A Tool-Chain for High-Assurance Cryptographic Software

by José Almeida, Manuel Barbosa, Hugo Pacheco and Vitor Pereira (INESC TEC)

Cryptography is an inherently interdisciplinary area and the development of high-quality cryptographic software is a time-consuming task drawing on skills from mathematics, computer science and electrical engineering, only achievable by highly skilled programmers. The challenge is to map high-level cryptographic specifications phrased using mathematical abstractions into efficient implementations at the level of C or assembly that can be deployed on a target computational platform, whilst adhering to the specification both in terms of correctness and security. The High Assurance Software Laboratory at INESC-TEC maintains a domain-specific toolchain for the specification, implementation and verification of cryptographic software centred on CAO, a cryptography analyses and operations-aware language.

There is a high risk associated with poor cryptographic implementations, as is shown by frequent (and in some cases catastrophic) security breaches directly attributed to implementation errors in widely used cryptographic libraries [L1,L2]. One of the causes of these breaches in widely tested software is the semantic gap between theoretical cryptographic specifications and their concrete implementations. Effectively closing this gap is a huge challenge, especially when attackers may exploit physical vulnerabilities not covered by the specification, commonly known as side-channel attacks.

To answer this demand, research in the crossover area between cryptography and programming languages has been growing steadily in the last decade, as demonstrated by the Computer Aided Cryptography Engineering (CACE) EU/FP7 project that focused on developing tools to automate the production of high quality cryptographic software at a lower cost. The CAO cryptographic domain-specific language [L3,1,2], initially developed at the University of Bristol and subsequently re-engineered within CACE, enables the natural translation of cryptographic constructions (as found in standards and scientific articles) to high-level prototype implementations. The driving principle behind the design of CAO is to support cryptographic concepts as first-class features.

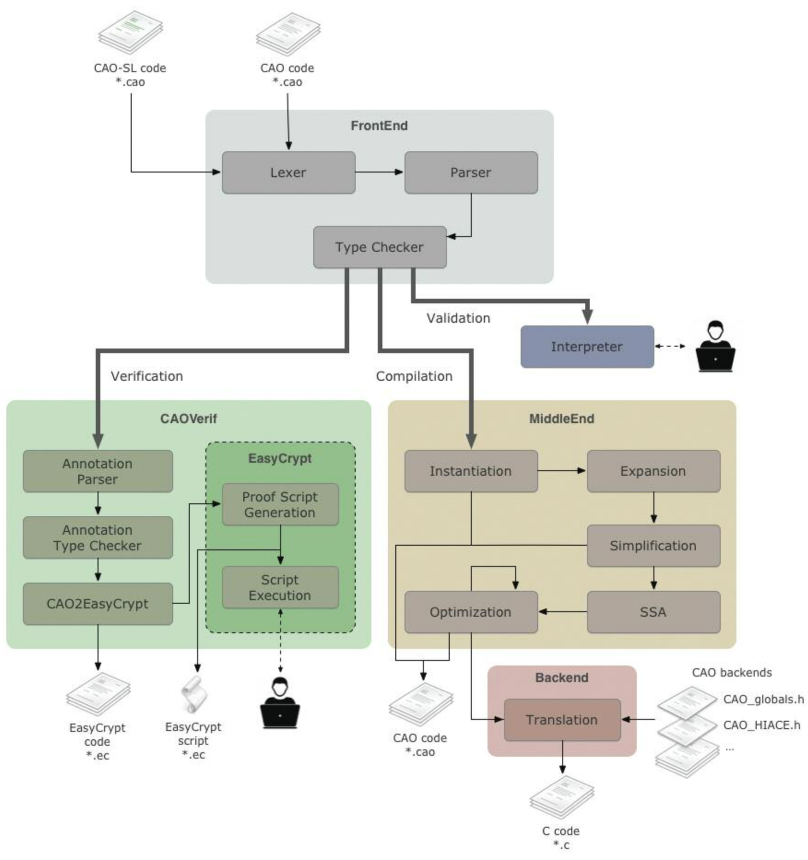

Figure 1: The CAO toolchain.

CAO adopts some features familiar to imperative programmers but has a very simple programming model by design. For instance, it does not support input/output, as it is targeted at implementing the core components of cryptographic libraries. Conversely, it offers a very rich type system tuned to the specific domain of cryptography. In recent versions of the language, CAO programs can be seen as generic specifications that, like pure theoretical cryptographic constructions, are defined abstractly for a set of parameters satisfying certain base assumptions. The CAO developer is assisted by an interpreter that enables fast prototyping and debugging, and a type-checker that enforces strong typing and performs extensive preliminary validation of the code, extracting rich crucial information for further processing down the chain. CAO specifications can also be validated in a fully automatic way for parameter consistency properties in later versions of the type checker.

The CAO tool chain (Figure 1) also provides an optimising compiler for the automatic generation of high-security and high-speed cryptographic C implementations from high-level CAO specifications. The inner workings of the CAO compilation process mimic those performed by cryptography practitioners.

The CAO specification is first converted into a canonical CAO subset through a series of both general and domain-specific CAO-to-CAO transformation and optimisation steps.

In a second phase, the intermediate CAO code is compiled into C code in which CAO native operations are implemented as a C backend library that may be either pre-compiled or dynamically generated. This flexibility allows adapting the CAO compiler to the wide variety of computational platforms in which cryptographic code is deployed in the real world. The CAO compiler offers a generic C backend supporting the entire functionality of the CAO language and capable of targeting any computational platform with a C/C++ compiler. In the context of the SMART ENIAC/JU project, a very specific backend supporting only a limited subset of the CAO language has been developed to target a severely constrained proprietary microcontroller that resides in standalone PCM memories, while preserving the remaining high-level infrastructure.

Seeing CAO programs as specifications, it becomes natural to express the properties of CAO programs in the same abstract setting, i.e., directly in the CAO language. For this reason, the CAO toolchain also incorporates a formal verification tool that permits reasoning about arbitrarily complex properties of CAO programs (specified as in-code annotations) in a semi-automated environment, by embedding them in EasyCrypt [L4,3], a tool-assisted framework for specifying and verifying the security of cryptographic constructions. Using EasyCrypt, the developer is now able to additionally perform safety, correctness and security proofs of cryptographic algorithms written in CAO.

The joint effort across two European projects brought the CAO toolchain to life and came to fruition by demonstrating that a domain-specific high-level cryptographic language can be used to guide, validate and automate the development of low-level high-assurance cryptographic implementations for diverse computational platforms. Ongoing and future work will broaden the applications of the CAO family of tools by further exploring the integration with EasyCrypt and developing new backends. In particular, we envision the implementation of a backend for a cryptography-oriented low-level language such as qhasm, in which assembly level programs are seen as first class representations of cryptographic computations.

Links:
[L1] http://heartbleed.com/
[L2] http://resources.infosecinstitute.com/beast-vs-crime-attack/
[L3] https://hackage.haskell.org/package/cao
[L4] http://www.easycrypt.info

References:
[1] M. Barbosa, D. Castro, Paulo F. Silva: "Compiling CAO: From Cryptographic Specifications to C Implementations." POST 2014: 240-244.
[2] M. Barbosa, et al.: "Type Checking Cryptography Implementations", FSEN 2011: 316-334.
[3] G. Barthe et al.: "Computer-Aided Security Proofs for the Working Cryptographer", CRYPTO 2011: 71-90.

Please contact:
Manuel Bernardo Martins Barbosa
HASLab, INESC TEC and DCC FCUP
mbb@dcc.fc.up.pt

Code-Based Cryptography: New Security Solutions Against a Quantum Adversary

by Nicolas Sendrier and Jean-Pierre Tillich (Inria)

Cryptography is one of the key tools for providing security in our quickly evolving technological society. An adversary with the ability to use a quantum computer would defeat most of the cryptographic solutions that are deployed today to secure our communications. We do not know when quantum computing will become available, but nevertheless, the cryptographic research community must get ready for it now. Code-based cryptography is among the few cryptographic techniques known to resist a quantum adversary.

Since their appearance in the mid seventies, public key (or asymmetric) cryptographic primitives have been notoriously difficult to devise and only a handful of schemes have emerged and have survived cryptanalytic attacks. In particular, the security of nearly all public key schemes used today relies on the presumed difficulty of two problems, namely factoring of large integers and computing the discrete logarithm over various groups.

The security of all these schemes was questioned in 1994 when Shor showed that a quantum computer could efficiently solve these two problems [1]. We do not know when large enough quantum computers will be built, but this will have dramatic consequences because it will break all popular public-key cryptosystems currently in use.

Clearly, the cryptographic research community has to get ready and prepare alternatives. Those alternatives have to be ready, not only for tomorrow in case of a scientific advance (which might even be of a different nature than those that are foreseen today), but also for now, in order to provide long term security – i.e., several decades – to the data that is encrypted or digitally signed today. This effort has started already with PQCRYPTO [L1] of the European Horizon 2020 program. Furthermore, in August, 2015, NSA announced that it is planning to transition 'in the not too distant future' to a