A Solution to the Long-Term Challenge in SPARK

Claire Dross, Johannes Kanig, and Yannick Moy

AdaCore, 75009 Paris

1 Introduction

SPARK [2] is a programming language designed to be amenable to formal verification. In addition to the standard features of a procedural programming language, it contains standard annotations such as pre- and postconditions and annotations of effects on global variables. SPARK is a subset of the Ada language [1], so it can be compiled to machine code, e.g. using the GNAT compiler. It can also be freely mixed with (unverified) Ada code.

We have applied SPARK to the key server described in the long-term challenge. Our goal was to get a working prototype, so we did some simplifications in order to move quicker. For example, our prototype uses in-memory storage (simple lists) to store the key server data, as opposed to using external storage. This means that if our key-server is switched off, it will lose all data.

The code is available at https://github.com/AdaCore/Lumos_Maxima.

2 Overview of the Code

The central piece of verified code is the implementation of the server API in `server.adb`. In this code we verified absence of runtime errors, but also quite strong postconditions. Here are the declarations for `Request_Add` and `Verify_Add`, including their contracts:

```
procedure Request_Add
  (Email : Email_Id;
   Key   : Key_Id;
   Token : out Token_Type)
with
  Pre  => Invariant,
  Post => Invariant
    and Contains (Seen_Tokens, Token)
    and Email = Get_Email (Seen_Tokens, Token)
    and Key = Get_Key (Seen_Tokens, Token)
    and Is_Add (Seen_Tokens, Token)
    and Seen_Tokens'Old ≤ Seen_Tokens;

procedure Verify_Add
  (Token  : Token_Type;
   Status : out Boolean)
with
  Pre  => Invariant,
  Post => Invariant
    and (if Status
```

```
      then Contains (Seen_Tokens, Token)
        and Is_Add (Seen_Tokens, Token)
        and Model'Old ≤ Model
        and Included_Except
          (Model, Model'Old,
            (Get_Key (Seen_Tokens, Token), Get_Email (Seen_Tokens, Token)))
        and Contains
          (Model,
            (Get_Key (Seen_Tokens, Token), Get_Email (Seen_Tokens, Token)))
      else Model = Model'Old)
  and Seen_Tokens'Old ≤ Seen_Tokens;
```

As one can see, we prove that `Request_Add` returns a token with the proper information attached, and this token represents a request to add a key (expressed with `Is_Add`). Once `Verify_Add` is called, the key/email pair is added to the database (this is expressed using the `Model` variable). The code for `Request_Remove` and `Verify_Remove` is similar.

In total, five units (files) with 500 lines of code have been verified with SPARK Pro 20.1, out of 10 units with 1300 lines. The total running time of the proof tool on an 8-core/16-threads AMD Ryzen 1700x is roughly 1 minute from scratch, and roughly 30 seconds to replay all proofs using an existing session.

3 Difficulties and Workarounds

We found that SPARK proofs worked best when storing basic data such as integers in the central database, and not more complex data such as strings. The reason seems to be that the containers that we use rely heavily on equality for their specifications, and the equality for complex types is much heavier and degrades proof performance. This issue prevented us from writing natural code where e.g. email addresses would be represented directly by strings. We worked around this situation by interning strings in a separate table, and referring to the table indices instead of the actual strings. The string interning code is also proved.

4 The Missions of the Challenge

We addressed mission 0 (identify relevant properties) by defining the design of our software and writing postconditions, mission 1 (safety) by proving absence of runtime errors, mission 2 (functionality) by proving the specified postconditions, and finally mission 6 (termination) by adding termination annotations. Proof of termination only required a single loop variant.

We did not address mission 3 (protocol), mission 4 (privacy), mission 5 (thread safety) and mission 7 (randomness), which are out of scope for SPARK. Proof of thread-safety is possible in SPARK, though the analysis is quite restrictive - threads cannot access memory that is potentially written by other threads unless these accesses are protected against race conditions. Also, this analysis requires the use of Ada tasking and no other mechanisms, and visibility over all tasks (not the case in our prototype, where tasks might be created by the web server).

5 Additional Unverified Code

To get a working example, we wrote some unverified Ada code. This code starts a web server using the AWS (Ada Web Server [1]) framework and dispatches incoming requests to the various API functions. It also extracts emails from the submitted public key, so that the API functions can be called directly with interned email addresses. With reasonable additional effort, the percentage of verified code could be increased, to leave only the code directly related to AWS requests unproved.

When a user connects to the interface, a welcome page is presented with links to the subpages to query existing keys by email or adding a new key. On the page to add a new key, the user can insert a public key (the begin and end markers of a typical public key need to be removed here). From this key, the code extracts the email address, creates the request token from the interned email/key pair and returns the token to the user. To simplify the setup, we show the confirmation link directly in the browser instead of emailing it. This would of course have to change for a production implementation. The code invoked by this link then inserts the (interned) email/key pair into the database.

References

1. Barnes, J.: Programming in Ada 2012. Cambridge University Press (2014)
2. McCormick, J. W., Chapin, P. C.: Building high integrity applications with SPARK. Cambridge University Press (2015)

[1] https://www.adacore.com/gnatpro/toolsuite/ada-web-server