AdaCore TECH PAPER

NVIDIA: Using RecordFlux and SPARK to Implement SPDM for Secure Computing

NVIDIA: Using RecordFlux and SPARK to Implement SPDM for Secure Computing

Executive Summary

One of the basic requirements for enterprise platform security is device attestation: trustworthy evidence of a device's identity and security properties. The industry standard Security Protocol and Data Model (SPDM) addresses this need, defining message formats and session behaviors that device suppliers can implement for attestation.

AdaCore partnered with NVIDIA in a project that has implemented a subset of the SPDM Version 1.1.0 specification; the resulting library will be integrated into the firmware of microcontrollers and microprocessors that NVIDIA designs. AdaCore's RecordFlux toolset was used to formally specify the message structure and protocol behaviors in RecordFlux's Domain Specific Language (DSL) and to generate source code in the mathematically analyzable SPARK subset of Ada. AdaCore's SPARK Pro toolset was then used to prove memory safety (no buffer underflow / overflow), the absence of integer overflow, and other integrity properties of the generated code, as well as specific functional properties in several critical components. The SPARK code was subsequently compiled by AdaCore's GNAT Pro Ada tool suites targeted to the RISC-V and Arm architectures.

The SPDM project has expanded NVIDIA's usage of the Ada/SPARK technology and has shown the benefits of using the RecordFlux and SPARK Pro tool suites to design and implement complex high-assurance code for mission-critical applications. The project gave NVIDIA higher confidence in integrity properties of the resulting firmware than would have been achieved using traditional manual methods or an SPDM technology based on a language with less extensive built-in checking. And using the RecordFlux formalism to capture SPDM semantics resulted in a precise and unambiguous specification, avoiding potential misunderstandings of natural language descriptions.

SPDM

As stated in its Introduction section, *"The Security Protocol and Data Model (SPDM) Specification* defines messages, data objects, and sequences for performing message exchanges over a variety of transport and physical media. The description of message exchanges includes authentication of hardware identities, measurements for firmware identities and session key exchange protocols to enable confidentiality and integrity protected data communication. The SPDM allows efficient access to low-level security capabilities and operations." The standard was developed and is being maintained by DMTF (dmtf.org), a not-for-profit association comprising major players in the device and telecommunications industries. DMTF creates open manageability standards for IT infrastructure.

SPDM messages are exchanged between two endpoints, known as a Requester and a Responder, following a specific sequential protocol. The flow of messages is shown in the SPDM standard, from which the accompanying figure is derived, and comprises several kinds of messages:

Capability discovery and negotiation

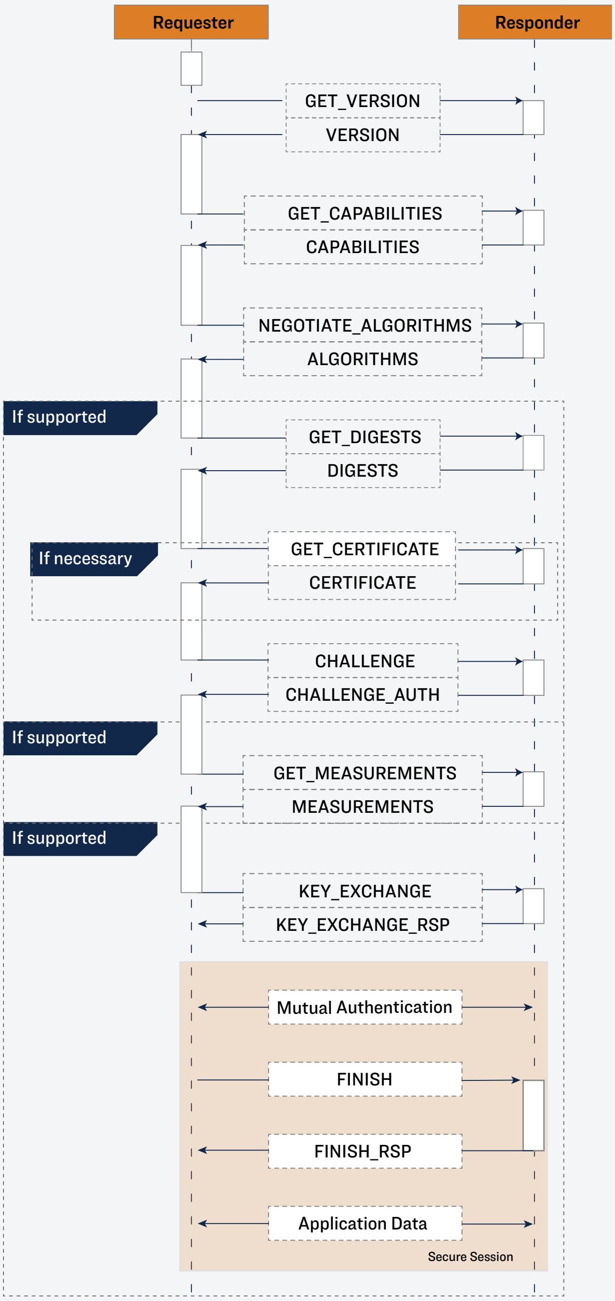
- The GET_VERSION / VERSION exchange establishes the major SPDM version to be used for subsequent messages.
- The GET_CAPABILITIES / CAPABILITIES exchange allows a Requester to discover the Responder's SPDM capabilities (for example, support for optional message exchanges).
- The NEGOTIATE_ALGORITHMS / ALGORITHMS exchange allows the Requester and Responder to agree on the cryptographic algorithms to be used.

Responder identity authentication, if supported / necessary

- The GET_DIGESTS / DIGESTS exchange allows the Requester to retrieve hashes of the Responder's certificate chains; these hash values (digests) can be used to determine whether the certificate chains have changed.
- The GET_CERTIFICATE / CERTIFICATE exchange allows the Requester to retrieve one or more certificate chains from the Responder, to establish trust in the Responder's identity.
- The CHALLENGE / CHALLENGE_AUTH exchange allows the Requester to verify that the Responder knows the private key associated with a certificate chain.

Firmware measurements, if supported

- The GET_MEASUREMENTS / MEASUREMENTS exchange allows the Requester to determine the configuration or other characteristics of the Responder, for example whether debug restrictions are in place.



Key agreement for secure channel establishment, if supported

- The KEY_EXCHANGE / KEY_EXCHANGE_RSP message pair is used to authenticate the Responder (or both parties), decide on cryptographic parameters, and establish shared keying data.
- The FINISH / FINISH_RSP exchange completes the handshake initiated by a KEY_EXCHANGE message.

The SPDM standard defines the detailed binary structure for each kind of message, specifying the size and interpretation of each field. The format is complex, with some messages containing interdependent fields and/or fields that are conditionally present.

RecordFlux

The AdaCore RecordFlux technology consists of a Domain-Specific Language (DSL) for precisely defining the structure of binary messages, and a toolset for generating formally verifiable code for parsers, message generators, and protocol sessions. The produced source code is in the SPARK language, an Ada subset that is amenable to mathematical verification analysis by AdaCore's SPARK Pro toolset. SPARK Pro can prove a range of program properties, ranging from valid information flow and memory safety, up to full functional correctness.

A simple but representative example to illustrate the RecordFlux DSL notation is a TLV ("Tag-Length-Value") message, which has two valid formats:

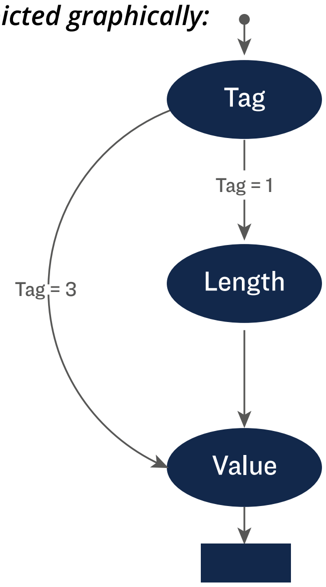
A structure with three components

- An 8-bit `Tag` field with value 1, indicating that two fields immediately follow,
- A 16-bit `Length` field, and
- A `Value` field (the payload), whose length (in 8-bit bytes) is specified by the Length field.

A structure with one component

- An 8-bit `Tag` field with value 3, indicating that no further fields are present.

The message structure can be depicted graphically:



A message with any other content in its `Tag` field is invalid.

The structure of messages is often non-linear because of optional fields. For this reason the RecordFlux DSL message syntax uses a graph-based representation (a directed acyclic graph). The order of fields is defined by `then` clauses, which are also used to state conditions and properties for the following field.

The TLV structure is specified as follows in RecordFlux, using an Ada-like notation. The valid Tag field contents are represented mnemonically by enumeration values `Msg_Data` (1) and `Msg_Error` (3).

TLV messages can carry `Value` fields of different types, and this flexibility is reflected in the special RecordFlux type `Opaque`. The actual type for a specific message will be specified in application code that generates or processes the message. The interpretation of an opaque field can also be specified in the RecordFlux language using a separate message type and a RecordFlux feature known as refinement.

```
package TLV is

   type Tag_Type is (Msg_Data => 1, Msg_Error => 3) with Size => 8;
   type Length_Type is range 0 .. (2 ** 16) - 1 with Size => 16;

   type Raw is
     message
        Length : Length_Type
           then Value
              with Size => Length * 8;
         Value : Opaque;
      end message;

   type Message_Type is
      message
         Tag : Tag_Type
            then Length
               if Tag = Msg_Data
            then null
               if Tag = Msg_Error;
         Length : Length_Type
            then Value
               with Size => Length * 8;
         Value : Opaque;
      end message;
end TLV;
```

Message processing sessions can typically be represented by a state machine with well-defined actions and transitions for each state. For the TLV example, assume that valid messages (i.e., those with `Tag = Msg_Data`) are processed by simply retransmitting the raw data that the message contains. RecordFlux uses Ada's syntax for generic templates to model this session behavior, reflecting the generic nature of sessions: the user must define the semantics of the formal functions and connect the channels when integrating the code that was generated for the session.

```
with TLV;
package TLV_Responder is
   generic
      Network : Channel with Readable, Writable;
      with function Plat_Create_Response
         (Message : TLV::Message_Type) return TLV::Raw;
   session TLV_Session is
      Request  : TLV::Message_Type;
      Response : TLV::Message_Type;
   begin
      state Idle is
      begin
         Network’Read(Request);
      transition
         goto Prepare_Response
            if Request.Tag = TLV::Msg_Data
         goto Send_Error
      end Idle;

      state Prepare_Response is
         Raw : TLV::Raw;
      begin
         Raw := Plat_Create_Response (Request);
         Response := TLV::Message_Type’(Tag    => TLV::Msg_Data,
                                        Length => Raw.Length,
                                        Value  => Raw.Value);
      transition
         goto Send_Response
      exception
         goto null
      end Prepare_Response;

      state Send_Response is
      begin
         Network’Write (Response);
      transition
         goto Idle
      end Send_Response;

      state Send_Error is
      begin
         Network’Write(Request);
      transition
         goto null
      end Send_Error;
   end TLV_Session;
end TLV_Responder;
```

The RecordFlux toolset can be used to generate Ada/SPARK source code from the specifications. This source code is then available to applications that need to process TLV messages and prove properties of the resulting code.

The Challenge

A modern enterprise platform is typically composed of a heterogeneous set of both reprogrammable and fixed-logic components and presents a rich target environment for malevolent actors. Device suppliers need to anticipate and thwart potential threats such as message tampering and confidentiality breaching.

A prerequisite for platform security is a secure authentication mechanism to establish trust between devices servicing requests for data ("Requesters") and devices responding to such requests ("Responders"). The Security Platform and Data Model (SPDM) specification addresses this need, but the message formats and message exchange protocols are complex. NVIDIA's challenge: implement a chosen subset of SPDM functionality, with high confidence that the implementation is provably correct (for example, knowing that their application is memory safe and would never raise a run-time exception), while staying within tight device storage and run-time memory constraints for the generated code. Achieving a high assurance level is critical, since the devices involved in attestation are roots of trust. They need to be bug free in order to provide confidence in the measurements that they report.

The Decision

NVIDIA has been using AdaCore's SPARK language and toolset to develop ultra-high reliability firmware in other contexts (see https://www.adacore.com/case-studies for more information), and their success in those efforts gave them a natural incentive to use the SPARK technology to implement SPDM. However, the complexities of the SPDM message formats, for example the interdependencies among message fields, are not easily modeled in the data definition facilities of a general-purpose language. Higher-level abstraction constructs in a domain-specific language oriented towards the specialized requirements of message communication would make the SPDM implementation more straightforward to develop and maintain, and allow for more rigorous verification.

AdaCore's RecordFlux product fulfills this need:

- It defines a DSL that can precisely express the structure of complex messages with interdependent and varying-size fields, as well as the logic of requester/responder protocols.
- It provides tools to generate SPARK code for generating and/or parsing such messages and for the protocol behavior (a finite-state machine).

Security properties of the resulting code can then be formally verified by the SPARK toolset. A benefit of RecordFlux is thus to make the power of SPARK's formal-methods-based verification more accessible in more use cases

Although other SPDM implementations are available (for example, OpenSPDM), NVIDIA wanted a technology with a more secure foundation, especially with respect to the programming language. All these considerations led them to choose the RecordFlux and SPARK approach.

The Project

The SPDM implementation for this project consisted of a number of tasks:

Architecture, Testbed & Integration

This phase of the project created a high-level architecture for the SPDM protocol implementation, defined the required interfaces, and set up a testbed to validate interoperability with OpenSPDM. It also integrated the resulting responder framework into the transport layer and implemented a mapping from session IDs to SPDM protocol instances.

SPDM Formalization

The structure of SPDM messages and the dynamic behavior of an SPDM responder were formalized in the RecordFlux DSL. The supported messages included capability discovery and negotiation, key exchange, and session termination.

Validation and Proof

The code generated from both the Architecture, Testbed & Integration task and the SPDM Formalization task were proven by the SPARK Pro toolset to be free of run-time errors, and thus verifiably free of any buffer overrun/underrun, integer overflow, or dereferencing of null pointers. Further, for conditionally structured messages, all field accesses were proved to be valid. The test tool was validated against an OpenSPDM responder to ensure that a complete key exchange and session establishment were possible. Additionally, the code size for the protocol implementation was verified to be within a specified limit (35kB when compiled standalone on a RISC-V 64-bit target).

Attestation

The SPDM RecordFlux model was extended to support attestation, for example with the formalization of the GET_DIGESTS, GET_CERTIFICATE, and GET_MEASUREMENTS messages and responses.

Session Key Update

The SPDM message and session models were extended to support the key update protocol.

As part of the joint project, AdaCore made a number of enhancements to RecordFlux based on SPDM-specific requirements and NVIDIA feedback:

- RecordFlux's Domain Specific Language was extended to support a more generalized message structure.
- The code generator was extended to handle the various SPDM message types and to ensure that the resulting SPARK code could be proved automatically for absence of run-time errors. Thus memory safety is assured (all buffer accesses are guaranteed to be within range), and “wraparound” integer overflow will not occur. Since these properties are proved statically, run-time code does not need to be generated (and indeed is not generated) to enforce these checks.
- The allocation scheme for session variables was changed, to avoid the need for dynamic allocation (thus better supporting the intended bare metal target environment and avoiding the need to demonstrate absence of heap exhaustion).

The Results

To both NVIDIA and AdaCore, the project demonstrated that RecordFlux and SPARK are appropriate technologies for implementing high-assurance, remote communication protocols, and SPDM in particular. Several factors stand out:

- RecordFlux's Domain-Specific Language has the expressiveness and precision needed to capture the complexities of the SPDM message structure and protocol sessions. The resulting specification serves as a "single source of truth" and prevents errors that can arise from misunderstandings of natural language descriptions of syntax and semantics.
- The SPARK code generated by RecordFlux was amenable to analysis by the SPARK Pro toolset to achieve several levels of assurance. For example, SPARK Pro proved that the code for the dynamic behavior of the SPDM responder for the implemented messages was memory safe and free of other run-time errors, and also proved specific security properties of the message structure code. The SPDM message specifications comprised around 3000 lines of code, from which 135K lines of provable SPARK code were generated.
- Code compactness is critical in the embedded applications where NVIDIA's firmware will be used, and the project demonstrated that RecordFlux can be competitive with C while allowing much higher confidence in the code's correctness. The code for the protocol implementation was less than 35KB on a RISC-V 64-bit target.
- AdaCore's responsive and helpful customer support is one of the company's hallmarks, and this project was no exception. AdaCore's engineers worked with the NVIDIA team to ensure that any questions concerning RecordFlux or SPARK were addressed promptly and thoroughly.

"We are very pleased with the results of our SPDM project with AdaCore," said Ron Koo, Senior System Software Engineer at NVIDIA. "We wanted a tool that would allow us to implement SPDM in SPARK and to prove important security properties, but to prove them at a higher level than the SPARK code itself. RecordFlux certainly facilitated that and resulted in a robust implementation, and we look forward to using it on other projects in the future."

"Our goal for RecordFlux from the outset has been to leverage formal verification and the SPARK language to address the vulnerabilities that communication protocols and complex data formats can bring," said Alex Senier, RecordFlux team lead. "The SPDM project with NVIDIA shows that this goal has been met. The RecordFlux DSL allows domain experts to precisely specify a protocol, and the SPARK toolset can formally prove a range of security properties for the generated code. In short, RecordFlux and SPARK can be a real game changer for highly trustworthy protocol implementations."

For more information

For details on SPDM:

- DMTF, Security Protocol and Data Model (SPDM) Architecture White Paper; DSP2058, Version 1.0.0; 2020-09-04.
https://www.dmtf.org/sites/default/files/standards/documents/DSP2058_1.0.0_1.pdf
- DMTF, Security Protocol and Data Model (SPDM) Specification; DSP0274, Version 1.1.0; 2020-07-15.
https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.1.0.pdf

A public version of the SPDM implementation, together with the validation platform, is available at https://github.com/AdaCore/spdm-recordflux

For information about RecordFlux see
https://adacore.com/recordflux

For information on SPARK see
https://docs.adacore.com/spark2014-docs/html/ug/