Verification of a Cryptographic Primitive: SHA-256 (Abstract)

Andrew W. Appel
Princeton University
appel@princeton.edu

Abstract

A full formal machine-checked verification of a C program: the OpenSSL implementation of SHA-256. This is an interactive proof of functional correctness in the Coq proof assistant, using the Verifiable C program logic. Verifiable C is a separation logic for the C language, proved sound w.r.t. the operational semantics for C, connected to the CompCert verified optimizing C compiler.

Categories and Subject Descriptors D.2.4 [*Software/Program Verification*]: Correctness proofs; E.3 [*Data Encryption*]: Standards; F.3.1 [*Specifying and Verifying and Reasoning about Programs*]

General Terms Verification

Keywords Coq

1. Introduction

> *[C]ryptography is hard to do right, and the only way to know if something was done right is to be able to examine it.... This argues very strongly for open source cryptographic algorithms.... [But] simply publishing the code does not automatically mean that people will examine it for security flaws.* *Bruce Schneier, 1999*

> *Be suspicious of commercial encryption software ... [because of] back doors.... Try to use public-domain encryption that has to be compatible with other implementations.... "* *Bruce Schneier, 2013*

That is, use widely used, well examined open-source implementations of published, nonproprietary, widely used, well examined, standard algorithms—because "many eyes make all bugs shallow" works only if there are many eyes paying attention.

To this I add: use implementations that are *formally verified with machine-checked proofs* of functional correctness, of side-channel resistance, of information-flow properties. "Many eyes" are a fine thing, but sometimes it takes them a couple of years to notice the bugs. Verification can guarantee program properties in advance of widespread release.

Formal verification is not necessarily a *substitute* for many-eyes assurance. For example, in this case, I present only the assurance of functional correctness (and its corollary, safety, including absence of buffer overruns). With respect to other properties such as timing side channels, I prove nothing; so it is comforting that this same C program has over a decade of widespread use and examination.

SHA-256, the Secure Hash Algorithm with 256-bit digests, is not an encryption algorithm, but it is used in encryption protocols. The methods I discuss in this paper can be applied to the same issues that appear in ciphers such as AES: interpretation of standards documents, big-endian protocols implemented on little-endian machines, odd corners of the C semantics, storing bytes and loading words, signed and unsigned arithmetic, extended precision arithmetic, trustworthiness of C compilers, use of machine-dependent special instructions to make things faster, correspondence of models to programs, assessing the trusted base of the verification tools.

This paper presents the following result: I have proved functional correctness of the OpenSSL implementation of SHA-256, with respect to a *functional specification:* a formalization of the FIPS 180-4 *Secure Hash Standard*. The machine-checked proof is done using the *Verifiable C* program logic, in the Coq proof assistant. Verifiable C is proved sound with respect to the operational semantics of C, with a machine-checked proof in Coq. The C program can be compiled to x86 assembly language with the CompCert verified optimizing C compiler; that compiler is proved correct (in Coq) with respect to the same operational semantics of C and the semantics of x86 assembly language. Thus, by composition of machine-checked proofs with no gaps, the assembly-language program correctly implements the functional specification.

In addition, I implemented SHA-256 as a functional program in Coq and proved it equivalent to the functional specification. Coq can execute the functional program on real strings (only a million times slower than the C program), and gets the same answer as standard reference implementations. This gives some extra confidence that no silly things are wrong with the functional spec.

PLDI'15, June 13–17, 2015, Portland, OR, USA
ACM. 978-1-4503-3468-6/15/06
http://dx.doi.org/10.1145/2737924.2774972