"They're not that hard to mitigate"

What Cryptographic Library Developers Think About Timing Attacks

Jan Jancar[1], Marcel Fourné[2], Daniel De Almeida Braga[3], Mohamed Sabt[3], Peter Schwabe[4], Gilles Barthe[5], Pierre-Alain Fouque[3] und Yasemin Acar[6]

This paper has been published at the 43rd IEEE Symposium on Security and Privacy in 2022.

Cryptographic protocols, such as TLS (Transport Layer Security), are the backbone of computer security, and are used at scale for securing the Internet, the Cloud, and many other applications. Quite strikingly, the deployment of these protocols rests on a small number of open-source libraries, developed by a rather small group of outstanding developers. These developers have a unique set of skills that are needed for writing efficient, correct, and secure implementations of (often sophisticated) cryptographic routines; in particular, they combine an excellent knowledge of cryptography and of computer architectures and a deep understanding of low-level programming. Unfortunately, in spite of developers' skills and experience, new and sometimes far-reaching vulnerabilities and attacks are regularly discovered in major open-source cryptographic libraries. One class of vulnerabilities are *timing attacks*, which let an attacker retrieve secret material, such as cryptographic keys, *"by carefully measuring the amount of time required to perform private key operations"*. Although timing attacks were first described by Kocher in 1996 [Ko96], they continue to plague implementations of cryptographic libraries.

At the same time, and most importantly for this paper, we know how to systematically protect against timing attacks. The basic idea of such systematic countermeasures was already described by Kocher in 1996 [Ko96]: we need to ensure that all code takes time independent of secret data. It is important here to not just consider the total time taken by some cryptographic computation, but make sure that this property holds for each instruction. This paradigm is known as *constant-time*[7] cryptography and is usually achieved by ensuring that

- there is no data flow from secrets into branch conditions;

1 Masaryk University, Brno, Czech Republic,
2 MPI-SP, Bochum, Germany, marcel.fourne@mpi-sp.org
3 Rennes University, CNRS, IRISA, Rennes, France,
4 MPI-SP, Bochum, Germany and Radboud University, Nijmegen, The Netherlands,
5 MPI-SP, Bochum, Germany and IMDEA Software Institute, Madrid, Spain,
6 The George Washington University, Washington D.C., USA,
7 The term constant-time, often referred as CT, is a bit of a misnomer, as it does not refer to CPU execution time but rather to a structural property of programs. However, it is well-established in the cryptography community.

doi:10.18420/sw2024_47

- addresses used for memory access do not depend on secret data; and
- no secret-dependent data is used as input to variable-time arithmetic instructions (such as, e.g., `DIV` on most Intel processors or `SMULL`/`UMULL` on ARM Cortex-M3).

We know how to verify that programs areconstant-time. This was first demonstrated by Adam Langley's ctgrind [La10], developed in 2010, the first tool to support analysis of constant-timeness. A decade later, there are now more than 30 tools for checking that code satisfies constant-timeness or is resistant against side-channels [Ba19; Ja21]. These tools differ in their goals, achievements, and status. Yet, they collectively demonstrate that automated analysis of constant-time programs is feasible; for instance, a 2019 review [Ba19] lists automatic verification of constant-time real-world code as one achievement of computer-aided cryptography, an emerging field that develops and applies formal, machine-checkable approaches to the design, analysis, and implementation of cryptography.

Based on this state of affairs, one would expect that timing leaks in cryptographic software have been systematically eliminated, and timing attacks are a thing of the past. Unfortunately, this is far from true, so in this paper we set out to answer the question: *Why is today's cryptographic software not free of timing-attack vulnerabilities?*

We find that, while all 44 participants are aware of timing attacks, not all cryptographic libraries have verified/tested resistance against timing attacks. Reasons for this include varying threat models, a lack of awareness of tooling that supports testing/verification, lack of availability, as well as a perceived significant effort in using those tools. We expose these reasons, and provide recommendations to tool developers, cryptographic libraries developers, compiler writers, and standardization bodies to overcome the main obstacles towards a more systematic protection against timing attacks. We also briefly discuss how these recommendations extend to closely related lines of research.

Literatur

[Ba19] Barbosa, M.; Barthe, G.; Bhargavan, K.; Blanchet, B.; Cremers, C.; Liao, K.; Parno, B.: SoK: Computer-Aided Cryptography. IACR Cryptol. ePrint Arch. 2019/, S. 1393, 2019, URL: https://eprint.iacr.org/2019/1393.

[Ja21] Jancar, J.: The state of tooling for verifying constant-timeness of cryptographic implementations, 2021, URL: https://neuromancer.sk/article/26.

[Ko96] Kocher, P. C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In (Koblitz, N., Hrsg.): Advances in Cryptology – CRYPTO'96. Bd. 1109. LNCS, Springer, S. 104–113, 1996, URL: http://www.cryptography.com/public/pdf/TimingAttacks.pdf.

[La10] Langley, A.: ctgrind, 2010, URL: https://github.com/agl/ctgrind.