# Expanding Gate Level Information Flow Tracking for Multi-level Security

Wei Hu, Jason Oberg, *Student Member, IEEE*, Janet Barrientos, Dejun Mu, and Ryan Kastner, *Member, IEEE*

***Abstract*—Embedded systems found in critical infrastructures require tight information flow controls to prevent unintended interference between different system components. These critical embedded systems require extensive testing and verification to ensure strict enforcement of information flow policy. To assist in this process, gate level information flow tracking (GLIFT) has been proposed to expose all flows of information through Boolean gates. However, the current work in this realm only considers a two-level linear security lattice (`LOW` $\sqsubset$ `HIGH`). In this letter, we expand the theoretical aspects of GLIFT to multi-level security lattices and provide an analysis of the overheads using *IWLS* benchmarks. Results show that expanding GLIFT to a multi-level lattice produces overheads and we discuss potential research on its applications.**

***Index Terms*—Critical Embedded Systems, Hardware Security, Gate Level Information Flow Tracking, Security Lattice**

## I. Introduction

CRITICAL embedded systems such as those found in the military, industrial infrastructures and medical devices all require strict guarantees on information flow security because of the extremely high cost of a failure. These systems require rigorous design and testing to ensure that untrusted information never affects trusted computation or that secret information never leaks to unclassified domains. The requirements, for both integrity and confidentiality, can be captured by the formal model of information flow security. This model classifies data objects in a system into different security levels, tracks the flow of information between security domains, and enforces a specific security policy such as non-interference [1]. While non-interference is a strong and useful security policy, it requires tight information flow control (IFC) to prevent unintended interactions between different system components resulting from harmful flows of information.

Information flow tracking (IFT) is a frequently used technique for enforcing IFC. IFT associates a label with data, and monitors the propagation of this label through the system to check if sensitive data leaks to an unclassified domain or if integrity-critical components are affected by untrusted data. With more functional units, such as security primitives, being built into hardware to meet performance and power constraints, it is required that embedded security be enforced from the underlying hardware up. In this process, hardware assisted IFT methods have been deployed to capture harmful flows of information including those through hardware specific timing channels. Implicit flows resulting from these timing channels have been shown to leak secret keys in stateful elements such as caches [2] and branch predictors [3]. In addition, such timing flows can cause violations in real-time constraints, hindering real-time operations of a system or even rendering the critical system useless. Further, these channels are so hard to detect that they are usually identified only after operational critical security policies have been violated.

To allow full account for information flow security in critical systems, researchers have proposed Gate Level Information Flow Tracking (GLIFT) [4]. GLIFT monitors all digital information flows by tracking individual bits through Boolean gates. At such a low level of abstraction, GLIFT is able to capture all transition activities including register to register timing. As a result, all digital information flows are made explicit, including timing channels that are inherent in the underlying hardware but invisible to programmers. Previous work has illustrated the application of GLIFT for building verifiably information flow secure high-assurance systems. GLIFT has been shown to be effective in detecting timing channels in bus protocols such as $I^2C$ and USB [5]. In [6], an execution lease architecture was developed to strictly bound the effects of untrusted programs, employing GLIFT to show provable information flow isolation between execution contexts. Further, GLIFT has been used to build a provably information flow secure system from the ground level up [7].

Although GLIFT provides an effective approach for enforcing information flow security, the existing GLIFT method targets a two-level linear security lattice [8] and thus only considers two-level security labels, e.g., `trusted` $\sqsubset$ `untrusted` or, the dual, `unclassified` $\sqsubset$ `confidential`. However, most systems benefit from or require multi-level security (MLS). For example, data objects are usually classified into at least four security levels, namely `Top secret`, `secret`, `confidential` and `unclassified` in military systems. A two-level linear security lattice simply cannot be used for modeling such a policy. In addition, many systems tend to be interested in non-linear lattices for modeling security policies. For example, it is often desirable to have a policy which requires isolation of the highest security level (`Top Secret`) from several incomparable entities (e.g., `Secret US` and `Secret UK`). That is, the model specifies that `Secret US` and `Secret UK` are at the same level but represent two different objects. More specifically, `Top Secret` might be the label for a data encryption process which requires that `Secret US` and `Secret UK` learn nothing other than the

Manuscript received Sep. 27, 2012; revised Feb. 27, 2013; accepted Apr. 15, 2013. This manuscript was recommended for publication by G. Qu.

This work was supported by the NSF, under Grant CNS - 0910581.

W. Hu and D. Mu are with the School of Automation, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China (e-mail: vinnie@mail.nwpu.edu.cn; mudejun@nwpu.edu.cn).

J. Oberg, J. Barrientos and R. Kastner are with the Department of Computer Science and Engineering, University of California at San Diego, San Diego, CA 92093, USA (e-mail: jkoberg@cs.ucsd.edu; jabarrie@ucsd.edu; kastner@cs.ucsd.edu).

cipher-text while it is perfectly secure for processes `Secret US` and `Secret UK` to learn information about one another. Thus, we need to expand GLIFT to more general security lattices in order to adapt to a wider range of systems.

This letter focuses on how GLIFT can be expanded to multi-level security lattices. It discusses this expansion by defining label propagation rules, deriving tracking logic and performing overhead analyses for several frequently used lattices. Such an expansion is intended to provide insight into how GLIFT can be expanded in this general way and what sort of area and performance overheads should be expected.

## II. Preliminaries

Denning first proposed the lattice model for describing information flows policies [8]. An information flow policy can be modeled as a finite security lattice $\{SC, \sqsubseteq\}$, where $SC$ is the set of security classes indicating different security levels of data objects and $\sqsubseteq$ defines the partial order on these security classes. Let $\mathcal{L} : \mathcal{O} \rightarrow SC$ be a function which returns the security class of object $\mathcal{O}$. For example, $\mathcal{L}(x)$ denotes the security class of an object $x \in \mathcal{O}$. The security class of $A$ is no higher (or more restrictive) than that of $B$ if $\mathcal{L}(A) \sqsubseteq \mathcal{L}(B)$. In this case, information flowing from $A$ to $B$ will not violate the policy specified by the lattice and thus is secure.

Figure 1 shows several frequently used security lattices. Take the square lattice in Fig. 1 (d) as an example. The symbols in the lattice represent security classes, i.e., $SC$ = {`Unclassified`, `Secret1`, `Secret2`, `Top Secret`}. Here, `Secret1` and `Secret2` are two incomparable security classes that reside between `Top Secret` and `Unclassified`. The arrows show the permissible information flows that do not lead to a security policy violation and reflect the partial order defined by the security lattice.

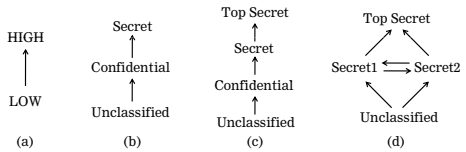


Fig. 1. Sample security lattices. (a) Two-level linear lattice. (b) Three-level linear lattice. (c) Four-level linear lattice. (d) A square lattice.

Let $\oplus$ denote the least upper bound operator on security classes. Given two security classes S1 and S2, S1 $\oplus$ S2 calculates the most restrictive security class S, satisfying that S1 $\sqsubseteq$ S and S2 $\sqsubseteq$ S. Consider the square lattice, we have `Unclassified` $\oplus$ `Secret1` = `Secret1` and `Secret1` $\oplus$ `Secret2` = `Top Secret`. Previous IFC methods tend to be conservative in calculating the security class for the output of an operation since they do not consider the *value* with which the objects can take but solely its security level. It is often the case that a higher security level object will not actually affect a lower one even if it is involved in a computation. Specifically, consider $n$ data objects $A1, A2, \cdots, An$ belonging to security classes S1, S2, $\cdots$, S$n$ respectively. When an operation is performed on these data objects, the security class of the output will be determined using (1). This is doubtlessly secure since we have $\mathcal{L}(Ai) \sqsubseteq$ S, $i = 1, 2, \cdots, n$. However, it can be conservative because information contained in $A1, A2, \cdots, An$ may not necessarily affect the output.

$$S = S1 \oplus S2 \oplus \cdots \oplus Sn \tag{1}$$

GLIFT provides a more precise approach to IFC in that the output is bounded to the most restrictive security class whose data actually affects the output. The existing GLIFT method targets the two-level linear lattice. Table I defines the label propagation rules for calculating the security class of the output on the two-input AND gate (AND-2). In the table, the symbol (S0, 0) represents a class S0 logic '0' input; (S1, 1) denotes a class S1 logic '1' input, and so forth.

TABLE I
GLIFT label propagation rule set for AND-2 on the two-level linear security lattice.

| ***AND*** | **(S0, 0)** | **(S0, 1)** | **(S1, 0)** | **(S1, 1)** |
|---|---|---|---|---|
| **(S0, 0)** | (S0, 0) | (S0, 0) | (S0, 0) | (S0, 0) |
| **(S0, 1)** | (S0, 0) | (S0, 1) | (S1, 0) | (S1, 1) |
| **(S1, 0)** | (S0, 0) | (S1, 0) | (S1, 0) | (S1, 0) |
| **(S1, 1)** | ***(S0, 0)*** | (S1, 1) | (S1, 0) | (S1, 1) |

For a better understanding, consider S0 and S1 as the `trusted` and `untrusted` security classes respectively From *row 4*, *column 1* of Table I, whenever one of the inputs of AND-2 is *trusted '0'*, the output will be dominated to (S0, 0). In this case, information contained in the other input will not be able to flow to the output since its output will be a constant '0'. This is opposed to the conventional operator which would have conservatively computed S0 $\oplus$ S1 = S1 as the output label even though (S1, 1) has no effect on the output.

Upon the basic ideas behind GLIFT, previous work [9] has formalized the theoretical aspects of GLIFT on the two-level linear lattice (`LOW` $\sqsubset$ `HIGH`). However, real systems usually employ MLS policies, which need to be modeled with multi-level security lattices. In the following sections, we expand the basic GLIFT technique to more general security lattices.

## III. GLIFT for Security Lattices

When $m$ is the level of lattice and $n$ is the number of inputs, the complexity of the label propagation rule set enumeration method would be $(2m)^n$. In the following subsections, we first restrict our discussion to two-input gates, whose GLIFT logic has to be create through naive enumeration. Then, we show how the general GLIFT logic generation problem actually can be reduced to solving just two-input gates in Section III-D.

### A. Three-level Linear Security Lattice

We focus on AND-2. Let S0, S1 and S2 denote `unclassified`, `confidential` and `secret` respectively in the three-level linear lattice as shown in Fig. 1 (b). One can expand two rows and columns upon Table I to include label propagation rules defined for S2 as shown in Table II.

Let $A$, $B$ and $O$ denote the objects representing the inputs and output of AND-2 while their security labels are denoted by $a_t$, $b_t$ and $o_t$ respectively. The GLIFT logic for AND-2 can be derived from Table II, which are shown in (2) and (3).

$$\begin{aligned} o_t[1] = {} & Ba_t[1]\overline{a_t[0]}\ \overline{b_t[1]} + A\overline{a_t[1]}b_t[1]\overline{b_t[0]} \\ & + a_t[1]\overline{a_t[0]}b_t[1]\overline{b_t[0]} \end{aligned} \tag{2}$$

TABLE II
GLIFT LABEL PROPAGATION RULES FOR AND-2 ON THE THREE-LEVEL LINEAR SECURITY LATTICE.

| ***AND*** | **(S0, 0)** | **(S0, 1)** | **(S1, 0)** | **(S1, 1)** | **(S2, 0)** | **(S2, 1)** |
|---|---|---|---|---|---|---|
| **(S0, 0)** | (S0, 0) | (S0, 0) | (S0, 0) | (S0, 0) | ***(S0, 0)*** | ***(S0, 0)*** |
| **(S0, 1)** | (S0, 0) | (S0, 1) | (S1, 0) | (S1, 1) | ***(S2, 0)*** | ***(S2, 1)*** |
| **(S1, 0)** | (S0, 0) | (S1, 0) | (S1, 0) | (S1, 0) | ***(S1, 0)*** | ***(S1, 0)*** |
| **(S1, 1)** | (S0, 0) | (S1, 1) | (S1, 0) | (S1, 1) | ***(S2, 0)*** | ***(S2, 1)*** |
| **(S2, 0)** | ***(S0, 0)*** | ***(S2, 0)*** | ***(S1, 0)*** | ***(S2, 0)*** | ***(S2, 0)*** | ***(S2, 0)*** |
| **(S2, 1)** | ***(S0, 0)*** | ***(S2, 1)*** | ***(S1, 0)*** | ***(S2, 1)*** | ***(S2, 0)*** | ***(S2, 1)*** |

$$\begin{aligned} o_t[0] = {} & B\overline{a_t[1]}a_t[0]\overline{b_t[1]} + \overline{B}a_t[1]\overline{a_t[0]}\ \overline{b_t[1]}b_t[0] \\ & + \overline{A}\ \overline{a_t[1]}a_t[0]b_t[1]\overline{b_t[0]} + A\overline{a_t[1]}\ \overline{b_t[1]}b_t[0] \\ & + \overline{a_t[1]}a_t[0]\overline{b_t[1]}b_t[0] \end{aligned} \tag{3}$$

For the three-level linear lattice, two-bit labels are used to denote three security classes. This leads to a *don't-care* input set since two binary bits can encode a total of four security classes. For a better understanding, assume S0, S1 and S2 are assigned binary codes "00", "01" and "10" respectively. Then, the input pattern "11" will be don't-care condition. Such *don't-care* input combinations will not lead to a security policy violation since the fourth security class never appears at the inputs of GLIFT logic. However, denoting such *don't-care* conditions to the logic synthesis tools will lead to better implementation results. Equations (4) and (5) give the GLIFT logic with consideration of the *don't-care* input set. These are less complex as compared to (2) and (3) respectively.

$$o_t[1] = Ba_t[1] + Ab_t[1] + a_t[1]b_t[1] \tag{4}$$

$$\begin{aligned} o_t[0] = {} & Ba_t[0]\overline{b_t[1]} + \overline{B}a_t[1]b_t[0] \\ & + A\overline{a_t[1]}b_t[0] + \overline{A}a_t[0]b_t[1] + a_t[0]b_t[0] \end{aligned} \tag{5}$$

For an $n$-level linear lattice, there are $(2n)^2$ elements in the label propagation rule set of AND-2. Thus, label propagation rule set enumeration soon becomes a complicated and error-prone process as $n$ grows. Instead, we use a more efficient way to derive GLIFT logic for AND-2 under arbitrary level of linear lattices in the next subsection.

### B. $N$-level Linear Security Lattice

Figure 2 provides a unified approach to deriving GLIFT logic under arbitrary level of linear lattices. Since each two of the security classes within an $n$-level linear lattice are comparable, we can use a comparator to convert the input labels, i.e., security classes, to two-level and use the GLIFT logic under the two-level linear lattice formalized in [9] for label propagation. At the output stage, a multiplexer is used to select the correct security class according to the output from the label propagation logic. This will enable the expanding of GLIFT to arbitrary linear security lattices.

As a sanity check, we generate the GLIFT logic for AND-2 under the four-level linear lattice both by expanding the rule propagation rule set upon Table II and using the method as shown in Fig. 2. These two methods lead to identical resulting circuits as given in (6) and (7).

$$o_t[1] = Ba_t[1] + Ab_t[1] + a_t[1]b_t[1] \tag{6}$$

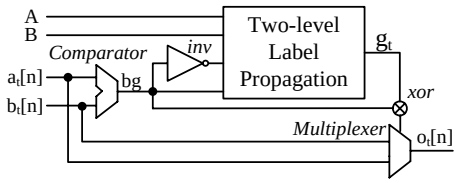


Fig. 2. Constructing GLIFT logic under $n$-level linear security lattice.

$$\begin{aligned} o_t[0] = {} & Ba_t[1]a_t[0] + \overline{B}a_t[1]\overline{b_t[1]}b_t[0] + Ba_t[0]\overline{b_t[1]} \\ & + A\overline{a_t[1]}b_t[0] + \overline{A}\ \overline{a_t[1]}a_t[0]b_t[1] + Ab_t[1]b_t[0] \\ & + a_t[0]b_t[0] \end{aligned} \tag{7}$$

### C. The Square Security Lattice

As shown in Fig. 1, the major difference between the square lattice and other linear ones lies in that it contains incomparable security classes, i.e., S1 and S2. This results in more subtle cases in label propagation. As an example, consider AND-2 with inputs (S1, 0) and (S2, 0). The output is doubtless logic '0'. In this case, both inputs have an influence at the output. Which label to choose at the output is not obvious since neither S1 nor S2 is more restrictive than the other. Fortunately, S1, S2 and even S3 are all safe security classes for the output since they will not lead to an information security policy violation. However, S1 and S2 are more restrictive than S3. Thus, either S1 or S2 can be selected as the security class for the output.

To make it different from the four-level linear lattice, we choose S1 for the AND-2 gate while S2 for the two-input OR gate (OR-2) when both S1 and S2 are safe. Under such convention, the GLIFT logic for AND-2 under the square lattice can be formalized as follows:

$$\begin{aligned} o_t[1] = {} & Ba_t[1]\overline{b_t[0]} + Ba_t[1]a_t[0] + A\overline{B}b_t[1] \\ & + \overline{A}Ba_t[1] + A\overline{a_t[0]}b_t[1] + Ab_t[1]b_t[0] \\ & + a_t[1]b_t[1] \end{aligned} \tag{8}$$

$$\begin{aligned} o_t[0] = {} & \overline{B}a_t[1]\overline{b_t[1]}b_t[0] + Ba_t[0] \\ & + \overline{A}\ \overline{a_t[1]}a_t[0]b_t[1] + Ab_t[0] + a_t[0]b_t[0] \end{aligned} \tag{9}$$

The problem with tracking information flows on a non-linear security lattices lies in that the security class of the output can be chosen too conservatively (although safe). A possible solution is to construct a candidate set of security classes that will not lead to an information flow policy violation and then choose the most restrictive one(s) from the candidate as the output label.

### D. A Constructive Approach

To generate GLIFT logic in linear time, we instantiate tracking logic for components in digital circuits discretely from a functionally complete GLIFT library. Usually, a minimized GLIFT library consisting of the tracking logic for the AND-2, OR-2 and the inverter is derived using the label propagation rule set enumeration method. More complex GLIFT libraries and circuits consisting of multiple-input gates can be created in a constructive manner. Figure 3 shows such a constructive method, which reduces the general GLIFT logic generation problem to solving just two-input gates.

TABLE III
AREA, DELAY AND POWER RESULTS OF GLIFT LOGIC CIRCUITS UNDER THE TWO TO FOUR LEVEL LINEAR AND SQUARE LATTICES.

| Benchmark | Area ($um^2$) | | | | Delay (ns) | | | | Power (mW) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2-lev | 3-lev | 4-lev | Square | 2-lev | 3-lev | 4-lev | Square | 2-lev | 3-lev | 4-lev | Square |
| alu2 | 9833 | 25787 | 30410 | 35571 | 2.10 | 2.71 | 2.66 | 5.02 | 1.69 | 4.99 | 6.45 | 6.95 |
| alu4 | 21242 | 54860 | 64457 | 75869 | 2.85 | 3.41 | 3.54 | 6.25 | 3.73 | 10.9 | 13.8 | 15.1 |
| pair | 44261 | 113885 | 133797 | 159606 | 1.63 | 2.03 | 1.99 | 3.63 | 7.29 | 19.9 | 26.9 | 28.8 |
| i10 | 60371 | 153896 | 183421 | 216059 | 3.48 | 4.61 | 4.24 | 8.32 | 9.00 | 28.3 | 34.6 | 37.7 |
| C1355 | 16854 | 42910 | 50449 | 59677 | 1.46 | 1.75 | 1.80 | 3.35 | 2.98 | 7.42 | 10.4 | 10.1 |
| C1908 | 13682 | 33978 | 40279 | 47558 | 2.26 | 2.66 | 2.73 | 4.92 | 2.75 | 4.75 | 9.22 | 9.55 |
| C2670 | 19670 | 50100 | 59216 | 69478 | 1.90 | 2.41 | 3.42 | 4.65 | 2.37 | 8.57 | 12.8 | 13.1 |
| C3540 | 32255 | 83947 | 98314 | 115755 | 2.66 | 3.20 | 3.10 | 5.62 | 5.26 | 14.3 | 19.8 | 21.0 |
| C5315 | 47318 | 122897 | 144400 | 171901 | 2.32 | 2.96 | 2.84 | 5.07 | 8.55 | 19.4 | 31.4 | 32.7 |
| C6288 | 83678 | 215020 | 250832 | 293322 | 8.73 | 9.84 | 10.2 | 17.1 | 15.7 | 23.8 | 53.8 | 52.8 |
| C7552 | 53603 | 135958 | 162224 | 190607 | 3.31 | 3.71 | 3.74 | 6.73 | 10.2 | 19.6 | 36.8 | 37.5 |
| DES | 102563 | 269418 | 314533 | 379610 | 1.30 | 1.64 | 1.62 | 3.19 | 18.6 | 48.1 | 71.6 | 83.8 |
| N. Average | 1.00 | 2.57 | 3.03 | 3.58 | 1.00 | 1.22 | 1.26 | 2.24 | 1.00 | 2.55 | 3.80 | 4.00 |

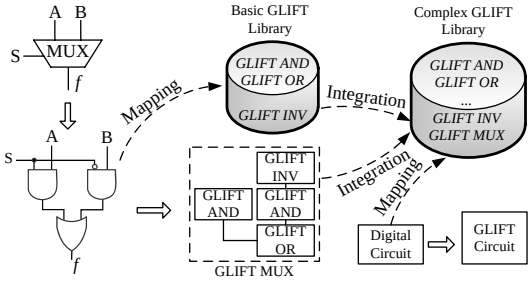


Fig. 3. The constructive method for GLIFT logic generation.

## IV. EXPERIMENTAL RESULTS

We generate GLIFT logic for several *IWLS* benchmarks under the two to four level linear and square security lattices. Tracking logic is augmented discretely in a constructive manner from a functionally complete GLIFT library. The resulting circuits are synthesized using *Synopsis Design Compiler* and targeted to its *90nm* standard cell library for area, delay and power reports, as shown in Table III.

From Table III, GLIFT logic typically reports larger area and delay and consumes more power as the security lattices grow more complex. Row *N. Average* shows the average area, delay and power normalized to those under the two-level linear lattice, reflecting design overheads. It should be noticed that GLIFT logic under the three-level linear lattice is relatively less complex than that for the four-level linear lattice. This is because we took the *don't-care* input set into consideration and denoted these *don't-cares* to the logic synthesis tools.

From the experimental results, expanding GLIFT to multi-level security lattices will result in considerable area and performance overheads. However, most systems require MLS policies modeled using more complex security lattices. The existing GLIFT technique should be expanded to meet such requirement. Security is a pressing problem in safety-critical embedded systems. Such overheads should definitely be tolerated since a single failure resulting from security issues will render critical embedded systems useless and cause tremendous losses. In real applications, there are usually partitions among security domains within a design. Only security critical portions of the design need to be augmented with GLIFT logic for dynamic IFT. In addition, GLIFT can also be used for static information flow security verification. The additional GLIFT logic can be removed when verification is complete. These will reduce the area and performance overheads and enable GLIFT to be employed for proving MLS.

## V. CONCLUSION

GLIFT provides an effective approach for enforcing tight information flow controls to prevent harmful flows of information, including those through hardware-specific timing channels. This letter expands the theoretical aspects of the basic GLIFT technique to more general security lattices, formalizing tracking logic and performing complexity analysis, which provides a possibility for proving MLS in critical embedded systems from the gates up. It also shows that significant area and performance overheads should be expected if MLS is required at this level of abstraction.

## REFERENCES

[1] J. A. Goguen and J. Meseguer, "Security policies and security models," *IEEE Symposium on Security and Privacy*, April 1982, pp. 11 –20.
[2] D. J. Bernstein, "Cache-timing attacks on aes," University of Illinois, Chicago, Tech. Rep. cd9faae9bd5308c440df50fc26a517b, 2005.
[3] O. A. Jean-Pierre, J. pierre Seifert, and C. K. Koc, "Predicting secret keys via branch prediction," in *Cryptology - CT-RSA 2007, The Cryptographers Track at the RSA Conference*. Springer-Verlag, 2007, pp. 225 –242.
[4] 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 (ASPLOS)*, New York, NY, 2009, pp. 109 –120.
[5] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, "Information flow isolation in i2c and usb," in *Proc. of Design Automation Conference (DAC)*, San Diego, CA, June 2011, pp. 254 –259.
[6] M. Tiwari, X. Li, H. M. G. Wassel, F. T. Chong, and T. Sherwood, "Execution Leases: A Hardware-Supported Mechanism for Enforcing Strong Non-Interference", In *Proc. of the International Symposium on Microarchitecture (Micro)*, Dec. 2009, pp. 189 –200.
[7] M. Tiwari, J. 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 *Proc. of the 38th annual international symposium on Computer architecture (ISCA'11)*, New York, NY, 2011, pp. 189 –200.
[8] D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.
[9] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood and R. Kastner, "Theoretical Analysis of Gate Level Information Flow Tracking", in *Proc. of the Design Automation Conference (DAC)*, 2010, pp. 244 –247.