The new Internet Protocol security IPsec testing with TTCN-3

Ariel Sabiguero[1,2], María Eugenia Corti[1], and César Viho[2]

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
{asabigue,mcorti}@fing.edu.uy
http://www.fing.edu.uy/inco

[2] IRISA
Campus de Beaulieu
35042 Rennes CEDEX, France
{asabigue,viho}@irisa.fr,
http://www.irisa.fr/armor

Abstract. IPsec is a set of protocols designed to provide security to the new version of the Internet Protocol (IPv6). It includes encryption and other complex operations. TTCN-3 offers several ways to develop test cases but no real methodological guide is provided to help in choosing where to put complexity.
Two orthogonal approaches were studied. On the one hand, everything can be done using coding and decoding functions. On the other hand external functions can be used for for complex operations, leaving the CoDec for only simple encoding operations. Both approaches were implemented for a selected test case. Strengths and weaknesses of each are unveiled, and ultimately compared.

Keywords: IPv6, IPsec, TTCN-3, ATS, ETS

1 Introduction

The Internet Protocol version 6 (IPv6) is the new version of current Internet Protocol (IPv4). The most popular enhancement of IPv6 is the growth of the IP address space, but several other changes are introduced. One important improvement is that the security aspects are included in the specification. In the IPv6 suite, confidentiality and authentication mechanisms has been specified since the initial drafts. Thus testing IPv6 must include the testing of the new Internet Protocol security features. This is already the case in the world wide IPv6 Ready Logo certification program that provides test suites for IPsec (Internet Protocol Security) [1–3]. IPsec is a set of protocols that provides cryptographically based security at the IP layer, protecting the network and upper layers. The services offered by IPsec includes: confidentiality, connectionless integrity and data origin authentication. TTCN-3 language was designed to specify and implement any kind of testing activity in an abstract and efficient way. It has been successfully applied for the new Internet Protocol testing. Moreover, in the recent years, different research groups have released public TTCN-3 libraries that ease IPv6 test case development.

The objective of this work is to present the use of TTCN-3 language and tools to test IPv6-IPsec, specifically conformance. The test cases themselves exchange only few messages between the tester and the Implementation Under Test (IUT), and could be considered quite simple to implement but they hold the inherent complexity of the encryption, decryption and authentication/integrity algorithms, among others. IPsec specification (by means of an RFC - Request for Comments) indicates which authentication/integrity and encryption algorithms are used. The RFC [1] does not specify the algorithms themselves, but describes how to use existing ones. Thus, in the test specification, these algorithms are not implemented in TTCN-3. Already existing libraries that implement the required algorithms are used.

This work compares different methodological approaches to reuse existing functions and distribute complexity of the task across TTCN-3 standard interfaces. One possibility is to model the encryption stage as an operation performed and specified in the TTCN-3 Abstract Test Specification (ATS) of the test case. Other possibility is to consider the encryption as a transmission problem. Consequently, making the TTCN-3 ATS unaware of the encryption/decryption task. Different decisions lead to different tester configuration and Executable Test Suites (ETS) for the same test requirement. We explore how these ATS design decisions impact the ETS, simplifying or hardening the test development process. Pros and cons are discussed.

This work shall help the reader to understand deeply the different interfaces present in TTCN-3 and how to use them effectively to address particular problems. Different decisions lead to different capabilities and expressiveness of the TTCN-3 ATS. Practical results are presented.

The work is organized as follows. Section 2 highlights the principal aspects of the IPsec protocol and presents a general description of IPsec tests. Section 3 introduce the test selected to be implemented, the requirements and available tools used. In Section 4 the two methodological approaches implemented are introduced and they are compared in Section 5. Conclusions are presented in Section 6.

2 IPsec highlights

IPsec is a suite of security protocols that offers access control, connectionless integrity, data origin authentication and confidentiality, among other services, for IPv4 and IPv6. These services offered protect the IP layer and upper layer protocols.

2.1 Protocol description

Two protocols are used by IPsec to provide security: Authentication Header (AH) and Encapsulating Security Payload (ESP). AH provides connectionless integrity, data origin authentication and optionally anti-replay service. Beside this, ESP may provide confidentiality too. Both of them also provide access controls by the use of cryptographic keys, that can be distributed manually or automatically. AH an ESP are used in conjunction with a set of cryptographic algorithms specified in RFC 4305 [4].

Both protocols, AH and ESP, can be used alone or can be combined. ESP can be used to provide both functionalities, integrity and confidentiality, or it can be used to provide only integrity, the same functionality provided by AH. This makes AH to be not only a specification requirement, but an option.

The IPsec protocols can be used in two modes: transport and tunnel. In transport mode security is provided for the upper layer protocols and not for the IP header. In the case of AH some portions of extension headers are also covered. In tunnel mode the security protocols are applied to the entire IP datagram, including the IP header.

The security protocol (ESP or AH), the mode, the cryptographic algorithms, how to combine the specified protocols and services and the traffic that will be protected, are specified by the Security Associations (SA) and the Security Policy Database (SPD).

As defined in [3] an SA is a simplex "connection" that affords security services to the traffic carried by it. For a typical communication two SA are required, one for each traffic direction. Also, if AH and ESP protocols are combined, two SA must be created, one for each protocol. Each SA is an entry in the SA Database (SAD). In the SA the security protocol and the mode are specified among other parameters that defines the connection.

The SPD control whether and how IPsec is applied to traffic transited or received. The SPD must be consulted while processing the traffic, incoming or outgoing, even in traffic that IPsec protection is not required.

2.2 General Test description

The IPv6 forum implements the IPv6 logo with the objective of give confidence to users that IPv6 is available and ready to use. They provide a suite of test that should be passed to get the logo. Specification conformance and Interoperability are tested. For this work we concentrate in the conformance test suites specified by the IPv6 Ready Logo Technical Committee (v6LC).

IPsec testing is about IPsec, and not about IPv6 testing. IPsec implementation is strongly encouraged in IPv6, but different parts of the protocol suite are tested separately. By the moment IPsec is addressed, IPv6 must have been tested before, and must have got a hundred percent of pass verdicts. The same principle of separation of concerns is applied to the encryption and privacy providers, with the difference that there is no test on the suite that addresses their correctness.

IPsec tests address, as said in Section 2.1, the two different modes present in IPsec: tunnel and transport. The mode requirement depends on the targeted usage. For each of them, it tests the different combinations of encryption algorithms and the authentication ones. For both algorithms' types two categories are defined: *base* and *advanced*. The algorithms included in the *base* category are mandatory for all equipment and the ones included in the *advanced* are required only for equipment that supports these algorithms.

Manual key configuration is used, but dynamic negotiation of keys is an accepted alternative, using Internet Key Exchange (IKE) protocol. Although, IKE is addressed in a different test suite, devoted to it.

From the two security protocols used by IPsec, AH and ESP, only ESP is required and tested.

During the execution of the selected test case, an IPsec-ICMPv6 Echo Request message is sent to the Node Under Test (NUT). The NUT must receive the message, process it and return an IPsec-ICMPv6 Echo Reply message.

3 Requirements on the testing platform

This work takes from granted the IPsec test suite definition engineered by the v6LC. It is not addressed *what* to test in order to ascertain the correctness of an IPsec implementation, but *how* to do it with TTCN-3.

IPsec test specification is published by the IPv6 Ready Logo as an English written document [5], complemented with some graphics. English ATS is translated into TTCN-3 specification, with additional, test specific, functions implemented through the standard TTCN-3 interfaces. Main requirements include IPv6 data type handling and cryptographic routines.

A TTCN-3 test system can be thought conceptually as a set of interacting entities, each implementing a specific test functionality. Figure 1 shows the general structure of a TTCN-3 test system. We will focus only on the main concepts addressed by this work.

The TTCN-3 Executable (TE) interprets and executes compiled ATS. SA, which stands for SUT Adaptor (System Under Test Adaptor), "adapts" communications between the TTCN-3 system and the SUT. The Platform Adaptor (PA) implements (amongst others) external functions. External functions are convenient ways of executing platform language code, in our case, ANSI C. The Test Management (TM) entity is responsible for overall management of a test system. Finally, the Coding and Decoding (CD) entity is responsible for the encoding and decoding of TTCN-3 values into bitstrings suitable to be sent to the SUT. All these definitions can be found in [6–8].

The comprehensive detail of the way this runtime components interact is beyond the scope of this work. Nevertheless, the following simplified examples give a grasp of the semantic behavior. It is important to understand the interaction of these elements during a `send` operation. The runtime behavior of a send operation is to take the template, hand it to the associated CoDec through

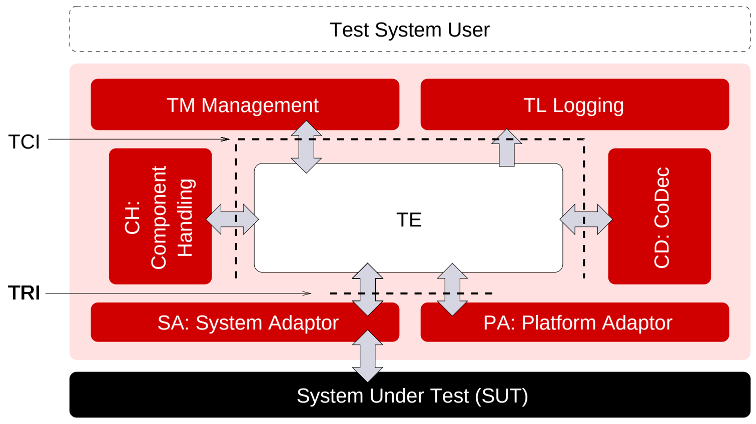

Fig. 1. Conceptual architecture of TTCN-3

the TCI/CD interface and obtain its representation as a BinaryString. The bit-oriented representation is passed then to the TRI/SA function that implements the `port` implementation, ultimate responsible of the transmission.

Another important operation is the invocation of an external function. External functions provide ways to extend TTCN-3 language with platform language functionality. When an external function is invoked from the TTCN-3 ATS, the runtime behavior requires that the TTCN-3 variable is converted into its BinaryString representation using the associated CoDec through the TCI/CD interface. The bit-oriented representation is passed trough the TRI/PA interface to the registered external function, that will perform the expected operation over the bitstring. Upon the function return, the BinaryString is again used to invoke the corresponding CoDec, this time to obtain a TTCN-3 variable out of the bitstring, through the TCI/CD interface again.

These are the basis for extending TTCN-3 functionalities with platform language. Section 4 explains different ways of using this API to generate IPsec messages.

3.1 The selected test case

This work bases its results on experiments made basically on a specific test case, number 5.2.3 of [5]. The test case has an extense preamble detailing Security Association Databases (SAD) configuration and Security Policy Databases (SPD) for each node. Exchanged packets are also detailed.

This work is focused in the implementation of transport mode test, with encryption algorithm 3DES-CBC and authentication algorithm NULL. Figure 2 shows the test topology: the way involved nodes are corrected.

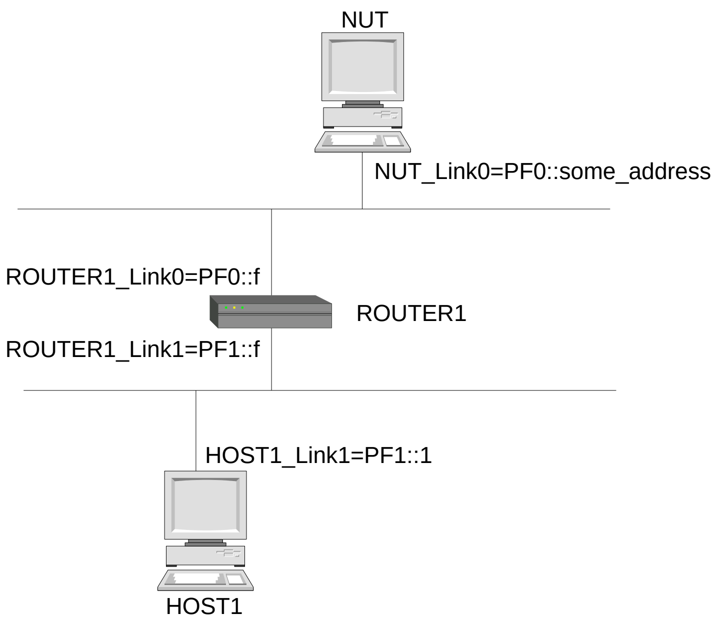

Fig. 2. Test Topology.

```
Procedure:
HOST1_Link1(TN)         Target(NUT)
|                            |
|--------------------------->| ICMP Echo Request with ESP
|                            |
|<---------------------------| ICMP Echo Reply with ESP
|                            |      (Judgment #1)

1. HOST1 sends "ICMP Echo Request with ESP"
2. Observe the packet transmitted by NUT

Judgment:
Judgment #1
Step-2: NUT transmits "ICMP Echo Reply with ESP"
```

Fig. 3. TestCase 5.2.3: TransportMode ESP=3DES-CBC NULL

Figure 3 extracted from [5], describes the test procedure (scenario) and verdict criteria. It is noticeable that the procedure consists of a stimulus and a response, with a statement regarding of the correctness (judgment).

3.2 Available tools

At the time this work began, there were no TTCN-3 IPv6 IPsec libraries or tools available on line that could be reused. IRISA's T3DevKit [9] with IPv6 examples and ETSI's TC MTS-IPT [10] TTCN-3 IPv6 test tools and suites were publicly available. Both of them seemed a suitable starting point for our development. IRISA's T3DevKit was selected due to existing knowledge of the tool and no particular aid to IPsec testing on ETSI's public Abstract Test Suites (ATS).

As encryption routines are not part of the test purpose, it was decided to reuse existing ones. GNU Libgcrypt was selected because it is freely available, there are good examples of its usage and there is experience of its IPsec usage.

The rest of this Section analyzes these building blocks and the test development challenges to be addressed.

T3DevKit The T3DevKit is a helper for implementing TRI-PA, TRI-SA and TCI-CD interfaces in order to build the executable test out of a TTCN-3 abstract specification. It provides the T3CDGen, an automatic generator that extract type definitions present in the TTCN-3 source files and generate most of the C++ code needed to implement the logic of the TCI-CD interface. The T3DevLib provides a framework of C++ classes that eases TTCN-3 data type handling, together with port and timer definitions suitable for working over Ethernet networks.

Examples provided with the T3DevKit provides types and functions for handling IPv6, ICMPv6, TCP and UDP. The T3DevKit not only eases our work because it provides several elements that were reusable, but because it automatizes the CoDec generation.

The T3DevKit is licensed under CeCILL-C license.

Libgcrypt Libgcrypt is a general purpose cryptographic library which works on POSIX systems. It is built based on the code from GNUPG and provides functions and support for several cryptographic ciphers, hash algorithms, message authentication codes (MAC), etc. It has a broad user base and provides all the functionality required for implementing IPsec.

3.3 What can be reused and what has to be developed

With the tools selected, we have enough building blocks to simplify our abstract test engineering. We already have implementations for IPv6 packets, ICMP messages, UDP datagrams and TCP segments. Most of this code can be reused to perform the IPsec test cases, and some just need to be adapted with minor modifications. Implementations for 3DES, SHA and other cipher related functions are also available. What is required now is to glue all these things together and to build the ETS.

It is clear that IPsec TTCN-3 data types have to be developed, together with their corresponding encoding and decoding functions. Also the TTCN-3 templates that will be used for the tests have to be defined. Beside this, we have to integrate the new code implemented and the reused one. Figure 4 shows the TTCN-3 data types defined for representing the ESP message.

```
type record ESPMessage {
        octetstring     SPI length(4),
        UInt32          SeqNum,
        EncPayload      Payload,
        octetstring     ICV optional
  }

type record EncPayload {
        IPDatagram Data,
        integer TFCPadding optional,
        octetstring Padding length(0..255) optional,
        UInt8 PadLength,
        UInt8 NextHdr
  }
```

Fig. 4. TTCN-3 data types for ESP message

Further explanation of the test cases implemented and details of the implementations are presented in the following Section.

4 Alternatives on test case engineering

CoDec task is to convert TTCN-3 objects into transmittable bitstrings. Particular details of the communication are removed from the TTCN-3 ATS and relayed to CoDecs. The direct usage of the T3DevKit suggests also to relay other functionalities to the CoDecs, like message size calculation or checksum computing. The natural way to extend this methodology would be to perform cipher operations on the CoDec, moving there all IPsec handling. This way of assembling IPsec messages does not seem natural, as all the IPsec assembly is done in C++ CoDecs.

Another approach is to embed cryptographic operations inside the TTCN-3 code using external functions. This approach provides more control to the ATS during the encryption process, lightening the weight of the CoDec. T3DevKit also offers a wrapper to give access to external functions, which we used.

Following Subsections describe these two test development implementation strategies.

4.1 First approach: encryption and codification relayed to the CoDec

The natural way of extending CoDec based, existing public ATS, to address IPsec was to implement cryptographic and authentication functions in the CoDecs too. We shall analyze independently transmission and reception operations.

Transmission Performing all the encoding in the CoDec, removes most of the cryptographic details from the TTCN-3 code. In this way the ESP packet is modeled in TTCN-3 without applying any cipher algorithm and passed to the CoDec. The C++ CoDecs perform the corresponding cryptographic operations and assemble the packet that will be finally sent to the NUT.

```
Link1.send(ICMPv6WithESP_EchoRequest_AuthNULL(SPI_SA1, DATA));
```

Fig. 5. Complete transmission processing in the ATS

The unencrypted message template is sent to the CoDec. The CoDec receives the TTCN-3 values and encode them into bitstring, but there are several things to be done. Before building the BinaryString with the transmittable representation, part of the message must be encrypted, and before encrypting some values must be calculated. T3DevKit provides Encode and Decode methods for each field of a packet. These methods simplify finding the fields that must be encrypted but it is an intricate task to calculate fields like checksum, padding and padlength.

The length of all the fields have to be calculated to be able to determine the padlength. Handling of the bitstring representation is not natural in C++. Even though the T3DevKit provides tools for handling the bitstring (a cursor and operations over the bitstring representation), the task is error prone. Indeed, as the T3DevKit works on the bitstring, but C++ native libraries work on memory addresses, byte oriented, several translations have to be performed from the bitstring into byte representation and vice versa. These operations are highly error prone.

Although the T3DevKit soothes the work, there is not a common representation of types and data between TTCN-3 world and C++ one.

Reception For the reception of messages, the same design decision of moving all the cryptographic operations to the CoDec can be applied, leading to a very clear abstract specification. Figure 6 shows the piece of code corresponding to the test case implementation.

```
alt
  //Receive the correct answer
  [] Link1.receive(ICMPv6WithESP_EchoReply_AuthNULL(SPI_SA2, ''O))
          setverdict(pass);
          replyTimer.stop;

  //Receive incorrect answer
  [] Link1.receive
          setverdict(fail);
          replyTimer.stop;

  //Receive no answer
  [] replyTimer.timeout
          setverdict(fail);
```

Fig. 6. Complete reception processing in the ATS

It is straightforward to follow the logic of the message reception. The `pass` verdict is only issued if the received message can be matched to the `ICMPv6With ESP_EchoReply_AuthNULL` template. In any other cases, a `fail` verdict is issued. All the logic regarding proper encryption is placed on C++ CoDecs.

The power given by the T3DevKit tool to the CoDec generation translated parts of the protocol complexity to the coding operation. Length calculation can be considered a simple operation, that can be handled during encoding. Checksum calculation is not a simple operation (at least not as simple as length calculation), but the CoDec is an elegant place to perform it. We shall analyze the result of removing cryptographic tasks from the CoDec in the following Subsection.

4.2 Second approach: encryption/decryption done in external functions

By "encryption/decrytion done in external functions" we describe the decoupling of purely coding operations from semantically rich ones. Even though it is possible to discuss what is purely coding, we think that performing cryptographic operations cannot be considered a simple operation.

Transmission The objective of this design decision is to be able to have a complete encoded value, accessible and represented in TTCN-3 variables before the final BinaryString encoding is performed. The CoDec do not need to perform operations to the objects received from TTCN-3, but just to convert the TTCN-3 value into its bit oriented representation.

```
template ESPMessage ICMPv6ESPMessage (IPv6AddressType src,
                    IPv6AddressType dst, octetstring m_spi,
                    octetstring m_data, UInt16 checksum) := {

          SPI:= m_spi,
          SeqNum := 1,
          Payload := EncryptPayload(src, dst, EchoRequestType,
                                   m_data, checksum),
          ICV :=omit
}
```

Fig. 7. TTCN-3 template for ESP with external functions

The Figure 7 shows how we use external functions to compute cryptographic generated values of the ESPMessage. It is possible to see how external function invocation is embedded in the template definition with the `EncryptPayload()` function. The ESP message template definition includes the parameters it receives, and the ones that has to be passed to the external function responsible for performing the encryption.

Before encrypting the payload, it's content (the ICMPv6 packet) must be built. Consequently, it's length and checksum need to be calculated and accessed from TTCN-3. Thus, we need to use external functions in this case too. We illustrate checksum calculation in Figure 8. The checksum is calculated calling the external function `GetCheckSum()` before assembling the packet. The calculated checksum is passed as a parameter to the template defined for the ESP message.

```
var UInt16 checksum := GetCheckSum(PF1_1, PF0_1, EchoRequestType,
                                   DATA, NextHeaderIcmpV6);

Link1.send(ICMPv6WithESP_EchoRequest(PF1_1, PF0_1, SPI_SA1,
                                   DATA, checksum));
```

Fig. 8. TTCN-3 checksum calculation

Reception This approach also introduces changes, challenges and differences in reception operations, and the way received information is treated. External functions can also help validating the message and are used to decrypt the message. It is pretty straightforward to see that TTCN-3 matching mechanisms based on wildcards does not apply inside encrypted structured data fields. They have to be decrypted first.

```
alt{
     //Receive correct answer, unverified encrypted payload
     [] Link1.receive(ICMPv6ESPMessage_Answer_AuthNULL
                    (PF0_1, PF1_1, SPI_SA2, DATA, checksum)) -> value Myvar {
           var bitstring encpayload := Myvar.Payload;
           var UInt8 payloadLength := lengthof(encpayload)/8;
           var EncPayload payload := DecriptPayload(encpayload, payloadLength);
           if (match(payload, ICMPv6EncPayload_Answer(PF0_1, PF1_1, DATA))) {
                  setverdict(pass);
           } else {
                  setverdict(fail);
           }
           replyTimer.stop;
     }
     //Receive incorrect answer
     [] Link1.receive {
           setverdict(fail);
           replyTimer.stop;
     }
     //Receive no answer
     [] replyTimer.timeout {
           setverdict(fail);
     }
}
```

Fig. 9. TTCN-3 code to validate received encrypted message using external functions

Figure 9 shows actual `alt[]` used for encrypted message reception and verdict issuing. Message is received and compared to the corresponding template, shown in Figure 10. The resulting value is assigned to the variable **MyVar**. From this variable the encrypted payload is extracted and passed to an external function to be decrypted. The decrypted value is then decoded into the type **EncPayload** and then passed to the function **match()** (provided by TTCN-3), to be compared with the corresponding template. Whether the result is true the issued verdict is `pass`. `fail` is issued in other cases.

```
template ESPMessageAnswer ICMPv6ESPMessage_Answer_AuthNULL
                    (Ipv6AddressType src, Ipv6AddressType dst,
                     octetstring m_spi, octetstring m_data,
                     UInt16 checksum) := {

        SPI:= m_spi,
        SeqNum := ?,
        Payload := ?,
        ICV := omit
  }

}
```

Fig. 10. ESPMessageAnswer template

The checksum is also verified with an external function invocation, defined in the template to check the incoming ICMP echo request. Figure 11 shows the template and the checksum calculation function.

```
template ICMPv6MessageType icmpv6_EchoReply (Ipv6AddressType src,
                      Ipv6AddressType dst, octetstring m_data) := {

      Type:=Icmpv6EchoReplyType,
      Code:=uint8_0,
      Checksum:=GetCheckSum(src, dst, Icmpv6EchoReplyType,
                            m_data , NextHeaderIcmpV6),
      ChecksumShouldBe:=omit,

      body := {
          echo :={
              Identifier := uint16_0,
              SequenceNumber := uint16_0,
              Data := m_data
          }
      },
      Options :=  omit
}
```

Fig. 11. icmp echo reply validation template

For handling external functions T3DevKit provides an implementation of *triExternalFunction()* for multiplexing calls and presenting the data and a class for manipulating the parameters. External functions permits building the complete message in TTCN-3 types and data structures simplifying and reducing the codification in the CoDec. Comparison between the two methods is done in the following section.

5 Comparison

We have implemented and shown two different approaches to TTCN-3 test case engineering of IPsec protocol test cases. We first shown a direct extension of the IPv6 examples provided with the T3DevKit, that performs all the required tasks at the CoDec level. The other approach presented was using external functions to control the complete message assembly from TTCN-3, using only CoDec for data representation conversion between TTCN-3 variables and transmittable bitstrings. In the following, we compare these two approaches, according to different criteria.

5.1 Code engineering

We analyze all the aspects required to produce an executable test case, not only the TTCN-3 ATS. We compare both TTCN-3 specification and platform language readability together, despite of the fact that maybe different groups of developers, with different backgrounds, address each part. First, we address message transmission and afterward, reception.

Without external functions, message assembly and encryption are performed in a single function. Moreover, as the CoDec (accessed from the `send()` operation) is not intended to be used as a regular function, it does not receive other extra parameters than the template to be transmitted. No control on the encryption keys or other arguments specified in the test specification is accessible from the TTCN-3 code. With the usage of external functions, message is passed to the CoDec with all the data fields calculated, thus only BinaryString encoding is required. The logical sequence of the code is simpler to understand, as separated abstract concepts are mapped to individual functions. The test case becomes simpler to implement, as divide&conquer principles apply now. Just specific functions are implemented in C++ to calculate some field values that could not be implemented with TTCN-3, yet the message assembly logic and sequence is handled from the TTCN-3 ATS.

While receiving the message for validation, an external function is used to decrypt part of the message and then compare it with the corresponding template. The complete validation process is done in TTCN-3. Without the usage of external functions, only part of the message construction is controlled from the TTCN-3 ATS. Some fields like checksum, padding and padlength are not calculated in TTCN-3 and have to be added in the CoDec. This becomes relevant for the payload generation. We need to have an ICMPv6 Echo Request message, whose creation has been delegated to the CoDec. To keep our implementation aligned and coherent with existing one, ESP assembly should be relayed to the CoDec too. This fact forces that a part of the ATS is moved to the CoDec and is not specified in TTCN-3 language. The test case is then split into TTCN-3 and C++ CoDecs. To understand the test you have also to know the code implemented by the CoDec. This approach seems not to follow the TTCN-3 philosophy and tends to put too much semantic in the CoDec.

5.2 Test specification size

As we are comparing two implementations of the same specification developed with the same language it is possible to compare the methodologies based on some code metrics. The first thing that is important to consider is that existing IPv6 types and definitions were reused, and they are the biggest part of the TTCN-3 code.

Table 12 shows the different parts of the code we measured. The metric used is the effective lines of code (*loc*). Comments, blank lines, lines with only block delimiters and other kinds of "empty" lines were removed, and only a single command per line was accepted.

		CoDec only	CoDec + Ext. Functions
TTCN-3	Test case	81	81
(loc)	Accessory	812	797
C++	ext	0	234
(loc)	CoDec	681	255
TTCN-3		893	878
C++		681	489

Fig. 12. Some *loc* based software metrics

Size of TTCN-3 code is equivalent in both methodologies, with only slight adjustments. It is noticeable that test case specific code accounts for a 10% of all the required code. Modeling of IPv6, ICMPv6, options accounts for most of the TTCN-3 code, which we managed to reuse from publicly available IPv6 ATS.

Differences arise when we consider platform language coding, both for CoDec and External functions. It was part of the methodological approach to avoid external functions in the first implementation, accounting only for CoDec implementation. The second implementation methodology splits the complexity, but it is more than splitting it. It diminishes the number of lines of code required. The total number of lines of C++ code shrunk almost 30%, spread over a bigger number of shorter functions. These properties suggests that the code is also more maintainable.

5.3 Performance

The main drawback found in the usage of external functions is the performance overhead. Every time an external function is invoked, the TTCN-3 values are encoded and passed to the external function. Upon exit, values are decoded and passed back to TTCN-3. None of this happens in the approach that does not require external functions.

The code requires 4 external function invocations during transmission and reception, thus 4 additional code and decode cycles. The performance impact of this is relevant for time sensitive Test case, but not in general case of conformance testing.

6 Conclusions

We acknowledge that the comparison was applied only to subset of the whole Conformance test suite, but we believe interesting conclusions can be taken. The TTCN-3 ATS developed when putting all the operations in the CoDec is very clean and readable, but we feel that important parts of the test specifications have to be moved to the CoDec. Too much IPsec behavior is not expressed in

TTCN-3 language and is relied to CoDec. CoDec abstraction level -even augmented with the T3DevKit- is too low and operations are hard to maintain and implement. We think that this approach diverges from TTCN-3 design strategy.

Moving all the operations to the external functions provide a much more comfortable framework. No changes in the size of TTCN-3 were found, and it is still abstract enough, while keeping all required semantic for more detailed test case operations. The weight of the CoDec is lightened, but the number of invocations grew significantly. The total number of *loc* in platform language shrunk, making the test case smaller and easier to develop. It seems that with this approach we obtain better engineered test cases, at the expense of performance degradation.

Further studies are in progress, but current findings seems to indicate that the best option is to design test cases making use of the external functions, whenever performance restrictions allows it. Despite of that, we think it is a good approach to leave simple operations in the CoDec. Without trying to define what *simple* mean for all possible ATS, our experience seems to indicate that all operations that are related to the experiment definition shall not be relayed to the CoDec. If there is a doubt, then is better to implement that operation as an external function.

Acknowledgment

We would like to mention the importance of the T3DevKit for this study. It proved solid and a powerful tool that eases test case engineering and it proved enough flexible to accommodate both design choices presented in this article. The authors would like to thank Anthony Baire for his thorough comments, ideas and support during this work.

References

1. K. Seo S. Kent. RFC 4301: Security Architecture for the Internet Protocol. http://www.rfc-editor.org/rfc/rfc4301.txt, 2005.
2. S. Kent. RFC 4303: IP Encapsulating Security Payload (ESP). http://www.rfc-editor.org/rfc/rfc4303.txt, 2005.
3. C. Kaufman. RFC 4306: Internet Key Exchange. http://www.rfc-editor.org/rfc/rfc4306.txt, 2005.
4. D. Eastlake. RFC 4305: Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). http://www.rfc-editor.org/rfc/rfc4305.txt, 2005.
5. IPv6 Ready Logo. Phase II Test Specification IPsec. http://www.ipv6ready.org/pdf/IPsec_1_8_0b3.pdf (last ckecked 22/04/2006), 2007.
6. ETSI. ES 201 873-1 Part 1: TTCN-3 Core Language, Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187301v030101p.pdf, 2005. [Online; accesed 19-April-2006].

7. ETSI. ES 201 873-5 Part 5: TTCN-3 Runtime Interface (TRI), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187305v030101p.pdf, 2005. [Online; accesed 19-April-2006].
8. ETSI. ES 201 873-6 Part 6: TTCN-3 Control Interface (TCI), Version: 3.1.1. http://webapp.etsi.org/exchangefolder/esi_20187306v030101p.pdf, 2005. [Online; accesed 19-April-2006].
9. T3DevKit. http://t3devkit.gforge.inria.fr/, 2007. [Online; accesed 22-April-2007].
10. TC MTS-IPT: IPv6 Testing an eEurope Project. http://www.ipt.etsi.org/deliverable.htm, 2007. [Online; accesed 22-April-2007].