Quantifying Hardware Security Using Joint Information Flow Analysis

Ryan Kastner, Wei Hu, and Alric Althoff
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
{kastner, vhu040, aalthoff}@ucsd.edu

***Abstract*—Existing hardware design methodologies provide limited methods to detect security flaws or derive a measure on how well a mitigation technique protects the system. Information flow analysis provides a powerful method to test and verify a design against security properties that are typically expressed using the notion of noninterference. While this is useful in many scenarios, it does have drawbacks primarily related to its strict enforcement of limiting all information flows – even those that could only occur in rare circumstances. Quantitative metrics based upon information theoretic measures provide an approach to loosen such restrictions. Furthermore, they are useful in understanding the effectiveness of security mitigations techniques. In this work, we discuss information flow analysis using noninterference and qualitative metrics. We describe how to use them in a synergistic manner to perform joint information flow analysis. And we use this novel technique to analyze security properties across several different hardware cryptographic cores.**

I. Introduction

Digital hardware is the foundation of modern computing systems. It explicitly controls the actions of every single piece of data across all of its execution. Thus, even the smallest hardware vulnerability opens up the door for attacks across the entire system stack. Simply stated, it is not possible to construct a secure application upon insecure hardware. Despite this fact, the vast majority of the security research has focused on software. And while the hardware security research has gained momentum in the past decade, there is still a desperate need for hardware security design tools.

The limited focus on hardware security may be a consequence of the lack of attacks focusing on hardware. However, the number of hardware specific exploits is consistently increasing. Hardware security flaws have broad ranging effects – downgrading performance and reliability [1], opening door to rootkits [2], or allowing root control over the system stack [3]. Security experts have shown that it is possible to downgrade the cryptographic capability of Intel processors through a simple yet difficult to detect modification, which may allow an attacker to retrieve personal secrets in seconds [4]. Recent attacks have also demonstrated the possibility to compromise hardware deployed on a commercial jet through a hidden back door, which provides cyber criminals a foothold to override the flight control system [5]. What's more, there is speculation that hardware backdoors are present in military weapons, which allows them to be remotely disabled [6].

Designing secure hardware is a challenging task. It goes far beyond traditional testing and verification techniques that look at functional correctness. First, it is impossible to fully verify that a multibillion transistor chip is functionally equivalent to its specification. Second, and perhaps more importantly, functional correctness does not imply security; the specification can never be entirely complete, and the uncertain portions of the specification provide an opportune place for vulnerabilities.

Existing hardware design tools provide limited support for detecting security flaws or deriving a measure on how well a mitigation technique protects the system. Even the largest semiconductor companies heavily rely on security auditing teams, e.g., Intel's Security Center of Excellence (SeCoE) and the Qualcomm Product Security Initiative (QPSI) group, to perform tedious, primarily manual inspection of hardware designs. Thus, system designers are left to make *qualitative* arguments such as: "our system will be more secure if we use encryption" or "a hardware security module makes the system less vulnerable to attack". This is obviously less than ideal.

Secure hardware design requires *quantitative* measures on the effectiveness of mitigation techniques. This would allows designers to effectively reason about need for enhancing the security of a system, which often is in direct competition with traditional design metrics such as area, delay, throughput, and power. A quantitative security metric enables the designer to say "this modification adds 5% more area, and it requires 10% more power, but makes the design 30% more secure" instead of "I know it is costly in terms of area and power, but trust me, it makes the design a lot more secure". Effective metrics that allow both hardware designers and security engineers to quantitatively argue about the security of hardware designs are an important and open research problem.

In this paper, we propose a novel hardware design methodology for analyzing and understanding the security of the system. To the best of our knowledge, this is first time that noninterference and information theoretic techniques have been combined for secure hardware design. The major contributions of this work are:

- Describing the benefits and drawbacks of noninterference and information theoretic measures for understanding the security of hardware designs.
- Introducing a information flow analysis that combines noninterference and information theoretic techniques to provide a metric for hardware security.
- Utilizing our novel approach to identify security vulnerabilities, and understanding the effectiveness of hardware mitigation techniques.

The reminder of this paper is organized as follows. Section II describes information flow analysis, noninterference, and quantitative security metrics. Section III introduces our methodology for identifying and mitigating security vulnerabilities. In Section IV, we demonstrate our method with design examples. Section V discuses about potential research opportunities and challenges. We conclude in Section VI.

II. Information Flow Analysis

Information flow analysis associates data with a security label (also called *taint*) and monitors the propagation of this label through the system. It can be used to determine whether the system adheres to security properties related to confidentiality and integrity. This analysis is used across the system stack – in the compiler [7], operating system [8], [9], architecture [10], [11], [12], and hardware [13], [14], [15].

Information flow analysis most commonly uses the notion of noninterference in order to specify security properties, which we discuss in Section II-A. In Section II-B, we argue that quantitative metrics are necessary in many scenarios.

A. *Noninterference*

The notion of noninterference was introduced by Goguen and Meseguer as a means to specify and prove security properties [16]. They stated that “one group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see”.

Figure 1 provides some insight on noninterference at the hardware level. Here we show the idea behind gate level information flow tracking (GLIFT). The original digital circuit under inspection is a four-input AND gate implemented using a cascade of two-input AND gates. Each of the three AND gates is augmented with GLIFT analysis logic as shown in Figure 1 (b). The analysis logic for one 2-input AND gate has four inputs – the two inputs to the original gate, and their security labels. It outputs one value, which is the label associated with the output of the original circuit. To facilitate explanation, assume that these labels can be `LOW` or `HIGH` and we wish to prove properties about confidentiality. Here a `LOW` label means low security (e.g., public information) and a `HIGH` value means high security (e.g., top secret information). The same logic can also be used for properties related to integrity, which is the dual of confidentiality.

Assume that `LOW = 0` and `HIGH = 1`. We wish to determine when the output can be `HIGH`, or logical `1`. If both inputs are `HIGH`, then the output is `HIGH`. A more interesting case is when there is a mix of `LOW` and `HIGH` inputs. A conservative approach would be to mark the output as `HIGH` if any of the inputs are `HIGH`. However, assume input A is labeled as `LOW` and its value is `0`, and input B is `HIGH`. The output of the AND gate is `0` regardless of the value of B. Thus, we can learn nothing about the `HIGH` input. This is the idea of noninterference – by observing the final `LOW` outputs one can learn nothing about the `HIGH` values. If either of the inputs is `1` and the other input is `HIGH`, then the output is `HIGH` since we know what the `HIGH` input value is in this case; this is the logic as shown in Figure 1 (b). We can constructively combine the GLIFT analysis logic in a manner as shown in Figure 1 (a). An interested reader can find information about generating the GLIFT logic in [17].

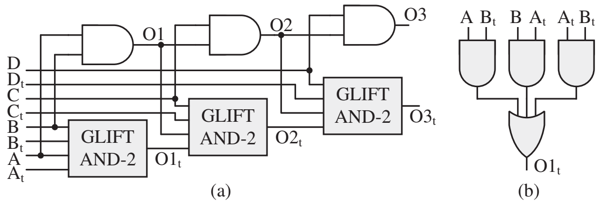

Fig. 1. GLIFT augments the original logic with analysis logic. Part (b) shows the analysis logic corresponding to the first GLIFT AND-2 component.

Continuing with the example in Figure 1, assume that only input A holds confidential information (e.g., a bit of your password). We would mark the security labels A_t = `HIGH` and B_t, C_t, D_t = `LOW`. Finally, assume the output $O3$ feeds into a memory location visible to any program. We want to know if $O3_t$ = `HIGH`, i.e., that output contains information about your password. Equivalently, we taint A, and then use information flow analysis to determine if $O3$ is tainted.

This seemingly simple approach provides a powerful method to analyze systems against different security properties. For example, GLIFT was used to test for isolation between users on shared communication protocols [18]; it was used to prove timing-based noninterference for a network-on-chip architecture [19]; and it verified that software processes running on a microkernel were provably isolated on processor modified to eliminate interference [20]. Additionally, GLIFT marks interference caused by a timing channel [21]. This was used to show that data is a cache was isolated, and shared resources on a system on chip would not leak any information either explicitly or through a timing channel [22], [23].

Yet, despite all these successes in analyzing systems for different security properties, information flow analysis based upon noninterference has its drawbacks. These are due to the fact that noninterference takes an *all or nothing* approach. For example, to determine properties related to integrity, we would label “untrusted” data as tainted, and determine if that data ever affects critical resources. The discrete nature of this labeling process can quickly lead to a “taint explosion” where the entire system is determined to be untrusted. However, in many scenarios this is not as serious as it may seem. Those critical cases are either exceedingly rare or difficult to exploit in a real-life attack. And many systems have a security policy that allows a limited amount of these flows. Thus, noninterference does not correctly reflect the security of the system in many situations.

B. *Quantitative Metrics*

Relaxing the strict property of noninterference, and developing a quantitative metric on “how much” information flows from one location to another allows the designer to understand the risk associated with leaking this information. And there are often cases where some leakage is justifiable. E.g., a password system leaks information by telling the user when the password is incorrect. Yet, we are often willing to allow this for a limited number of attempts.

Quantitative security metrics are a generalization of noninterference. And there is past research that uses information theoretic measures to determine the exact amount of information contained in a tainted label. Denning proposed using entropy to model relationships between statements in a program [24]. Miller provided a relationship between noninterference and mutual information using deterministic state machines [25]. McLean described the flow model security property [26]; Gray formalized this by relating noninterference to the maximum rate of flow between variables [27]. Mica and Morgan use conditional entropy to calculate the channel capacity of a program [28]. Newsome et al. [29] use channel capacity as a quantitative measure of the influence of the inputs on the outputs of a program using x86 binaries. Information theory measures, e.g., the worst-case mutual information [30] and min-entropy [31], are used to determine the difficulty of breaking into the system. Kopf and Durmuth use information theory to analyze time-based side channel attacks [32].

We aim to create similar quantitative metrics for hardware design. Figure 2 uses the same example as Figure 1. Here, we measure the mutual information from input A to signals $O1$, $O2$, and $O3$, i.e., $I(A;O1)$, $I(A;O2)$, and $I(A;O3)$, respectively. It also shows $I(O1;O2)$ and $I(O2;O3)$. Intuitively, mutual information $I(X;Y)$ is a measure of how much information we learn about variable X given that we have complete knowledge of variable Y.

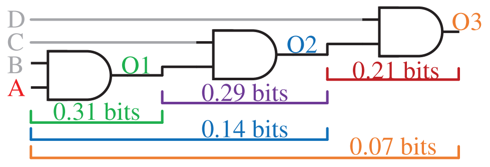

Fig. 2. Information flow through four-input AND gate.

Notice that the mutual information through an AND gate is not always the same: $I(A;O1) = 0.31$ bits, $I(A;O2) = 0.14$ bits, and $I(A;O3) = 0.07$ bits. Mutual information is not a local property; it is a more complex function of the entire system and more precisely the distribution of the inputs. In this example, we use a uniform distribution on the inputs A, B, C, and D. The distribution on the inputs to the "later" AND gates changes. E.g., the distribution of $O1$ is skewed toward `0` and away from `1`; this changes the mutual information. This is even more prevalent between A and $O3$. Again, A has an even distribution between `0` and `1`. But $O3$ is mostly `0`. Thus it is difficult to learn anything about A because $O3$ is almost always `0`. And this value occurs when A is both `0` and `1`.

III. Joint Information Flow Analysis

Information flow analysis is a powerful technique for determining whether a system adheres to a variety of different types of security properties. We have built many different hardware designs that act in accordance to security properties based upon noninterference. However, in many cases, we found these properties to be overly strict. In these cases, we need a more quantitative assessment of the security.

A. Motivation

Consider again our running example from Figures 1 and 2. Assume that we want to ensure that the password (stored in variable A) does not leak to the unprotected memory location (denoted by output $O3$). Specifying this as a noninterference property would mark A as tainted (or `HIGH`), and information flow analysis (e.g., GLIFT) would show that A can flow to $O3$, i.e., the label of $O3$ could be `HIGH`. If we are relying solely on noninterference properties, at this point we would have to redesign the system until there is no flow between A and $O3$, i.e., GLIFT indicates that the label of $O3$ is `LOW`.

Instead of going through a redesign, we could otherwise quantify the amount of information flowing from A to $O3$. Using mutual information (as shown in Figure 2) determines that there are on average 0.07 bits of information moving from A to $O3$ after propagating along only three levels of gates. Depending on the threat model, this may be an acceptable risk. It is not possible to provide such a quantitative measurement using information flow analysis built upon noninterference.

We propose a joint analysis technique using noninterference and information theoretic measures. The key idea is to use noninterference properties as a first step. This allows us to leverage our substantial work using GLIFT. However, whenever these properties are violated, we can fallback on more quantitative metrics to provide a better understanding about the security of hardware designs.

B. Usage Scenarios

Analyzing Cryptographic Cores: Cryptographic engines are ubiquitous in secure systems for protecting confidentiality, data integrity, authentication, and non-repudiation. However, there is a fundamental problem when attempting to prove properties about the cryptographic system – the ciphertext output is by definition a function of the secret key.

Many security properties related to cryptographic cores attempt to insure that the key is not leaked. Thus we wish to mark the key as `HIGH` and write properties about where it should and should not flow. However, information flow analysis will rightly determine that the key flows to the ciphertext, and then this often leads to a taint explosion; it flows to a large part of the system. However, the key is mathematically secure at the ciphertext. Thus, it should not be marked as flowing there. There are ways around this using noninterference based information flow analysis by declassifying the ciphertext data (e.g., forcing the labels of the ciphertext to `LOW`). But this requires separate analysis to insure that it is sound. In Section IV-A show that quantitative information theoretic methods can handle this in a more natural manner.

Quantifying Effectiveness of Mitigation Techniques: Security mitigation techniques are often not formally validated. For example, a technique may be shown to be more effective against certain attacks in a particular situation. But it is often laborious to formally prove that a system is secure with respect to that property. Additionally, even if a mitigation technique is known or proven to be effective, it might be difficult to implement precisely to the specification. And a subtle deviation could lead to additional security vulnerabilities. Finally, there are often many different mitigation techniques that one can use to secure a system. For example, there are a large number of techniques to make a memory system more secure. Yet, it may not be clear which technique is best especially for the system under development. The joint analysis technique that we propose will enable designers to measure the effect of mitigation techniques on their hardware design. Section IV provides an example of how we can use quantitative measures to understand the effect of different mitigation techniques on the security of RSA IP cores.

IV. Design Examples

In this section, we analyze hardware cryptographic designs using noninterference and quantitative security properties. Throughout the section, we use an AES and a RSA core from *opencores.org*. Both cores are implemented according to their standard specifications and tested by the designers to make sure that they are functionally correct. We also use a Mini-AES core that we designed (see Figure 3); it is small-scale version of AES that we use largely for illustration purposes. Each of the crypto cores encrypt plaintexts under given key to produce ciphertexts. They use a *ready* signal to indicate when the ciphertext is valid.

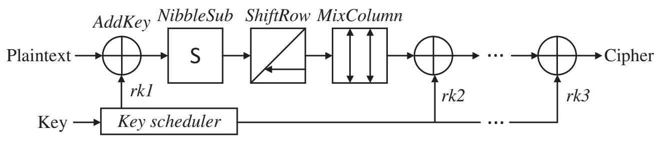

Fig. 3. The Mini-AES cryptographic function. Other cryptographic cores have a similar interface but perform different computations.

A. Functional and Timing Flow Analysis

It is important to understand where the key flows in the cryptographic system. Thus, we label it as tainted (`HIGH`) to indicate that it is something that we wish to track. Then we use the Mentor Graphics Questa Formal tool to check which signals can be tainted, i.e., we want to know where the key can flow. The results show that the key flows to the ciphertext for Mini-AES, AES, and RSA, which is expected. Information flow tracking techniques using noninterference will always indicate that there are flows from both the key and plaintext to the ciphertext. That is, if we mark either the key or the plaintext as tainted, the ciphertext should be tainted at the time when the ready signal is asserted. This is because both the key and plaintext always have an influence on the ciphertext. From the perspective of noninterference, a change in the key or plaintext will always be observed at the ciphertext.

Information theoretic measures differ in an important way – the mutual information between the key and ciphertext is *zero* bits when the key is a constant. Thus, it correctly states that one cannot learn any information about the key by merely observing the varying ciphertexts. This states that a correctly implemented and configured cryptographic function will not leak information about the key through the ciphertext. Additionally, the mutual information will indicate that the plaintext fully flows to the ciphertext. This is because cryptographic functions essentially perform one to one mapping from the plaintext to the ciphertext using a given key. This is a necessary condition to insure that the ciphertext can be decrypted. It should be noted that the mutual information would be nonzero if the key varies. In this case, you are learning information about the key, i.e., the key has been changed.

This shows that information flow tracking can be overly conservative. Despite the fact that there is no information flow from the key to the ciphertext as shown by the mutual information results, the GLIFT results tend to indicate otherwise. Nevertheless, it is possible to use these GLIFT and mutual information in concert to analyze the security of the system more efficiently and effectively. If GLIFT indicates no information flow (i.e., noninterference), the mutual information measurements will be zero. If GLIFT indicates a possible flow, we can further use mutual information to determine the actual amount of information flow.

The noninterference analysis declares that the key flows to the *ready* signal in the RSA core. In other words, GLIFT shows that the *ready* output of the RSA core may reveal information about the secret key. More specifically, this indicates that there is a timing channel in the RSA implementation since the key affects *ready* in a timing manner. By comparison, there is no timing channel in the AES or Mini-AES cores. This is because the time needed to compute these encryptions is invariant, and does not depend on the key.

We have detected a timing channel in the RSA core using GLIFT. The next question that one may ask is how much information if conveyed through this timing channel? Thus, we investigate if it is possible to use quantitative security measures to determine the amount of information that leaks from the key through the timing channel.

B. Analysis of Mitigation Techniques

In this subsection, we will use information theoretic metrics to measure the timing leakage in a number of RSA architectures each of which employs different timing channel mitigation techniques. We compare these measurements with the success of the attacking the timing side channel in order to understand if these metrics can serve as a measure of security.

The basic operation of RSA is modular exponentiation, which can further implemented using modular multiplication. The runtime variations originate from the unbalanced conditional branches in these two operations. We apply mitigation techniques to these two modules and create six unique 32-bit RSA architectures as shown in Table I.

R-to-L and *L-to-R* implement the right-to-left and left-to-right repeated squaring algorithms respectively. These two architectures are reference designs without any mitigation. *L-to-R always* adds a dummy modular multiply operation to the conditional branches in the modular exponentiation module of the *L-to-R* implementation in order to balance the delay difference. *Power ladder* re-designs the algorithm flow to balance

TABLE I

RSA architectures with different mitigation techniques. The first column is the number used Figure 4. The second column has the name we use in the text. Mitigation technique describes the change in the design. Time is the average number of cycles required by each design.

Number	Name	Mitigation Technique	Time
1	R-to-L	No mitigation	3023
2	L-to-R	No mitigation	1558
3	L-to-R always	Dummy modular multiplication	1975
4	Power ladder	Re-design algorithm flow	2099
5	Montgomery	Constant modular mult	1768
6	Constant time	Constant total runtime	4500

the delay difference of the conditional branches. *Montgomery* uses a constant time Montgomery multiplier so that the total runtime is dominated by the key. One cannot attack the design by specifying different plaintexts and observing the runtime differences. *Constant time* adds additional delay to make the total runtime constant for all keys and plaintexts. The last column of the table shows the average number of clock cycles needed by each architecture to compute a RSA encryption.

We use *entropy* and *mutual information* as security metrics to quantify the amount of timing leakage. We take the above mentioned six RSA implementations and test 2000 non-redundant keys for each to collect the total runtime measurements T. The plaintext is set to constants in our test so that the runtime measurements is only dependent on the key. We calculate the entropy of runtime samples T and the mutual information between key bit vectors k_i $(1 \leq i \leq 32)$ and T. Then, we use Kocher's method [33] to attack each of the designs and obtain the success rates.

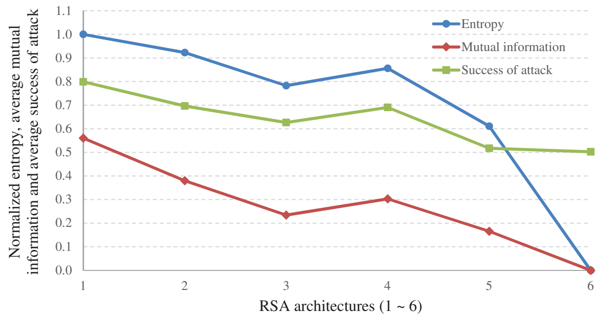

Fig. 4. Normalized entropy, average mutual information and average attack success rate over all key bits for different RSA architectures.

Figure 4 plots the normalized entropy, the average mutual information measurements along with the average success of attack over all key bits. Both entropy and mutual information measurements increase or decrease in accordance with the attack success rates, which indicates that entropy and mutual information may serve as a metric of timing leakage. It also provides insight about how effective a mitigation technique could be. Specifically, the designs with no mitigation techniques have relatively higher timing leakage. The *Power ladder* technique is relatively less efficient than *L-to-R always* in mitigating a RSA timing channel. By contrast, the *Montgomery* is the most effective yet practical technique. The *Constant time* implementation completely mitigates the timing channel, resulting an attack success rate around 50%, which is pretty much random guess. However, it comes at a significant cost of performance.

From our observations, the number of unique runtime samples is the key parameter to model timing characteristics. If the runtime samples spread in a wider range, the probability of observing each sample will decrease. According to information theory, a rare event carries more information when it happens. Thus, observing each runtime sample will reveal more information about the key. If the runtime samples distribute within a smaller range, each runtime sample may correspond to multiple possible keys and thus provides less information about the key. Mathematically, the amount of timing leakage revealed by each runtime sample can be modeled by self-information. Let $p(t_j)$ denote the probability of observing t_j, the following formalizes the amount of leakage.

$$I_s = \log_2 p(t_j), t_j \in T$$

Entropy is the expected value of self-information. It measures the average amount of information leakage over all runtime observations. Thus, entropy can be a metric for quantifying timing leakage. However, entropy only considers the runtime observations. This works well in our test, where we use non-redundant keys. Entropy measurements can be imprecise when considering different distributions of the key. Mutual information moves a step further and takes the distribution information of key into account. Thus, mutual information can be a more effective metric for quantifying the amount of timing information flow.

V. Design Challenges and Opportunities

Methods for Efficiently Calculating Security Metrics: Our security metrics are typically dependent on statistical analysis. The primary computational step in calculating these metrics is to estimate the distribution of random variables. An interesting question would be how to achieve an estimation close to the actual distribution while collecting as few samples as possible. This process also benefits from a hardware accelerated emulation framework for collecting and analyzing data.

Nonparametric Measures for Hardware Security Metrics: We have identified that the strengh of mutual information lies in its incorporation of knowledge about the key, and the importance of distribution estimation. Alternative measures which do not require assumptions about the data distribution—whatever data we find to be useful—are nonparametric values such as Kendall's τ or Spearman's ρ. These rank correlation methods allow testing for associations regardless of distribution, and may provide insight when the connection between observations and design security is highly nonlinear.

Languages for Specifying Security Properties: The security properties must be specified in a formal and succinct manner. This must be balanced with expressibility and usability of the language. Ideally, it does not require a security expert to write the properties especially considering they require understanding of the hardware. Thus the language must be usable by both security and hardware experts. Furthermore, a language similar to that used in hardware design (e.g., System Verilog) would be easier to invoke using existing hardware testing and verification tools.

VI. Conclusion

We need better tools to determine the security of hardware designs. Up until this point, hardware information flow analysis has used noninterference as a model of security. However, information theoretic metrics can more precisely quantify the amount of information flow. We propose a joint information flow analysis that combines noninterference with quantitative measures that enables hardware designers to specify and check if a system adheres to the desired security properties. Design examples have demonstrated that our method can be used to model security flaws and measure the effectiveness a different mitigation techniques.

Acknowledgment

This work was supported by NSF grant 1527631.

References

[1] S. Wasson, "Errata prompts intel to disable tsx in haswell, early broadwell cpus," August 2014, http://techreport.com/news/26911/.

[2] L. Constantin, "Design flaw in intel processors opens door to rootkits, researcher says," August 2015, http://www.pcworld.com/article/2965872/.

[3] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in *Proc. of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08, Berkeley, CA, USA, 2008, pp. 5:1–5:8.

[4] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Proc. of the 15th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 197–214.

[5] R. Waugh, "Could a vulnerable computer chip allow hackers to down a boeing 787? 'back door' could allow cyber-criminals a way in," May 2012, http://www.dailymail.co.uk/sciencetech/article-2152284/.

[6] S. Adee, "The hunt for the kill switch," *IEEE Spectr.*, vol. 45, no. 5, pp. 34–39, May 2008.

[7] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE J. Sel. Areas Commun.*, vol. 21, p. 2003, 2003.

[8] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information flow control for standard os abstractions," in *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2007, pp. 321–334.

[9] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris, "Labels and event processes in the asbestos operating system," in *Proc. of the Twentieth ACM Symposium on Operating Systems Principles*, ser. SOSP '05. New York, NY, USA: ACM, 2005, pp. 17–30.

[10] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," *SIGOPS Oper. Syst. Rev.*, vol. 38, no. 5, pp. 85–96, Oct. 2004.

[11] J. Newsome and D. Song, "Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software," in *Proc. of the Network and Distributed Systems Security Symposium*, Feb. 2005.

[12] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: A flexible information flow architecture for software security," in *Proc. of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 482–493.

[13] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *Proc. of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV. New York, NY, USA: ACM, 2009, pp. 109–120.

[14] X. Li, V. Kashyap, J. K. Oberg, M. Tiwari, V. R. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong, "Sapper: A language for hardware-level security policy enforcement," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1. ACM, 2014, pp. 97–112.

[15] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," in *Proc. of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS15)*, 2015.

[16] J. A. Goguen and J. Meseguer, "Security Policies and Security Models," in *IEEE Symp. on Sec. and Privacy*. Los Alamitos, CA, USA: IEEE, 1982, pp. 11–20.

[17] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner, "Theoretical fundamentals of gate level information flow tracking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 8, pp. 1128–1140, 2011.

[18] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, "Information flow isolation in i2c and usb," in *48th Design Automation Conference (DAC'11)*. IEEE, 2011, pp. 254–259.

[19] H. M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 583–594, 2013.

[20] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood, "Crafting a usable microkernel, processor, and i/o system with strict and provable information flow security," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 189–199.

[21] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner, "A practical testing framework for isolating hardware timing channels," in *Proc. of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1281–1284.

[22] J. Oberg, T. Sherwood, and R. Kastner, "Eliminating timing information flows in a mix-trusted system-on-chip," *IEEE Des. Test. Comput.*, vol. 30, no. 2, pp. 55–62, 2013.

[23] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner, "Leveraging gate-level properties to identify hardware timing channels," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 9, pp. 1288–1301, 2014.

[24] D. E. Robling Denning, *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.

[25] J. K. Millen, "Covert channel capacity," in *2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1987, pp. 60–60.

[26] J. McLean, "Security models and information flow," in *Security and Privacy, Proc. of 11th IEEE Computer Society Symposium on*. IEEE, 1990, pp. 180–187.

[27] J. W. Gray III, "Toward a mathematical foundation for information flow security," *Journal of Computer Security*, vol. 1, no. 3, pp. 255–294, 1992.

[28] A. McIver and C. Morgan, "A probabilistic approach to information hiding," in *Programming methodology*. Springer, 2003, pp. 441–460.

[29] J. Newsome, S. McCamant, and D. Song, "Measuring channel capacity to distinguish undue influence," in *Proc. of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*. ACM, 2009, pp. 73–85.

[30] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden, "Anonymity protocols as noisy channels," in *Trustworthy Global Computing*. Springer, 2007, pp. 281–300.

[31] G. Smith, "On the foundations of quantitative information flow," in *Foundations of Software Science and Computational Structures*. Springer, 2009, pp. 288–302.

[32] B. Kopf and M. Durmuth, "A provably secure and efficient countermeasure against timing attacks," in *Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE*. IEEE, July 2009, pp. 324–335.

[33] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *CRYPTO '96: Proc. of the 16th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1996, pp. 104–113.