"You-Know-Why": an Early-Stage Prototype of a Key Server Developed using Why3

Diego Diverio[1], Cláudio Lourenço[2], and Claude Marché[2]

[1] Université Paris-Saclay, Inria, 91120 Palaiseau, France
[2] Université Paris-Saclay, Univ. Paris-Sud, CNRS, Inria, LRI, 91405 Orsay, France

Abstract. This is a preliminary report on a solution we designed for the VerifyThis Collaborative Long Term Challenge 2019 [6].

1 Used verification approach and tools

We used the Why3 verification framework [1] to design, from scratch, a simple but running prototype implementation of a PGP key server. The current version does not have a Web interface but a basic command-line interface. We exploit the ability of Why3 to extract OCaml code from verified WhyML code (getting rid of formal specifications and *ghost* code [3]) so as to produce code that can be compiled into an executable.

Our prototype is made of a combination of unverified handwritten OCaml code and of WhyML code whose functional behaviour is formally specified and proven correct. As Why3 is a framework for sequential programs only, our approach does not address any issue related to concurrency of execution of server requests.

2 How the challenge was adapted to make verification possible

The main approach we took was to follow the document presenting the challenge [5]. We also had a look at the implementation of the Hagrid key server [4] written in Rust, to get more precise ideas on its implementation. In particular, we discovered that it was using the file system of the host computer as a database, instead of any more sophisticated database management system.

From the challenge document [5] we addressed mission 1 (Safety), mission 2 (Functionality) and mission 6 (Termination). The mission 0 (Identify Relevant Properties) was not deeply investigated: we turned the informal properties of the five operations given in the challenge documentation [5, Section 5] into formal specifications in WhyML. For that we reused the theories for sets and maps from the Why3 standard library. Yet it appeared that such theories were in fact under reorganisation and we are using the new versions, so that replaying our proofs today requires to install the Why3 version in the master branch of development[3].

[3] Or presumably the upcoming next release 1.3.0.

We also used character strings, which is a newly available theory in the Why3 standard library.

We did not really address mission 3 (Protocol), though our functional specifications implicitly express properties about the server protocol, such as the fact that to confirm a key addition, only a token previously issued by the ADD operation can be accepted. Missions 4 (Privacy), 5 (Thread Safety) and 7 (Randomness) are not addressed at all.

We also did not implement a Web front-end though we could have done it with unverified OCaml code.

3 What has been achieved

Our implementation is divided into three parts as it is described in the challenge presentation [5]: a front-end (here a basic CLI program), a back-end (which actually implements the requirements) and a database (here a file-based implementation of dictionaries). The development is publicly available from the GITLAB repository `https://gitlab.inria.fr/why3/verifythis2020`. The `README` file there explains how to replay the proofs, how to compile an executable and how to run the latter.

The back-end interface is specified by a WhyML module `Spec` containing an abstract view of the server global *state* - under the form of a record type with ghost fields and invariants - and the expected five operations QUERY, ADD, CONFIRMADD, DEL and CONFIRMDEL. The back-end is implemented in another module `Impl` which refines `Spec`, the type *state* being extended with concrete fields. Gluing invariants relate those concrete fields with the ghost fields. The refinement is proved correct. The proofs are not difficult: they proceed smoothly using automated provers only.

The module `Impl` makes use of an extra module `Table` providing an interface to a general data structure of dictionaries mapping strings to strings. The `Table` module is only a WhyML interface, its implementation is written in OCaml and is not verified. Yet, this OCaml code makes use of an implementation of `Base64` encoding-decoding that is used to turn arbitrary strings into valid filenames. This `Base64` module is a completely independent library that was recently verified with Why3. This constitutes the database part of our prototype.

While the back-end operations rely on preconditions to ensure properties on inputs, the front-end adds a layer which checks inputs and raises exceptions when they do not conform to preconditions. The front-end is written in WhyML but is actually completed with OCaml code that provides the simple CLI interface. The front-end also implements an initialisation function that performs sanity checks on database when starting the server (see details below).

4 Successes and challenges

The main success is that we could actually write the specifications down in WhyML, develop WhyML code that is fully proven conforming to the specifica-

tions, and also extract an executable program from it that we can interface with OCaml external code, to effectively run a basic prototype server.

The proofs were not particularly challenging. Each verification condition was most of the time discharged by one of the automated theorem provers available in Why3 (one of Alt-Ergo, CVC4, or Z3 for this proof). Adding a few extra assertions in the code was enough in the remaining cases but one. This extra case has to be handled using a few manual Why3 transformations to be discharged [2], but was quite easy to prove anyway. The most challenging part was in fact the initialisation function: the code of this function must indeed do a lot of checks on the data read from disc, so as to ensure that the resulting server state satisfies the invariant. It is also the only part of the code using loops, thus requiring loop invariants to be added.

Of course, the remaining challenges are numerous, starting from the fact that we did not address at all the issues related to concurrency, privacy, or randomness. Another challenge would be to get closer to the current Hagrid server which uses clever techniques to store less information on disc. For example, it encodes all the needed information in the confirmation tokens: Hagrid does not have to store any information about the token, as it can recover it from the token itself, whereas we have to store locally for each token which email and key it was associated with.

Another challenge in a more general point of view is the amount of time, or human effort, to achieve such a verified server: we had quite little human resources to dedicate to this case study, and we had to aim at a very simple implementation if we wanted to have a working basic prototype. Designing a verified alternative to Hagrid, with all required guarantees regarding privacy and concurrency, should start by planning significant allocation of human resources.

Acknowledgements

For their contributions and feedback, we would like to thank François Bobot, Sylvain Dailler, Jean-Christophe Filliâtre and Mário Pereira.

References

1. Bobot, F., Filliâtre, J.C., Marché, C., Paskevich, A.: Let's verify this with Why3. International Journal on Software Tools for Technology Transfer (STTT) **17**(6), 709–727 (2015). https://doi.org/10.1007/s10009-014-0314-5, see also `http://toccata.lri.fr/gallery/fm2012comp.en.html`
2. Dailler, S., Marché, C., Moy, Y.: Lightweight interactive proving inside an automatic program verifier. In: Proceedings of the Fourth Workshop on Formal Integrated Development Environment, F-IDE, Oxford, UK, July 14, 2018 (2018)
3. Filliâtre, J.C., Gondelman, L., Paskevich, A.: The spirit of ghost code. Formal Methods in System Design **48**(3), 152–174 (2016). https://doi.org/10.1007/s10703-016-0243-x
4. Hagrid, a verifying OpenPGP key server. Web page `https://gitlab.com/hagrid-keyserver/hagrid`

5. Huisman, M., Monti, R., Ulbrich, M., Weigl, A.: VerifyThis collaborative long term challenge: The PGP key server — challenge manual. https://verifythis.github.io/VerifyThisLongTerm.pdf (Aug 2019)
6. VerifyThis collaborative long term challenge. Web page https://verifythis.github.io/challenge/ (2019)