 "Description" : "Fuse values contains keys for AES",
 "Family" : ["Storage"],
 "Type" : ["Data", "Critical", "Secret"],
 "Database_ID" : ["CWE VIEW: Hardware Design"]
}
```

Fig. 4. Asset Definition Object.

The second step is for the IP provider to decide which database to identify weaknesses related to the asset to use. In this case MITRE's CWE is used. A Database object (Fig. 6) is created that will be referenced in the Asset Definition object.

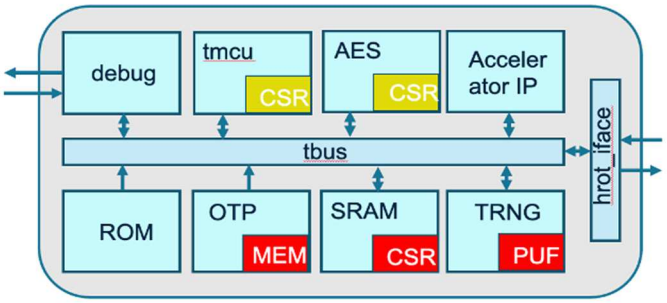

Fig. 5. Hardware Root of Trust Module Example.


```
{
 "ID" : "CWE VIEW: Hardware Design",
 "Description" : "A community developed list of hardware
weakness types.
https://cwe.mitre.org/data/definitions/1194.html",
 "URI" : "LocalDrive:/data/SWKB/cwe/cwe43.csv",
 "Version" : "4.3"
}
```


Fig. 6. Database Object.

The third step is to generate two Element Objects for the asset. Each Element respectively identifies which IP ports are in the fan-in and fan-out cone of the asset and may thus affect Integrity and Confidentiality of the asset. The Description, Family and Type fields in the Asset Definition are used to determine relevant CWE entries for the asset. Here, two applicable CWE entries are:

- CWE-1274: Insufficient Protections on the Volatile Memory Containing Boot Code **Error! Reference source not found.**
- CWE-1243: Exposure of Security-Sensitive Fuse Values During Debug **Error! Reference source not found.**

The Element Object (Fig.) is tool generated as manually determining fan-in and fan-out for an asset in a complex design is not feasible.


```
{
 "Asset Name" : "hrot.otp.mem.mem_out",
 "Direction" : "Output",
 "Security Weakness Reference" : ["CWE-1274","CWE-1243"],
 "Ports" : ["dbg_dout", "hrot_dout", "hrot_rdy"],
  "Parameters" : ["wd_top.COUNT_SIZE"]
 }
{
 "Asset Name" : "hrot.otp.mem.mem_out",
 "Direction" : "Input",
 "Security Weakness Reference" : ["CWE-1274","CWE-1243"],
 "Ports" : ["dbg_mode", "dbg_addr", "dbg_din", "hrot_addr",
"hrot_rwn", "hrot_din"]
}
```


Fig. 7. Element Objects.

The security objective for the fuse values is confidentiality. The fourth step is to manually create APSO objects for the corresponding Element object. Here, there are two different attack surfaces, the debug interface and the hrot_iface interface. They have different conditions under which Confidentiality of the asset will be violated so two APSO objects for the two cases (Fig. 8) are created.


```
{
 "Name" : "OTP_CONF_HROT",
 "Asset Name" : " hrot.otp.mem.mem_out ",
 "Security Objective" : "Confidentiality",
 "Description" : " Fuse values contains keys for AES that can be
read from hrot_iface",
 "Condition" : "(hrot_rwn==1) && (addr > 'h3500 &&
addr<'h3600)",
 "Security Weakness Reference" : ["CWE-1274"],
 "Attack Points" : ["hrot_dout,", "hrot_rdy" ]
}
{
 "Name" : "OTP_CONF_DBG",
 "Asset Name" : " hrot.otp.mem.mem_out ",
 "Security Objective" : "Confidentiality",
 "Description" : " Fuse values contains keys for AES that can be
read from debug interface",
 "Condition" : "(dbg_mode==1) && (addr > 'h3500 &&
addr<'h3600)",
 "Security Weakness Reference" : ["CWE-1243"],
 "Attack Points" : ["dbg_dout" ]
}
```


Fig. 8. Attack Point Security Objective Objects.

All SA-EDI data objects for the IP are now created. The last step is to include them in the IP Bundle delivered to the IP integrator. Before delivering the IP Bundle, the IP creator needs to verify that the attack points and conditions specified are correct and that no additional attack points exist.

The tool that generated the Element objects may also create information flow security rules or path verification assertions to be used in simulation or formal verification, respectively. For example, using the fields in the OTP_CONF_DBG APSO JSON object, it will generate a rule such as the following in an executable form.

"Information from hrot.otp.mem.mem_out will not flow to the dbg_dout port when dbg_mode is true and the address is in range ['h3600:'h3500]"

Fig. 9. Example of a tool-generated rule.

This rule is expected to fail since the APSO object communicates the attack points and conditions for how the security objective is violated. It is also easy to verify that information flow does not occur when the condition is false or that information doesn't flow to other outputs. Information flow rules to verify there is no information flow to any other output of the IP are easily tool generated.

The IP Integrator receives the IP Bundle and uses an EDA tool to generate Element objects from the IP RTL source and the Asset Definition object for each asset. The generated Element objects are compared to the Element objects delivered in the IP

Bundle and they are expected to match. If not, the integrity of the IP Bundle is violated and should be resolved before proceeding.

Next, the IP Integrator reviews the APSO objects and decides which ones are applicable to the product being designed. The debug port of the SoC will not be connected in the end product so the security concern communicated in the OTP_CONF_DBG object does not need to be mitigated. An SoC-level APSO object communicating the attack points to the SoC user may be created. The attack points identified by the OTP_CONF_HROT object needs to be mitigated. The IP integrator implements access control in the module connected to the hrot_iface interface to prevent any read of the AES keys from outside the HRoT module. To verify that the mitigation is effective, the integrator generates a similar information flow rule from the APSO object as was used by the IP provider. The only difference is that the destination is an SoC port or signal. When verifying the rule in SoC level simulations, it is expected to not fail since the leakage was mitigated. This ensures that no security weaknesses are introduced when the IP is integrated in the SoC.

D. *Section Summary*

We have seen how the required SA-EDI data objects are created by the IP Provider and how both the IP Provider and IP Integrator can use them for verifying security requirements of the IP and system. It lowers the risk for the IP Integrator since the security concerns of the IP are known and can be mitigated. Using SA-EDI is easy since data objects and verification code can be automatically generated by EDA tools.

V. SUMMARY

We gave an overview of our CAD for Security journey and how we planned to bring out our vision to reality to have proper EDA support for Design for Security and Security Assurance. Although, the industry have progressed from the stage that security assurance was considered overhead to the current stage that it is an essential part of the product development, the CAD for Security supporting systematic, scalable, and comprehensive Design for Security and Security Assurance has just started evolving for a while. We expect that with collaborative research and development among academic and industry experts: (a) it will get to a level that we have standard and portable security models to be understood by development team and EDA tools, (b) EDA vendors will provide multiple tools and many options for Design for Security and Security Assurance (similar to their portfolio of tools Design for Test and Validation), (c) design houses will have established flows to use the EDA tools with acceptable overhead to hit time-to-markets, and (d) product design and development teams will be security aware and use the security tools and standards to model the security of their designs approbatory, built security into their products, and perform the security assurance required easily and efficiently.

We introduced a couple of solutions to contribute to the maturity of CAD for Security tools. Solutions like CWE-IFT leverage information flow tracking verification tools to cover CWE's to integrate into a property driven hardware security design flow. Standards such as SA-EDI can be used to create portable threat models to be handled by EDA tools for providing better security assurance for IP's and their integration within a subsystem or a system. Solutions like SOFI based on security-properties for assessment and mitigation of advanced attacks become available to product development teams.

There is still a long way to get to our destination and we would invite other likeminded experts who believe on creating systemic CAD for Security solutions to address the needs of the future, to join us.

REFERENCES

[1] S. Bhunia and M. Tehranipoor, Hardware Security - A Hands on Learning Approach, S. Bhunia and M. Tehranipoor, Eds. Morgan Kaufmann, 2019

[2] M. Tehranipoor and C. Wang, Introduction to Hardware Security and Trust. Springer Publishing Company, Incorporated, 2011.

[3] H. Khattri, N. K. V. Mangipudi and S. Mandujano, "HSDL: A Security Development Lifecycle for hardware technologies," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, San Francisco, CA, USA, 2012, pp. 116-121, doi: 10.1109/HST.2012.6224330 .

[4] R. Cammarota, S. Aftabjahani, et al., Security and Privacy Chapter, *Semiconductor Research Opportunities – An Industry Vision and Guide*, Semiconductor Research Corporation and Semiconductor Research Association, 2017, pp. 45-51. [Online]. Available: Semiconductor Research Opportunities – An Industry Vision and Guide

[5] M. Tehranipoor, R. Cammarota, S. Aftabjahani, et al., "Chapter 3: Microlectronics Secuity and Trust - Grand Challenges", *TAME: Trusted and Assured MicroElectronics Working Group Report*, Dec., 2019. [Online]. https://dforte.ece.ufl.edu/wp-content/uploads/sites/65/2020/08/TAME-Report-FINAL.pdf

[6] D. Gardner, P. Ramrakhani, S Jeloka, P. Song, C. Vishik, S. Aftabjahani, R. Cammarota, M. Chen, A. Xhafa, J. Oakley, and D. Yeh, Research Needs: Trustworthy and Secure Semiconductors and Systems (T3S), Semiconductor Research Corporation, 2019. [Online]. Available: https://www.src.org/program/grc/t3s/research-needs/2019/2019-t3s.pdf .

[7] M. Abadir and S. Aftabjahani, "An Overview of the International Microprocessor/ SoC Test, Security and Validation (MTV)Workshop," in *2019 IEEE International Test Conference (ITC)*, Washington, DC, USA, 2019, pp. 1-2, doi: 10.1109/ITC44170.2019.9000128..

[8] M. Abadir and S. Aftabjahani, "An Overview of the International Verification and Security Workshop (IVSW)," in *2019 IEEE International Test Conference (ITC)*, Washington, DC, USA, 2019, pp. 1-2, doi: 10.1109/ITC44170.2019.9000165.

[9] Welcome to Trust-Hub, *Trust-Hub*. Accessed on: Apr. 08, 2021. [Online]. Available URL: https://www.trust-hub.org

[10] "CAD/IP for Security," *Trust-Hub*. Accessed on: Apr. 08, 2021. [Online]. Available URL: https://www.trust-hub.org/#/cad-ip-sec/cad-solutions

[11] "The Vulnerability Database," *Trust-Hub*. Accessed on: Apr. 08, 2021. [Online]. Available URL: https://www.trust-hub.org/#/vulnerability-db/physical-vulnerabilities

[12] CAD for Assurance, https://cadforassurance.org

[13] H. Foster, Applied assertion-based verification: An industry perspective, Now Publishers Inc., 2009.

[14] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, Werner, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, et al, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 973–990, 2018.

[15] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Hass, M Hamburg, M. Lipp, S. Mangard, T Prescher, et al, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[16] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," in *IEEE Design & Test of Computers*, vol. 18, no. 4, pp. 36-45, July-Aug. 2001, doi: 10.1109/54.936247.

[17] H. Chockler, O. Kupferman, and M. Vardi, “Coverage metrics for formal verification,” in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pp.111–125, 2003.

[18] W. Hu, A. Althoff, A. Ardeshiricham and R. Kastner, "Towards Property Driven Hardware Security," in *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, Austin, TX, USA, 2016, pp. 51-56, doi: 10.1109/MTV.2016.12.

[19] Common Weakness Enumeration (CWE). Accessed on: Apr. 08, 2021. [Online]. Available URL: https://cwe.mitre.org/

[20] “CWE View: Hardware Design,” *Common Weakness Enumeration (CWE)*. Accessed on: Apr. 08, 2021. [Online]. Available URL: https://cwe.mitre.org/data/definitions/1194.html

[21] M. Clarkson, F. Schneider, “Hyperproperties,” in *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[22] W. Hu, A. Ardeshiricham, and R. Kastner, “Hardware Information Flow Tracking,” in *ACM Computing Surveys*, 2021.

[23] *Radix Coverage for Hardware Common Weakness Enumeration (CWE) Guide.* Accessed on: Apr. 08, 2021. [Online]. Available URL: https://tortugalogic.com/wp-content/uploads/2020/03/RadixCWEGuide_20210126.pdf

[24] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, “Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures,” in *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[25] M. Karpovsky and A. Taubin, “New class of nonlinear systematic error detecting codes,” in *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818–1819, 2004.

[26] M. Hutter, J. Schmidt, and T. Plos, “Contact-based fault injections and power analysis on rfid tags,” in *2009 European Conference on Circuit Theory and Design*, pp. 409–412, 2009.

[27] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz, “Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller,” in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 77–88, 2013.

[28] D. F. Kune, J. Backes, S. S. Clark, D. Kramer, M. Reynolds, K. Fu, Y. Kim, and W. Xu, “Ghost talk: Mitigating emi signal injection attacks against analog sensors,” in *2013 IEEE Symposium on Security and Privacy*, pp. 145–159, 2013.

[29] S.Tajik and F. Ganji, “Artificial Neural Networks and Fault Injection Attacks”, arXiv:2008.07072, 2021.

[30] A. Nahiyan, F. Farahmandi, P. Mishra, D. Forte, and M. Tehranipoor, “Security-aware fsm design flow for identifying and mitigating vulnerabilities to fault attacks,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 6, pp. 1003–1016, 2019.

[31] R. E. Lyons and W. Vanderkulk, “The use of triple-modular redundancy to improve computer reliability,” in *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[32] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, “An efficient hardware-based fault diagnosis scheme for aes: performances and cost,” in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2004. DFT 2004. Proceedings., pp. 130–138, 2004.

[33] B. Ning and Q. Liu, “Modeling and efficiency analysis of clock glitch fault injection attack,” in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 13–18, 2018.

[34] N. Timmers and C. Mune, “Escalating privileges in linux using voltagefault injection,” in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 1–8, 2017.

[35] M. Dumont, M.Lisart and P. Maurine., “Electromagnetic Fault Injection : how faults occur ?,” in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, 19 September 2019.

[36] H. Wang, Q. Shi, A. Nahiyan, D. Forte, and M. M. Tehranipoor, “A physical design flow against front-side probing attacks by internal shielding,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2019.

[37] H. Wang, H. Li, F. Rahman, M. Tehranipoor, and F. Farahmandi, “SoFI: Security Property-Driven Vulnerability Assessments of ICs Against Fault-Injection Attacks,”in *IEEE Transactions on Computer-Aided Design (TCAD), 2021*.

[38] Loic Zussa, Amine Dehbaoui, Karim Tobich, Jean-Max Dutertre, Philippe Maurine Ludovic Guillaume-Sage, Jessy Clediere, Assia Tria, “Efficiency of a Glitch Detector against Electromagnetic Fault Injection,” in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 21 April 2014.

[39] Dennis R.E. Gnad, Fabian Oboril, Saman Kiamehr, Mehdi B. Tahoori., “Analysis of Transient Voltage Fluctuations in FPGAs,” in *2016 International Conference on Field-Programmable Technology (FPT)*, Xi'an, 18 May 2017.

[40] Accellera IP Security Assurance Group, Accellera System Initiative^{tm}. Accessed on: Apr. 08, 2021. [Online]. Available: https://www.accellera.org/activities/working-groups/ip-security-assurance

[41] Brent Sherman, Mike Borza, Jonathan Valamehr, Sohrab Aftabjahani, et al., “IP Security Assurance Standard”, September 4th , 2019. Accessed on: Apr. 08, 2021. [Online]. Available: https://www.design-reuse.com/articles/46877/ip-security-assurance-standard.html

[42] Introducign JSON (JavaScript Object Notation). Accessed on: Apr. 08, 2021. [Online]. Available: http://www.json.org/