A review of hardware timing channel detection and mitigation

Gang Chen¹, Zhenyan Zhu¹, Baolei Mao¹*, Wenzuan Chen¹, Jie Jian¹, Zijiao Zhang¹ and Yan ZHUANG¹

¹ Zhengzhou University, Zhengzhou, Henan, 450001, China
*Corresponding author’s e-mail: maobaolei@zzu.edu.cn

Abstract. Hardware timing channels are likely to leak information and easily ignored, which has gradually become the target of attacker. However, there are few investigations systematically analyse hardware timing channel detection and mitigation. In this article, we perform in-depth analysis for the detection technologies such as taint analysis, fuzzing, symbolic analysis and information statistics. And we further analyse some mitigation techniques such as taint analysis and algorithm analysis. Moreover, we compare and analyse the methodologies and characteristics of detecting and mitigating timing channel attack in recent years, and proposes a feasible perspective.

1. Introduction
The researches of traditional security problems focus on the methods of software to detect the vulnerabilities in the software. The detection of software vulnerabilities also brings some innovative inspiration for the realization of hardware structural security. For detecting side channels in software using information flow trace, He et al. extend the test-based analysis method from traditional security properties to dual security properties, the method of double security attributes are used to ensure the comparability of the security information flow and to detect the information leakage of the time-measured channel to the cache[1]. McIver et al. detect the side channel by combination of symbolic execution and taint analysis on the control flow graph to realize the transition from software vulnerabilities detection to software side channels detection [2]. The timing channel can be alleviated by branch prediction [3], and the leakage boundary is determined by tracking the information leakage in each round. Symbol execution can be utilized to determine if there are abnormal variables in timing channel related to instruction and cache, and the timing channel can be mitigated by balancing each execution time [4].

However, it is far from enough to rely on software to ensure the security of the system. Security problems exist not only in the software, but also in the hardware. Invalid hardware security can lead to information leakage, information modification, or system failure. The available attacks of hardware include timing side channel, power side channel and electromagnetic side channel. It is necessary and challenging to eliminate timing channel at low cost. Timing leakage can occur in many places such as cache, hardware accelerator and bus, which are difficult to prevent. In this paper, we perform in-depth analysis for detection technologies such as taint analysis, fuzzing, symbolic analysis and information statistics. In addition, we analyze some mitigation techniques such as taint analysis and algorithm analysis. Moreover, we compare the methodologies and characteristics of detecting and mitigating timing channel attack in recent years, and proposes a feasible perspective.
2. Detection of timing channels

2.1. Taint analysis

2.1.1. Isolating timing flows. Using the same set of labels and propagation rules, the existed hardware information flow tracking approaches track both functional and timing flows. To assess the security of side channels, isolating timing flows is necessary. As shown in Figure 1, the input is code from Verilog/RTL/Gate. There are two steps: expanding each variable with sensitive label and timing label; and inserting logic for updating these labels when their corresponding variables change. Finally, identify timing channels through analysis results of verification tools. To identify influence of the timing variation of the output from input variations, a formal model proposed by integrating time label enhanced tracking logic in HDL/RTL/Gate \[5,6,7,8,9\] level code.

![Figure 1. Isolating timing flows.](image)

2.1.2. Based on hardware description language (HDL). Taint analysis and HDL combine to protect information flow is a promising strategy. For example, Sapper \[10\] detects explicit, implicit and timing information flows through statically-inserted logic for dynamic tracking. SecVerilog \[11\] adds a dependent label with specified security level for each variable to statically check time-sensitive information flows properties in the hardware.

2.2. Fuzzing

Fuzzing-based methodologies work without user interaction and even source code in a customized platform, but there is only a few suitable hardware (e.g. cache). Two-safety property violations revealed by ct-fuzz \[1\] through coverage-guided grey-box to detect Timing leaks. As shown in Figure 2, two copies of a given program in isolation and each independent copy executed. Input and output values of this equivalence relates executions differ solely by secret content. Furthermore, timing variations distinguished through observing divergences in control flow decisions and accessed memory locations. Dudect \[12\], a tool based on the detection of constant-time on a given platform, is utilized to test timing variability in cryptographic functions by black-box testing to identify timing channels.

2.3. Symbolic analysis

2.3.1. Quantitative information flow. Timing side channel can be detected by combination of quantitative information flow (QIF) and symbolic execution due to these technologies are utilized to measure precise bound of leaks of confidential information. Leakage of Shannon’s Information theory computed via cost and information leakage \[4\]. Symbolic execution is utilized to collect all symbolic paths of the program \[13,14\]. As shown in Figure 3, an adversary simulated by introducing the variable “time” observing the timing channel. By observing “time” at the end of the program to deduce the timing leakage of “high” (bit value of a secret key is 1) so to recover the timing channel. Based on the measurement of the leakage by quantitative information flow, coarse-grained clock might leak more than a fine-grained \[15\]. Leak competitiveness \[16\] compares the leakage of two cache algorithms for every possible program to detect timing channels for cache through formal symbolic analysis.
2.3.2. Flow tracking. Control dependence graph is an efficient method to identify implicit flow, while data dependence graph only detects explicit flow. There is an example in Figure 4, ● represents a source of low security information and ○ represents a source of high security information. Variable b is bound to the type H in this code. The branch at L3 controls the assignments at L4 and L5. FlowTracker [17], CacheD [18], Triggerflow [19] are based on trace method with symbolic execution to identify potential timing channel for cache variations or Cryptography. Gate level information flow tracking (GLITF) monitor information flow through hardware timing channels from Boolean functions [20] [21].

![Flow tracking example](Image)

Figure 2. Fuzzing for timing leaks.  

![Modular Exponentiation](Image)

Figure 3. Modular Exponentiation.  

![Control flow graph](Image)

Figure 4. Control flow graph

2.4. Information Statistics

2.4.1. Based on Entropy. Entropy is utilized to measure the uncertainty of variables in information mechanics, it can be seen as the standard by which a piece of information contains useful information. Mao et al [4] quantify information leakage through mutual information by analyzing the effect of time change on the secret key, and quantify timing leakage across different encryption hardware architectures with information theory methods, such as entropy and mutual information [2][16]. Attackers gather information to quantify the amount of leaked information based on different entropy by observing the runtime of the program.

| Features                        | Issues                                      |
|---------------------------------|---------------------------------------------|
| Extra label and propagation rule| Customized HDL, overtaining, undertaining    |
| Without user interaction        | A few suitable hardware (e.g. cache)        |
| Identifying implicit flow       | Requiring automatic tools to reduce workload|
| Detecting precise leakage       | Few automatic tools                         |
2.4.2. Combined with Power-Side Channel. The amount of information leaked by the power side channel may be the same on the time side channel in the case of constant power. Doychev et al. quantify the information leakage caused by whether the cache is hit or not [22]. Dong et al. monitor the power consumption traces generated by the input in the DNN model, and the timing difference of floating-point multiplication of different operands on modern processors is analyzed by microcontroller [23]. As shown in Table 1, we summarized the characteristics of detection timing channels’ approaches.

3. Mitigation methods

Clock fuzzing only reduces the bandwidth of timing channel without entirely eliminating it. Information can still be leaked through timing channel in safe execution pipeline. The technology of hiding timing flow leakage by adding noise can be denoised through statistical analysis. Based on the above issues, this section mainly discusses the methods of mitigating timing channels proposed in recent years, and makes a brief summary as shown in Table 2.

Table 2. Cache timing channel method characteristics.

| Method               | Characteristics                                      | Performance overhead      |
|----------------------|------------------------------------------------------|---------------------------|
| Taint analysis       | Language-level Information flow label tracking, pipeline control | Security label storage   |
| Constant-time        | Program transformation, path balance, internal and external timing decoupling | Resource duplication     |
| Algorithm analysis   | Quantitative information leakage, limit output       | Algorithm calculation    |

3.1. Taint analysis

3.1.1. Language-level technology. Besides detecting the timing flow, the secure-type HDL often combines with taint analysis to eliminate the timing channel, such as Caisson and ChiselFlow. Caisson prevents unexpected information flow and eliminates timing channels through using language-level technology that enforces timing-sensitive non-interference [24]. However, Caisson has a pure static security level and cannot achieve fine-grained resource sharing. Instead, ChiselFlow supports not only static information flow tags but also dynamic tags, allowing more flexible hardware resource sharing. HyperFlow uses information flow tags to protect the timing channel and eliminates the cache timing channel by emptying the processor pipeline [25]. Similarly, the AES accelerator eliminates the timing channel by assigning a security level label to each pipeline phase [26].

3.1.2 Constant-time technology. Constant-time technology is commonly used to eliminate timing channels in hardware, which can be implemented by program transformation or path balance scheduling. The general idea is that adding security dependency flow labels to the input and output and inferring the information policy violations in the system, and using code-switching can eliminate the timing channel leakage caused by the unbalanced condition jump [3,27]. Actually, program transformation exists resource duplication [3]. Jiang et al. [27] proposed an Assure framework, which decouples the internal and external timing behavior of the accelerator and eliminates the timing channel with lower performance overhead. In fact, many password implementations are not constant-time, a solution that introducing a weak form of constant-time into stealth memory can resist concurrent cache attacks [28].
3.2. Algorithm analysis
Askarov et al. [29] limit the amount of information leaked through the timing channel as a function of elapsed time, and specify a bound for the leaked information. Bucketing algorithm can resist adaptive timing channel observation attacks by limiting the output of the system to a specified time interval, but it exists system performance overhead. Therefore, Terauchi et al. [30] proposed to combine bucketing with constant time technology to reduce the timing channel.

4. Conclusion
To make future research more efficient, our paper summarizes the recent detection and mitigation of timing channels, as well as the efforts made to alleviate the leakage and eliminate the existence of timing flow in literature.

In the future, we consider that development of an automatic framework for precise quantitative information flow leakage through statistics and fuzzing is possible in timing channel due to the characteristics of these approaches.

Acknowledgments
This work was supported in part by the Natural Science Foundation of China, under Grant 61672433, the Natural Science Foundation of Shanxi Province, under Grant 2019JM-244 and the National Crypto Development Foundation, under Grant MMJJ20170210.

References
[1] He, S., Emmi, M., & Ciocarlie, G. (2019). ct-fuzz: Fuzzing for Timing Leaks. arXiv preprint arXiv:1904.07280.
[2] McIver, A. K., Morgan, C. C., & Rabeheja, T. (2019). Program algebra for quantitative information flow. Journal of Logical and Algebraic Methods in Programming, 106, 55-77.
[3] Wu, M., Guo, S., Schaumont, P., & Wang, C. (2018, July). Eliminating timing side-channel leaks using program repair. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 15-26).
[4] Mao, B., Hu, W., Althoff, A., Matai, J., Tai, Y., Mu, D., ... & Kastner, R. (2017). Quantitative analysis of timing channel security in cryptographic hardware design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(9), 1719-1732.
[5] Ardeshiricham, A., Hu, W., & Kastner, R. (2017, November). Clepsydra: Modeling timing flows in hardware designs. In 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 147-154). IEEE.
[6] Qin, M., Wang, X., Mao, B., Mu, D., & Hu, W. (2020). A formal model for proving hardware timing properties and identifying timing channels. Integration, 72, 123-133.
[7] Ardeshiricham, A., Takashima, Y., Gao, S., & Kastner, R. (2019, November). VeriSketch: Synthesizing Secure Hardware Designs with Timing-Sensitive Information Flow Properties. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 1623-1638).
[8] Hu, W., Oberg, J., Irturk, A., Tiwari, M., Sherwood, T., Mu, D., & Kastner, R. (2011). Theoretical fundamentals of gate level information flow tracking. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30(8), 1128-1140.
[9] Oberg, J., Meiklejohn, S., Sherwood, T., & Kastner, R. (2014). Leveraging gate-level properties to identify hardware timing channels. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 33(9), 1288-1301.
[10] Li, X., Kashyap, V., Oberg, J. K., Tiwari, M., Rajarathinam, V. R., Kastner, R., ... & Chong, F. T. (2014, February). Sapper: A language for hardware-level security policy enforcement. In Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (pp. 97-112).
[11] Zhang, D., Wang, Y., Suh, G. E., & Myers, A. C. (2015). A hardware design language for timing-sensitive information-flow security. ACM Sigplan Notices, 50(4), 503-516.

[12] Reparaz, O., Balasch, J., & Verbauwhede, I. (2017, March). Dude, is my code constant time?. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017 (pp. 1697-1702). IEEE.

[13] Brennan, T., Saha, S., Bultan, T., & Păsăreanu, C. S. (2018, July). Symbolic path cost analysis for side-channel detection. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 27-37).

[14] Pasareanu, C. S., Phan, Q. S., & Malacaria, P. (2016, June). Multi-run side-channel analysis using Symbolic Execution and Max-SMT. In 2016 IEEE 29th Computer Security Foundations Symposium (CSF) (pp. 387-400). IEEE.

[15] Vasilikos, P., Nielson, H. R., Nielson, F., & Köpf, B. (2019, June). Timing leaks and coarse-grained clocks. In 2019 IEEE 32nd Computer Security Foundations Symposium (CSF) (pp. 32-3215). IEEE.

[16] Cañones, P., Köpf, B., & Reineke, J. (2018). On the Incomparability of Cache Algorithms in Terms of Timing Leakage. arXiv preprint arXiv:1807.01240.

[17] Rodrigues, B., Quintão Pereira, F. M., & Aranha, D. F. (2016, March). Sparse representation of implicit flows with applications to side-channel detection. In Proceedings of the 25th International Conference on Compiler Construction (pp. 110-120).

[18] Wang, S., Wang, P., Liu, X., Zhang, D., & Wu, D. (2017). Cached: Identifying cache-based timing channels in production software. In 26th {USENIX} Security Symposium ({USENIX} Security 17) (pp. 235-252).

[19] Gridin, I., Garcia, C. P., Tuveri, N., & Brumley, B. B. (2019, June). Triggerflow: Regression testing by advanced execution path inspection. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 330-350). Springer, Cham.

[20] Hu, W., Oberg, J., Irturk, A., Tiwari, M., Sherwood, T., Mu, D., & Kastner, R. (2012). On the complexity of generating gate level information flow tracking logic. IEEE Transactions on Information Forensics and Security, 7(3), 1067-1080.

[21] Qin, M., Hu, W., Wang, X., Mu, D., & Mao, B. (2019). Theorem proof based gate level information flow tracking for hardware security verification. Computers & Security, 85, 225-239.

[22] Doychev, G., Köpf, B., Mauborgne, L., & Reineke, J. (2015). Cacheaudit: A tool for the static analysis of cache side channels. ACM Transactions on Information and System Security (TISSEC), 18(1), 1-32.

[23] Dong, G., Wang, P., Chen, P., Gu, R., & Hu, H. (2019, August). Floating-Point Multiplication Timing Attack on Deep Neural Network. In 2019 IEEE International Conference on Smart Internet of Things (SmartIoT) (pp. 155-161). IEEE.

[24] Li, X., Tiwari, M., Oberg, J. K., Kashyap, V., Chong, F. T., Sherwood, T., & Hardekopf, B. (2011). Caisson: a hardware description language for secure information flow. ACM Sigplan Notices, 46(6), 109-120.

[25] Ferraiuolo, A., Zhao, Y., Suh, G. E., & Myers, A. C. (2018). HyperFlow: A Processor Architecture for Timing-Safe Information-Flow Security.

[26] Jiang, Z., Jin, H., Suh, G. E., & Zhang, Z. (2019, June). Designing Secure Cryptographic Accelerators with Information Flow Enforcement: A Case Study on AES. In Proceedings of the 56th Annual Design Automation Conference 2019 (pp. 1-6).

[27] Jiang, Z., Dai, S., Suh, G. E., & Zhang, Z. (2018, November). High-level synthesis with timing-sensitive information flow enforcement. In 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 1-8). IEEE.

[28] Barthe, G., Betarte, G., Campo, J., Luna, C., & Pichardie, D. (2014, November). System-level non-interference for constant-time cryptography. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 1267-1279).
[29] Askarov, A., Zhang, D., & Myers, A. C. (2010, October). Predictive black-box mitigation of timing channels. In Proceedings of the 17th ACM conference on Computer and communications security (pp. 297-307).

[30] Terauchi, T., & Antonopoulos, T. (2019, April). A formal analysis of timing channel security via bucketing. In International Conference on Principles of Security and Trust (pp. 29-50). Springer, Cham.