Proving the Absence of Timing Side Channels in Cryptographic Applications

2023

Denis Mazzucato
denis.mazzucato@inria.fr
Inria & École Normale Supérieure | PSL

Abstract

Cryptographic applications provide strong mathematical guarantees regarding the difficulty of extracting secret information from output observations. However, these guarantees often fall short in preventing side-channel attacks that reveal sensitive data. In particular, timing side-channel attacks exploit variations in execution time to infer (part of) the secret information. An application is deemed resistant to such attacks when its execution time remains unaffected by secret input data, rendering timing patterns not exploitable by malicious users. Our project focuses on the s2n-bignum library, a critical component of the AWS-LC cryptographic library. While all the primitives in s2n-bignum have been verified to be correct in HOL-Light and are intentionally designed to be immune to timing variations caused by user data, a crucial element is currently absent: a formal proof establishing timing side-channel freedom. We aim to establish such formal verification for s2n-bignum, defining first an abstract timing cost model, then the side-channel freedom could be expressed as a non-interference property via program self-composition. Our project will contribute to the development of secure cryptographic applications.

Introduction

In the realm of digital security, cryptographic systems serve as fortresses protecting sensitive information. Traditionally, the security of their systems has been quantified through their mathematical guarantees. However, practical experience has shown that attackers rarely exploit direct computational complexity to breach cryptographic primitives. Instead, they usually exploit vulnerabilities in the implementation of cryptographic algorithms, much like a skilled thief bypasses a locked safe not by brute-forcing combinations but by understanding its physical weaknesses [1].

In the context of program security, introduced by Kocher [2], such attacks are called side-channel attacks, which exploit the physical and external observable characteristics of the system implementation to deduce secret information. These attacks may require physical tampering with the cryptographic device or exploits the internal operation of the processor, such as power consumption [3] or speculative executions [4], [5], whereas other attacks, for instance timing side-channel attacks [2], [6], exploit the information leaked during computation without physical contact; see Hong's special issue [7] for a broad overview of possible side-channel attacks.

The s2n-bignum library[1] emerges as a pivotal player. This library consists of a collection of high-performance big-number operations for cryptographic applications, written in low-level assembly code and formally verified to be correct in the HOL-Light[2] proof assistant [8]. Furthermore, it is designed to be timing side-channel free, meaning that timing observations cannot be exploited as side-channels. However, despite these meticulous design considerations, the s2n-bignum library lacks formal verification against timing attacks.

[1] https://github.com/awslabs/s2n-bignum
[2] https://www.cl.cam.ac.uk/~jrh13/hol-light/

Formal verification entails proving system correctness and security properties with mathematical guarantees. In cryptographic applications, formal verification is akin to reinforcing a safe's steel walls, ensuring that no such vulnerability exists for attackers to exploit. Various techniques have been developed to certify the correctness, or prove incorrectness, of program systems, such as interactive theorem provers [9], model checking [10], [11], symbolic execution [12], and abstract interpretation [13], [14]. Certifying a cryptographic system as timing side-channel free is particularly challenging [15], [16]. Such certification could potentially render security breaches in cloud-based cryptographic systems nearly impossible, substantially enhancing the system's robustness.

Our research objective is to first design an abstract timing cost model, which can be subsequently specialized to different architectures (x86, arm), then rigorously formalize the timing side-channel freedom of the s2n-bignum library (which will be parameterized by the cost model). To achieve this, the classic approach is to self-compose a program [17] and annotate timing information. Whenever the self-composed program, with different secret input data for the two copies of the program, retains the same time consumption then it means that no timing information could be exploited to recover the secret data. To verify the self-composed program, we will explore theorem proving and model checking, complemented with abstract interpretation techniques. These techniques will enable exhaustive analysis of the codebase, obtaining sound proofs of timing side-channels and identifying potential timing side-channel vulnerabilities.

In summary, our research project lies at the intersection of cryptography, formal methods, and cybersecurity. By protecting the s2n-bignum library against side-channel attacks, we improve security throughout the entire cryptography pipeline, as this library is already used in the cryptographic library maintained by the AWS Cryptography team (AWS-LC[3]) and will be adopted even more in the future. We aspire to lay the foundation stone towards a complete security certification of the cryptographic library.

Methods

Our property of interest, namely timing side-channel freedom, holds whenever two executions of the same program starting from different secret inputs, terminate with the same execution time. In such a way an attacker observing only the execution time cannot infer any information about the secret data. A classic approach to address this property in security is through self-composition [17]. Given a program P, we first partition its input parameters into low/high variables, where "low" denotes public values known to a malicious user, and "high" denotes private secrets. Part of this project, we plan to develop a simple tool or DSL for annotating high and low variables. The self-composed program is P(low, high1); P(low, high2), where "high1" and "high2" are two different secrets. The key idea behind self-composition is to compare the timing cost of any two executions (P(low, high1) and P(low, high2)). The timing cost model should be abstract and iteratively improved until we reach a faithful representation of the processor execution model. At the beginning, it can be naively implemented by instrumenting the program P with a ghost (not used by the program P) variable called "cost", incremented with each executed instruction. The program is timing side-channel free whenever "cost1" (the cost variable of the first program copy, P(low, high1)) is equal to "cost2" (the variable of the second program copy, P(low, high2)). Formally, P is secure to timing side-channel attacks whenever **P(low, high1); P(low, high2) {cost1 = cost2}** holds. Note that the approach can be extended to detection of unexpected leakage of sensitive data through 'low' buffers.

The above property could be addressed using different formal methods techniques. Each formal verification technique has its own benefits and drawbacks. For example, symbolic execution is excellent for automatically synthesizing test cases to demonstrate the possibility of a timing attack leveraging

[3] https://github.com/aws/aws-lc

side-channels. Model checking, such as CBMC [18], instead provides proof of the absence of timing side-channels automatically, under the assumptions of our cost model and an iteration limit k. Counterexamples would indicate potential vulnerabilities crucial to our Amazon collaborators. As our goal is towards an automatic and sound approach to proving the absence of such attacks, abstract interpretation will also play a key role. Furthermore, we plan to explore the application of HOL-light to obtain the proofs for the cryptographic libraries, leveraging the existing infrastructure developed at AWS for proving functional correctness.

Our approach involves a comprehensive study of the timing patterns in the s2n-bignum library, employing both the CBMC model checker and the HOL-Light theorem prover. The use of mechanized proofs with the HOL-Light theorem prover is quite appealing due to its integration into the s2n-bignum's development pipeline for proving functional correctness. In this instance, to ease the verification process, we will build our verification framework on AWS's existing infrastructure, including the machine code decoder. The decoder can also be exploited to obtain an intermediate representation amenable for abstract interpretation. Indeed, we will study the use of abstract interpretation afterward. Throughout the entire project, we will establish a close collaboration with the s2n Team at Amazon AWS, who are the primary customer of our work.

Additionally, an automatic counterexample checker tool will help to prevent many false alarms without notifying the s2n Team every time our analysis loses precision (abstract interpretation) or the assumptions on the underlying model are not faithful enough (model checking). Leveraging EC2 instances with different processor architectures, this additional tool can confirm real counterexamples in "production" environments.

Expected Results

The timeline and milestones below will guide the progress and monitoring of the 1-year postdoc position.

- **Month 1-3**: Onboarding; apply bytecode decoder and explore connection with CBMC goto-programs; experiments with CBMC; translate to HOL-Light for a subset of s2n-bignum.
  Milestone: Demo of findings on limited subset of s2n-bignum.
- **Month 4-6**: Extend process to additional s2n-bignum primitives and validate with AWS partner. Identify performance bottlenecks.
  Milestone: Extended demo and report.
- **Month 7-9**: Develop an abstract interpretation framework to address performance bottlenecks.
  Milestone: Integration of abstract interpretation in HOL-Light and CBMC verification pipelines.
- **Month 10-12**: Strengthen the developed techniques by incorporating additional s2n-bignum features; evaluate against AWS customer.
  Milestone: Final product and publication.
- **Beyond 1 Year (Stretch Goal)**: Certification for the whole s2n-bignum library.

Conclusion

In conclusion, the aim of this research project is to secure the s2n-bignum library against timing side-channel attacks, contributing to the broader landscape of secure cryptographic applications. I believe that my background in formal methods and abstract interpretation positions me as a strong candidate for this project. Moreover, the collaboration with the s2n Team at Amazon AWS, including key members such as Harrison, the creator of HOL-Light [8], is advantageous to the success of this research.

Bibliography

[1] B. Schneier, *Information Management & Computer Security*, 1998.

[2] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, N. Koblitz, Ed., in Lecture Notes in Computer Science, vol. 1109. Springer, 1996, pp. 104–113. doi: 10.1007/3-540-68697-5_9.

[3] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis", in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, M. J. Wiener, Ed., in Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397. doi: 10.1007/3-540-48405-1_25.

[4] P. Kocher *et al.*, "Spectre Attacks: Exploiting Speculative Execution", in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, IEEE, 2019, pp. 1–19. doi: 10.1109/SP.2019.00002.

[5] M. Lipp *et al.*, "Meltdown: Reading Kernel Memory from User Space", in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds., USENIX Association, 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[6] W. H. Wong, "Timing attacks on RSA: revealing your secrets through the fourth dimension", *ACM Crossroads*, vol. 11, no. 3, p. 5, 2005, doi: 10.1145/1144396.1144401.

[7] S. Hong, "Special Issue on "Side Channel Attacks", *Applied Sciences*, vol. 9, no. 9, 2019, doi: 10.3390/app9091881.

[8] J. Harrison, "HOL Light: An Overview", in *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, Eds., in Lecture Notes in Computer Science, vol. 5674. Springer, 2009, pp. 60–66. doi: 10.1007/978-3-642-03359-9_4.

[9] N. G. de Bruijn, "AUTOMATH, a Language for Mathematics", in *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967--1970*, J. H. Siekmann and G. Wrightson, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 159–200. doi: 10.1007/978-3-642-81955-1_11.

[10] E. M. Clarke and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic", in *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, D. Kozen, Ed., in Lecture Notes in Computer Science, vol. 131. Springer, 1981, pp. 52–71. doi: 10.1007/BFB0025774.

[11] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR", in *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, M. Dezani-Ciancaglini and U. Montanari, Eds., in Lecture Notes in Computer Science, vol. 137. Springer, 1982, pp. 337–351. doi: 10.1007/3-540-11494-7_22.

[12] J. C. King, "Symbolic Execution and Program Testing", *Commun. ACM*, vol. 19, no. 7, pp. 385–394, 1976, doi: 10.1145/360248.360252.

[13] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints", in *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, R. M. Graham, M. A. Harrison, and R. Sethi, Eds., ACM, 1977, pp. 238–252. doi: 10.1145/512950.512973.

[14] P. Cousot and R. Cousot, “Systematic Design of Program Analysis Frameworks”, in *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979*, A. V. Aho, S. N. Zilles, and B. K. Rosen, Eds., ACM Press, 1979, pp. 269–282. doi: 10.1145/567752.567778.

[15] B. Köpf and D. A. Basin, “An information-theoretic model for adaptive side-channel attacks”, in *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds., ACM, 2007, pp. 286–296. doi: 10.1145/1315245.1315282.

[16] Q.-S. Phan, L. Bang, C. S. Pasareanu, P. Malacaria, and T. Bultan, “Synthesis of Adaptive Side-Channel Attacks”, in *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, IEEE Computer Society, 2017, pp. 328–342. doi: 10.1109/CSF.2017.8.

[17] J. B. Almeida, M. Barbosa, J. S. Pinto, and B. Vieira, “Formal verification of side-channel countermeasures using self-composition”, *Sci. Comput. Program.*, vol. 78, no. 7, pp. 796–812, 2013, doi: 10.1016/J.SCICO.2011.10.008.

[18] E. M. Clarke, D. Kroening, and F. Lerda, “A Tool for Checking ANSI-C Programs”, in *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, K. Jensen and A. Podelski, Eds., in Lecture Notes in Computer Science, vol. 2988. Springer, 2004, pp. 168–176. doi: 10.1007/978-3-540-24730-2_15.