Research Topic in Secure Hardware Verification

# Introduction

Security verification is a challenging task, since it requires to cover a wide scenario space, usually much wider than for the functional verification. To address the security verification of software, the Common Weaknesses Enumeration taxonomy was introduced in 2006 [1] and became a de facto catalogue of software security issues. Today, wide variety of tools exist that try to find these weaknesses in the software code. Among them, static analysis tools prevail [2]. Following the software path, hardware CWE database was added in 2020 [3] and now includes hundreds of entries. Adoption of the static analysis approach for hardware security verification looks like a natural choice.

An interesting question is whether specific methods from the software domain can be used in hardware security analysis. Absolute majority of those methods are domain-specific and use heuristics tailored to software specific paradigms, such as buffer overflow, SQL injection and bound crossings. Moreover, various lint tools already adopted for hardware verification don’t go much deeper than language semantics or simple structural tasks, such as clock domain crossing, and therefore are not an answer for the complex, usually architecture or microarchitecture level hardware security issues. One software verification paradigm that may work also for hardware is taint analysis [4], which is known in the hardware world as information flow tracking [5]–[7]. Such analysis may use graph methods to track the information flow between different security domains. Commercial use of the information-flow analysis tools has been reported. Nevertheless, early methods are limited to small scale, due to the required intimate knowledge of every design detail and intrinsic algorithm limitations.

Recently, an urgent need for CAD tools for hardware security was announced [8], which resulted in more extensive investment in the security verification tools In particular, hardware information flow tracking (IFT) model was adopted by several research and commercial organizations [9], [10]. The model adopts a formal language, defining rules, which are usually added to RTL in the form of assertions. The assertions can then be verified either statically or dynamically. Brute-force static methods lead to fast explosion of the state space, therefore dynamic methods are proposed. In the developer language, it means creating verification tests that cover as many scenarios as possible (including out-of-spec) that will run with the assertions. This is a substantial amount of work yet with partial coverage only.

## Assessing the quality of the verification tools and methods

What is secure verification for hardware? Ideally, we want our tool to give us an automatic answer as to whether our design has security issues, preferably pointing out the problematic locations. Of course, without some input from the designer that would not be possible. But, to make the process robust, we wish to minimize the size of the user input, because bigger size means more manual intervention, which in turn results in higher sensitivity to a human error. So, we can arguably claim that a verification tool developer will strive to build a tool that takes as few user inputs as possible. We may also try to quantify the user input in the information-theoretic domain to derive a quality metric for various tools. This is an interesting research topic for the future. The user input can come in various forms, such as test vectors for dynamic verification or rule definition for formal approach.

# Research Topic: Using Graph Neural Networks (GNN) to detect security vulnerabilities

Graph Neural networks are popular way to reason about many “real-life situations” such as social networks, chemical structures and more. Since Netlist is a graph presentation of hardware, using GNN seems to be the right infrastructure for reasoning about hardware design, in general, and for reasoning regarding security in particular. First, the design under test will be mapped to a generic cell library. Let us refer to this mapping as a generic netlist. This netlist will be loaded to our framework in the form of a graph. There are several ways of representing a logical circuit (netlist) as a graph. A preferred representation at this point is a ‘flip-flop dependency graph’. On one hand, this representation loses some structural information of the netlist. On the other hand, it has the advantage of implementation invariance.

The detection of the security issues should be based on the property-based soft decision techniques. This technique addresses the weakness of the existing approach in requiring a fine-grained rule definition. In many cases, defining such a rule is as complex as finding the issues manually. The proposed method, rather than directly defining the rules, will search for structural or functional properties that may hint for existence of a security problem [11]. Examples of structural properties are fan-in, connection density and other characteristics from the graph theory. Boolean function analysis can serve as a method to define functional properties, for example Boolean influence, linearity/affinity or specific Boolean relationships between signals

For example, let’s consider CWE-1233: Improper Hardware Lock Protection for Security Sensitive Controls from [3]. According to [9], to find this vulnerability, one needs to know in advance all the lock bits and all the user-controlled signals. Hence, the added value of such a rule check over the classical functional verification is minimal. Using the graph structural and functional properties of the design’s graph representation, we can for example detect anomalies in the fan-in or Boolean influence in the sub-graph that contains the lock bit and the lockable resource. A Graph Neural Network (GNN) can be trained using the features above using synthetic correct and faulty circuits.

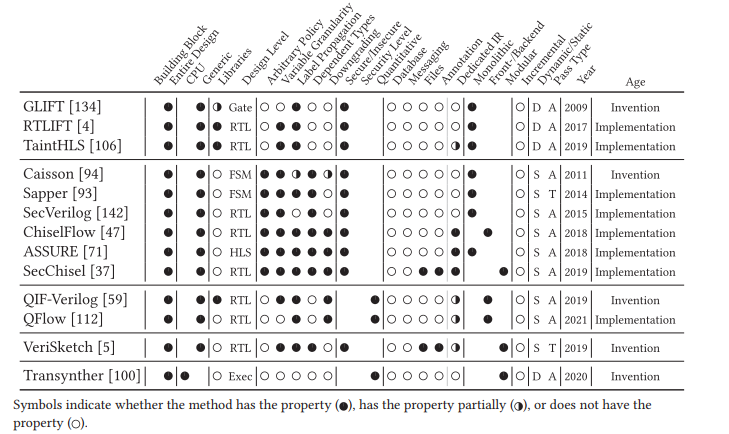
# Research Topic: Using Quantitative Information Flow (QIF) for secure hardware verification

The Quantitative Information Flow (QIF) analysis is a relatively new field of research that measures the amount of information leakage rather than searching for a binary answer of whether such leakage exists. Based on probabilistic analysis and information theory, it may reduce the analysis complexity by using statistical models for the signals in the system. QIF can also be used for applications that look for a quantitative answer, such as side channel leakage measurements.

## Information Flow Tracking (IFT) Tools Overview

Feldtkeller *et al.* [12] provide a taxonomy of various security verification tools for hardware.

### Information Flow Tracking



Quantitative Information Flow

Shadow

logic

Language

based

Figure Information Flow Analysis Methods for Hardware in Literature (Feldtkeller et al, "Challenges and Opportunities of Security-Aware EDA)

IFT looks either for secret leakage (confidentiality) or for integrity (non-interference) failures. Associate security labels with the resources and make sure data from high level doesn’t leak to lower levels. This is called non-interference. It is also common to model the policy with information flow lattices constructed from a set of security labels and partial order. Non-interference allows for information to flow up in the lattice, i.e. from less to more secure.

GLIFT [6] augments the gate-level netlist with ‘trust’ labels, and for each gate defines a truth table that includes the ‘trust’ label of the output. So, each gate is accompanied by a shadow gate. The augmented netlist can be simulated dynamically. The augmented netlist with the ‘shadow gates’ can also be fabricated for in-field verification leading to about 3X overhead in area. GLIFT can also be used to monitor information leakage through timing by observing events. RTLIFT [12] is an extension of GLIFT to RTL. STAR-logic, an evolution of GLIFT, is a static tool that verifies a minimal SW/HW trusted base statically.

[13] brings a modified HDL that includes information of the information flow (IFC HDL), known as SecVerilogBL, built upon the SecVerilog language. The SecVerilog language adds the notion of security domains attached to signals and timing labels. SecVerilogBL adds the notion of downgrade mainly to mark exceptions.

Sapper [14] is a hardware description language that is based on a synthesizable subset

of Verilog. Sapper compiler automatically ensures non-interference in the generated hardware

logic, and is able to generate Verilog code with added dynamic information flow tags. Figure 2

shows the verification flow. Sapper statically analyzes the hardware logic and automatically inserts dynamic IFT logic and generates Verilog code with extra logic for the dynamic information flow tracking.

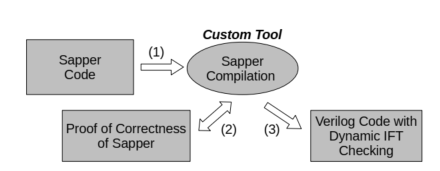
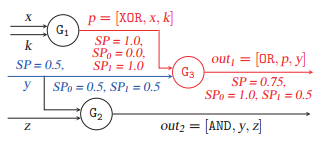


Figure 2 Dynamic Information Flow Verification in Sapper

### Quantitative Information Flow (QIF)

While the classical Information Flow analysis searches for a binary “black or white” answer on the question of information leakage /non-interference, Quantitative Information Flow (QIF) [15] tries to measure the amount of leakage. It does so by using Shannon’s information theory approach to measure the entropy of a leaked secret versus the secret’s a priori entropy. It introduces the notion of information channels that change the statistical properties of the signals.

Lakshmy et. al. [16] proposed a framework for a static side channel leakage assessment. Although not explicitly called QIF, it is actually using the same approach albeit at a basic level. It assigns conditional probabilities of each signal in the system being 1 or 0 given the reference by recursive evaluation of the netlist gates. The leakage is estimated based on the weighted diff of the conditional probabilities.



Signal probability estimation illustration [16]

This method has several limitations. First, its precision relies on the assumption of independent inputs to the gates, i.e. assuming no reconvergent paths. This assumption is not realistic. In addition, due to the simplicity of the method, it cannot cover complicated statistical scenarios.

QIF-Verilog [17] and Qflow [18] propose an augmented Verilog-based method. The augmented mode contains information on the signals’ security domain. The tools further calculate the leakage based on the logic composition.

### Dynamic

Muller [19] proposes a framework for confidentiality verification of data flow in a RISC-V-based SoC, combining SAT with symbolic checking. It extends Unique Program Execution Checking (UPEC) [20] from a core level to SoC level and more. In general, the approach is comparison of executions with different secret data. It employs SVA to implement the method.

## QIF for security verification research path

QIF is a promising direction of research for security verification. QIF in general is an emerging topic and thus brings vast research opportunities.

For applications, such as the information flow tracking, QIF may address the scalability issue, a general issue for most of the formal methods. It can do so by switching from the strict rule-based problem definition that usually leads to a fast state explosion to the statistical models.

There are also applications, which inherently use quantitative metrics. The most appealing of them is the side channel attack vulnerability estimation, where we are interested in measuring the information leakage.

### QIF for IFT

Several research groups are already actively working on the WIF-based information flow tracking tools and methods. Yet the current offer is quite rudimentary and has numerous limitations. However, this is just a beginning, and it looks promising. Let us consider the QIF-Verilog tool. Currently, the input to the tool is a binary label that marks each input either as tainted or as not tainted. The tool can be expanded to support a continuous value from 0 to 1 that will indicate the signal’s entropy also called a remaining uncertainty in the paper. The value can be assigned both to secret and to non-secret signals. Further, the tool will propagate the entropy value through the code to calculate the value at the output as shown in Figure 3. Additional topics are enhancing the tool to work with System Verilog (purely technical), working with sequential logic and handling reconverging paths.

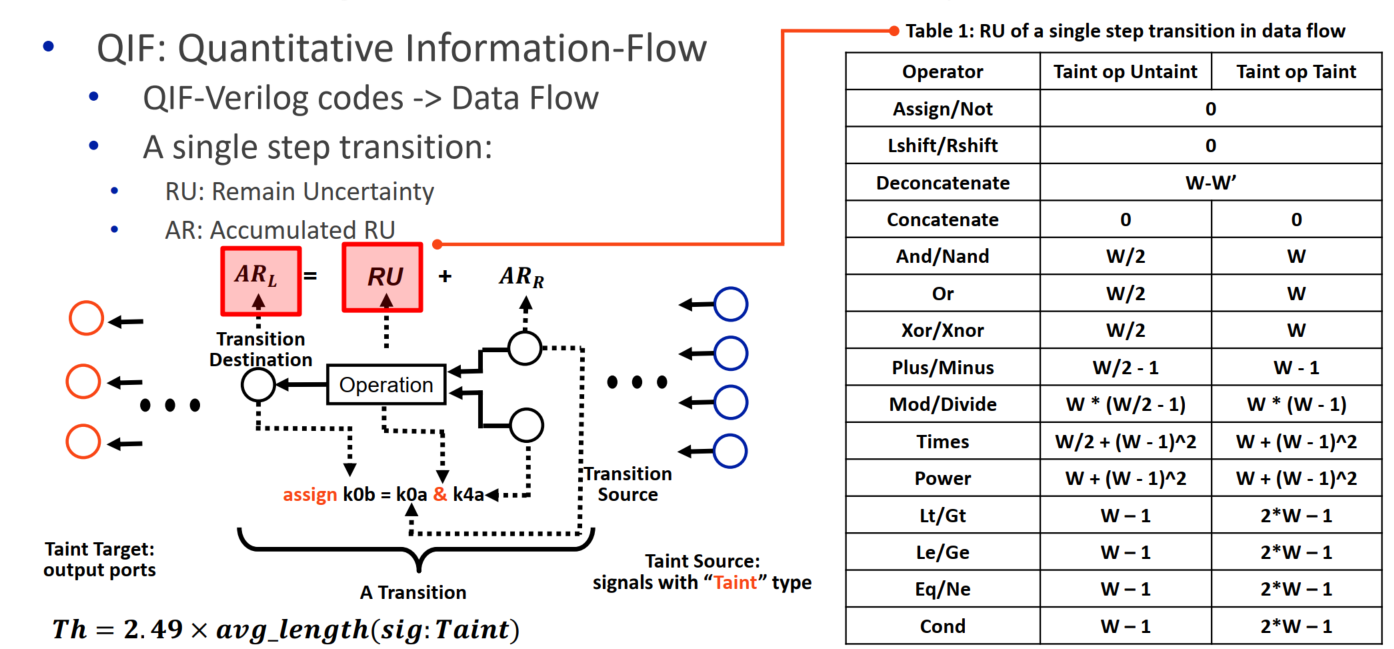


Figure 3 QIF-Verilog remaining uncertainty calculation

### QIF for Side-channel Analysis

Side-channel analysis (SCA) may be the most appealing application of Quantitative Information Flow. Various SCA techniques, such as the differential power analysis (DPA) or timing side-channel attacks employ statistical methods. We estimate a circuit’s vulnerability to SCA either by application of specific attacks or by a more generic and naturally more pessimistic methods of information leakage measurements (e.g. by statistical t-test). This used to be done only on a finished device. However, recently several tools have emerged that offer pre-silicon information leakage estimation by running power simulations of various accuracy levels. Due to their dynamic nature, it takes long time and vast resources to acquire sufficient number of traces (can be millions). QIF is a static analysis tool that can be an ab efficient alternative to the simulation-based approach.

Building a QIF-based framework for side channel analysis starts from definition of a leakage model. For example, for the DPA attack, the leakage model will express a correlation between the power consumed by a gate at the switching time and the logical state of it inputs. Naturally, the tool will need to run on the netlsit rather than on RTL as the current tools do.

# Bibliography

[1] “CWE - Common Weakness Enumeration.” [Online]. Available: https://cwe.mitre.org/. [Accessed: 25-Mar-2022].

[2] P. Ferrara, A. K. Mandal, A. Cortesi, and F. Spoto, “Static analysis for discovering IoT vulnerabilities,” *Int. J. Softw. Tools Technol. Transf.*, vol. 23, no. 1, pp. 71–88, Feb. 2021, doi: 10.1007/S10009-020-00592-X/FIGURES/7.

[3] “CWE - CWE-1194: Hardware Design (4.6).” [Online]. Available: https://cwe.mitre.org/data/definitions/1194.html. [Accessed: 26-Mar-2022].

[4] J. Newsome and D. Song, “Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software,” in *NETWORK AND DISTRIBUTED SYSTEMS SECURITY SYMPOSIUM*, 2005.

[5] T. Mccomb and L. Wildman, “SIFA : A Tool for Evaluation of High-Grade Security Devices,” pp. 230–241, 2005.

[6] M. Tiwari *et al.*, “Gate-Level Information-Flow Tracking for Secure Architectures,” *IEEE Micro*, vol. 30, no. 1, pp. 92–100, 2010, doi: 10.1109/MM.2010.17.

[7] D. W. Palmer and P. K. Manna, “An efficient algorithm for identifying security relevant logic and vulnerabilities in RTL designs,” in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 61–66, doi: 10.1109/HST.2013.6581567.

[8] S. Aftabjahani *et al.*, “Special session: CAD for hardware security - Automation is key to adoption of solutions,” *Proc. IEEE VLSI Test Symp.*, vol. 2021-April, Apr. 2021, doi: 10.1109/VTS50974.2021.9441032.

[9] “Radix Coverage for Hardware Common Weakness Enumeration ( CWE ) Guide.” pp. 1–47, 2020.

[10] “360 DV-Verify – OneSpin Solutions.” [Online]. Available: https://www.onespin.com/products/360-dv-verify. [Accessed: 27-Mar-2022].

[11] L. Azriel, R. Ginosar, and A. Mendelson, “Sok: An overview of algorithmic methods in IC reverse engineering,” in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, pp. 65–74, doi: 10.1145/3338508.3359575.

[12] FeldtkellerJakob, SasdrichPascal, and GüneysuTim, “Challenges and Opportunities of Security-Aware EDA,” *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 43, Apr. 2023, doi: 10.1145/3576199.

[13] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, “Register transfer level information flow tracking for provably secure hardware design,” in *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, 2017, pp. 1691–1696, doi: 10.23919/DATE.2017.7927266.

[14] A. Ferraiuolo, R. Xu, D. Zhang, A. C. Myers, and G. E. Suh, “Verification of a Practical Hardware Security Architecture Through Static Information Flow Analysis,” in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, vol. 52, no. 4, doi: 10.1145/3093336.

[15] M. S. Alvim, K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith, *The Science of Quantitative Information Flow*. Cham: Springer International Publishing, 2020.

[16] A. V. Lakshmy, C. Rebeiro, and S. Bhunia, “FORTIFY: Analytical Pre-Silicon Side-Channel Characterization of Digital Designs,” in *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2022, vol. 2022-Janua, pp. 660–665, doi: 10.1109/ASP-DAC52403.2022.9712551.

[17] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin, “QIF-Verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment,” in *Proceedings of the 2019 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019*, 2019, pp. 91–100, doi: 10.1109/HST.2019.8740840.

[18] L. M. Reimann, L. Hanel, D. Sisejkovic, F. Merchant, and R. Leupers, “QFlow: Quantitative Information Flow for Security-Aware Hardware Design in Verilog,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, vol. 2021-Octob, pp. 603–607, doi: 10.1109/ICCD53106.2021.00097.

[19] J. Muller, M. R. Fadiheh, A. L. D. Anton, T. Eisenbarth, D. Stoffel, and W. Kunz, “A Formal Approach to Confidentiality Verification in SoCs at the Register Transfer Level,” in *Proceedings - Design Automation Conference*, 2021, vol. 2021-Decem, pp. 991–996, doi: 10.1109/DAC18074.2021.9586248.

[20] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, and W. Kunz, “Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking,” *Proc. 2019 Des. Autom. Test Eur. Conf. Exhib. DATE 2019*, pp. 994–999, May 2019, doi: 10.23919/DATE.2019.8715004.