Screaming fast symmetric cryptography: purely functional & typed with class

Pierre-Evariste Dagand (dagand@irif.fr)

Usuba[1] is a high-level domain-specific programming language to write high-throughput and constant-time cryptographic primitives, generating low-level C code based on a generalization of bitslicing. The Usuba compiler targets both high-end, superscalar Intel machines as well as low-end, embedded devices (such as Arm Cortex).

Whilst Usuba has demonstrated significant performance benefits on high-end processors, its performance profile on embedded devices has received limited attention. So far, we have focused solely on a single ARM Nucleo STM32F401RE development board.

Besides, the Usuba programming language is currently specified through a pen-and-paper dynamic and static semantics. Moreover, the internal representation of the compiler, dubbed `usuba0`, does not fully exploit the equational theory offered by a purely functional language.

This internship aims at addressing either (or both!) of these limitations.

Targeting embedded systems

One line of research would aim at extending the `usubac` compiler with optimization passes specifically designed for embedded platforms of the ARM Cortex family. Our goal is to deliver performance on par with hand-tuned implementations on this platform.

A concrete starting point for this endeavor is the seminal work of Schawbe et al.[2] that describes the process of optimizing a cryptographic primitive –in this case, AES– for an embedded ARM Cortex M platform. We wish to implement this proposal as part of a dedicated `usubac` back-end, with the intent of closing the performance gap between usuba-generated and hand-tuned cryptographic implementations. As part of this process, we intend to provide a synthetic benchmark of the Usuba implementation of the lightweight cryptographic primitives

[1] https://usubalang.github.io/usuba/

[2] https://doi.org/10.1007/978-3-319-69453-5_10

proposed to the NIST LWC competition, following Renner et al.[3] and expanding upon our earlier benchmark[4] (which was focused on masked implementations).

Our current testing workbench consists in

- a Skiva softcore processor deployed on the main FPGA (Cyclone IV EP4CE115) of an Altera DE2-115 board (the processor is clocked at 50 MHz and has access to 128 kB of RAM);
- a STM32F030 discovery kit featuring a 48Mhz ARM Cortex-M0 (8kB of SRAM, 64 kB of flash memory)
- a STM32G031 discovery kit featuring a 64MHz ARM Cortex-M0+ (8kB of SRAM, 32 kB of flash memory)
- a STM32L100 discovery kit featuring a 32MHz ARM Cortex-M3 (16kB of SRAM, 256kB of flash memory)
- a STM32F407 discovery kit featuring a 168MHz ARM Cortex-M4 (192 kB of SRAM, 1 MB flash memory)
- a STM32F756 Nucleo featuring a 216MHz ARM Cortex-M7 (320kB of SRAM, 1 MB of flash memory)
- a Raspberry Pi 3 (model A+) featuring a 1.4GHz ARM Cortex-A53 (512MB of RAM)
- a Raspberry Pi 4 (model B) featuring a 1.5GHz ARM Cortex-A72 (2GB of RAM)
- a BeagleBone Black (rev C) featuring a 1GHz ARM Cortex-A8 (512MB of RAM)

Because of our initial interest in higher-order masking, we have been focused on the ARM platform for which one can easily find a configuration with enough flash memory to hold the compiled code. If experiments with the Cortex-M0 are encouraging, we would certainly foray into the world of 8-bit microcontrollers, such as the Atmel AVR family, and 16-bit microcontrollers, such as the MSP430 family.

Delivering high performance on embedded devices calls for dispensing with existing C compilers and directly producing machine code. This issue has long been recognized by the cryptographic community. Jasmin[5] is such a "high-level" assembly language, offering a mechanized semantics and a machine-checked assembler for x86 and ARM architectures. Crossing the gap between Usuba and Jasmin boils down to implementing a register allocator, Jasmin already supporting all the other Usuba features. Such an effort to adapt a register allocator tailored for bitsliced code will be an opportunity to improve the quality and predictability of `usubac` optimizations across-the-board: the brittleness of the general-purpose register allocators provided by C compilers has led us to rely on ad-hoc, unnatural code patterns to coerce certain allocation strategies, sometimes at the expense of other optimization opportunities.

[3] https://doi.org/10.1007/978-3-030-61078-4_28

[4] https://usubalang.github.io/usuba/assets/documents/tornado-eurocrypt20.pdf

[5] https://hal.archives-ouvertes.fr/hal-01649140

Developing a general-purpose register allocator is no small feat. Here, we are hoping to take advantage of the simplicity of the source language (little to no control-flow, essentially straight-line code) to benefit from a combined instruction scheduling / register allocation scheme. Encouraging results on ciphers have been reported in the literature.

Trustworthy Usuba

Another line of research would aim at developing a mechanized dynamic and static semantics for Usuba in the Coq theorem prover. This work would be put to the test by proving the correctness of the `usubac` compiler front-end, which performs a syntactic reduction down to `usuba0`, the compiler internal representation that amounts to combinational circuits.

Once we have obtained `usuba0` code, the compiler back-end is responsible for aggressively optimizing these circuits through source-to-source transformations. We plan to refine this internal language, adapting ideas and techniques from Equality Saturation[6] to streamline the implementation of optimizations. To ensure the correctness of the back-end, we will rely on translation validation to ensure *post facto* that the meaning of a given program has been preserved throughout the pipeline. We will extensively exploit the fact that `usuba0` maps straightforwardly to SMT formulae, where the verification problem reduces to (combinational) circuit equivalence checking.

In the process and to show-case this approach, we wish to implement a generic fixslicing[7] optimization pass. Fixslicing is a whole-cipher transformation that aims at removing the linear layer of Substitution-bitPermutation-Network (SbPN) ciphers (such as GIFT and Present) and, beyond, to AES-like designs (such as AES itself or Skinny). In those ciphers, the linear layer operates a representation change over the matricial state of the cipher at run-time (which, for example, represents 30% of the execution time on AES). Fixslicing –when possible– turns this run-time operation into a compile-time code transformation by specializing the subsequent code to work over a non-standard representation of the matricial state. As it turns out in the case of AES, this representation converges back to the identity after 4 rounds. As a consequence, a round needs only be specialized into 4 distinct implementations operating over 4 non-standard layouts, thus limiting the code size blowup to a factor 4. Identifying a suitable data layout and synthesizing the corresponding specialized rounds calls for highly non-trivial heuristics, which would represent a formidable challenge to prove correct. Instead, we intend to lean on the translation validation framework to check *a posteriori* that the semantic of an individual round is preserved when specialized to operate over a non-standard representation.

[6] https://rosstate.org/publications/eqsat/

[7] https://doi.org/10.46586/tches.v2021.i1.402-425

References:

- https://usubalang.github.io/usuba/
- Darius Mercadier, Pierre-Evariste Dagand. Usuba: High-Throughput and Constant-Time Ciphers, by Construction[8]. PLDI 2019.
- Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, Raphaël Wintersdorff. Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations[9]. EUROCRYPT 2020.
- Peter Schwabe, Ko Stoffelen. All the AES You Need on Cortex-M3 and M4[10]. SAC 2016.
- José Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, Pierre-Yves Strub. Jasmin: High-Assurance and High-Speed Cryptography[11]. CCS 2017.
- Ross Tate. Equality saturation[12]
- Alexandre Adomnicai and Thomas Peyrin.Fixslicing AES-like Ciphers New bitsliced AES speed records on ARM-Cortex M and RISC-V[13]

[8] https://usubalang.github.io/usuba/assets/documents/usuba-pldi19.pdf
[9] https://usubalang.github.io/usuba/assets/documents/tornado-eurocrypt20.pdf
[10] https://doi.org/10.1007/978-3-319-69453-5_10
[11] https://hal.archives-ouvertes.fr/hal-01649140
[12] https://rosstate.org/publications/eqsat/
[13] https://doi.org/10.46586/tches.v2021.i1.402-425