A Simplifying Logic Approach for Gate Level Information Flow Tracking

Yu Tai, Wei Hu(✉), Dejun Mu, Baolei Mao, Lantian Guo, and Maoyuan Qin

School of Automation, Northwestern Polytechnical University, Xi'an, China
`taiyu@mail.nwpu.edu.cn`, `weihu@nwpu.edu.cn`

Abstract. With the increase of design scale and complexity, security vulnerabilities residing in hardware designs become hard to detect. Existing functional testing and verification methods cannot guarantee test and verification coverage in design phase. Fortunately, gate level information flow tracking (GLIFT) has been proposed to enforce bit-tight information flow security from the gate level to detect security vulnerabilities and prevent information leakage effectively. However, there is a significant limitation that the inherent high complexity of GLIFT logic causes significant overheads in static verification and physical implementation. In order to address the limitation, we propose a simplified GLIFT method that incorporates more detailed optimization logic routes to reduce its complexity and allow don't care to simplify original GLIFT logic. Experimental results have demonstrated that the simplified GLIFT method can reduce the design overhand in several gates by sacrificing a fraction of GLIFT precision.

Keywords: Hardware security · Gate level information flow tracking
Security lattice · Don't care · Optimization

1 Introduction

Existing functional verification methods cannot ensure testing and verification coverage or detect design flaws, security vulnerabilities, malicious code and other security vulnerabilities. Guaranteeing hardware security is a big challenge for designers due to the increasing size of circuit design and unpredictable supply chain. For example, an implementation flaw in QualComm's TrustZone is used to break Android's full disk encryption [1]. However, there is a fact that the traditional hardware design flows focus on function verification and provide limited security verification. It is almost inevitable that these designs will have design vulnerabilities and security flaws. Recent researches revealed that security vulnerabilities in hardware devices are more than designers might previously have thought [2]. There is a relatively high estimated rate of design flaws in millions of lines of hardware description language (HDL) code [10].

There are two information security policies non-interference [7] and Bell and LaPadula [3], which are frequently used to isolate data in the systems. While

B. Li et al. (Eds.): ChinaCom 2017, LNICST 237, pp. 302–311, 2018.
https://doi.org/10.1007/978-3-319-78139-6_31

these common security policies are very strong and strict, which they all require that trusted information should be never affected by untrusted and secret information should never leak to a public domain.

As a technique for enforcing information policies, information flow tracking (IFT) that has been proven to be effective in detecting security vulnerabilities and preventing attacks through side channels in the end-to-end designs, can be used to apply the principle of hardware designs to the specification of their security requirements. IFT is a frequently used method to analyze the security of a system [5,19]. IFT provides an effective approach for preventing unintended interaction between different components in a system. It is useful for deploying information flow security and enforcing its security properties at various levels of the entire system, including compiler/programming language [14], operating system [11,17], instruction set architecture [15] and hardware level [16,20]. However, the coarse-grained design rules are generally used in these previous IFT methods, which lead to higher design complexity [13,19], the huge number of additional overhead [11,20] or overly conservative propagation policies [4,18].

To fully account for information flow security, gate level information flow tracking (GLIFT), which uses fine granularity labels and propagation policies, has been proposed to precisely measure how information flows through Boolean gates in hardware system [16]. It captures and monitors all logical flows in a unified manner that enables the verification of important security properties related to confidentiality, integrity, and logical side channels. Recent work has shown that GLIFT can be used to investigate the precision and complexity tradeoffs of GLIFT through logic synthesis optimizations [8], guarantee information flow security in embedded systems [12], and detect hardware Trojans that violate information flow security properties related to confidentiality and integrity [9].

The original GLIFT logic method generates information flow tracking logic through direct GLIFT library mapping, which can be confronted with high overheads in terms of design complexity and verification performance. In this paper, we propose a new simplified method that use don't care to search out some optimized routes to generate the most simplified GLIFT logic. Specifically, this paper makes the following contributions:

1. Proposing an approach to search out the most simplified logic routes for gate level information flow tracking;
2. Providing this simplified GLIFT logic method to implement reduced GLIFT logic and optimized design;
3. Presenting optimized and comparative analysis using primitive gates with don't care to evaluate area and delay.

The reminder of this paper is organized as follows. Section 2, we briefly cover the background in security lattices and GLIFT. In Sect. 3, we discuss how to carry out the label propagation and mapping. We propose the simplified method to optimize GLIFT logic in Sect. 4. Section 5 presents experimental results using don't care optimizations. We conclude the paper in Sect. 6.

2 Background

We explain background work in this section that enable a better understanding of our research. We focus on various background literatures including security lattice model, gate level information flow tracking and two-level information flow tracking.

2.1 Security Lattice Model

The lattice model was first proposed for describing information flow policy by Denning [6]. An information flow policy can be modeled as a finite security lattice L = {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.

The symbols in the lattice represent security classes i.e., SC = {Unclassified, Confidential}. Defined function L: x $\rightarrow$ SC to represent safety data object x belongs to the security class. The arrow indicates the direction of information flows that reflect security policy and the partial order defined by the security lattice. Let U, C, S and T denote Unclassified, Confidential, Secret and Top Secret respectively. For beter understanding, Let U, C, and S denote Unclassified, Confidential and Secret respectively in the three-level linear security lattice. So security class set is simplified to SC = {U, C, S}. Security class set can be simplified to SC= {U, C, S, T} in the four-level linear security lattice.

2.2 Gate Level Information Flow Tracking

Gate level information flow tracking (GLIFT) is established for monitoring the flow of information that models digital information flows using bit level labels at the Boolean level [16]. GLIFT provides a model for understanding how data propagates through a design. In GLIFT, data are typically assigned a label to characterize their trustworthiness or security level, such as untrusted/trusted, low/high or unclassified/confidential/secret. It adopts fine-grained labels to implement strict and precise analysis at Boolean level, which this label is propagated through the system under pre-defined policies to prevent unintended information flows. Each binary data bit is associated with a tag called taint label (security label). When the data involve in computing in the hardware system, the taint label also will be propagated in this hardware circuit. The information included in the data is called to be tainted when its taint is logic true ‘1’ and untainted when its taint is logic false ‘0’. Taint is propagated from the input to the output of the hardware design if and only if the value of any tainted input has an influence on the output.

2.3 Two-Level Information Flow Tracking

Binary security properties can be verified on a two-level information flow tracking model. The information flow security property enforced by the corresponding security lattice states that `HIGH` information should never flow to a `LOW` portion.

Two-level information flow tracking model targets a two-level security lattice `LOW` $\sqsubseteq$ `HIGH`. Under such a lattice, all signals in the hardware design will be classified as either `LOW` or `HIGH`. Then, a two-level information flow model will be used to understand the flow of `HIGH` information. Consider a two-input AND (AND-2) gate and let input A be `LOW` while input B be `HIGH`. The information flow model should precisely track when the `HIGH` information in B could flow to the output O. Table 1 illustrates such a model.

Table 1. A partial two-level information flow model for AND-2

A	B	O
(LOW, 0)	(HIGH, 0)	(LOW, 0)
(LOW, 0)	(HIGH, 1)	(LOW, 0)
(LOW, 1)	(HIGH, 0)	(HIGH, 0)
(LOW, 1)	(HIGH, 1)	(HIGH, 1)

According to the notion of information flow, the `HIGH` information in B flows to O *if and only if* B has an effect on the output. In the first two rows of Table 1, the `LOW` input dominates the output and thus there is no `HIGH` information flow.

Table 2. A partial two-level information flow model for OR-2

A	B	O
(LOW, 0)	(HIGH, 0)	(HIGH, 0)
(LOW, 0)	(HIGH, 1)	(HIGH, 1)
(LOW, 1)	(HIGH, 0)	(LOW, 1)
(LOW, 1)	(HIGH, 1)	(LOW, 1)

Similarly, consider a two-input OR (OR-2) gate and let input A be `LOW` while input B be `HIGH`, the partial results for OR-2 can be shown in Table 2. Information of a higher security level cannot flow to the output when an input of lower security level dominates the output. The difference is that the model allows a finer classification of security labels, which is desirable when reasoning about the security between multiple parties.

3 Label Policy and Mapping

In this section, we provide an approach for information flow tracking. A fundamental problem in this process is to define label propagation policies and map label encoding for GLIFT library.

3.1 Label Propagation Policy

The two-level IFT model provides a precise measurement of if there is a flow. Take a two-input AND (AND-2) gate for example. Let A, B and O be its inputs and output respectively. Use At, Bt and Ot to denote their security labels under two-level IFT. The label propagation policy for AND-2 under two-level IFT can be formalized as (1).

$$Ot = A \cdot Bt + B \cdot At + At \cdot Bt \tag{1}$$

Equation (1) indicates that when A is logical `1` and B is `HIGH`, the output will be `HIGH`; when B is logical `1` and A is `HIGH`, the output will also be `HIGH`; Or when both A and B are `HIGH`, the output will doubtlessly be `HIGH`. It cares about if there is any `HIGH` information flow from either A or B to the output.

As mentioned, the two-level IFT model precisely indicates if there is a flow. When there is no flow, the security label should be all zeros. The subtle case is when there are `HIGH` inputs. The label propagation policy should track the `HIGH` information flow from each of these inputs to the output. Figure 1 describes such a label propagation policy for AND-2 gate and OR-2 gate, which track the information flow from both inputs to the output.

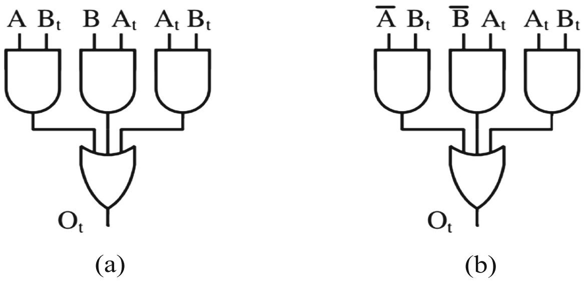

A B_t B A_t A_t B_t
O_t
(a)
$\overline{A}$ B_t $\overline{B}$ A_t A_t B_t
O_t
(b)

Fig. 1. (a) Label propagation policy for AND-2 gate; (b) Label propagation policy for OR-2 gate

From Fig. 1(a), A and B are the original inputs of AND; At and Bt are the two-level security labels; We define Ot as an output variable, which reflects the two-level security label for the output. The OR-2 gate is the dual of AND-2 according to Demorgan's law. Thus, its label propagation policy can be derived by inverting the original inputs A and B. Using the same notation as AND-2, it can be seen from Fig. 1(b).

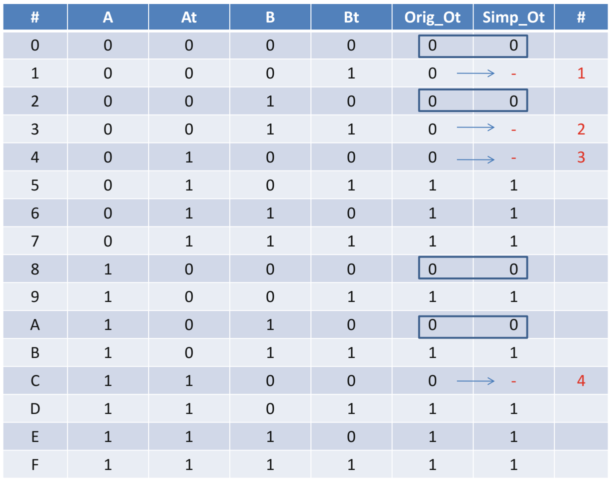
3.2 Label Mapping

To track the propagation of information flows, we need to set up the single bit security label for two-level IFT or binary security label for multi-level IFT (three-level or four-level) to a one-hot encoding. In this way, each bit in the security label only tracks the propagation of one bit. For a better understanding, the two-level IFT method targets a two level security label. Thus, we have the taint labels $At[w] \in \{0, 1\}$, where $0 \leq w \leq 3$. The multi-level IFT method allows a finer classification of security labels. If each individual bit in A takes a different label, we have the taint labels $At[w] \in \{00, 01, 10, 11\}$. Using a binary encoding, two bits will be enough for encoding the taint labels.

4 Simplified Model

4.1 Logic Optimization with Don't Care

As a common optimization method for logic synthesis in integrated circuit design, don't care computation can be used to optimize GLIFT logic, where the different routes in the logic network may affect the way in which signal flows through hardware design. Consider two-level linear security lattice (such as Unclassified and Confidential). If don't care is not considered, AND-2 GLIFT logic is described as first row in Table 3. In the scheme as Fig. 2, we can study the various routes for don't care to achieve an optimized GLIFT logic. After the above don't care is considered, AND-2 GLIFT logic can be simplified as Table 3 at simplified nodes in different logic synthesis.



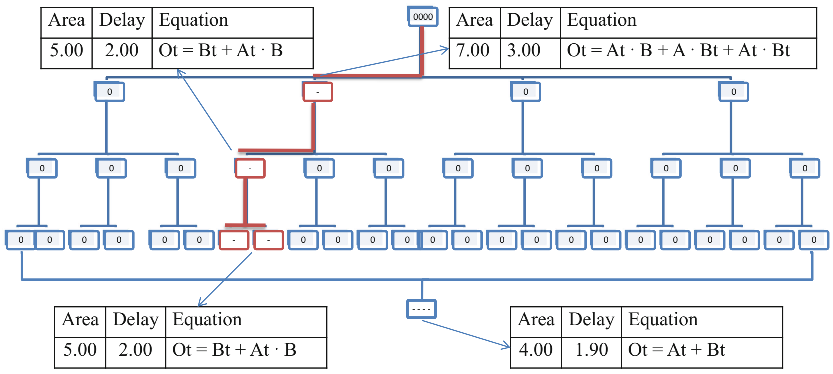
#	A	At	B	Bt	Orig_Ot	Simp_Ot	#
0	0	0	0	0	0	0	
1	0	0	0	1	0	$\rightarrow$ -	1
2	0	0	1	0	0	0	
3	0	0	1	1	0	$\rightarrow$ -	2
4	0	1	0	0	0	$\rightarrow$ -	3
5	0	1	0	1	1	1	
6	0	1	1	0	1	1	
7	0	1	1	1	1	1	
8	1	0	0	0	0	0	
9	1	0	0	1	1	1	
A	1	0	1	0	0	0	
B	1	0	1	1	1	1	
C	1	1	0	0	0	$\rightarrow$ -	4
D	1	1	0	1	1	1	
E	1	1	1	0	1	1	
F	1	1	1	1	1	1	

Fig. 2. Optimized logic using don't care

Logic synthesis has an effect on the complexity and precision of the GLIFT logic in two ways. On one hand, logic synthesis can reduce design redundancy to be frequently used for logic optimization. By comparison, the GLIFT logic equations are shown in Fig. 3, we see that logic synthesis can possibly make the GLIFT logic more simplified. On the other hand, this will typically make the resulting GLIFT logic less precise.

4.2 Logic Library Mapping

For the above approach, we can derive different simplified versions of GLIFT logic for the AND-2, which has 4 different minterms between the most and least precise GLIFT logic. This yields a total of 16 possible combinations for the simplified logic with don't care on each route. This reveals how many versions of GLIFT logic can be derived, which shows in Fig. 3 using AND-2 as an example. From Fig. 3, in the second optimized route, we have the most precise GLIFT logic for AND-2 At the top level, where A, B and O are the inputs and output of AND-2; At, Bt and Ot are their security labels respectively. By setting any possible combinations of the inputs with don't-care as Fig. 3, we can obtain three simplified versions of GLIFT logic as shown at the different nodes in this route. When the four observing zeros are all don't cares, we reach the most simplified GLIFT logic for AND-2, shown at the last node. In this way, we reduce the complexity to achieve a simplified version GLIFT logic through a tractable route with don't care. Now that we can create a simplified GLIFT library, we can map logic primitives to each alternative version of its GLIFT logic in order to obtain simplified logic, i.e. equations and reduce the overhead, i.e. area and delay. It can be seen that the area, delay and equations from precise version to simplified one of AND-2 GLIFT logic from Table 3.



Area	Delay	Equation
5.00	2.00	Ot = Bt + At · B

Area	Delay	Equation
7.00	3.00	Ot = At · B + A · Bt + At · Bt

Area	Delay	Equation
5.00	2.00	Ot = Bt + At · B

Area	Delay	Equation
4.00	1.90	Ot = At + Bt

Fig. 3. Simplified tracking logic model

Table 3. Area, delay and equation with don't care for AND-2

#	A	B	Area	Delay	Equation
1			7.00	3.00	$Ot = At \cdot B + A \cdot Bt + At \cdot Bt$
2	-		5.00	2.00	$Ot = Bt + At \cdot B$
3		-	5.00	2.00	$Ot = At + A \cdot Bt$
4	-	-	4.00	1.90	$Ot = At + Bt$

5 Experimental Results

This section presents our experimental results. It shows how GLIFT logic can be simplified while introducing don't care, and demonstrates that the most precise and simplified GLIFT logic can result in overheads as an example of AND-3. Area and delay need to be taken into consideration in GLIFT logic design for information flow tracking. ABC is used as the synthesis tool. The resyn2 synthesis script in ABC is used to optimize the GLIFT logic circuits that can provide a good tradeoff between area and delay. The ABC mcnc library does not provide units for area and delay results.

We carried out experiments on several N input AND gate (AND-N) to obtain area and delay results of the different GLIFT logic. In our experiments, the GLIFT logic is generated using the method discussed in the Sect. 4. Table 4 shows some statistics of the original and simplified GLIFT logic area in our experiments including the different AND-N gates. Figure 4 shows some original and simplified delay reports of GLIFT logic in these gates. From Table 4 and Fig. 4, we can discover that GLIFT logic represented in the simplified method gives significantly smaller area and delay. As an example, the GLIFT logic for AND-4 described using the original GLIFT reports an area/delay of 24.00/4.60, while that represented using the simplified one reports a result of 8.00/2.30; there are significant reduction in area and delay.

Subsequently, we carried out experiments to obtain area and delay reports of AND-3 for GLIFT logic circuits represented the schemes in Sect. 4. GLIFT logic formalization for AND-3 results are also generated using the schemes to verify optimized GLIFT logic. Table 5 shows some statistics of the area,

Table 4. Original and simplified GLIFT logic area for AND-N

AND-N	AND-2	AND-3	AND-4	AND-5	AND-6	AND-7	AND-8
Area_orig	7.00	11.00	24.00	123.00	176.00	489.00	1082.00
Area_simp	4.00	6.00	8.00	8.00	9.00	11.00	12.00

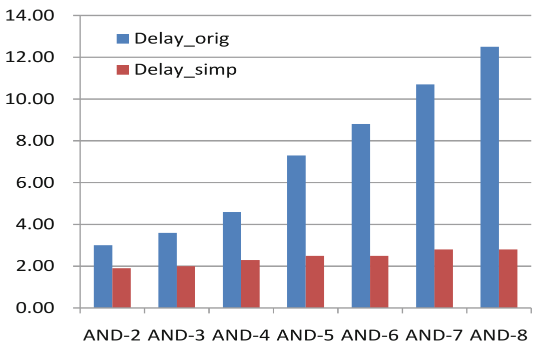


Fig. 4. Original and simplified GLIFT logic delay for AND-N

delay and equations of these logics in our experiments. It can be seen that the area/delay/equation of 6.00/2.00/$Ot = At + Bt + Ct$ are the most optimized results when all three variables are introduced as don't cares.

Table 5. Area, delay and equation with don't care for AND-3

#	A	B	C	Area	Delay	Equation
1	-			9.00	3.20	$Ot = At \cdot B \cdot C + Bt \cdot C + B \cdot Ct + Bt \cdot Ct$
2		-		9.00	3.20	$Ot = At \cdot C + A \cdot Bt \cdot C + A \cdot Ct + At \cdot Ct$
3			-	9.00	3.20	$Ot = At \cdot B + A \cdot Bt + A \cdot B \cdot Ct + At \cdot Bt$
4	-	-		6.00	2.60	$Ot = At \cdot C + Bt \cdot C + Ct$
5	-		-	6.00	2.60	$Ot = At \cdot B + Bt + B \cdot Ct$
6		-	-	6.00	2.60	$Ot = At + A \cdot Bt + A \cdot Ct$
7	-	-	-	6.00	2.00	$Ot = At + Bt + Ct$
8				11.00	3.60	$Ot = At \cdot B \cdot C + A \cdot Bt \cdot C + A \cdot B \cdot Ct + At \cdot Bt \cdot C + At \cdot B \cdot Ct + A \cdot Bt \cdot Ct + At \cdot Bt \cdot Ct$

6 Conclusion

The GLIFT method can effectively monitor and measure all logical information flows at gate level to detect security vulnerabilities and prevent leakage of sensitive information in hardware designs. This paper presents the method of optimized GLIFT logic that search out some simplified routes to decrease logic complexity using don't care in design logic synthesis. The experimental results show that the method can effectively reduce the area and delay of original GLIFT logic.

Acknowledgments. This research was financially supported by the National Natural Science Foundation of China under Grant 61303224 and Grant 61672433, the National Cryptography Development Fund under Grant MMJJ20170210.

References

1. Extracting qualcomms keymaster keys - breaking android full disk encryption (2016). http://bits-please.blogspot.com/2016/06/extractingqualcomms-keymaster-keys.html
2. Becker, G.T., Regazzoni, F., Paar, C., Burleson, W.P.: Stealthy dopant-level hardware trojans. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 197–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_12
3. Bell, D.E., LaPadula, L.J.: Secure computer systems: mathematical foundations. Technical report, DTIC Document (1973)

4. Dalton, M., Kannan, H., Kozyrakis. C.: Raksha: a flexible information flow architecture for software security. In: ACM SIGARCH Computer Architecture News, vol. 35, pp. 482–493. ACM (2007)
5. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
6. Denning, D.E.: Cryptography and Data Security. Addison-Wesley Longman Publishing Co., Inc., Boston (1982)
7. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and privacy, vol. 11, p. 77 (1982)
8. Hu, W., Becker, A., Ardeshiricham, A., Tai, Y., Ienne, P., Mu, D., Kastner, R.: Imprecise security: quality and complexity tradeoffs for hardware information flow tracking. In: Proceedings of the 35th International Conference on Computer-Aided Design, p. 95. ACM (2016)
9. Hu, W., Mao, B., Oberg, J., Kastner, R.: Detecting hardware trojans with gate-level information-flow tracking. Computer **49**(8), 44–52 (2016)
10. Keating, M.: The Simple Art of SoC Design: Closing the Gap Between RTL and ESL. Springer Science & Business Media, Heidelberg (2011). https://doi.org/10.1007/978-1-4419-8586-6
11. Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: ACM SIGOPS Operating Systems Review, vol. 41, pp. 321–334. ACM (2007)
12. Mu, D., Hu, W., Mao, B., Ma, B.: A bottom-up approach to verifiable embedded system information flow security. IET Inf. Secur. **8**(1), 12–17 (2014)
13. Pottier, F., Simonet, V.: Information flow inference for ML. ACM Trans. Program. Lang. Syst. (TOPLAS) **25**(1), 117–158 (2003)
14. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J. Sel. Areas Commun. **21**(1), 5–19 (2003)
15. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S.L Secure program execution via dynamic information flow tracking. In: ACM Sigplan Notices, vol. 39, pp. 85–96. ACM (2004)
16. Tiwari, M., Wassel, H.M., Mazloom, B., Mysore, S., Chong, F.T., Sherwood, T.: Complete information flow tracking from the gates up. In: ACM Sigplan Notices, vol. 44, pp. 109–120. ACM (2009)
17. Vandebogart, S., Efstathopoulos, P., Kohler, E., Krohn, M., Frey, C., Ziegler, D., Kaashoek, F., Morris, R., Mazières, D.: Labels and event processes in the asbestos operating system. ACM Trans. Comput. Syst. (TOCS) **25**(4), 11 (2007)
18. Venkataramani, G., Doudalis, I., Solihin, Y., Prvulovic, M.: Flexitaint: a programmable accelerator for dynamic taint propagation. In: 2008 IEEE 14th International Symposium on High Performance Computer Architecture, pp. 173–184. IEEE (2008)
19. Volpano, D., Irvine, C., Smith, G.: A sound type system for secure flow analysis. J. Comput. Secur. **4**(2–3), 167–187 (1996)
20. Zhang, D., Wang, Y., Suh, G.E., Myers, A.C.: A hardware design language for timing-sensitive information-flow security. In: The Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2015, pp. 503–516, New York, NY, USA (2015)