AFRL-RY-WP-TR-2022-0191

AUTOMATED RULE CHECKING FOR HARDWARE SECURITY (ARCHS)

Nusrat Farzana Dipu
University of Florida

OCTOBER 2022
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2022-0191 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//Signature//

CHRISTOPHER A. BOZADA
Program Manager
Aerospace Components & Subsystems Division

//Signature//

GENE M. WILKINS, Lt Col, USAF
Deputy Chief, Aerospace Components & Subsystems Technology Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
October 2022	Final	3 March 2020	31 December 2021

4. TITLE AND SUBTITLE

AUTOMATED RULE CHECKING FOR HARDWARE SECURITY (ARCHS)

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8650-20-2-7014	N/A	62716E
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	Y23D

6. AUTHOR(S)

Nusrat Farzana Dipu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Florida Division of Sponsored Research
1523 Union RD RM 207
Gainesville FL 32611-1941

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'SREPORT NUMBER(S)
Air Force Research Laboratory, Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command, United States Air Forces	Defense Advanced Research Projects Agency (DARPA/MTO) 675 North Randolph Street Arlington, VA 22203	AFRL/RYD	AFRL-RY-WP-TR-2022-0191

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

This material is based on research sponsored by the Air Force Research Lab (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-20-2-7014.The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Labs (AFRL), the Defense Advanced Research Projects Agency (DARPA) or the U.S. Government. Report contains color.

14. ABSTRACT

The Automated Rule Checking for Hardware Security (ARCHS) program aims to develop a comprehensive framework to analyze security vulnerabilities in System-on-Chips (SoCs) including potential untrusted third-party intellectual property (IP) from register-transfer level (RTL) to layout. The objective of ARCHS is to have an extended and holistic solution for the next generation microelectronics by incorporating security properties to the different levels of complex hardware designs. ARCHS focuses on developing security aware properties and rules to systematically identify vulnerabilities at pre-silicon design stages; assessing the severity level of vulnerability with design level metrics, rules and threat levels; providing design level fixes with automation and integration of tool sets to protect security assets and mitigate security vulnerabilities at early design stages; guiding designers to trade off risk vs effort assessment based on the security level.

15. SUBJECT TERMS

security property; security rule; security asset; security vulnerability; threat model; system-on-chip (SoC)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	43

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Christopher Bozada	

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Table of Contents

Section **Page**

List of Tables ii
1 TASK 1: ESTABLISHING SECURE DESIGN INFRASTRUCTURE 1
2 TASK-2: DEVELOPMENT OF NOVEL SECURITY RULES AND METRICS 4
3 TASK-3: DEVELOPMENT OF SECURITY AWARE AUTOMATED CAD TOOLS 10
4 SECURITY PROPERTY DATABASE 22
5 SECURITY RULES: 36
6 SUMMARY: 37
LIST OF SYMBOLS, ABBREVIATION, AND ACRONYMS 38

List of Tables

Table **Page**

Task 1.1 : Asset and Vulnerability Space Analysis Against Respective Threat Models..... 1
Task 1.2: Identification of Security Assets 2
Task 1.3: Development of Vulnerability Database..... 3
Task 2.1: Formal Representation of SECURITY & Risk-aware Rules..... 4
Task 2.2 : Metric Development 6
Task 2.3: Comprehensive Set of Formally Defined Properties, Rules & Metrics for Quantitative Security Assessment 9
Task 3.1: Development of Tools and Methodologies for Checking Rules/Properties 10

1 TASK 1: ESTABLISHING SECURE DESIGN INFRASTRUCTURE

Task 1.1 : Asset and Vulnerability Space Analysis Against Respective Threat Models	
Abstraction Level	Task Update
RTL	• Literature reviews have been done on diverse attack and relative vulnerabilities for developing security rules and properties. For example, literature reviews have been done on- • The security vulnerabilities introduced due to power and timing optimization during logic synthesis. • Power side channel attack related vulnerabilities for micro-controllers. • Side channel attack resilient Simon threshold implementations. Simon threshold implementations are the designs on which masking schemes are applied to protect the intermediate values of the design. • Post-silicon side-channel assessment techniques like TVLA, SNR and masking protection. • Countermeasures by increasing noise variation at amplitude domain. • Machine learning-based delay predication to address the potential scalability issues. • Post-quantum cryptographic implementations and their side-channel analysis. • Cache side channel attacks. • In-house documentations to protect AES implementation. • Quantitative Information flow analysis for security path verification. • RSA and SHA algorithms to develop security properties. • Existing tools to identify finite state machines of an RTL design. • Identification and mitigation of vulnerabilities in FSM Literature review on SoC security verification using property checking. • Multiple lightweight AEAD (Authenticated Encryption with Associated Data) implementations including ASCON, COMET, Romulus, etc. • CWE database on hardware vulnerability to categorize the gathered information into system level and design level vulnerabilities. • CWE database to get understanding of vulnerabilities which may be present in Verilog HDL based RTL. • RTL security properties published in Trust-Hub property database so far. • Good coding styles designers follow, and bad styles designers try to avoid for writing better RTL in terms of optimization performed in synthesis process.
GATE	• Literature reviews have been done on diverse attack and relative vulnerabilities for developing security rules and properties. For example, literature reviews have been done on- • Quantitative Information flow analysis for security path verification. • Side channel vulnerability introduced due to scan chain based DFT infrastructures. • Existing tools to identify finite state machines of a gate level netlist. • We analyzed the fault-injection vulnerability in the implementation of SHA256 controller and found the vulnerability in binary state encoding techniques. An attacker can exploit the vulnerability cleverly by violating the setup time of some state flip-flops and getting into a protected state from unauthorized states.

Task 1.2: Identification of Security Assets	
Abstraction Level	Task update
RTL	We have defined security assets in two categories- i) primary and ii) secondary. Those that are the ultimate target for protection as deemed by the designer are defined as primary asset. On the other hand, the infrastructures that require protection to ensure the security of the primary assets during rest or transition, are defined as secondary asset. We have developed a "Secondary asset identification" tool to identify the secondary assets in a given design. Based on the user input e.g. primary asset, trusted observable outputs and untrusted observable outputs, the tool performs forward and backward structural analysis to find out common components between the primary asset and trusted observable outputs. Then, the tool performs different security vulnerability analysis to prune out the common components which are not susceptible to information leakage, power side channel or fault injection vulnerability. Thus, a short list of common components will be filtered, and we will consider them as secondary assets to protect. We also developed a "Register to signal converter" tool as a part of the "Secondary asset identification" tool to extract information about RTL signals and utilize them for other CAD tools. We have fully automated the tool and utilized it for vulnerability case studies of MSP430 and AES designs. We applied security asset identification tool to a sub-system of CEP SoC. A paper has been published in VLSI teste conference on secondary asset identification framework. We are working on to extend the paper to a journal version. We have utilized security asset identification tool to find out secondary asset for different lightweight cryptographic applications. We will use the findings in the result section of the journal paper-under development. We have integrated the secondary asset identification tool with a graphical user interface (GUI). We have derived security attributes which are the features- important to define secondary assets out of immense number of supporting components. The security attributes are- i) Close Correlation to Primary Asset, ii) High Controllability and Observability from Primary I/O, iii) Inheritance of Vulnerable characteristics.
GATE	For mapping security assets, identified at RTL to gate level, we have developed a script to parse the Verilog code of a design to identify each input, register, wire and output of the design along with the length of these signals. We developed a basic script that can parse the Verilog to identify the syntax using 'pyverilog' library.

Task 1.3: Development of Vulnerability Database	
Abstraction Level	Task update
RTL	We have enlisted the vulnerability database focusing on- • Test and debug infrastructure • Encryption mechanism • True random number generator • Controller design • Hardware interconnect • Exception handler • Locking mechanism • Comparison logic • Different hardware logic • Common core and computation issues • Memory and storage issues • Cache side channel issues • Processor
GATE	We have enlisted the vulnerability database focusing on- • Test and debug infrastructure • Encryption mechanism • State encoding techniques of FSM. • Vulnerabilities that can happen in any design example.

2 TASK-2: DEVELOPMENT OF NOVEL SECURITY RULES AND METRICS

Task 2.1: Formal Representation of SECURITY & Risk-aware Rules	
Abstraction Level	Task Update
RTL	• We have developed 67 security properties focusing on- • Test and debug infrastructure • Encryption mechanism • Controller design • Hardware IP interconnect • Exception handler • Locking mechanism • Comparison logic • Different hardware logic • Common core and computation issues • Memory and storage issues • Cache side channel issues. • Processor issues • We have published a security property database on Trust-hub website. To do so, we have converted 28 RTL security properties into packages with appropriate design example. • We have developed 5 security rules to be followed by RTL designers to make their written Verilog HDL based RTL robust in terms of security. Violations of these rules being developed so far will result in bad RTL implementation in terms of security in later implementation stages of the VLSI design flow. These rules are based on bad coding styles in terms of security an RTL designer may use unintentionally. The RTL static analyzer tool being developed on detection of violations of these RTL security rules will provide security warnings to the RTL designer if the RTL fails to comply with these. • We developed a tool named as "FSM transition probability extraction tool". For a given FSM design, the tool can obtain the transition probability for each state.
Gate level	• We have developed 31 security properties for gate level designs focusing on- • Test and debug infrastructure • Encryption mechanism • State encoding techniques of FSM. • Vulnerabilities that can happen in any design example. • We have published a security property database on Trust-hub. To do so, we have converted 7 gate level security properties into packages with appropriate design example. • We have identified two important properties which must followed by flip-flops constituting state register of an FSM for successful identification of logic gates forming control FSM in an unstructured flattened gate level netlist. These observations have been made after manually analyzing lots of synthesized flattened gate level netlist containing one or multiple FSMs. Synthesis of the known RTLs were performed using

	Cadence Genus and Cadence's free 45 nm PDK's technology library 'fast_vdd1v0_basicCells.lib' was used as the technology library for synthesis. • Property 1: State FFs present in a particular FSM state register must have pure combinational self-feedback loop between its data input and data output ports. This property ensures self-updating structure of state FFs prevalent in FSM structures. This property also ensures the self-influence behavior of state FFs. • Property 2: State FFs present in a particular FSM state register must influence other state FFs and should also be influenced by other state FFs. This property stands for cross-FF influence prevalent in FSM structures. Only true FSMs exhibit this property. False positive FSMs satisfying only property 1 can be removed using this property since accumulator register FFs satisfy property 1 but fails to satisfy property 2 completely. These observations are also backed up by FSM automata theory and by design and synthesis process of Mealy and Moore type FSMs. These observations are also backed up by FSM automata theory and by design and synthesis process of Mealy and Moore type FSMs.
Physical Layout	• 3 rules have been developed for physical layout level designs to detect or prevent probing attacks. First rule identifies the working frequency of the circuit path so that its total delay will exceed its maximum value and lead to faults in the design once being probed. The other two rules define the allowable probing delay vulnerability and allowable probing path vulnerability for the circuits with probing detector to prevent probing attack. • 2 rules have been developed to identify the maximum probabilities and glitch width respectively for the transition to occur to prevent glitches-based fault injection. • 1 model has been developed that helps designers who wants to build CMOS gates more robust against SEU effects or fault injection in the security field by studying the relations between the layout of cells, its different laser-sensitive areas and their associated fault model using laser pulse duration in the nanosecond range. • 1 model has been developed which helps designers who wants to achieve an optimal protection to the vulnerable nets in the design with internal shields in it while keeping the overhead (e.g., power, routing, etc.) in a proper range. • 1 model has been developed that enables designers to measure the vulnerability of the design to reroute attack based on added traces length metric.

Task 2.2 : Metric Development	
Abstraction Level	Task update

RTL	Metrics have been developed for-
	• Power Side-Channel Analysis: Success Rate (The number of successful attacks divided by the total number of performed attacks), SNR (Signal to Noise Ratio); SCV; KL-Divergence; TVLA.
	• Information Leakage: Property Coverage (The percentage of covered security properties that pass property checking using available toolsets), Property Severity (Importance/weight of a property in terms of security requirements)
	• Trojan Detection:
	  • Rareness of individual nets.
	  • Net controllability and observability.
	  • Statement hardness and signal observability
	• Secondary asset identification: Observation hardness: The percentage of a primary asset that propagates to a secondary asset.
	• KL Divergence metric: A metric to measure how one probability distribution differs from another. Switching activity of intermediate nets of a design differ for different input patterns. Thus, KL divergence can be utilized for power side-channel leakage comparison. Higher KL-divergence value corresponds to higher leakage.
	• Vulnerability factor of fault injection: The cumulative probability of propagating a fault from primary asset to intermediate registers (IRs) and from IRs to an untrusted output port.
	• Rare branch coverage: A metric for trojan detection. The Rare Branch coverage is a sorted bit-vector in which each bit corresponds to a branch statement in the C++ translation of the RTL model. The sorted bit-vector means that the Most Significant bits correspond to rare branches while the Least significant bits correspond to the functional branches. The process of distinguishing rare versus functional branches is done via random simulation of the design for a sufficient number of cycles and keeping account of how many times each branch statement is observed in the random simulation traces. Each branch has a counter and based on the sorting that happens on the counter, the place of that branch in the bit-vector is defined. This metric is used to prune the search space by eliminating the test-cases that have similar Rare Branch coverage metrics and indicate moving towards the same target.

Gate level	Metrics have been developed for-
	• Information Leakage: Property Coverage (The percentage of covered security properties that pass property checking using available toolsets), Property Severity (Importance/weight of a property in terms of security requirements)
	• Trojan Detection:
	  • Rareness of individual nets.
	  • Net controllability and observability.
	  • Statement hardness and signal observability

	• **Fault Injection: Vulnerability factor of fault injection** (Measurement of the overall vulnerable transition of the FSM to fault injection attacks), **Susceptibility factor** (Quantitative measurement of the vulnerability of each transition of the FSM to fault injection attacks) • **Secondary asset identification: Observation hardness:** The percentage of a primary asset that propagates to a secondary asset. • **KL Divergence metric**: A metric to measure how one probability distribution differs from another. • Switching activity of intermediate nets of a design differ for different input patterns. Thus, KL divergence can be utilized for power side-channel leakage comparison. Higher KL-divergence value corresponds to higher leakage. • **Vulnerability factor of fault injection:** The cumulative probability of propagating a fault from primary asset to intermediate registers (IRs) and from IRs to an untrusted output port. • **Fault Injection Feasibility (FFI):** FFI metric identifies whether an FSM encoding is vulnerable to the fault-injection attack. Given all the normal and protected states' encoding information in an FSM, the metric is calculated as follows. $\text{FIF}=\prod_{i=0}^{n-1}(S_{yi}\Theta\, S_{pi})\mid (S_{xi}\Theta\, S_{pi})$ Where "x" and "y" represent the present and next state respectively. "S" represents the bit of ith position in the encoding. Depending on the value of FIF, the fault-injection vulnerability in a certain encoding is identified as follows. $FIF=\begin{cases}0, Not\ vulnerable\\1, vulnerable\end{cases}$ • **FSM Transition Probability:** 1 metric is currently under development. This metric helps measure the vulnerability of each transition in the FSM design to fault injection attack. • **FSM Probability Metric of Registers:** This metric calculates probability of being finite state machine for a register present in the input unstructured flattened gate level netlist. This metric considers both two properties of state registers enlisted in Task 2.1 which must be strictly followed by a register to be considered as an FSM successfully. For true FSM registers, this metric should have a value of 100%. For false positive FSM registers, the value of this metric may go up to 65% from a very low percentage value but will never reach 100%. For registers failing to satisfy both property 1 and 2, will have a metric value of 0%. This metric can clearly distinguish between a real FSM register and a false-positive FSM register. Accumulator type registers whose FFs satisfy property 1 only but fails to satisfy property 2 properly are treated as false-positive FSM registers in our consideration.
Physical Layout	• 2 metrics have been developed based on shield coverage, shield security and allowable exposed area in a design. A design may contain protective metal shields and mashes in the front side. The shield coverage metrics determines which layers are optimal for two-layer parallel shield. Shield security metric and exposed area metric has been extended to backside which were defined for frontside. • 3 metrics have been developed based on probing resistant working frequency, probing delay vulnerability, and probing path vulnerability in a design.

	For the designs with probing detector, probing delay vulnerability and probing path vulnerability metrics are proposed to assess the design's vulnerability to probing attacks in terms of the number of vulnerable nodes and paths respectively. For the designs without probing detector, probing resistant working frequency metric is proposed to identify the working frequency of the circuit path so that faults will appear once its value is exceeded. • 1 metric, *added traces length*, has been developed based on the traces added to the design to perform reroute attack. It is proposed to evaluate reroute attack difficulty on different shield structures based on the calculation of added traces length. The number of vias, number of traces, and total length of added traces are calculated in this metric. The longer the added traces, the more time and complexity to perform reroute attack. • Utilizing two metrics, transition event probabilities and glitch width, glitches-based fault injection difficulty can be evaluated. The smaller the probabilities and glitch width is, the more time and complexity to perform glitches-based fault injection. • TLS setup time violation metric is proposed to measure the amount of time required to be added when the TLS is applied to the fan-in of a flip-flop. • 1 metric has been developed. This metric is proposed against fault injection attack with constraints on the way that state flip-flops are distributed in the physical layout. • 2 metrics have been developed. These metrics are proposed to measure the FSM's susceptibility to fault injection attacks.

Task 2.3: Comprehensive Set of Formally Defined Properties, Rules & Metrics for Quantitative Security Assessment	
Abstraction Level	Task update
RTL	• All metrics developed for secondary asset identification tool, have been formally represented in equations with necessary parameters. • For FSM transition probability extraction tool, the whole process has been automated with Matlab, Tetramax and Design Compiler. • In total, 35 security properties have been translated into assertion and checked using formal verification tool.
Gate level	• All metrics developed for secondary asset identification tool, have been formally represented in equations with necessary parameters. • Out of 25 security properties, 11 have been translated into assertions and checked using formal verification tool. • Vulnerability factor metric is defined based on the path delay distribution. Maintained/ violated transition refers to the transition that needs to be maintained/violated in a fault injection attack. This metric is defined as the probability of the event that, when the clock period is shorted to a certain value, the delay of the violated transition is larger than the clock period and the delay of the maintained transition is smaller than the period.
Physical Layout	• We have summarized the equations for the rules and metrics mentioned in 2.1 and 2.2, and they are formally presented using necessary parameters. • State flip-flops distancing metric is proposed against fault injection attack. The main idea is this, distribute state flip-flops with some distance away from each other in the layout to increase the difficulty of the attack. • Vulnerability Metric is defined as the FSM's susceptibility to x laser-based faults in one clock cycle. For example, for the example transition, 001 000, assuming laser fault capability (x) = 1, then vulnerable states (VS_x) are the states that are susceptible to x laser-based fault, VS_x=1={001,010,100}. So, $VM_{x=1}=\frac{|VSx=1|=3\{001,010,100\}}{\text{Total states}=8}$ • Spatial Vulnerability Metric is defined as, the FSM's susceptibility to x laser-based faults in one clock cycle given that F is the FF set, W is the laser set, and the possible FF attacks: $SVM_x=\frac{|UAA \in f(W) \subseteq F|}{Total\ state\ in\ the\ FSM\ design}$

3 TASK-3: DEVELOPMENT OF SECURITY AWARE AUTOMATED CAD TOOLS

Task 3.1: Development of Tools and Methodologies for Checking Rules/Properties	
Abstraction Level	Task update
RTL	RTL-PSC Test generation Tool: The goal of PSC-TG tool is to assess the power side channel leakage of an RTL design. The tool consists of 3 parts: i) target variable identification, ii) pattern generation and iii) metric calculation. • Cadence Jaspergold SPV scripts have been developed for automatic target variables identification to search for those variables whose switching activities are sensitive to side-channel attacks. • Formal verification tool- Cadence Jaspergold FPV has been used to derive the patterns corresponding to different hamming distance of the target variables before and after specified clock cycle. • Synopsys Spyglass is used for RTL power estimate to calculate the metric SCV (side-channel vulnerability) which can reflect the vulnerability of the target design. We validated the effectiveness of our tool on AES-LUT and AES-GF implementations. We have also validated the effectiveness of our tool on parallel and bit-serialized Simon implementations. • We have extended our framework to evaluate the side-channel resistance of Simon threshold implementations. We propose to use pre-silicon non-specific TVLA to analyze the designs by generating a fixed pattern and a group of random patterns for simulation. The toggle counts are collected at each clock cycle as the simulated power traces and used to calculate the first order and second-order t-statistical value to see if the threshold ($\|t\| = 4.5$) is exceeded. If the threshold value is exceeded, it means that the design is vulnerable to first/second-order side-channel attacks. • Developed a mechanism to determine how many patterns should be passed to claim a design is d-th order secure in t-test based on central limit theory and law of large numbers. • Validate the analysis results at RTL with the gate-level and post-silicon (FPGA) results. The pearson coefficient is calculated to demonstrate that there is a good correlation for RTL v.s. gate/FPGA, which means that our method can predict the results very well. • Analyzing Simon 2-share threshold implementation at first- and second order using our pre-silicon t-test methodology. The conclusion of pre-silicon analysis is consistent with the post-silicon results, i.e., first-order secure but second-order vulnerable. • Combining our test generation work with the previous simulation-based assessment technique to address the potential scalability issues. • Utilizing our technique to generate key pair for previous simulation-based RTL-PSC technique to address its accuracy problem. • Utilizing our formal technique to generate specific patterns for previous simulation-based approach to cross-check our results. • Covering more experimental designs of NIST 2nd lightweight cryptographic IPs (e.g., SAEAES) • Identifying vulnerable modules in the post-quantum cryptographic implementations like Saber and assess the side-channel leakage with our method.

- Analyzing more NIST 2nd round lightweight AEAD implementations like ASCON and COMET as well as make a comparison among their pre-silicon leakage.
- Utilizing our technique to generate key pairs for previous simulation-based RTL-PSC technique to address its accuracy problem.
- Utilizing our formal technique to generate specific patterns for previous simulation-based to cross-check our results.
- Working on the hardware (FPGA) setup for validating the pre-silicon side-channel assessment results of recently covered implementations like Simon, SAEAES, etc.
- Working on extending the tool to enable non-specific pre-silicon TVLA for scanning possible side-channel vulnerabilities of arbitrary implementations at RTL. This TVLA should follow the methodology of its post-silicon counterpart instead of what we have done for the masked designs which only changes mask value and perform the t-test later.

<u>Trojan trigger pattern generation tool:</u>

We have developed a test generation tool to detect hardware Trojans hidden in RTL codes. We used Verilator to convert the RTL codes to C level. Test generation at C level can be done using an established formal verification tool, KLEE. To overcome the scalability issues with KLEE engine, we are incorporating a new method that tries to divide each clock cycle of execution into a separate symbolic session. The main idea is to execute the last cycle of execution symbolically using the concrete values for previous cycles that were generated in any number of last clock cycles. We are currently developing abovementioned platform. The concrete values are extracted from the KTEST files provided by KLEE. The steps done until today include:

- Decoding the syntax of KTEST files to extract the values.
- Implementing the iterative approach of reading files and concrete simulation.
- Implementing the platform to run an example on one of the AES Trojan inserted designs from trust-hub.
- Implementing the new approach for iterative symbolic execution at the last cycle of the execution.
- The Generic C++ Wrapper for Verilator classes is implemented that takes the test case numbers and unroll counter as its input.
- The Previous part that introduces symbolic values for each cycle of execution is finished and we are writing a paper about its improvement over other proposed methods.

To have an iterative unrolling of the RTL design in C++, we have a python script that issues the KLEE search for each cycle and selects test cases for concrete execution up to last cycle. Basically, with this iterative approach the problem of full space search using symbolic engine, that was exponential in its order with regard to size of the design, now translates to how good we can select the test cases for concrete execution in previous unrolling cycles. We wrote a paper on multi-cycle symbolic execution. The Result section of the paper includes benchmarks where we inserted Trojan into OpenRisc_1200 cache controller and introducing it to the toolchain.

Three main steps of this tool include: Test case Selection, Symbolic Execution, and test pool analysis. The selection is a simple method that favors new test inputs in order to explore a more diverse range of branches of the design and test pool analysis keeps track of the assertions that have been hit and the data extraction from files generated by KLEE.

	The Scalability of Hardware Trojan detection using Symbolic execution and specifically KLEE engine is the main reason for moving towards and iterative approach that concretely executes the design and faster unrolls the simulation to a desirable state that is closer to the rare branches which are our candidates for Trojan insertion. We developing some metrics for the iterative unrolling approach and also developed the main steps of the work in a proper manner using Python. In order to strengthen our claims over better performance and memory usage compared to bounded model checkers, we introduced the same benchmarks as used in our paper to the EBMC tool, which is an open-source model checker and gathered timing and memory information. The core reason behind getting a better performance while unrolling a design iteratively is the fact that concrete values selected in previous cycles are already pre-defined and this depends on how close to target, we select these concrete values in test case selection step. In order to formalize this selection, a metric is proposed that depends on static analysis of all branches inside the evaluation function. This metric is based on the distance of each branch to any target rare branch and is obtained from building the CFG of the C++ design and finding the shortest path between branches. For obtaining this CFG at C++ level, an static analysis tool is written in python that analyzes the function call tree and stiches each function's individual CFG in order to build the complete CFG of the design. Another metric has been developed to prune the test-cases that lead to same rare targets. This metric is based on rareness vectors of each concrete execution trace. The values of rareness vectors increases if we get close to the rare target branches based on a sorted bit-vector in which each MSB corresponds to the most rare branch and the LSB to the most functional branch. This way we have a vector associated with each test-case and this metric is used to prune the search space. We worked more on the iterative last cycle Symbolic execution. The Static Analysis of the C++ model of the design is used implemented and the C++ code is divided to basic blocks and the whole graph of the design is obtained from LLVM static analyzer. The graph is further analyzed to develop a metric. This metric is the distance of each branch statement to each of the target rare branches obtained from the random simulation. This metric in conjunction with rareness metric, are incorporated in the iterative approach source code in order to direct the execution of Symbolic engine by getting the design to a certain state that is closer to the rare targets. The coding of the iterative last cycle approach is close to an end and the plan is to introduce AES-Trojan benchmarks to the tool and evaluate the results. We finished the iterative Symbolic execution on AES Trojan inserted benchmarks. • The Static Analysis of the C++ code is done and the pruning of targets is included in the Python script. • The Iterative Approach is finalized and the metric obtained from static analysis of the conditional graph is incorporated in selecting best fitted test-cases for further expansion using KLEE engine. • The results of timing and memory measurements are attained for Trust-hub AES benchmarks which show improvement over previous approaches like EBMC and Conquest. However, we did not investigate trojans that are more complex and do not only depend on input values using the current version of the tool during our symbolic sessions. The reason for this is the fact that only regions of code that depend on symbolic values are

searched and code lines that are controlled by concrete values are only investigated in case of a satisfiable concrete assignment which is hard to get to using only random simulation. Our tool (SymbA) generated Triggers for the Symbolic input variables in reasonable time for AES Trojan infected designs from trust-hub but it failed in generating triggers for more complex trojans of WishBone bus architecture. The concrete variables contributing to the masking of some regions of the code should be identified and test should be generated to set them to the right values. For this purpose, the information flow at RTL was investigated. Tools like Goldmine and PyVerilog where used to generate the CDFG of any flattened design and finding the dependency amongst all the variables in the design. The CDFG generation is complete but the further analysis is being investigated with incorporating concepts like information flow and static analysis at RTL. There were also some minor problems with previous version of the tool that we solved. We have made it generic to all the benchmarks which were fixed and resolved.

Data-flow analysis is needed to help the Symbolic engine in reaching to the sections of the translated RTL model at C-level. This analysis will enable the user to define more symbolic variables and unlock the sections that are controlled by concrete variables in order to achieve better line coverage and detect the Trojan triggers. Two different Data-flow analysis tools were investigated -

- JasperGold static analyzer: This gives a full control and data flow analysis in form of a whole design graph that needs to be further analyzed to detect the actual variables involved in data-flow analysis.
- Goldmine: This tool uses PyVerilog compiler and has a static data-flow analyzer. Using this tool, the Data-flow is obtained in a readable format that is used for more analysis.

After having all the variables that are in the input cone of an output variable, all of them are defined Symbolic and this is done on RS232 benchmarks as an example prototype for now. The problem now translates to justify the intermediate symbolic variables through only input ports and coming up with full multi-cycle exploits to address how to achieve to the state of intermediate variables attained from symbolic execution, from a reset state of the model and only using changes in input port values.

Now, the problem is the difficulty of justifying the intermediate symbolic variables through only input ports and coming up with full multi-cycle exploits to address how to achieve to the state of intermediate variables attained from symbolic execution, from a reset state of the model and only using changes in input port values. SymbA tool suffers from the following issues:

- Test-bench generation was a manual task that should have been implemented at C-level. · The assertions are tailored only for Trojan detection and line coverage while there is possibility of targeting any KLEE assertion statements with some modifications in the platform.
- The Concrete values that are set only in the RTL code and are not controlled by inputs. Like embedded counter values.
- Static analysis that is not yet generic and leads to some errors like BFS not finding the closest nodes in case of bad parsing of Basic Block presentation.

The python script to generate the test-benches for both random and symbolic simulation has been programmed which gets the Verilog files as input and makes the test-benches

	in C. Also, smaller examples of how to translate SystemVerilog Assertions to KLEE-ASSERTION were investigated and the tool worked for a small FIFO buffer. Assertion Generation: Using the set of vulnerability database and the security assets identified, we want to generate assertions for a circuit to check if an asset leakage occurs. Assertion generation can be used to verify the design as well. By designing a tool that can generate complex assertions and then running them through formal verification, we can get a solid set of security properties for an asset. This assertion generation followed by formal verification can be integrated with the test generation, which complements each other. The counterexamples we obtain for failed assertions can be used to develop the tests further to include them, and better tests will give us better assertions for the asset. The design can be checked against the present vulnerabilities from the vulnerability database, and later new properties can be added which we have obtained through the testing and assertion generation. We started working on the automated assertion generation tool. Towards this goal, we worked on a high-level plan for the tool. By parsing the Verilog for various threat models, the framework will generate assertions. We started with the plans for Information Leakage, Access Violations and Extending assertions to Secondary assets. As part of this, we started working on a script to generate data flow for a design and then extract all control signals from the data path of the assets. We also started working on the part where we use these control signals to generate assertions for fault injection. We completed script to extract the control and controlled signals for an If-Else scenario in Verilog. We completed 90% of the script to extract the control and controlled signals for a Case Statement. Current version of tool can identify control signals for an input of primary and secondary assets. We integrated an assertion mining tool, that can mine for assertions based on the given design with minimum user input or provide a feature where the designer who has a better understanding of the design can provide a template to look for based on the previous design versions or his/her own knowledge. We are also looking forward to using structural analysis of the input RTL design to derive assertion relating to path propagation between signals. We incorporated a DFS algorithm in the structural analysis methodology to search for all possible paths from the Security Asset to the Observable Point at RTL level. We generated 21 paths from the asset(key) to output using the DFS algorithm then cross verified with JasperGold for one of paths. We have also been working on developing a roadmap for the automated property generation for pre-silicon security verification and developed various stages for the roadmap. We have developed a framework for Automatic Assertion Generation Tool. The framework uses the assets defined for each IP in the property database. We also plan to develop a library with various threat models for each IP module and then define templates for each of the IP to be used for assertion generation. We are working on incorporating the Automatic Assertion Generation Tool into Automatic Property Generation framework. Also we have been trying to improve some of the steps to be automatic. We have developed a script for identifying the control signals in a design. The script can parse a Verilog file and then identify If/Else and case scenarios in each design. It can then identify the control and controlled signals. Given the list of PAs and SAs from the designer, we then identify the signals that control the primary assets and secondary

	assets in the If/Else scenarios. This information of control signals can help the designer to protect them against fault injection attacks. We started working on the script for information leakage. We have the script ready for comprehensive methodology. It can generate Jaspergold SPV assertions that check for information leakage from secondary assets to untrusted observable points. We updated the security property database. We have developed a workflow on utilizing security property database for automatic property generation. We have prepared automatic assertion generation tool flow. We consider that the tool flow will have 4 parts Based on what type of assertion the user wants to generate. The assertion types will be- information-leakage based assertion, access control/fault injection type assertion, Trojan detection type assertion and mapped assertion from the security property database. For information leakage based assertion, after identifying all the primary and secondary security assets using SAIF, the tool can generate a set of assertions in the form of check security path from a source to a destination. If the targeted assertions are Access Control/Fault Injection- based assertions, then instead of generating assertions in security path checking format, we plan that- the tool will generate a CDFG of the design and from that CDFG, annotate the nodes where security assets are present as security nodes and from the control and data flow at that node, it will identify the control signals to formulate Access Control/Fault Injection- based assertion. If the targeted assertions are trojan detection-based assertions, then by using standard simulation techniques. we simulate a given design for a long period of time (~1 million clock cycles.). From the obtained simulation traces we try to identify the least controllable/ observable signals (with least switching in values). We can use metrics like "Rareness of individual nets"; "Net controllability and observability"; "Statement hardness and signal observability" . The rare signals are then marked as possible trojans triggers and we try to generate assertions for these signals to ensure desired functionality. Using a database of security properties available on Trust-Hub for various standard modules (AES encryption core, SHA-256 etc.). We can obtain a set of assertion templates for these security properties and map them to the same modules in our design and generate assertions for these modules using template-based mining. To implement the above-mentioned flow- We have been working on Control and Data Flow Graph of the design. To do so, we- • Extract VDGs and modify them to contain CDFG node information. • Construct Data/Control Dependency Variable Graphs. • Combine the procedural block CDFGs to get a fused CDFG for the module containing variable information. • Identify security critical nodes using primary and secondary assets given as input. • Using the DVDG/CVDG to extract relation between the nodes of CDFG. We are also developing necessary scripts for trojan based assertion generation. The trojan based assertion generation tool has two parts for trigger and payload. For trigger part identification, we have implemented a "Rare Signal Identification" flow. We implemented a "Trigger condition Identification" flow for Trojan Detection-based assertion generation for Automatic Assertion Generation tool. At first, we identified the rare signals from the flow developed from the switching activity format file. By negating the rare condition, we developed a set of assertions and fed to Jasper Gold tool to check if any counterexample can be achieved. The counterexample helps us to perform the

	Fan-in analysis of the rare signals and extract the rare conditions for which malicious implants in a design can be activated. Transition Probability Extraction Tool: Given an FSM design, our FSM transition probability extraction tool can first extract its state transition graph with the help of Matlab. Design Compiler is in charge of generating all the required files of the FSM design, such as its FSM report, containing state encoding information, state register names, etc. Then, Tetramax will report all the possible next states for each state, and their corresponding transition probability. We tested the FSM transition probability extraction tool on different designs and some modifications have been made to the tool, mainly on the way to calculate the probability, which is finished in MATLAB. Template-based RTL Assessment Tool: With increasing complexity of the design, the process of RTL design is getting much more sophisticated since the designer needs to augment functionality of his/her RTL quite often. It is very common now a days that industry grade RTL designs contain thousands of lines of codes written in Verilog HDL. Primarily, the RTL designer tries to make his/her RTL code optimal for getting an efficient design as much possible after gate level synthesis and other later implementation stages. In tradition, the designers keep only performance and area overhead of their designs in their minds, but security aspects are totally absent. It is because either the RTL designers have no idea on security vulnerabilities which their designs may face in later implementation stages of the chip design cycle, or they write vulnerability-prone RTL codes without knowing even. The main goal of this 'Template Based RTL Assessment Tool' is to make the RTL designers aware of design issues in their written RTL codes which may turn into security vulnerabilities in later implementation stages of the design flow. The main motivation for our work is that if the designer can clear security bugs in RTL level, then it will be very unlikely that security issues will be raised due to unsafe RTL coding styles. This tool is targeted to provide the designers warnings on certain security issues detected in their RTL codes and it will tell the designer which steps they need to take to make their RTL more secure. Our tool will read and parse the provided RTL written in Verilog HDL. It will first identify all the state register variables, inputs and output ports of the design. Then it will search for templates. These templates come from security rules which has been developed so far or being developed. Next templates match checking is performed. If the templates have been matched successfully, then the designer can be sure that the provided RTL is free from security bugs in this development stage otherwise they will encounter warnings from this tool to take appropriate steps according to the guidance provided by this tool. There are two inputs to our tool. One input will be the RTL written in Verilog HDL which may contain thousands of lines of codes. The other input will be the defined RTL security rules converted into templates. These rules are stored in a text file as a database of the rules and the users must have to go through it to know about the defined RTL security rules in details and be aware to implement in their design. The output of this framework is the security rule checking report in text format which provides the designer whether any security issues has been detected in the provided RTL or not. It also mentions the line numbers where security issues have been detected and the required action. Provided RTL written in Verilog HDL language will be fed to the PyVerilog tool as the input first. The PyVerilog toolkit is an open-source Python based Verilog HDL analyzer and it functions as an RTL code compiler in this framework. This toolkit generates intermediate representation of the RTL which is the main input to the RTL security rules checker tool. The intermediate representation of the RTL can be of three types: the AST report, control flow report and dataflow report. The secondary input to this

	rules checker tool is the provided RTL. Then the tool will perform template matching as described earlier and will generate the rules checking report in text format. This RTL security rules checker tool is a bunch of scripts written in TCL which implements those templates matching techniques individually (one template for one rule) in primary implementation phase. Later, it will be upgraded to have one template checking for multiple security rules. In the final implementation phase, this rules checker tool will be implemented using machine learning and AI techniques to achieve efficient implementation of the tool in terms of runtime. It is very important for the designer to have a look in the generated rules checking report since it not only tells the designer whether their provided RTL is free from security warnings or not but also guides them what he/she needs to do to achieve a secure RTL implementation according to the rules in this very early design stage. We successfully detect violations of RTL security rule no. 1 and 2 as mentioned in Task 2.1 in a provided Verilog HDL based RTL. Both these two rules are targeted to security critical control units. We have developed and implemented sophisticated parsing algorithms for this purpose. The tool is being written in TCL so that in can be incorporated with any type of CAD platform. For detection of violations of security rule 1 in the provided RTL, control flow report and dataflow report generated by PyVerilog were analyzed. A complex TCL script have been written to implement this sub-tool of the complete tool. The control flow report helped us to get all the state encodings and transitions present in the provided RTL. The dataflow report was analyzed to get the state variable names which correspond to their respective state encoding values. The set of protected states was taken as input from the user stored in a text file. After analyzing it, all the unprotected states were found. Finally, only unprotected to unprotected state transitions were considered and hamming distance between these two states was checked. If the hamming distance was greater than 1, a security warning was raised, and a report was generated indicating the user which state encodings failed to satisfy this rule. The user can then modify the state encoding to clear this security issue in the RTL. In short, detection of violations this rule in Verilog HDL based RTL is focused on detection of unsafe state encoding styles in RTL which may turn into security issues in later implementation stages of the design flow as the synthesized gate level netlist obtained from such bad RTL (in terms of security) will be prone to fault injection attacks. If the RTL designer clears this security bug, he/she will be sure at least in this scenario that the designed RTL won't suffer from fault injection vulnerabilities due to bad state encoding practice. For detection of violations of security rule 2, we have analyzed AST report generated from PyVerilog. This rule is focused on ensuring default statements where it is required. This rule also targets security critical control unit like rule 1. First, we have identified all the case statement variables of the design. Next, all the widths of these variables were found and from that the number of maximum possible cases were found. Finally, number of case statements in each of the blocks were calculated and it was checked whether there was a 'default' statement if the number of case statements was less than the maximum number of possible cases. If 'default' case was present, then no security flags were raised otherwise security issues were detected and the line numbers containing such case statements without 'default' was reported for the designer to take appropriate actions. Our tool can successfully handle nested case structures and it was the most difficult part of this implementation. It doesn't depend on how much deep the top 'case' structure is. Our tool starts analysis from the deepest 'case' structure and gradually moves upward in the hierarchy. For this reason, the search area gradually reduces with each iteration, and it improves runtime since we don't have to do searching again. We have performed search for only one time for a certain 'case' statement block. We have also been able to develop a top TCL script which acts as the main tool script. From this script, all the sub-tools for detecting violations of rule 1 and 2 (written in TCL) are called. For this reason, this tool can be

	easily extended to incorporate more security rules which are being developed gradually. This also indicates a good scalability of the tool. For a new rule being developed, we just need to develop and implement parsing algorithms and combine with the main script. We are currently working on developing sub-tools for detecting violations of other security rules so far developed. We are also working on enhancing our rules set. For this, we are exploring extensively good and bad RTL design practices in terms of security.
Gate level	<u>Property mapping tool:</u> We have developed of the security property tool for mapping security properties from RTL to Gate-level. Using in house script, the properties to check fault-injection vulnerability in the state encoding of SHA256 and RSA is generated. These properties have been used to expand the properties for RTL. We have worked on security equivalence checking of SHA256 and RSA: We have developed and verified security properties in RTL, then mapped the gate-level properties and rechecked the security equivalence. For example, we have identified a security property related to the "valid" signal representing the end of the hash computation. Each successful hashing must take the same number of clock cycles in RTL implementation, indicating that there is no possibility of skipping any round. As the finite state machine is introduced at the gate-level, more opportunity arises for an attacker to attack the design. We extended and expanded the AES Security properties from RTL to gate-level and formally verified it in Jaspergold. We also worked on the opencores RSA controller and SHA256 controller for the security property extension from RTL to gate-level. As a part of the property expansion, we checked the leakage of the asset after scan-chain insertion and verified the property in Jaspergold SPV. We found that the FSM register for both the AES and RSA controller is observable to the test output. We also developed an algorithm to expand the properties based on the fault-injection attack's possible don't-care state utilization. The algorithm takes the states and the transitions of a finite-state-machine and creates a list of the possible reachable don't-care states. Using these don't-care states, our tool creates additional properties by expanding corresponding properties in RTL implementation. Our algorithm considers the one-bit flip for the binary encoding and the two-bit flip for the one-hot encoding scheme. We developed a python script to implement the identification of the reachable don't-care state extraction algorithm. This script takes the state transition graph, state encoding, and the protected state as input and outputs the possible reachable don't-care states. We had considered a one-bit fault for the binary encoding and two-bit faults for the one-hot encoding during the extraction process. However, we plan to tackle more bit-fault in the future. We also developed a python script that can utilize the extracted don't care states and expand the RT-level access control properties to the gate-level properties. We extended and expanded security properties during the design transition from RTL to gate level. For now, we considered the DFT and the fault-injection vulnerability. Currently, we are interested in the vulnerability introduced due to followings in gate-level: 1. Insertion of Design-for-debug (DFD) 2. Design optimization for low power 3. Design optimization for increasing performance 4. Design optimization for area 5. Clock tree synthesis Towards this goal, we explored different possible security issues that might be introduced by the debugging structure. We found that the potential malicious

	hardware vulnerability can be introduced while the DFD insertion is outsourced to the third party. We found that the JTAG interface can be exploited to leak and modify the confidential information in the SoC. There is also a risk of malicious instruction implemented by the attacker in the instruction register of the IP wrapper. We plan to consider these issues in our security property extension and expansion module of our framework. To expand the security properties from RTL to Gate-level, one important task is to extract and analyze the finite state machine (FSM) at the gate level, since automated property mapping framework takes the FSM states and the encoding information as inputs and expands the mapped security properties to identify if any potential access control, don't-care, and Fault-injection vulnerabilities are introduced in the FSM. Automated Finite State Machine Extractor Tool: The main purpose of this tool currently being developed is to extract finite state machines from an unstructured flattened gate level netlist. First, an input RTL needs to be synthesized using Cadence Genus. Cadence's free 45 nm technology library file 'fast_vdd1v0_basicCells.lib' has been used as the technology file for synthesis. Flattening process was performed during synthesis so that the synthesized netlist does not contain any type of hierarchy. This results in a highly optimized netlist. We have been successful to develop a framework for the complete automation process of the tool. There are two important things that need to be provided as inputs to our tool. First one is the flattened synthesized netlist obtained using Cadence Genus synthesizer. Second one is the technology library which was used in the synthesis process. Technology library is required by our tool to understand which cells are present in the technology library, what are the types of the cells present in the library, what are the pins and types of the pins of the standard cells. This information is very critical for successful and reliable operation of our tool. In our developed framework, there are two main parts and each of them are very crucial for successful extraction of the FSMs. We are currently working extensively on the implementation of first part. The first part is the topological analyzer tool. The main purpose of this tool is to perform graph algorithmic operations on the highly unstructured complex gate level netlist to identify potential FSM candidates successfully. This tool is again divided into two sub-tools: 'Gate Level Netlist to Graph Representation Converter' and 'FSM Candidates Identifier'. The first sub-tool 'Gate Level Netlist to Graph Representation Converter' has been implemented using TCL. The main purpose of this sub-tool is to generate graph representation of the input unstructured flattened netlist composed of logic gates and interconnecting wires. The entire gate level netlist can be treated as a directed graph and this graph is cyclic in nature since it has multiple feedback loops composed of various types of logic gates present inside the netlist. All the logic gates present in the netlist is treated as vertices/nodes of the directed graph and the interconnecting wire from the output of one gate to the input of other gate is considered an edge of the graph. In this way, the netlist directed graph representation is constructed as collection of huge numbers of vertices pairs. The graph representation is stored in a text file for further processing. Types of the logic gates present in an edge will also be stored along with the gates. This also helps us to process information more efficiently in later stages. Implementation of this part was very tricky, and we have been able to implement a sophisticated TCL script for this sub-tool implementation. A sophisticated algorithm was developed for this purpose. Some minor bugs were detected, and they have been cleared successfully. We call the output of this sub-tool as connected components report. The second sub-tool 'FSM Candidates Identifier' has been implemented using Python since Python offers NetworkX library which is a collection of all most efficient graph

	algorithms developed so far. The main purpose of this sub-tool is to perform the main graph topological analysis of the input netlist directed cyclic graph. We are currently working on the complete implementation of this part. In our first implementation flow, we used Tarjan's algorithm for identifying strongly connected components since FSMs form strongly connected component regions in mathematical point of view. This narrows down our search area from a global region to a more localized region. However, the game does not end here. We need to get into much more localized region in order to successfully identify state registers present in FSMs. For this reason, we implemented Johnson's algorithm to identify all cycles present in the netlist graph. However, this is an NP hard problem and algorithmic complexity is too high for Johnson's algorithm. Due to this, Johnson's algorithm suffers from scalability issues and can't analyze medium and large sized netlists. The runtime is too much high to be acceptable. Moreover, it is really very complicated task to analyze a cyclic graph having hundreds or thousands of nodes. So, we needed to figure out an alternative way in order to improve runtime. We have addressed the problem using an indirect method to identify the feedback loops present in the state flip-flops of the FSM. We have decomposed the complete directed cyclic netlist graph into two directed acyclic sub-graphs. One is the sequential directed sub-graph which is composed of only sequential elements and few combinational gates connected to them. The other one is the combinational directed acyclic graph which is composed of all other combinational logic elements. This decomposition process improves runtime since both these two sub-graphs are acyclic in nature and we don't need to think about infinite recursion which may happen while analyzing large cyclic netlist graphs. We have indirectly identified all the flip-flops having pure combinational feedback between their inputs and outputs. These flip-flops are potential state FFs and the registers they form are potential FSM state registers. However, it is just one property for being considered as FSMs. Accumulator registers also exhibit this property, but they are not FSMs. So, we must have some distinguishing property which can separate FSMs from accumulators. In this case, property 2 helps us since FSM state FFs influence other state FFs and are also influenced by other state FFs. Only FSM state FFs exhibit this property strongly. Accumulators register FFs fail to satisfy this property properly. Based on these two properties, we have developed a metric named 'FSM probability metric' which calculates the probability of being FSM for a certain register. Based on this metric, gates in FSM region can be successfully extracted. Only FSM registers have this metric value of 100%. Accumulator registers may have value maximum up to 65-70% but this metric value never reaches 100% like the FSM registers. We have verified this result by analyzing large number of gate level netlists. We are currently working on the successful extraction of the gates in the FSMs. These gates include all the state FFs present in the FSM registers, combinational gates present in the self-feedback loop path and combinational gates present in the cross-FF influence paths since all these gates form the state transition combinational logic. After this process, part of the entire unstructured netlist which is responsible for FSM state transitions will be identified along with the FSM state FFs. By analyzing this small part of the netlist using some sort of Boolean Function Analyzer tool, conditions of the state transitions and the state encoding information will be found.
Physical Layout	<u>Exposed area visualization tool</u> A complete set of scripts have been developed for visual outputs of our assessment tools. We implemented of rule 5 and 7, which describes the proper working frequency of the circuits to detect the FIB and identify the vulnerable paths. Parasitic information has now been extracted from the design layout with the assistance of Synopsys StarRC, with

	which, physical-level static timing analysis can be implemented using Synopsys Primetime. Critical path delay has been identified. Next, fault resistant frequency will be applied to determine the paths with violation. We implemented rule 15, used to determine the optimal number of shield gates and size of keep out region given constraints on design overhead and exposed area. We have established the model and are currently verifying it with several designs. Interpolating and fitting function has been used to depict the relation between size of keep out region, number of shield gates and design overhead. We have finished the implementation of rule 5 and currently working on the rule 7. For rule 5, parasitic information will first be extracted from the design layout with the assistance of Synopsys StarRC, with which, physical-level static timing analysis (STA) will be implemented using Synopsys Primetime. Critical path delay will be identified with STA results, and fault resistant frequency can then be determined. We have finished the implementation of rule 6 and 7. For rule 6, the gate-level implementation of calibratable detector structure is achieved using 32nm library, which will be attached to a gate to determine if extra capacitance has been added. For rule 7, parasitic information will first be extracted from the design layout with the assistance of Synopsys StarRC tool. Then, static timing analysis will be performed using Synopsys PrimeTime tool, with which, probing resistant frequency will be determined and vulnerable paths will be identified. We have finished the implementation of rule 11 and 13. For rule 11, using Synopsys HSPICE tool, we will vary clock frequency to generate glitches at a gate and collect path delay distribution at its fanout gate. For rule 13, path delay distribution will be collected by obtaining joint probability density function using Synopsys HSPICE tool given process variation in a design layout. We finished implementing rule 12. Based on the correlation between temperature and propagation delay, we perform the timing analysis using Synopsys PrimeTime to take the temperature effects into consideration. We applied two different temperatures and corresponding capacitance and propagation delay has been obtained. With this, setup violation check can be performed and we can get the amount of margin that should be added to avoid the setup time violation. We implemented rule 15. Our goal is to evaluate the effects of the keep out region size, number of shield gates on the design overhead and exposed area. We have already comprehensively analyzed one AES implementation in this regard and expect to cover more benchmarks to support our conclusions.

4 SECURITY PROPERTY DATABASE

Abs. Level	Family of Entity	Vul. Source	Vulnerabilities	Associated Assets	Threat	No. of Prop.	Property	Conv. to Assert.	No. of Assertion
	Test & debug structure	Halt mode debugging	Readability of / writability to sensitive information through Debug Access Port (DAP) during HALT operation	Program counter's value	Integrity	1	Program Counter's value should not be modified during HALT operation.	Done	1
				Privilege memory	Confidentiality violation Integrity	1	Privileged memory access should be denied in HALT operation through DAP.	Done	2
			Loading any sensitive information of an SoC into interconnects or bus during HALT operation	Crypto key, TRNG value	Confidentiality violation	1	Assets of an SoC should not be loaded into bus/inter-connects in HALT operation.	NOTE DONE YET	NOTE DONE YET
			Readability of / writability to sensitive information through Debug Access Port (DAP) during HALT operation	Key registers	Confidentiality violation	1	Security-sensitive data stored in registers, such as keys, etc. should be cleared when entering debug mode.	NOTE DONE YET	NOTE DONE YET
			The location of the Stack Pointer contains the last value pushed onto the stack of memory (RAM), serving as the starting address from where user space memory can be accessed. If the stack pointer's content is modified, it points the wrong starting address memory of user space and an unintended function can be executed.	Stack pointer's co tent	Integrity violation	1	Stack pointer's content should not be modified during HALT mode debugging.	NOT DONE YET	NOT DONE YET
		Inter-processor debugging	Different privilege level of **host** and **target** processors during inter-processor debugging	Assets of the target processor	Confidentiality violation Integrity	1	For inter-processor debugging, both target and host should be in same privilege level	NOTE DONE YET	NOTE DONE YET
		Trace Buffer	Trace signals are observable to the debugger and can be exploited to extract information	Trace buffer contents	Confidentiality violation	2	Trace Signals should be encrypted before being stored/recorded in the trace buffer	NOTE DONE YET	NOTE DONE YET
				Any asset carrying signal			Any asset carrying signal / sensitive signal should not be stored in trace buffer.	NOTE DONE YET	NOTE DONE YET

		Coresite External Debug Registers	Using external debug registers, debugged core's general-purpose registers can be overwritten to initiate debug mode just after reset which skips kernel text read-only register (KTRR) initialization, and lead to turn off memory management unit (MMU).	Debug features	Confidentiality violation Integrity Availability violation violation	2	A core should not enter into debug state just after reset.	NOTE DONE YET	NOTE DONE YET
							The debug registers should only be initialized in real-address mode/protected mode	NOTE DONE YET	NOTE DONE YET
			Ability to access debug register during halt mode	Debug features	Confidentiality violation	1	Access to debug registers should be authenticated / Hardware assisted authentication mechanism should be implemented while accessing debug registers	NOTE DONE YET	NOTE DONE YET
		Debug interface	An unprotected debug interface may lead to	Control ability to debug features	Confidentiality violation Integrity Availability violation	1	Only for valid debug certificate, a secure CPU should authenticate a Debugger's request. [debug certificate which includes the requested Debug Control Unit (DCU) values for this debug session.]	NOTE DONE YET	NOTE DONE YET
		Extensive use of external debug interfaces	Extensive use of external debug interfaces as a mechanism to transport configuration data to the SoC. It allows device programmers to transfer data to non-volatile memories like the flash, which often stores the firmware of the system. This capability can be exploited by attackers to overwrite the firmware.	Firmware/ Configuration data	Confidentiality violation Integrity Availability violation	1	External debug interface should not be used to transport configuration data of an SoC to non-volatile memories	NOTE DONE YET	NOTE DONE YET
			Usage of external debug interfaces as a mechanism to transport configuration data to the SoC allows device programmers to transfer data to non-volatile memories like the flash, which often stores the firmware of the system. This capability can be exploited by	Firmware/ Configuration data	Confidentiality violation Integrity violation Availability	1	Using JTAG as an interconnect should be denied transferring configuration data to an SoC	NOT DONE YET	NOT DONE YET
		Debug registers (Breakpoint Exception (BP) and Debug	In ARM debugging unit, the access to the debug registers is achieved by memory mapped interface, instead of hardware traps or interrupts. If the restriction is implemented by software running in the nonsecure mode (e.g., the OS), the malware with kernel privilege may bypass debug authentication easily	Debug configuration bits	Integrity violation Availability violation	1	Access to debug registers should be authenticated / Hardware assisted access control should be applied for debug registers	NOT DONE YET	NOT DONE YET

		Exception (DB))							
	Encryption/Cryptographic Module	Output ports of AES	Observability of AES key from the output ports of AES	Key	Confidentiality violation	2	AES key should not flow to the output port	Done	1
							AES key should not be shared with other modules	Done	1
		Weakness in AES implementation	Incomplete implementation of standard security compliances	Intermediate result of encryption	Confidentiality violation	1	The cipher-text ready signal should be only asserted from final round state	Done	1
				Round Key	Confidentiality violation	1	Round key should not be generated /available before it is being in use.	NOTE DONE YET	NOTE DONE YET
				Intermediate state	Confidentiality violation	2	Each intermediate state should be derived correctly by taking the bitwise XOR of round key and corresponding state	Done	1
							The intermediate encryption results can only flow to the output in debug mode but prohibited during normal operation	NOTE DONE YET	NOTE DONE YET
			Incomplete implementation of standard security compliances can ease the process of performing side channel and fault injection attacks.	Intermediate result of encryption	Confidentiality violation	3	Ready signal should be high after certain cycles of the INITIAL_ROUND	Done	1
							DO_ROUND must occur before the FINAL_ROUND	Done	1
							FINAL_ROUND should only be accessed from DO_ROUND	Done	1
				Key	Confidentiality violation	1	Key cannot propagate to the peripherals.	Done	1
		Too small modulus of RSA encryption	If the RSA key is too short, the modulus can be factorized by applying brute force attack	Plain text	Confidentiality violation Integrity	1	Key generation algorithm should create strong primes so that modulus p and q are not close to each other	NOTE DONE YET	NOTE DONE YET

		Low public exponent of RSA encryption	If the exponent used for encryption is very low, the message can be recovered	Plain text	Confidentiality violation	2	If low public exponent is utilized, proper randomized padding should be used for RSA	NOTE DONE YET	NOTE DONE YET
							Public exponent e must be coprime with λ(n), where $\lambda(n) = \text{lcm}(p-1, q-1)$	NOTE DONE YET	NOTE DONE YET
		Low private exponent of RSA encryption	Low private exponents are easier to recover than high exponents	Plain Text	Confidentiality violation	1	If private exponent (e) is lower than a threshold t1, the public exponent (d) should be higher than a threshold t2	NOTE DONE YET	NOTE DONE YET
		Lack of entropy in key pair generation of RSA	If prime numbers are not random enough, the value of N is easier to factorize		Confidentiality violation	1	Devices should not create their private key on first entropy, because it may not have enough entropy to create a random number	NOTE DONE YET	NOTE DONE YET
		Weakness in RSA implementation	Availability violation of "data_valid" signal	Intermediate result of encryption	Integrity Confidentiality violation Availability violation	2	The interval between initial and final round of SHA algorithm should not be less than 64 cycles	Done	1
							"data_valid" signal of SHA256 algorithm should become high only after final round	Done	1
			Incomplete implementation of standard security compliances can ease the process of performing side channel and fault injection attacks.	Intermediate result of encryption	Confidentiality violation	3	RESULT state can only be accessed from SQR state	Done	1
							MULT and SQR both states must occur before reaching the RESULT state.	Done	1
							finished signal should be high 9 cycles after the initial state.	Done	1
		Hash computation of SHA encryption	Lack of uniqueness of hash value	Protected data	Confidentiality violation	1	Every successful hash computation must take same number of clock cycles	Done	1
		Intermediate variables sensitive to DPA/CPA	Sensitive variables could be utilized by differential power analysis/correlation-based power analysis	Secret Key	Confidentiality violation	1	The value of the intermediate variables should not depend on both plain text and limited number of key bits at the same time	Done	4

	Controller Design	Current program status register (CPSR)	Copying content of CPSR to general purpose registers in user mode	CPSR content	Integrity Confidentiality violation Availability violation	1	Only in privilege mode, CPSR content can be copied to/from any general-purpose register	NOTE DONE YET	NOTE DONE YET
		Erroneous **FSM** of controller circuit	Computational or execution fault may occur due to erroneous FSM	FSM States	Integrity violation	2	When state transition takes place between two consecutive unprotected states, then at least one bit of encoding (MSB) should be fixed. or When state transition takes place between two consecutive unprotected states, then hamming distance between the encoded states should be 1.	DONE	2
							All unused states of an FSM should be handled through default statement which ensures the system to be in secure state. [CWE 1245]	DONE	1
	Exception Handler	Position of Exception vector table	If exception vector table can be overwritten, the execution flow of a program can be altered.	Exception vector table	Integrity Availability violation	1	Exception vector table (EVT) should not be stored into low memory address like 0x0	NOTE DONE YET	NOTE DONE YET
		Mode switching	Initialization of exception handler can provide invalid access to an adversary	Mode handling	Integrity	1	A reset/ software interrupt exception should only be handled in supervisor mode	NOTE DONE YET	NOTE DONE YET
		Exception register value updating	Incorrect updating of register value can provide a user to overwrite the register value	Exception register value	Integrity	1	Exception register value should be updated in supervisor mode	NOTE DONE YET	NOTE DONE YET
	Hardware IP interconnect	Incorrect Synchronization	De-synchronization between data and permissions checking logic can violate Confidentiality violation requirements		Availability violation Integrity Confidentiality violation	1	The data flow and permission checking logic between all masters and slave IPs integrated on bus must obey the bus protocol and follow certain sequence of operations.	NOTE DONE YET	NOTE DONE YET
		Incorrect Synchronization between master and slave IP modules	De-synchronization between data and permissions checking logic can violate confidentiality requirements.	Sensitive information of targeted slave	Confidentiality violation 3		Information written to targeted Slave should never be leaked to other IPs	NOT DONE YET	NOT DONE YET
							Only one slave is permitted to send data to the data channel when one read address is sent to the address channel.	NOT DONE YET	NOT DONE YET
							Only one slave is permitted to receive data from the data channel when one writes address is sent to the address channel.	NOT DONE YET	NOT DONE YET

Miscellaneous	Incorrect signal connection	Incorrect signal connection between a hardware IP and the parent system design	Integrity violation data inside the IP	Integrity Confidentiality violation Availability violation	1	Individual hardware IP must communicate with the parent system in order to function correctly and as intended.	NOTE DONE YET	NOTE DONE YET
	Reserve bits	Reserve bits of a design that are not disabled before production can be exploited by an adversary to perform writes to reserve space in hopes to change the behavior of the hardware	Access to the hardware IP	Integrity Confidentiality violation Availability violation	1	Any write to the reserve bits should be blocked	NOTE DONE YET	NOTE DONE YET
	Instantiation of register module	If hardware registers defaults and IP parameters are incorrectly defined to insecure values in Hardware description language code, the system security settings can be affected by the vulnerability	Access to the hardware IP	Availability violation Integrity Confidentiality violation	1	Registers containing device identifier values are required to be read only to prevent any possibility of software modifying these values.	NOTE DONE YET	NOTE DONE YET
	Write once register	A write-once register in hardware design is programmable by an untrusted software component earlier than the trusted software component, resulting in a race condition issue.	Write access to write once registers	Availability violation Integrity Confidentiality violation	1	After writing to the write once register, the trusted software can issue a read to confirm that the valid setting has been programmed.	NOTE DONE YET	NOTE DONE YET
Locking mechanism	**Improper Implementation of Lock Protection to Registers**	Design or coding errors in the implementation of the lock bit protection feature may allow the lock bit to be modified or cleared by software after being set to unlock the system.	Registers protected by lock bit	Integrity Confidentiality violation Availability violation	2	A set of registers – implemented for certain functionality (temperature sensing) should be locked by single security bit such that they become write once, when configured by firm-wire/ trusted software.	NOTE DONE YET	NOTE DONE YET
		A process located in one memory region can read and modify the content of shared memory objects created by a process in another memory region	Shared memory	Availability violation		For each region of Read Access Memory (RAM), permissions should be set and then locked to prevent subsequent modifications by using the security lock bit	NOTE DONE YET	NOTE DONE YET
	Improper Lock Behavior After Power State Transition	After a power state transition, register's lock bit is set to unlocked. The values of the protected registers get reset untrusted software	Access to the Protected registers, Memory configuration	Integrity Confidentiality violation Availability violation	1	After a power state transition, protected registers should not be reset	NOTE DONE YET	NOTE DONE YET
	Debug mode	System configuration protection may be bypassed during debug mode.	System configuration	Integrity Confidentiality violation	2	Any system configuration override mode should be removed or protected during debug mode	NOTE DONE YET	NOTE DONE YET

							Security Lock bit protections should not support any bypass/override modes	NOTE DONE YET	NOTE DONE YET
	Comparison logic	Comparison logic failure	The product's comparison logic is performed over a series of steps rather than across the entire string in one operation. If there is a comparison logic failure on one of these steps, the operation may be vulnerable to a timing attack that can result in the interception of the process for nefarious purposes.	Access to the Protected registers, Memory configuration	Integrity Confidentiality violation Availability violation	1	A comparison logic should be implemented in one operation instead in smaller chunks.	NOTE DONE YET	NOTE DONE YET
	Hardware Logic	A race condition	A race condition in the hardware logic results in undermining security guarantees of the system.	Protected states of FSM	Availability violation Integrity Confidentiality violation	1	Any race condition should be recognized and eliminated from hardware logic	NOTE DONE YET	NOTE DONE YET
	Core and computation issue	CPU configuration	If CPU is not configured to provide hardware support for exclusivity of write and execute operations on memory, its allows an attacker to execute data from all of memory.	Memory Content	Integrity Confidentiality violation	1	Data area of memory should be non-executable.	NOTE DONE YET	NOTE DONE YET
		nstruction set architecture	Unexpected behavior from certain instruction combinations can arise because of implementation details such as speculative execution, caching etc.	Unlock state of CPU	Availability violation	1	Random instruction sequence should be explored during test which are unlikely to appear during normal workload	NOTE DONE YET	NOTE DONE YET
	Memory and storage issues	Improper write handling in limited-write nonvolatile memory	Non-volatile memories such as NAND Flash, EEPROM, etc. have individually erasable segments, each of which can be put through a limited number of program/erase or write cycles. For example, the device can only endure a limited number of writes, after which the device becomes unreliable. A high concentration of writes may cause a memory line unstable and change a program flow.	Memory Content	Integrity Confidentiality violation Availability violation	1	Status register of a nonvolatile memory line should be write disable after a certain number of write operations	NOTE DONE YET	NOTE DONE YET
		Mirrored Regions with Different Values	Whenever there are multiple, physically different copies of the same value that might change and the process to update them is not instantaneous and atomic, there will always be a time period when they are out of sync.	Memory Content	Integrity Confidentiality violation Availability violation	1	The out-of-sync time period should be as small as possible	NOTE DONE YET	NOTE DONE YET

		Improper Access Control Applied to Mirrored or Aliased Memory Regions	Aliased or mirrored memory regions may have inconsistent read/write permissions enforced by the hardware. A possible result is that an untrusted agent/software is blocked from accessing a memory region but is not blocked from accessing the corresponding aliased memory region.	Memory Content	Confidentiality violation Integrity Availability violation	2	The checks should be applied for consistency access rights between primary memory regions and any mirrored or aliased memory regions.	NOTE DONE YET	NOTE DONE YET
							If different memory protection units (MPU) are protecting the aliased regions, their protected range definitions and policies should be synchronized.	NOTE DONE YET	NOTE DONE YET
		Immutable Data is Stored in Writable Memory	If immutable data, such as a first-stage bootloader, device identifiers, and "write-once" configuration settings are stored in writable memory, it can be re-programmed or updated in the field	Security service assets such as the first stage bootloader, public keys, golden hash digests	Integrity Confidentiality violation	1	All immutable code or data should be programmed into ROM or write-once memory.	NOTE DONE YET	NOTE DONE YET
	Any design	Exposed target net	Exposed target net can be exploited for probing attack	Sensitive information	Confidentiality violation	1	Each vulnerable transition does not pose the same level of threat to the implemented FSM when confronted with fault injection attack. An FSM design with vulnerability transition metric, greater than a threshold value will be considered as susceptible.	NOTE DONE YET	NOTE DONE YET
	Cache	Remaining pre-fetched data of privileged software in cache	Because of speculative execution, data and instructions are fetched to the cache beforehand. When a system return instruction is executed and privilege mode changes the residue of pre-fetching remains in cache. Results in visibility of privileged software's data to user-space software.	Privileged software data	Confidentiality violation	1	When the execution mode of a core is changed to unprivileged level, shared resources/ CACHE registers should be flush.	Not done yet	Not done yet

		Vulnerability in cache inclusion policy	Cache inclusion policy means if data is evicted from last level cache of hierarchy it should be evicted from core caches. if an attacker can evict the victim's critical data from the shared LLC, the inclusive property guarantees that this data will also be evicted from the local core caches of the core where the victim process executes. the next access to the critical data by the victim will miss into the private caches and thus will be visible to the attacker through the LLC. This vulnerability is often used to extract crypto key when plaintext is known and mostly round keys are extracted because computation of round key requires round constants which are stored in the memory.	Encryption key, secret data	Confidentiality violation	1	When read only data is loaded into the cache it should only be evicted by the victim process.	Not done yet	Not done yet
		Information obtained from the access pattern of a cache configuration.	An access-driven attack takes place in different ways. Then, after victim's operation, data is forcefully evicted from the shared cache by the attacker. Or in case of a cache miss, attacker can bypass the cache replacement step to perform differential timing analysis on victims' program.	Encryption key, secret data	Confidentiality violation	1	All state transition of cache controller FSM should take place in an authorized way.	Not done yet	Not done yet
		User's ability of debugging cache.	A debugger 's facility to debug the cache hierarchy can leak sensitive information.	Encryption key, secret data	Confidentiality violation	1	When privileged software utilizes cache, or any encryption or logic locking key is loaded to the cache, debug structure should not have access to it	Not done yet	Not done yet
Gate level (31)	**Test & debug structure**	Halt mode	Readability of sensitive information through Debug Access Port (DAP) during HALT operation	Program counter's value	Confidentiality violation	1	Program Counter's value should not be modified during HALT operation	Done	1
				Privilege memory	Confidentiality violation Integrity	1	Privileged memory access should be denied in HALT operation through DAP	Done	1
			Loading of the sensitive information of an SoC into interconnects or bus during HALT operation	Crypto key, TRNG value	Confidentiality violation	1	Assets of an SoC should not be loaded into bus/inter-connects in HALT operation	NOTE DONE YET	NOTE DONE YET

		Test mode	Corruption/extraction of states, error injection during test mode	Functionality of an SoC	Integrity	1	Halt operation should be initiated only in debug mode not in functional mode	NOTE DONE YET	NOTE DONE YET
			By switching the chip between functional mode and test mode, secret information can be scanned out or calculated. And prone to scan-based attack	Crypto key	Confidentiality violation	1	A reset signal should be enabled for the scan flip-flops whenever there is a switch from the functional mode to the test mode	NOTE DONE YET	NOTE DONE YET
		Trace Buffer	Trace signals are observable to the debugger and can be exploited to extract information	Trace buffer con tents	Confidentiality violation	1	Trace Signals should be encrypted before being stored/recorded in the trace buffer	NOTE DONE YET	NOTE DONE YET
		Controllability of DFT structure	Protected signals can be intercepted and modified to change the output or circuit response	Sensitive signals	Integrity	1	Inserted scan chain should not interfere with protected signals of the design	NOTE DONE YET	NOTE DONE YET
			Memory control signals can be manipulated to gain unauthorized read/write access to memory	Memory Content	Confidentiality violation	1	While entering test mode, reading Random Access Memory (RAM) should be blocked	NOTE DONE YET	NOTE DONE YET
			State signals can be modified to force Finite State Machines (FSMs) to go into unauthorized states or generate different FSM outputs	FSM	Integrity	2	FSM State flip-flop should not be accessed from the debug infrastructures	NOTE DONE YET	NOTE DONE YET
							If the protected states are accessed from an unauthorized state, state flip flops should be reset to initial state	NOTE DONE YET	NOTE DONE YET
			Assets can be controlled to enable or disable certain functionalities	Secret key, password or any sensitive information	Integrity violation	1	Test mode should be disabled when any secret asset is loaded into any interconnect/register	NOTE DONE YET	NOTE DONE YET
		Observability of DFT structure	Data buses used for communication, such as a memory bus, can be leaked at observation points	Sensitive information in data bus	Confidentiality violation	1	Shared Data bus which contains sensitive information should not be observable through DFT structures	NOTE DONE YET	NOTE DONE YET
			SFFs storing intermediate keys of encryption		Confidentiality violation	1	Intermediate registers of an encryption modules should never be a part of Scan chain.	NOTE DONE YET	NOTE DONE YET
		State transition of Flip-flops	State transitions of FFs with longer path difference are vulnerable to fault injection attack	State FFs	Integrity Confidentiality violation	1	FSM state flip flops should be uniformly distributed	NOTE DONE YET	NOTE DONE YET

	Encryption/Cryptographic module	Output ports of AES	Observability of AES key from the output ports of AES	Key	Confidentiality violation	2	AES key should not flow to the output port	Done	1
							AES key should not be shared with other modules	Done	1
		Weakness in AES implementation	Incomplete implementation of standard security compliances	Intermediate result of encryption	Confidentiality violation	1	The cipher-text ready I signal should be only asserted from final round state	Done	1
		Any observable ports of AES	Observability of the ports of AES	Key	Confidentiality violation	1	Key can't propagate to test and debug ports	Done	1
		Accessibility of debug units to intermediate Encryption	Controllability of the debug ports	Internal FSM flip flops	Confidentiality violation Integrity Availability violation	1	FSM State flip-flop of encryption module should not be accessed from the additional debug ports	NOTE DONE YET	NOTE DONE YET
		Timing information of computation	The computational time of encryption operation can be exploited for timing-based side channel attack	Sensitive information	Confidentiality violation	1	The number of cycles between the start and finish of the encryption operation should always be constant.	NOTE DONE YET	NOTE DONE YET
		Intermediate states of encryption algorithm	Bypassing intermediate states of encryption	Encryption Key/plain text	Confidentiality violation	1	FSM encoding of encryption algorithm should be implemented completely to resist fault injection vulnerability	Done	2
		Intermediate states of encryption algorithm	Bypassing intermediate states of encryption	Encryption Key/plain text	Confidentiality violation	2	Any protected state should not be accessed from unauthorized (don't care) states.	Done	1
							Final_round of AES should only be accessed from the do_round	Done	1

		Intermediate states of encryption algorithm	Bypassing intermediate states of encryption Using test and debug structures	Encryption Key/plain text	Confidentiality violation Integrity violation	2	RESULT state should only be accessed from SQR state and should not be accessed from any don't care states.	Done	1
							"st_sha_data_valid" state should only be accessed from the "st_sha_blk_nxt" state. and should not be accessed from any don't care states.	Done	1
		FSM State encoding	For synthesis tool optimization, fault-injection attack vulnerability can be introduced to FSM encoding scheme.	Protected States of FSM	Confidentiality violation Integrity violation	1	For a certain FSM state encoding scheme, the FI Feasibility metric should not have the value 1.	Done	1
	Any design	**Exposed target net**	Exposed target net can be exploited for probing attack	Sensitive information	Confidentiality violation	4	Vulnerability factor metric is proposed to evaluate the vulnerability of the FSM design against fault injection attack.	NOTE DONE YET	NOTE DONE YET
							For designs without probing attempt detector, the working frequency of paths should be in range	N/A	N/A
							For the circuits with probing detector, the probing delay vulnerability should be smaller than % of the total number of the nodes in a design	N/A	N/A
							For the circuits with probing detector, the vulnerability should be smaller than % of the total number of the paths in a design	N/A	N/A
Physical Layout (17)	**Any Design**	**Exposed target net**	Exposed target net can be exploited for probing attack	Sensitive information	Confidentiality violation	6	The exposed area from frontside should be smaller than x% of the whole target net area	NOTE DONE YET	NOTE DONE YET
							The exposed area from backside should be smaller than z% of the whole target net area	NOTE DONE YET	NOTE DONE YET
							To protect against backside probing attacks on lower metal layers, target nets (i.e., those carrying sensitive data and/or assets that an attacker seeks to probe) should be placed above layer l where $RFIB,max(l)\geq y$, where $RFIB,max(l)$ is FIB aspect ratio	NOTE DONE YET	NOTE DONE YET

							For the circuits with probing detector, the probing delay vulnerability should be smaller than % of the total number of the nodes in a design	NOTE DONE YET	NOTE DONE YET
							For the circuits with probing detector, the probing path vulnerability should be smaller than % of the total number of the paths in a design	NOTE DONE YET	NOTE DONE YET
							For the designs without probing attempt detector, their working frequency should be in the range:	NOTE DONE YET	NOTE DONE YET
			Exposed target net can be exploited for probing attack	Sensitive information	Confidentiality violation	2	**The added traces length of the design (w/ and w/out countermeasures) should be larger than times of the length of the longer edge of target area.**	NOTE DONE YET	NOTE DONE YET
							The exposed area of the design (w/ and w/out countermeasures) should be smaller than % of the whole target area	NOTE DONE YET	NOTE DONE YET
			Exposed target net can be exploited for probing attack	Sensitive information	Confidentiality violation	1	**Develop a complete set of scripts that will generate an image that highlights the vulnerable region on the target area**	NOTE DONE YET	NOTE DONE YET
			Transition in an FSM can be exploited for fault injection attack	Sensitive information	Confidentiality violation	2	For a specific transition in the FSM, when there are adjacent events at an input for that gate, the transition event probability should be smaller than the probability of only one of the adjacent events happens	NOTE DONE YET	NOTE DONE YET
			Setup time violation might happen with TLS being applied to the fanin of the flip-flop	Sensitive information	Confidentiality violation	2	Suppose there is a TLS being applied to a fan-in of a flip-flop, with the following parameters: power of the laser, P, size of the spot, S, temperature, T, the amount of margin that should be added to avoid the setup time violation is $(P,,T)$.	NOTE DONE YET	NOTE DONE YET
			Setup time violation might happen with TLS being applied to the fanin of the flip-flop	Sensitive information	Confidentiality violation	1	Suppose there is a TLS being applied to a fan-in of a flip-flop, with the following primary parameters: laser pulse duration, $tpulse$, focus of the laser beam, z, summary of spatial parameters, S, (location, geometry, wafer thickness), the amount of margin that should be added to avoid the setup time violation is $f(tpulse,z,S)$.	NOTE DONE YET	NOTE DONE YET
			Single event effects (SEEs) would happen with TLS being applied.	Sensitive information	Confidentiality violation and integrity	1	A laser fault sensitivity mapping on a CMOS 40 nm D Flip-Flop is presented in this model. This model could help create a new layout of a standard D Flip-Flop cell more robust against SEU.	NOTE DONE YET	NOTE DONE YET

			Single event effects (SEEs) would happen with TLS being applied.	Sensitive information	Confidentiality violation	1	This model will help designers identify the relation between the shielding ability of the design and the overhead. Therefore, given different requirements for the overhead, designers are able to achieve the relatively optimal protection to the target nets	NOTE DONE YET	NOTE DONE YET
			Exposed target net can be exploited for reroute attack	Sensitive information	Confidentiality violation	1	This model will help designers describe the level of vulnerability of the design is to reroute attack based on added traces length metric. The longer the added traces length is, the less vulnerable the design will be	NOTE DONE YET	NOTE DONE YET
			Exposed target net can be exploited for reroute attack	Sensitive information	Confidentiality violation	3	State flip-flops distancing metric is proposed against fault injection attack by adding constraints on the way that state flip-flops are placed in the physical layout.	NOTE DONE YET	NOTE DONE YET
							Vulnerability metric is defined as the FSM's susceptibility to x laser-based faults in one clock cycle.	N/A	N/A
							Spatial vulnerability metric is defined as the FSM's susceptibility to laser-based faults in one clock cycle, which can be described with the Flip-flop set, the laser set, and the possible Flip-flop attacks.	N/A	N/A

5 SECURITY RULES:

Rule 1	When state transition occurs between two consecutive unprotected states, hamming distance (HD) between the encoded states should be 1, there should be exactly one bit difference between these two encodings.
Rule 2	All unused states of an FSM should be handled through 'default' statement
Rule 3	Control signals of a control unit FSM should come from trusted input ports only
Rule 4	IF statements containing operations with KEY must be associated with ELSE statements.
Rule 5	Always block containing operations with KEY should not have any variable assigned to don't care

6 SUMMARY:

Abstraction Levels	No. of Properties/Rules	Formally Defined metrics
RTL	71/5	11
Gate level	31	10
Physical Layout	20	9

LIST OF SYMBOLS, ABBREVIATION, AND ACRONYMS

ACRONYM	DESCRIPTION
AES	
AFRL	Air Force Research Lab
AHEAD	Authenticated Encryption with Associated Data
ARCHS	Automated Rule Checking for Hardware Security
ASCON	
CDFG	
COMET	
DARPA	Defense Advanced Research Projects Agency
DFT	
FMS	
FPGA	Field Programmable Gate Arrays
FPV	
GUI	Graphical User Interface
RSA	
RTL	Register-Transfer Level
SNR	
SoCs	System-on-Chips
TVLA	