## Contents

1. Disclaimer ............................................ 5
2. Introduction ....................................... 5
3. Executive Summary .................................. 5
4. Technical Overview
   4.1 Component Description .......................... 6
   4.2 Scope ........................................... 7
   4.3 Synthesis of Analysis .......................... 7
   4.4 Secure Deployment Guide ....................... 8
5. SSL Server Component Analysis ................. 14
   5.1 SSL Server Verification Summary .............. 14
   5.2 SSL Server API ................................ 15
   5.3 SSL Server Sub-component Integration .......... 15
   5.4 SSL Server Analysis Context .................... 16
   5.5 SSL Server Coverage Analysis .................. 17
   5.6 SSL Server Reviewed Alarms ..................... 18
   5.7 SSL Server Intermediate Annotations .......... 19
6. MD5 Sub-component Analysis ..................... 20
   6.1 MD5 Verification Summary ....................... 20
   6.2 MD5 API ........................................ 20
   6.3 MD5 Sub-component Integration ................. 20
   6.4 Verification of md5_starts ...................... 20
   6.5 Verification of md5_process .................... 22
   6.6 Verification of md5_update ..................... 24
   6.7 Verification of md5_finish ..................... 26
7. AES Sub-component Analysis ....................... 30
   7.1 AES Verification Summary ....................... 30
   7.2 AES API ........................................ 30
   7.3 AES Sub-component Integration ................. 30
   7.4 Verification of aes_crypt_cbc ................... 30
8. SHA-1 Sub-component Analysis ................... 35
   8.1 SHA-1 Verification Summary ..................... 35
   8.2 SHA-1 API ....................................... 35
   8.3 SHA-1 Sub-component Integration ............... 35
   8.4 Verification of sha1_starts ..................... 35
   8.5 Verification of sha1_process .................... 37
   8.6 Verification of sha1_update ..................... 39
   8.7 Verification of sha1_finish ..................... 41
9. MPI Sub-component Analysis ....................... 44
   9.1 MPI Verification Summary ....................... 44
   9.2 MPI API ........................................ 44
   9.3 MPI Sub-component Integration ................. 44
   9.4 MPI Analysis Stategy ........................... 44
   9.5 Verification of mpi_add_mpi ..................... 48
   9.6 Verification of mpi_sub_mpi ..................... 52
   9.7 Verification of mpi_mul_mpi ..................... 56

PolarSSL 1.1.8 verification kit – V1.0 – Evaluation version 3/116
1. Disclaimer

THIS REPORT IS PROVIDED BY TrustInSoft AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TrustInSoft BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS REPORT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2. Introduction

This first-of-its-kind verification report demonstrates as formally as allowed by current techniques how the PolarSSL software component, in a described configuration, is immune to popular attack families including buffer overflows.

The intended licensees of this document are software integrators and project managers working in industries and organizations where security is critical.

A PolarSSL configuration is described, including the selection of a cryptographic suite and the definition of compile-time options. The description of this configuration is intended to make it easy to integrate PolarSSL as part of a larger system. A plan for the formal verification of the PolarSSL software component thus configured is formulated and implemented. This formal verification has been carried out with TrustInSoft Analyzer, a source code analyzer that relies on state-of-the-art formal verification techniques. All the elements that, taken together, allow to conclude that PolarSSL is safe from a large number of attack families are provided. By themselves, these elements justify the choice of PolarSSL in the described configuration for use in a security-critical context. Together with TrustInSoft Analyzer, these elements can be used to re-check guarantees when slight changes occur: patches applied to the PolarSSL source code, new architectures, changes in the compilation process, etc.

The first three sections of this document provide an overview of the guarantees provided. If you want to compile, configure and use PolarSSL the way it has been verified, see sections 3 and 4. Sections 5 to 9 provide descriptions of the internal specifications and arguments that have been applied to the PolarSSL implementation in order to verify its security.

3. Executive Summary

This report states the total immunity of the SSL server implemented by the PolarSSL 1.1.8 library to the set of security weaknesses enumerated below if it is deployed according to the Secure Deployment Guide detailed in section §4.4.

<table>
<thead>
<tr>
<th>Security Weakness</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
<tr>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>CWE-121</td>
<td>Stack-based Buffer Overflow</td>
</tr>
<tr>
<td>CWE-122</td>
<td>Heap-based Buffer Overflow</td>
</tr>
<tr>
<td>CWE-123</td>
<td>Write-what-where Condition</td>
</tr>
<tr>
<td>CWE-124</td>
<td>Buffer Underwrite ('Buffer Underflow')</td>
</tr>
<tr>
<td>CWE-125</td>
<td>Out-of-bounds Read</td>
</tr>
<tr>
<td>CWE-126</td>
<td>Buffer Over-read</td>
</tr>
</tbody>
</table>
4. Technical Overview

4.1. Component Description

PolarSSL provides an implementation of the TLS (Transport Layer Security) and SSL (Secure Sockets Layer) protocols. These protocols use symmetrical and asymmetrical cryptography to authenticate and ensure the confidentiality and integrity of a point-to-point communication. The protocol is employed amongst other applications in the HTTPS protocol used on the World Wide Web.

<table>
<thead>
<tr>
<th>PolarSSL</th>
<th>Version 1.1.8 with patches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target architecture</td>
<td>IA-32</td>
</tr>
<tr>
<td>Endianness</td>
<td>Little endian</td>
</tr>
<tr>
<td>ABI</td>
<td>GCC/Linux IA-32</td>
</tr>
<tr>
<td>Provider</td>
<td>Offspark B.V.: <a href="https://polarssl.org/">https://polarssl.org/</a></td>
</tr>
<tr>
<td>Copyright holder</td>
<td>Brainspark B.V.</td>
</tr>
<tr>
<td>License</td>
<td>Dual licensing GPL and closed source commercial license</td>
</tr>
<tr>
<td>Pricing policy</td>
<td>Free for GPL version, see website¹ for details on other licenses.</td>
</tr>
</tbody>
</table>

If PolarSSL 1.1.8 is deployed according to the Secure Deployment Guide:

- No extra precaution needs to be taken in order to prevent security breaches exploiting these CWEs,
- No specific mitigation in case of attacks through these CWEs needs to be implemented,
- Security/Penetration testing activities do not need to look for these CWEs,

The conclusions of this report stem from formal methods tools and may contribute to a more general assessment of a complete system:

- In the context of a global risk analysis according to security norms, the evidence provided in this report may be used to demonstrate that state-of-the-art formal methods have been applied to the SSL server and state its robustness as well as its high integrity,
- In the context of a safety critical system that needs to be certified according to safety norms, the conclusions of this report may be integrated as one of the qualification artifacts to state compliance to the highest level of the applicable norms. As this may allow to integrate off-the-shelf Open Source Software, instead of redeveloping an SSL stack from scratch, this may significantly lower the costs of developments of certifiable highly critical systems.
- In the context of a business critical system, the conclusions of this report guarantee that the embedded SSL stack will not be subject to some known Denial-of-Service attacks. This fact can be used to optimize the risk management strategy.

¹[https://polarssl.org/how-to-get#commercial](https://polarssl.org/how-to-get#commercial)
4.2. Scope

The PolarSSL library is one software component of a complete system. It interfaces with the C standard library by calling some of the functions the latter provides. Similarly, an application, such as an HTTPS server, interfaces with PolarSSL by calling functions provided and documented by PolarSSL.

This report does not pretend to provide protection against security flaws originating in the hardware, in the kernel, in the C standard library or in the application. The standard functions used by PolarSSL are listed and the behavior expected from them is described. PolarSSL is only guaranteed to behave as described in this study as long as the standard library functions used behave as expected. A list of the standard functions used by PolarSSL is provided with a formal description of their expected behavior.

![Diagram showing the SSL library in its environment](image)

Figure 1: SSL library in its environment

Similarly, the application must use the PolarSSL library correctly for the library to function. A generic use pattern for PolarSSL is described. PolarSSL is immune to the listed CWEs as long as the application uses it in conformance to the described pattern.

4.3. Synthesis of Analysis

This report states the immunity of the PolarSSL software component to widespread CWEs, provided that PolarSSL is deployed in a context where:

- A single SSL server session is created at a time
- Input/Output buffers passed to the communication functions `ssl_read` and `ssl_write` are properly allocated with a size of 50 bytes
- Server-side certificates are well formed
- The single activated cryptographic suite is SSL_RSA_AES_256_SHA
- The server is configured as to never ask for client-side certificates
- All error codes returned by functions of the API are explicitly tested and properly handled
- API functions `ssl_init`, `ssl_set_rng`, `ssl_set_ciphersuites`, `ssl_set_endpoint`, `ssl_set_bio`, `ssl_set_own_cert`, `ssl_set_session`, `ssl_handshake` are used in sequence in order to initiate the session
- Communication with the client is subsequently done with functions `ssl_read` and `ssl_write` in any order
In order to guarantee the immunity of the PolarSSL library to the listed CWEs, the PolarSSL library has itself been divided in sub-components. For each sub-component, a formal description of its interface is provided. Each of sections 4 through 9 of this report describe the interface of a sub-component of PolarSSL, shows that its implementation corresponds to the interface, and shows that if it relies on another sub-component, it uses that sub-component according to the latter’s interface.

The immunity result is obtained through the application of TrustInSoft Analyzer 1.6 to each sub-component. TrustInSoft Analyzer is a formal verification tool, derivative of the Open-Source Frama-C analysis framework, that emits “alarms” for any risk of a CWE listed in §3. The claim that to each possibility of an error corresponding to one of the listed CWEs corresponds a Frama-C alarm has been evaluated\(^2\) by NIST as part of Frama-C’s participation in SATE V.

All alarms emitted during each analysis are reviewed. The alarms that turn out to correspond to actual security flaws are reported to the maintainer and a patch to fix the PolarSSL library is provided. The alarms that do not correspond to actual risks are justified as such. The justifications can be found in Appendix §A. The security issues identified during this study have been reported to Paul Bakker, principal developer of PolarSSL, and were fixed between PolarSSL versions 1.1.7 and the (currently unreleased) version 1.1.9.

### 4.4. Secure Deployment Guide

#### 4.4.1 Compile-time configuration

PolarSSL’s included sub-components and cryptographic suites are selected at compile-time by editing the file `include/polarssl/config.h`. The analysis in this report is for a PolarSSL 1.1.8 library configured with the following options:

```c
#include/polarssl/config.h

#ifndef POLARSSL_CONFIG_H
#define POLARSSL_CONFIG_H

// SECTION: System support

// portable C implementation for everything
#define POLARSSL_HAVE_LONGLONG

```

4.4.2 Callbacks

PolarSSL provides a generic SSL implementation instantiated for varied uses by providing callback functions (through function pointers). For the server functionality as verified in this study, three functions must be provided, for generating random data, and for receiving and sending data to the peer. The functions are provided at run-time during the set-up phase, before the first connection.

For the results of the verification to guarantee the security of a PolarSSL deployment, the behaviors of the callback functions passed to PolarSSL must be captured by the corresponding generic function used during the verification. The description of the generic functions used during the verification follows. These functions are written with help from Frama-C auxiliary functions described in §B.

Generating random data

A function generating random data must be passed to ssl_set_rng. For the sake of verification the function tis_rng is used as indicated in the source code of the server component:
The function passed to `ssl_set_rng`, when called with arguments `p`, `output` and `output_len`, must set `output_len` consecutive bytes to random values, starting from `output`, and return 0. The argument `p` can be used to hold a context that the function might need. No other visible effects shall be made by this function.

From the point of view of the verification, any function that sets `output_len` bytes from `output` to arbitrary contents works: this is exactly the meaning of the function body provided for `ssl_set_rng` in the analysis context.

```c
int tis_rng(void* p, unsigned char * output, size_t output_len) {
    Frama_C_make_unknown(output, output_len);
    return 0;
}
```

However, in order to ensure the confidentiality and authentication that SSL is supposed to provide in the first place, the function should be a cryptographic-grade random number generator: this property has to be verified by the function provided by the application. Cryptographic properties are not in the scope of this verification kit.

Functions such as `Frama_C_make_unknown`, that have the `Frama_C_` prefix, provide access to internal analyzer functionality; details are in Appendix §B.

**Receiving from the peer**

A function to receive data from the peer must be passed as second argument to `ssl_set_bio`. For the sake of the verification, the function `tis_recv` is used:

```c
int tis_recv(void* p, unsigned char * output, size_t output_len) {
    if (Frama_C_interval(0,1))
        return Frama_C_interval(-1,0);
    size_t r = Frama_C_interval(1, output_len); Frama_C_make_unknown(output, r);
    return r;
}
```

This modelization encompasses all possible messages sent to the SSL server, including **all the malevolent messages** that could possibly be imagined. This means that the results of the verification apply for all these messages.

The application has to provide a function that fulfills the given specification.
Sending to the peer

A function sending data to the peer must be passed as the fourth argument to `ssl_set_bio`. For the sake of the verification, the function `tis_send` is used:

```c
ssl_set_bio( &local_ssl_context, &tis_recv, (void*)0, &tis_send, (void*)0);
```

The function passed, when called with arguments `p`, `output` and `output_len`, can access up to `output_len` consecutive bytes of content, starting from `output`. The function may return `-1` or `output_len` to the exclusion of any other value. The argument `p` can be used to hold a context that the function might need. No other visible effects shall be made by this function.

From the analysis point of view, the body of function `tis_send` encompasses all possible behaviors for such a function:

```c
int tis_send(void* p, const unsigned char* output, size_t output_len) {
    if (Frama_C_interval(0,1))
        return -1;
    return output_len;
}
```

The application has to provide a function that fulfills the given specification.

4.4.3 Patches to apply

The following patches have been applied to the component for the verification. The first patch is intended to make the component easier to analyze, so that its security can be formally guaranteed. The subsequent patches fix issues that are present in the latest 1.1.x PolarSSL release to date, 1.1.8. These issues need to be fixed for the security property to hold.

**MPI: allocate big integers of a fixed size**

This patch simplifies the allocation of multi-precision integers in order to make that sub-component amenable to formal verification. The maximum size chosen for multi-precision integers is 3200 bits, sufficient for RSA-1024 cryptography. The `mpi_grow` function in `bignum.c` must be replaced by the one presented in §9.4.2, and the constant `POLARSSL_MPI_MAX_LIMBS` in `include/polarssl/bignum.h` must be adjusted from 10000 to 100.

**MPI: check for errors when calling `mpi_mod_mpi`**

The patch of `bignum.c` below fixes a lack of validation of a function’s result (CWE-391: Unchecked Error Condition). The called function, `mpi_mod_mpi`, can fail. The call should be wrapped inside the error-handling macro `MPI_CHK` as explained in §A.3.18.

```c
--- ../../../original/library/bignum.c 2013-12-19 10:57:16.770292145 +0100
+++ bignum.patched1.c 2014-03-03 12:56:02.368500307 +0100
@@ -1447,8 +1447,8 @@
+     */
+     if( mpi_cmp_mpi( A, N ) == 0 )
     -     mpi_mod_mpi( &W[I], A, N );
     +     MPI_CHK( mpi_mod_mpi( &W[I], A, N ) );
```
4. Technical Overview

4.4. Secure Deployment Guide

MPI: check for errors in mpi_div_mpi

Two similar potential problems have been identified in mpi_div_mpi. Some unprotected calls to mpi_copy can fail if one of Q or R points to an initialized struct, but the p field of the struct does not point yet to an allocated block. The patch below, for bignum.c, fixes a lack of validation of the results of these function calls (CWE-391: Unchecked Error Condition).

--- ../../../original/library/bignum.c 2013-12-19 10:57:16.770292145 +0100
+++ bignum.patched2.c 2014-03-03 11:53:01.676526755 +0100
@@ -1188,17 +1188,17 @@
  
 if ( Q != NULL )
  {
-  mpi_copy( Q, &Z );
+  MPI_CHK( mpi_copy( Q, &Z ) );

 Q->s = A->s * B->s;
  }

if ( R != NULL )
  {
-  mpi_copy( R, &X );
+  MPI_CHK( mpi_copy( R, &X ) );

 if ( mpi_cmp_int( R, 0 ) == 0 )
  R->s = 1;
  }

Server: fix buffer overflow caused by maliciously crafted message

The patch of ssl_tls.c below fixes buffer overflows that can occur during validation of the message’s padding if a malicious interlocutor sends messages declaring incoherent padding lengths (either a short message declaring an impossibly long padding length, or a long message declaring an impossibly short padding length). In the patch below, the suppression of the nearby empty lines aims to preserve line numbers for the rest of the file.

--- ../../../original/library/ssl_tls.c
+++ ssl_tls.patched.c
@@ -796,10 +796,10 @@
 */
  size_t pad_count = 0, fake_pad_count = 0;
  size_t padding_idx = ssl->in_msglen - padlen - 1;
-  
+  if (padlen == ssl->in_msglen) padding_idx = 0;
+  if ( padding_idx > SSL_MAX_CONTENT_LEN + ssl->maclen) padding_idx = 0;
  for( i = 1; i <= padlen; i++ )
    pad_count += ( ssl->in_msg[padding_idx + i] == padlen - 1 );
for( ; i <= 256; i++)
    fake_pad_count += (ssl->in_msg[padding_idx + i] == padlen - 1);
5. **SSL Server Component Analysis**

5.1. **SSL Server Verification Summary**

This section describes the security analyses results for a generic SSL server using the documented API functions of PolarSSL 1.1.8, configured and patched as in §4.4. Any server implemented with the same logic, detailed in §5.4, is immune to the given list of CWEs (§3).

The “SSL server” sub-component is the topmost sub-component in the overall verification architecture. Its verification relies on the other sub-components’ respective specifications.

In the following table, the notations FV, V and U respectively stand for Formally Verified properties, Verified properties and Unchecked properties. A “Formally Verified” property is guaranteed to hold by one of the formal verification tools used. A “Verified” property is guaranteed to hold by a rigorous argument in natural language. Definitions for these and other abbreviations can be found §C.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>SSL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>ssl_init, ssl_set_rng, ssl_set_cipher_suites, ssl_set_endpoint, ssl_set_bio, ssl_set_own_cert, ssl_set_session, ssl_handshake, ssl_read, ssl_write</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Validated server-side certificates</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>5882/6170</td>
</tr>
<tr>
<td>Sub-components</td>
<td>On-site check: Certificate Parsing</td>
</tr>
<tr>
<td></td>
<td>Checked: AES, MD5, SHA-1, RSA</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>120</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>1</td>
</tr>
<tr>
<td>Required properties</td>
<td>0</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>1/0</td>
</tr>
<tr>
<td>Guaranteed properties (FV/V/U)</td>
<td>0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>6/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memcpy, time, memmove, strlen, memcmp, memset</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>90s</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (everything reviewed)</td>
</tr>
</tbody>
</table>
5. SSL Server Component Analysis

5.2. SSL Server API

The SSL Server component is a typical usage pattern for the PolarSSL library. It is represented by figure 3. The generic implementation is presented in §5.4.

5.3. SSL Server Sub-component Integration

The specifications of the following sub-component functions have been used:

<table>
<thead>
<tr>
<th>Sub-component</th>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Integration Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>rsa_private</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA1</td>
<td>sha1_process</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§10.4.4</td>
<td>✓</td>
</tr>
<tr>
<td>SHA1</td>
<td>sha1_finish</td>
<td>requires finish_r_buffer</td>
<td>§A.2.7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§8.5.4</td>
<td>✓</td>
</tr>
<tr>
<td>SHA1</td>
<td>sha1_update</td>
<td>requires update_r_buffer</td>
<td>§A.2.7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§8.6.4</td>
<td>✓</td>
</tr>
<tr>
<td>MD5</td>
<td>md5_update</td>
<td>requires update_r_buffer</td>
<td>§A.2.7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§6.6.4</td>
<td>✓</td>
</tr>
<tr>
<td>MD5</td>
<td>md5_finish</td>
<td>requires finish_r_buffer</td>
<td>§A.2.7</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>other preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§6.7.4</td>
<td>✓</td>
</tr>
<tr>
<td>AES</td>
<td>aes_crypt_cbc</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§7.4.4</td>
<td>✓</td>
</tr>
</tbody>
</table>
Moreover, the specifications of the following standard library functions (see §B) have been used:

- `malloc`
- `strlen`
- `memset`
- `memcpy`
- `memcmp`
- `time`
- `memmove`

All the preconditions are automatically formally verified, except the property `valid_src` for `memmove`, which is justified by a code review (see §A.2.3).

5.4. SSL Server Analysis Context

The SSL server component is exerted according to figure 3, implemented by the following source code. Any usage pattern explored by this code (represented by figure 3) is guaranteed to be immune to the listed CWEs. In contrast, there may be erroneous usage patterns that are not immune to the CWEs, such as using the library without proper initialization.

The C comment `/*@ slevel 40000 ; */` in the following code is a directive to the analyzer. It has no meaning inside the analysis; rather, its purpose is to help the analyzer use resources appropriately. More generally, any text inside a comment delimited by `/*@` is processed by the analyzer.

```c
server/server.c

ssl_session tis_session={0};
int tis_cyphersuites[] = { SSL_RSA_AES_256_SHA, 0 }; x509_cert tis_cert={0};

int main() {
  rsa_context tis_rsa;
  int ret;
  ssl_context local_ssl_context;
  rsa_init( &tis_rsa, RSA_PKCS_V15, 0 );
  tis_rsa.len = 128;
  tis_rsa.N = make_mpi();
  tis_rsa.E = make_mpi();
  tis_rsa.D = make_mpi();
  tis_rsa.P = make_mpi();
  tis_rsa.Q = make_mpi();
  tis_rsa.DQ = make_mpi();
  tis_rsa.DP = make_mpi();
  tis_rsa.QP = make_mpi();

  int init_result = ssl_init(&local_ssl_context);
  ssl_set_rng(&local_ssl_context, &tis_rng, (void*)0);
  ssl_set_ciphersuites( &local_ssl_context, tis_cyphersuites );
  ssl_set_endpoint( &local_ssl_context, SSL_IS_SERVER );
  ssl_set_bio( &local_ssl_context, &tis_recv, (void*)0, &tis_send, (void*)0);
  ssl_set_own_cert( &local_ssl_context, &tis_cert, &tis_rsa );
  ssl_set_session(&local_ssl_context, 0, 0, &tis_session);
  ret = ssl_handshake(&local_ssl_context);
  if (ret != 0) return ret;
  L1: ;
  while (Frama_C_interval(0,1)) {
    if (Frama_C_interval(0,1)) {
      unsigned char buf[50];
```
5. SSL Server Component Analysis

5.4. SSL Server Analysis Context

```c
/*@ slevel 40000 ; */
ret = ssl_read(&local_ssl_context, buf, 50);
if (ret <= 0) return ret;
/*@ slevel default ; */
}
if (Frama_C_interval(0,1)) {
unsigned char buf[50];
Frama_C_make_unknown(buf, 50);
/*@ slevel 40000 ; */
ret = ssl_write(&local_ssl_context, buf, 50);
if (ret <= 0) return ret;
/*@ slevel default ; */
}
return ret;
}
```

5.5. SSL Server Coverage Analysis

The following table shows the amount of SSL server code that was covered by the analysis. Uncovered statements are dead code and are reviewed to check that no misconfiguration caused security-related code to be omitted from the scope of the verification. The table refers to these abbreviations for justifying the presence of dead code:

- **config-1** stands for "code inactive when RSA-AES-256-SHA suite is selected"
- **config-2** stands for "code inactive because RSA-AES-256-SHA does not feature key exchange"
- **config-3** stands for "code inactive because session resuming is inactive"
- **config-4** stands for "code inactive because analysis is for the server side of the protocol"
- **config-5** stands for "code inactive because no client certificate is requested"

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl_calc_finished</td>
<td>38/38</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_mac_sha1</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_write_server_hello_done</td>
<td>10/10</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_write_change_cipher_spec</td>
<td>11/11</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_parse_change_cipher_spec</td>
<td>17/17</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_flush_output</td>
<td>12/12</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_fetch_input</td>
<td>15/15</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_read_record</td>
<td>87/87</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_hmac</td>
<td>7/7</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_hmac_finish</td>
<td>2/2</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_hmac</td>
<td>5/5</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_hmac_finish</td>
<td>7/7</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_hmac_update</td>
<td>2/2</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>44/44</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>make_mpi</td>
<td>5/5</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>tis_send</td>
<td>6/6</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>tis_recv</td>
<td>10/10</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>tis_rng</td>
<td>3/3</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_handshake</td>
<td>4/4</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>
5.6. SSL Server Reviewed Alarms

A single alarm is raised:

```
assert \valid(output+(0 .. output_len-1));
```

It comes from a call to `Frama_C_make_unknown` in `tis_recv`:

```
# server/server.c
12 size_t r = Frama_C_interval(1, output_len); Frama_C_make_unknown(output, r);
```

The alarm is false since this property is ensured by `tis_recv` precondition `recv_r1` (see §A.2.2):

```
# server/server.acsl
107 requires recv_r1_rv: \valid(output+(0..output_len-1));
```
This precondition, and all other formal specifications in this report, are written in the ANSI/ISO C Specification Language (ACSL).

### 5.7. SSL Server Intermediate Annotations

The following annotations were added to support verification of the server. Their correctness was validated by manual code review; details are in Appendix §A.2.

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl_read</td>
<td>requires rd_r2_rv</td>
<td>§A.2.4</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_read</td>
<td>assert rd_a9_rv</td>
<td>§A.2.5</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_write</td>
<td>ensures wrt_e1_rv</td>
<td>§A.2.6</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_write</td>
<td>ensures wrt_e2_rv</td>
<td>§A.2.6</td>
<td>✓</td>
</tr>
<tr>
<td>tis_recv</td>
<td>requires recv_r1_rv</td>
<td>§A.2.2</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_parse_client_hello</td>
<td>assert spch_a1_rv</td>
<td>§A.2.1</td>
<td>✓</td>
</tr>
</tbody>
</table>
6. MD5 Sub-component Analysis

6.1. MD5 Verification Summary

This section describes the security analyses results for the MD5 sub-component deployed in the context of the SSL Server component. The MD5 sub-component provides an implementation of the eponymous cryptographic hash function.

In this context, this section states that the MD5 sub-component is immune to the given list of CWEs, and that the properties used to validate the server component given by the specification are correct.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>MD5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>md5_starts, md5_update, md5_finish, md5_process</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Used as part of the SSL Server component</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>305/413</td>
</tr>
<tr>
<td>Sub-components</td>
<td>None</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>45</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>18004</td>
</tr>
<tr>
<td>Required properties</td>
<td>17</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Guaranteed properties (F/V/U)</td>
<td>13/0/0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memcpy</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>2 days</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (all alarms reviewed)</td>
</tr>
</tbody>
</table>

6.2. MD5 API

The functions md5_starts, md5_update, and md5_finish offer a standard interface to the hash function. The same interface was for instance required for the NIST hash function competition³ announced in 2007.

The function md5_process is more of an internal function. It is verified separately because it can be called directly from other PolarSSL modules, and in order to facilitate the verification of functions md5_update and md5_finish that call it.

6.3. MD5 Sub-component Integration

The MD5 sub-component does not rely on any other delimited sub-component and uses only memcpy from the standard library.

6.4. Verification of md5_starts

6.4.1 md5_starts Formal Specification

The md5_starts formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
</table>

³http://en.wikipedia.org/wiki/NIST_hash_function_competition
It is defined by:

```c
#include <polarssl/md5.h>
#include <__fc_builtin.h>
#include "md5_spec.h"

int main(){
    md5_context ctx;
    md5_starts( &ctx );
}
```

6.4.2 `md5_starts` Analysis Context

This analysis context is built by the function below. The analysis context is written with help from Frama-C auxiliary functions described in §B.

```c
#include <polarssl/md5.h>
#include <__fc_builtin.h>
#include "md5_spec.h"

int main(){
    md5_context ctx;
    md5_starts( &ctx );
}
```

In this context, `md5_starts`’s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_starts</td>
<td>requires starts_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

Note: the sizes 2 and 4 that appear in the specification of `md5_starts` and in the analysis context above come from the definition of MD5 as using a size counter of two 32-bit words and a state of four 32-bit words (RFC 1321).
6.4.3 md5_starts Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>3/3</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_starts</td>
<td>7/7</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.4.4 md5_starts Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_starts</td>
<td>ensures starts_e_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_starts</td>
<td>ensures starts_e_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.4.5 md5_starts Assigns Properties

The above dependencies are compared to the dependencies computed automatically for `md5_starts`. The automatically computed dependencies below, in which `c` is the name of the array the address of which is passed to `md5_starts`, match the assigns clause in `md5_starts`'s specification.

```c
{from} Function md5_starts:
    c{.total[0..1]; .state[0..3]} FROM indirect: ctx
```

6.4.6 md5_starts Reviewed Alarms

No alarms.

6.4.7 md5_starts Intermediate Annotations

No annotations.

6.5. Verification of md5_process

6.5.1 md5_process Formal Specification

```c
/*@
   requires process_r_ctx: \valid(ctx);
   requires process_r_state: \initialized(ctx->state + (0..3));
   requires process_r_data_valid: \valid(data+(0 .. 63));
   requires process_r_data_init: \initialized(data+(0 .. 63));
   assigns ctx ->state[0..3]
   \from
   // indirect: ctx; data;
   ctx->state[0 .. 3], data[0 .. 63];
*/
```
6. MD5 Sub-component Analysis

6.5. Verification of *md5_process*

```c
// ensures process_e_state: \initialized{ctx->state + (0 .. 3));
/*
void md5_process( md5_context *ctx, const unsigned char data[64] );
*/
```

### 6.5.2 md5_process Analysis Context

This context is built by the function below:

```c
#include "md5_spec.h"
#include "__fc_builtin.h"

main(){
    md5_context c;
    unsigned char d[64];
    int i;
    for (i=0; i < 4; i++)
        c.state[i] = Frama_C_unsigned_int_interval(0, -1U);
    for (i=0; i < 64; i++)
        d[i] = Frama_C_interval(0, 255);
    md5_process( &c, d );
}
```

In this context, `md5_process`'s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_process</td>
<td>requires process_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_process</td>
<td>requires process_r_data_init</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_process</td>
<td>requires process_r_data_valid</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_process</td>
<td>requires process_r_state</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

Note: the sizes 4 and 64 that appear in the specification of `md5_process` and in the analysis context above come from the definition of MD5 as using a state of four 32-bit words and as operating on 64-byte blocks (RFC 1321).

### 6.5.3 md5_process Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>19/19</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_process</td>
<td>233/233</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 6.5.4 md5_process Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_process</td>
<td>ensures process_e_state</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
6.5.5 md5_process Assigns Properties

TIS Analyzer can be used to infer conservative dependencies expressed in terms of the pointed-to blocks c and d. The hand-written dependencies in the assigns clause from §6.5.1 are compared to the dependencies automatically computed for md5_process.

The automatically computed dependencies below, in which c is the name of the array the address of which is passed to md5_process for the formal argument ctx, and d the array the address of which is passed for the formal argument data, match the assigns clause in md5_process’s specification.

```plaintext
[from] Function md5_process:
c.state[0..3] FROM indirect: ctx; data; direct: c.state[0..3]; d[0..63]
```

6.5.6 md5_process Reviewed Alarms

No alarms.

6.5.7 md5_process Intermediate Annotations

No annotations.

6.6. Verification of md5_update

6.6.1 md5_update Formal Specification

```
MD5/md5_update.h

/*@ 
   requires update_r_ctx: \valid(ctx);
   requires update_r_ilen: ilen <= 18000;
   requires update_r_total: \initialized(ctx->total + (0..1));
   requires update_r_state: \initialized(ctx->state + (0..3));
   requires update_r_buffer: \initialized(ctx->buffer + (0..(ctx->total[0] % 64)-1));
   requires update_r_input_valid: \valid_read(input+(0 .. ilen - 1));
   requires update_r_input_init: \initialized(input+(0 .. ilen - 1));
   
assigns
   *ctx FROM // indirect: ctx, input,
      *ctx, ilen, input[0 .. ilen - 1] ;

   ensures update_e_total_val: \initialized(ctx->total + (0..1));
   ensures update_e_state_val: \initialized(ctx->state + (0..3));
   ensures update_e_buffer_rv: \initialized(ctx->buffer + (0..(ctx->total[0] % 64)-1));
*/
void md5_update( md5_context *ctx, const unsigned char *input, size_t ilen );
```
6.6.2 md5_update Analysis Context

This context is built by the function below:

```c
#include <polarssl/md5.h>
#include <__fc_builtin.h>
#include "md5_spec.h"

int main(){
    md5_context tis_ctx;
    Frama_C_make_unknown(&tis_ctx.total, sizeof tis_ctx.total);
    Frama_C_make_unknown(&tis_ctx.state, sizeof tis_ctx.state);
    unsigned char left = Frama_C_interval (0, 63);
    L: Frama_C_make_unknown(&tis_ctx.buffer, left);
    if (tis_ctx.total[0] % 64 != left) {
        return 1;
    }
    unsigned char t[N?N:1];
    Frama_C_make_unknown(t, N);
    md5_update( &tis_ctx, t, N );
    return 0;
}
```

The analysis is done for all values of \( N \) between 0 and 18000. The limit 18000 comes from the fact that SSL and TLS limit messages to 16KiB of useful data, with some padding on top, the maximum quantity of which varies with the exact protocol version (RFC 5246). In other words, this specification of `md5_update` covers all calls coming from other parts of PolarSSL.

```bash
export N=0
while [ "$N" -ne "18001" ] ; do
    echo SIZE:$N
    frama-c -cpp-extra-args=-DN="$N" $BUILTIN $SRC $PP $OPT $OPT1 $*
    export N='expr $N + 1'
done
```

In this context, `md5_update`’s preconditions are valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_update</td>
<td>requires update_r_buffer</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_ilen</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_input_init</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_input_valid</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>requires update_r_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
6. MD5 Sub-component Analysis

6.6. Verification of md5_update

6.6.3 md5_update Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>12/12</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>25/25</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.6.4 md5_update Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_update</td>
<td>ensures update_e_buffer</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>ensures update_e_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_update</td>
<td>ensures update_e_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.6.5 md5_update Assigns Properties

The above dependencies are compared to the dependencies computed automatically for md5_update. For ilen between 1 and 18000, the automatically computed dependencies below, in which c is the name of the array the address of which is passed to md5_update for the formal argument ctx, and t the array the address of which is passed for the formal argument input, match the assigns clause in md5_update’s specification.

```
[from] Function md5_update:
  ctx.total[0] FROM ctx; ilen; ctx{.total[0]; {.state[0..3]; .buffer[0..63]}}; }; t[0..%d]
  .total[1] FROM ctx; ilen; ctx{.total[0..1]; .state[0..3]; .buffer[0..63]}; t[0..%d] (and SELF)
  {.state[0..3]; .buffer[0..63]; .ipad[0..63]; .opad[0..63]} FROM ctx; ilen; ctx{.total[0]; {.state
  [0..3]; .buffer[0..63]}; }; t[0..%d] (and SELF)
```

For the case where ilen is 0, TIS Analyzer determines that md5_update has no effects, which also conforms to the specification written for the function.

6.6.6 md5_update Reviewed Alarms

No alarms.

6.6.7 md5_update Intermediate Annotations

The function md5_update calls md5_process and is verified according to the specification of this function. The pre-conditions of md5_process are formally verified at all call sites.

6.7. Verification of md5_finish

6.7.1 md5_finish Formal Specification

```
MD5/md5_finish.h
extern const unsigned char md5_padding[64];
```
6. MD5 Sub-component Analysis

6.7 Verification of md5_finish

```c
/*@
requires finish_r_ctx: \valid(ctx);
requires finish_r_output: \valid(output + (0..15)) ;
requires finish_r_total: \initialized(ctx->total + (0..1));
requires finish_r_state: \initialized(ctx->state + (0..3));
requires finish_r_buffer: \initialized(ctx->buffer + (0..(ctx->total[0] % 64)-1));
assigns *ctx \from
  // indirect: ctx,
  *ctx, md5_padding[0 .. 63];
assigns output[0 .. 15] \from
  // indirect: ctx, output
  *ctx, md5_padding[0 .. 63];
ensures finish_e_output: \initialized(output+(0..15));
*/
void md5_finish( md5_context *ctx, unsigned char output[16] );
```

6.7.2 md5_finish Analysis Context

This context is built by the function below:

```c
#include <polarssl/md5.h>
#include <__fc_builtin.h>
#include "md5_spec.h"

int main(){
    md5_context c;
    
    Frama_C_make_unknown(&c.total, sizeof c.total);
    Frama_C_make_unknown(&c.state, sizeof c.state);
    
    unsigned char left = Frama_C_interval (0, 63);
    L: Frama_C_make_unknown(&c.buffer, left);
    
    if (c.total[0] % 64 != left) {
        return 1;
    }
    
    unsigned char t[16];
    md5_finish(&c, t);
    return 0;
}
```

In this context, md5_finish’s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_finish</td>
<td>requires finish_r_buffer</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_finish</td>
<td>requires finish_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5_finish</td>
<td>requires finish_r_output</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
6. MD5 Sub-component Analysis

6.7. Verification of \texttt{md5\_finish} Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{md5_finish}</td>
<td>requires finish_r_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>\texttt{md5_finish}</td>
<td>requires finish_r_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.7.3 \texttt{md5\_finish} Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>11/11</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>\texttt{md5_finish}</td>
<td>39/39</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.7.4 \texttt{md5\_finish} Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{md5_finish}</td>
<td>ensures finish_e_output</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

6.7.5 \texttt{md5\_finish} Assigns Properties

The above dependencies are compared to the dependencies computed automatically for \texttt{md5\_finish}. The automatically computed dependencies below, in which \texttt{c} is the name of the array the address of which is passed to \texttt{md5\_finish} for the formal argument \texttt{ctx} and \texttt{t} is the array passed for the formal argument \texttt{output}, match the \texttt{assigns} clause in \texttt{md5\_finish}'s specification.

The function \texttt{md5\_finish} reads from the const-qualified table \texttt{md5\_padding}, that contains the bytes to add at the end of the message in order to make its total length a multiple of the block size.

\begin{verbatim}
[from] Function \texttt{md5\_finish}:
  c FROM indirect: ctx; c.total[0];
    direct: c; md5\_padding\_0[0..63] (and SELF)
  t[0..15] FROM indirect: ctx; output; c.total[0];
    direct: c; md5\_padding\_0[0..63]
\end{verbatim}

6.7.6 \texttt{md5\_finish} Reviewed Alarms

No alarms.

6.7.7 \texttt{md5\_finish} Intermediate Annotations

The function \texttt{md5\_finish} calls \texttt{md5\_update} and is verified according to the specification of this function.

\begin{verbatim}
MD5/md5\_c
171 \texttt{md5\_update( ctx, (unsigned char *) md5\_padding, padn );}
172 \texttt{md5\_update( ctx, msglen, 8 );}
\end{verbatim}
The pre-conditions of `md5_update` are formally verified at the first call site, and all of them, except `update_r_buffer`, are also formally verified at the second call site. The last precondition to verify is:

```c
MD5/md5_update.h
requires update_r_buffer: \initialized{ctx->buffer + (0..(ctx->total[0] % 64)-1)};
```

It is ensured by the postcondition `update_e_buffer` of the first call:

```c
MD5/md5_update.h
ensures update_e_buffer_rv: \initialized{ctx->buffer + (0..(ctx->total[0] % 64)-1)};
```
7. AES Sub-component Analysis

7.1. AES Verification Summary

This section describes the security analyses results for the AES sub-component deployed in the context of the SSL Server component. The AES sub-component handles symmetric cryptography for the exchanges between client and server. The key used has been agreed on through the asymmetric cryptography implemented in the RSA sub-component.

In this context, this section states that the AES sub-component is immune to the given list of CWEs, and that the properties used to validate the server component given by the specification are correct.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>aes_crypt_cbc</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Used as part of the SSL Server component</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>241/670</td>
</tr>
<tr>
<td>Sub-components</td>
<td>None</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>10</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>1100</td>
</tr>
<tr>
<td>Required properties</td>
<td>12</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Guaranteed properties (F/V/U)</td>
<td>2/2/0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memcpy</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>5h</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (all alarms reviewed)</td>
</tr>
</tbody>
</table>

7.2. AES API

The only function studied separately as representing the AES sub-component is aes_crypt_cbc.

7.3. AES Sub-component Integration

The AES sub-component does not rely on any other delimited sub-component and uses only memcpy from the standard library.

7.4. Verification of aes_crypt_cbc

7.4.1 aes_crypt_cbc Formal Specification

This section presents the aes_crypt_cbc function specification that is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by SSL Server</td>
<td>§5.3</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§7.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§7.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§7.4.5</td>
<td>✓</td>
</tr>
</tbody>
</table>
The `aes_crypt_cbc` function encodes information from an input buffer to an output buffer of the same length. The function receives an encryption context `ctx` and initial vector `iv` that contain the key and the information relating to the CBC chaining mode. The function reads from pre-computed, `const`-qualified tables `FSb`, `FT0`, `FT1`, `FT2`, `FT3`, `RSb`, `RT0`, `RT1`, `RT2`, `RT3`.

```c
#include <polarssl/aes.h>
extern const unsigned char FSb[256];
extern const unsigned long FT0[256], FT1[256], FT2[256], FT3[256];
extern const unsigned char RSb[256];
extern const unsigned long RT0[256], RT1[256], RT2[256], RT3[256];

/*@
   requires aes_r_ctx: \valid(ctx);
   requires aes_r_buf: \initialized(ctx->buf + (0 .. 63));
   requires aes_r_rk: ctx->rk == ctx->buf;
   requires aes_r_nr: ctx->nr == 14;
   requires aes_r_mode: mode == 0 || mode == 1;
   requires aes_r_length: 16 <= length <= 16672;
   requires aes_r_length_mod: length % 16 == 0;
   requires aes_r_input_valid: \valid_read(input + (0 .. length-1));
   requires aes_r_input_init: \initialized(input + (0 .. length - 1));
   requires aes_r_output: \valid(output + (0 .. length-1));
assigns output[0 .. length - 1]
   \from
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   Fsb[0..255], FT0[0..255], FT1[0..255], FT2[0..255], FT3[0..255],
   RSb[0..255], RT0[0..255], RT1[0..255], RT2[0..255], RT3[0..255],
   ctx->nr, ctx->buf[0..59], mode, length;
ensures aes_e1:\initialized(output + (0 .. length - 1));
assigns iv[0 .. 15]
   \from
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   // indirect: input, output, ctx, ctx->rk,
   Fsb[0..255], FT0[0..255], FT1[0..255], FT2[0..255], FT3[0..255],
   RSb[0..255], RT0[0..255], RT1[0..255], RT2[0..255], RT3[0..255],
   ctx->nr, ctx->buf[0..59], mode, length;
ensures aes_e2:\initialized(iv + (0 .. 15));
*/
int aes_crypt_cbc( aes_context *ctx,
    int mode,
    size_t length,
    unsigned char iv[16],
```
7. AES Sub-component Analysis

7.4. Verification of aes_crypt_cbc

```c
const unsigned char *input,
unsigned char *output );
```

### 7.4.2 aes_crypt_cbc Analysis Context

AES is analyzed in a context built with the following source code. The analysis context is written with help from Frama-C auxiliary functions described in §B.

#### AES/aes_crypt_cbc.c

```c
#include <polarssl/aes.h>
#include <__fc_builtin.h>
#include "aes_spec.h"

int main(){
    aes_context tis_ctx;
    Frama_C_make_unknown(&tis_ctx.buf, 64 * sizeof(tis_ctx.buf[0]));
    tis_ctx.rk = tis_ctx.buf;
    tis_ctx.nr = 14;

    int mode = Frama_C_interval(0, 1);
    int length = N;

    unsigned char iv[16];
    Frama_C_make_unknown(iv, 16);

    unsigned char input[N];
    Frama_C_make_unknown(input, N);

    unsigned char output[N];
    aes_crypt_cbc(&tis_ctx, mode, length, iv, input, output);
}
```

In this context, all the preconditions of aes_crypt_cbc are formally verified:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_buf</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_input_init</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_input_valid</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_iv_init</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_iv_valid</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_length_mod</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_length</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_mode</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_nr</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_output</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>requires aes_r_rk</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

A manual review ensures that all the input contexts defined by the preconditions are covered.
7. AES Sub-component Analysis

7.4. Verification of aes_crypt_cbc

7.4.3 aes_crypt_cbc Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_crypt_cbc</td>
<td>37/39</td>
<td>94.9%</td>
<td>check for length multiple of 16</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_ecb</td>
<td>194/194</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

7.4.4 aes_crypt_cbc Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_crypt_cbc</td>
<td>ensures aes_e1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>aes_crypt_cbc</td>
<td>ensures aes_e2</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

7.4.5 aes_crypt_cbc Assigns Properties

The assigns properties are verified by checking that the computed dependencies are included in the expected dependencies.

```c
assigns output[0 .. length - 1]
\from input[0 .. length - 1], iv[0 .. 15],
FSb[0..255], FT0[0..255], FT1[0..255],
FT2[0..255], FT3[0..255], RSB[0..255], RT0[0..255],
RT1[0..255], RT2[0..255], RT3[0..255],
ctx.nr, ctx.buf[0..59], mode, length ;
assigns iv[0 .. 15],
\from iv[0 .. 15],
FSb[0..255], FT0[0..255], FT1[0..255],
FT2[0..255], FT3[0..255], RSB[0..255], RT0[0..255],
RT1[0..255], RT2[0..255], RT3[0..255],
ctx.nr, ctx.buf[0..59], mode, length ;
```

The above dependencies are compared to the dependencies computed automatically for each value of length multiple of 16 between 16 and 16672. In each case the automatically computed dependencies are:

```c
[from] Function aes_crypt_cbc:
iv[0..15] FROM indirect: FSb[0..255]; FT0[0..255]; FT1[0..255]; FT2[0..255]; FT3[0..255]; RSb[0..255];
RT0[0..255]; RT1[0..255]; RT2[0..255]; RT3[0..255]; ctx; mode; length; iv; input; output; ctx{.nr; .rk; .buf[0..59]}; iv[0..15]; input[0..%d]; output[0..%d]; direct: FSb[0..255]; RSb[0..255]; output ; ctx.buf[56..59]; iv[0..15]; input[0..%d]; output[0..%d] (and SELF)
output[0..%d] FROM indirect: FSb[0..255]; FT0[0..255]; FT1[0..255]; FT2[0..255]; FT3[0..255]; RSB[0..255];
RT0[0..255]; RT1[0..255]; RT2[0..255]; RT3[0..255]; ctx; mode; length; iv; input; output; ctx{.nr; .rk; .buf[0..59]}; iv[0..15]; input[0..%d]; output[0..%d]; direct: FSb[0..255]; RSB[0..255]; output ; ctx.buf[56..59]; iv[0..15]; input[0..%d]; output[0..%d] (and SELF)
\result FROM length
```

The direct dependencies of output[0..%d] and iv[0..15] towards output[0..%d] are necessarily false positives of the dependency analysis because in the context used, the array output is uninitialized. The analyzer would emit a warning and prevent using the values contained in output if they were effectively accessed.
7. AES Sub-component Analysis

7.4. Verification of aes_crypt_cbc

7.4.6 aes_crypt_cbc Reviewed Alarms
No alarm.

7.4.7 aes_crypt_cbc Intermediate Annotations
No annotation.
8. SHA-1 Sub-component Analysis

8.1. SHA-1 Verification Summary

This section describes the security analyses results for the SHA-1 sub-component deployed in the context of the SSL Server component. The SHA-1 sub-component provides an implementation of the eponymous cryptographic hash function.

In this context, this section states that the SHA-1 sub-component is immune to the given list of CWEs, and that the properties used to validate the server component given by the specification are correct.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>SHA-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>sha1_starts, sha1_update, sha1_finish, sha1_process</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Used as part of the SSL Server component</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>617/737</td>
</tr>
<tr>
<td>Sub-components</td>
<td>None</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>45</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>18004</td>
</tr>
<tr>
<td>Required properties</td>
<td>17</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Guaranteed properties (FV/V/U)</td>
<td>13/0/0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memcpy</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>2 days</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (all alarms reviewed)</td>
</tr>
</tbody>
</table>

8.2. SHA-1 API

The functions sha1_starts, sha1_update, and sha1_finish offer a standard interface to the hash function. The same interface was for instance required for the NIST hash function competition⁴ announced in 2007.

The function sha1_process is more of an internal function. It is verified separately because it can be called directly from other PolarSSL modules, and in order to facilitate the verification of functions sha1_update and sha1_finish that call it.

8.3. SHA-1 Sub-component Integration

The SHA-1 sub-component does not rely on any other delimited sub-component and uses only memcpy from the standard library.

8.4. Verification of sha1_starts

8.4.1 sha1_starts Formal Specification

The sha1_starts formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
</table>

⁴http://en.wikipedia.org/wiki/NIST_hash_function_competition


8. SHA-1 Sub-component Analysis

8.4. Verification of sha1_starts

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by SSL Server</td>
<td>§5.3</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§8.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§8.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§8.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§8.4.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

It is defined by:

```c
#include "sha1_spec.h"

int main(){
    sha1_context c;
    sha1_starts( &c );
}
```

8.4.2 sha1_starts Analysis Context

This context is built by the function below. The analysis context is written with help from Frama-C auxiliary functions described in §B.

```c
#include <polarssl/sha1.h>
#include <__fc_builtin.h>

static void sha1_starts( sha1_context *ctx );

/*@
   requires starts_r: \valid(ctx);
   assigns ctx->total[0..1] \from
   // indirect: ctx,
   \nothing;
   assigns ctx->state[0..4] \from
   // indirect: ctx,
   \nothing;
   ensures starts_e1: \initialized(ctx->total + (0..1));
   ensures starts_e2: \initialized(ctx->state + (0..4));
*/

void sha1_starts( sha1_context *ctx );
```

In this context, sha1_starts’s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_starts</td>
<td>requires starts_r</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.4.3 sha1_starts Coverage Analysis
8. SHA-1 Sub-component Analysis

8.4. Verification of sha1_starts

<table>
<thead>
<tr>
<th>Function</th>
<th>LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>3/3</td>
<td>100.0%</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>sha1_starts</td>
<td>8/8</td>
<td>100.0%</td>
<td>-</td>
<td>✔</td>
</tr>
</tbody>
</table>

8.4.4 sha1_starts Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_starts</td>
<td>ensures starts_e1</td>
<td>formal</td>
<td>✔</td>
</tr>
<tr>
<td>sha1_starts</td>
<td>ensures starts_e2</td>
<td>formal</td>
<td>✔</td>
</tr>
</tbody>
</table>

8.4.5 sha1_starts Assigns Properties

The above dependencies are compared to the dependencies computed automatically for sha1_starts. The automatically computed dependencies below, in which c is the name of the array the address of which is passed to sha1_starts, match the assigns clause in sha1_starts’s specification.

```c
[from] Function sha1_starts:
    c{.total[0..1]; .state[0..4]} FROM indirect: ctx
```

8.4.6 sha1_starts Reviewed Alarms

No alarms.

8.4.7 sha1_starts Intermediate Annotations

No annotations.

8.5. Verification of sha1_process

8.5.1 sha1_process Formal Specification

```c
/*@ 
   requires process_r1: \valid(ctx);
   requires process_r2: \valid_read(data+(0 .. 63));
   requires process_r3: \initialized(ctx->state +(0 .. 4));
   requires process_r4: \initialized(data+(0 .. 63)) ;
   assigns ctx ->state[0..4]
   \from
   // indirect: ctx; data;
   ctx->state[0 .. 4], data[0 .. 63] ;
   ensures process_e: \initialized(ctx->state +(0 .. 4));
*/
void sha1_process( sha1_context *ctx, const unsigned char data[64] );
```
8. SHA-1 Sub-component Analysis

8.5. Verification of sha1_process

8.5.2 sha1_process Analysis Context

This context is built by the function below:

```c
#include "polarssl/sha1.h"
#include "__fc_builtin.h"

main(){
    sha1_context c;
    unsigned char d[64];
    int i;
    for (i=0; i < 5; i++)
        c.state[i] = Frama_C_unsigned_int_interval(0, -1U);
    for (i=0; i < 64; i++)
        d[i] = Frama_C_interval(0, 255);
    sha1_process( &c, d );
}
```

In this context, sha1_process’s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_process</td>
<td>requires process_r1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_process</td>
<td>requires process_r2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_process</td>
<td>requires process_r3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_process</td>
<td>requires process_r4</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.5.3 sha1_process Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>19/19</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_process</td>
<td>539/539</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.5.4 sha1_process Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_process</td>
<td>ensures process_e</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.5.5 sha1_process Assigns Properties

The above dependencies are compared to the dependencies computed automatically for sha1_process. The automatically computed dependencies below, in which c is the name of the array the address of which is passed to sha1_process for the formal argument ctx, and d the array the address of which is passed for the formal argument data, match the assigns clause in sha1_process’s specification.
8. SHA-1 Sub-component Analysis

8.5. Verification of \texttt{sha1_process}

[from] Function \texttt{sha1_process}:
\hspace{1em} \texttt{c.state[0..4]} FROM indirect: \texttt{ctx; data}; direct: \texttt{c.state[0..4]; d[0..63]}

8.5.6 \texttt{sha1_process} Reviewed Alarms

No alarms.

8.5.7 \texttt{sha1_process} Intermediate Annotations

No annotations.

8.6. Verification of \texttt{sha1_update}

8.6.1 \texttt{sha1_update} Formal Specification

\begin{verbatim}
/*@
   requires update_r_ctx: \valid(ctx);
   requires update_r_ilen: ilen <= 18000;
   requires update_r_total: \initialized(ctx->total + (0..1));
   requires update_r_state: \initialized(ctx->state + (0..4));
   requires update_r_input_valid: \valid_read(input+(0 .. ilen - 1));
   requires update_r_input_init: \initialized(input+(0 .. ilen - 1));
   requires update_r_buffer: \initialized(ctx->buffer + (0..(ctx->total[0] \mod 64)-1));
   assigns
   *ctx \from
   // indirect: ctx, input,
   *ctx, ilen, input[0 .. ilen - 1] ;
   ensures update_e_total: \initialized(ctx->total + (0..1));
   ensures update_e_state: \initialized(ctx->state + (0..4));
   ensures update_e_buffer_rv: \initialized(ctx->buffer + (0..(ctx->total[0] \mod 64)-1));
*/

void sha1_update( sha1_context *ctx,
                   const unsigned char *input, size_t ilen );
\end{verbatim}

8.6.2 \texttt{sha1_update} Analysis Context

This context is built by the function below:

\begin{verbatim}
#include <polarssl/sha1.h>
#include <__fc_builtin.h>
#include "sha1_spec.h"

int main(){
    sha1_context tis_ctx;

    Frama_C_make_unknown(&tis_ctx.total, sizeof tis_ctx.total);
    Frama_C_make_unknown(&tis_ctx.state, sizeof tis_ctx.state);
\end{verbatim}
8. SHA-1 Sub-component Analysis

8.6. Verification of sha1_update

```c
unsigned char left = Frama_C_interval (0, 63);
L: Frama_C_make_unknown(&tis_ctx.buffer, left);

if (tis_ctx.total[0] % 64 != left) {
    return 1;
}

unsigned char t[N?N:1];
Frama_C_make_unknown(t, N);

sha1_update( &tis_ctx, t, N );
return 0;
```

The analysis is done for all values of \( N \) between 0 and 18000:

```
export N=0
while [ "$N" -ne "18001" ] ; do
    echo SIZE:$N
    frama-c -cpp-extra-args=-DN="$N" $BUILTIN $SRC $PP $OPT $OPT1 $*
    export N='expr $N + 1'
done
```

In this context, sha1_update’s preconditions are valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_update</td>
<td>requires update_r_buffer</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_ilen</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_input_init</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_input_valid</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>requires update_r_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.6.3 sha1_update Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>12/12</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>25/25</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.6.4 sha1_update Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_update</td>
<td>ensures update_e_buffer_rv</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>ensures update_e_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_update</td>
<td>ensures update_e_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
8.6.5 sha1_update Assigns Properties

The above dependencies are compared to the dependencies computed automatically for sha1_update. For ilen between 1 and 18000, the automatically computed dependencies below, in which c is the name of the array the address of which is passed to sha1_update for the formal argument ctx, and t the array the address of which is passed for the formal argument input, match the assigns clause in sha1_update’s specification.

[from] Function sha1_update:
ctx.total[0] FROM indirect: ctx; ilen; ctx.total[0]; direct: ilen; ctx{.total[0]; {.state[0..4]; .buffer[0..63]}; }; t[0..%d]
.tot[1] FROM indirect: ctx; ilen; ctx.total[0]; direct: ctx{.total[1]; .state[0..4]; .buffer [0..63]]; t[0..%d] (and SELF)
{.state[0..4]; .buffer[0..63]; .ipad[0..63]; .opad[0..63]} FROM indirect: ctx; ilen; ctx.total[0];
direct: ctx{.state[0..4]; .buffer[0..63]; }; t[0..%d] (and SELF)

For the case where ilen is 0, TIS Analyzer determines that sha1_update has no effects, which also conforms to the specification written for the function.

8.6.6 sha1_update Reviewed Alarms

No alarms.

8.6.7 sha1_update Intermediate Annotations

The function sha1_update calls sha1_process and is verified according to the specification of this function. The pre-conditions of sha1_process are formally verified at all call sites.

8.7. Verification of sha1_finish

8.7.1 sha1_finish Formal Specification

```c
SHA-1/sha1_finish.h

extern const unsigned char sha1_padding[64];

/*@
requires finish_r_ctx: \valid(ctx);
requires finish_r_output: \valid(output + (0..19));
requires finish_r_total: \initialized(ctx->total + (0..1));
requires finish_r_state: \initialized(ctx->state + (0..4));
requires finish_r_buffer:
 \initialized(ctx->buffer + (0..(ctx->total[0] % 64)-1));

assigns *ctx \from
 // indirect: ctx,
 *ctx, sha1_padding[0 .. 63];
assigns output[0 .. 19] \from
 // indirect: ctx, output
 *ctx, sha1_padding[0 .. 63];

ensures finish_e: \initialized(output+(0..19)) ; */
void sha1_finish( sha1_context *ctx, unsigned char output[20] );
```
8.7.2  

**sha1_finish Analysis Context**

This context is built by the function below:

```c
#include <polarssl/sha1.h>
#include <__fc_builtin.h>
#include "sha1_spec.h"

int main(){
    sha1_context c;
    Frama_C_make_unknown(&c.total, sizeof c.total);
    Frama_C_make_unknown(&c.state, sizeof c.state);
    unsigned char left = Frama_C_interval (0, 63);
    L: Frama_C_make_unknown(&c.buffer, left);
    if (c.total[0] % 64 != left) {
        return 1;
    }
    unsigned char t[20];
    sha1_finish(&c, t);
    return 0;
}
```

In this context, `sha1_finish`'s precondition is valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1_finish</td>
<td>requires finish_r_buffer</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_finish</td>
<td>requires finish_r_ctx</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_finish</td>
<td>requires finish_r_output</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_finish</td>
<td>requires finish_r_state</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_finish</td>
<td>requires finish_r_total</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.7.3  

**sha1_finish Coverage Analysis**

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>11/11</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>sha1_finish</td>
<td>44/44</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

8.7.4  

**sha1_finish Output Properties**
8.7.5 sha1_finish Assigns Properties

The above dependencies are compared to the dependencies computed automatically for sha1_finish. The automatically computed dependencies below, in which c is the name of the array the address of which is passed to sha1_finish for the formal argument ctx and t is the array passed for the formal argument output, match the assigns clause in sha1_finish’s specification.

The function sha1_finish reads from the const-qualified table sha1_padding, that contains the bytes to add at the end of the message in order to make its total length a multiple of the block size.

```
[from] Function sha1_finish:
  c FROM indirect: ctx; c.total[0]; direct: ctx; c;
  sha1_padding[0..63] (and SELF)
  t[0..19] FROM indirect: ctx; c.total[0]; direct: ctx; output; c;
  sha1_padding[0..63]
```

8.7.6 sha1_finish Reviewed Alarms

No alarms.

8.7.7 sha1_finish Intermediate Annotations

The function sha1_finish calls sha1_update and is verified according to the specification of this function.

```
SHA-1/sha1.c
306  sha1_update( ctx, (unsigned char *) sha1_padding, padn );
307  sha1_update( ctx, msglen, 8 );
```

The pre-conditions of sha1_update are formally verified at the first call site, and all of them, except update_r_buffer, are also formally verified at the second call site. The last precondition to verify is:

```
SHA-1/sha1_update.h
4  requires update_r_buffer: \initialized{(ctx->buffer + (0..(ctx->total[0] % 64)-1))};
```

It is ensured by the postcondition update_e_buffer of the first call:

```
SHA-1/sha1_update.h
15  ensures update_e_buffer_rv: \initialized{(ctx->buffer + (0..(ctx->total[0] % 64)-1))};
```
9. MPI Sub-component Analysis

9.1. MPI Verification Summary

This section describes the security analyses results for the MPI (multi-precision integer library) used in the context of the RSA sub-component. In this context, this section states that the MPI sub-component is immune to the given list of CWEs (§3), and that the properties used to validate the RSA sub-component given by the specification are correct.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>mpi_add_mpi, mpi_sub_mpi, mpi_mul_mpi, mpi_div_mpi, mpi_exp_mod</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Used as part of the &quot;RSA&quot; sub-component</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>1340/2204</td>
</tr>
<tr>
<td>Sub-components</td>
<td>None</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>add:32, sub:32, mul:29, div:24, exp:22</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>add:1, sub:1, mul:10201, div:1, exp:1</td>
</tr>
<tr>
<td>Required properties</td>
<td>add:14, sub:14, mul:15, div:14, exp:19</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>add:0/0, sub:0/0, mul:0/0, div:1/0, exp:1/0</td>
</tr>
<tr>
<td>Guaranteed properties</td>
<td>add:5/5/0, sub:5/5/0, mul:4/5/0, div:4/4/0, exp:2/14/0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>add:4/0, sub:4/0, mul:4/0 div:17/0, exp:17/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memset, memcpy, malloc, free</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>add:5s, sub:5s, mul:3h40, div:30s, exp:1h10</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (everything reviewed)</td>
</tr>
</tbody>
</table>

9.2. MPI API

The verification is done on five of the API functions of the MPI sub-component:

- `mpi_add_mpi`: addition on MPI numbers,
- `mpi_sub_mpi`: subtraction on MPI numbers,
- `mpi_mul_mpi`: multiplication on MPI numbers,
- `mpi_div_mpi`: division on MPI numbers,
- `mpi_exp_mod`: modular exponentiation on MPI numbers.

9.3. MPI Sub-component Integration

The MPI sub-component does not rely on any other sub-component, and uses the following external standard library functions specifications (see §B):

- `malloc`,
- `free`,
- `memset`,
- `memcpy`.

9.4. MPI Analysis Stategy

In the MPI library, the numbers are represented by a three fields structure:

- `.s` represents the sign: -1 for negative numbers, and 1 for positive ones,
9. MPI Sub-component Analysis

9.4. MPI Analysis Strategy

- \( n \) is the size of the number (number of digits).
- \( p \) is an array in which each cell represents a 32-bit digit.

9.4.1 MSD notation

Let us introduce the notation \( \text{MSD}(X) \) that represents the index of the most significant digit defined by:

```plaintext
RSA/msd.acsl
axiomatic MostSignificantDigit {
  // MSD (X) is the number of the most significant digit in X.
  logic int MSD (mpi *X);

  // MSD (X) == 0 means that X == 0.
  // MSD (X) == 1 means that X has only one digit (X[0] != 0 && X[..1] == 0)
  // MSD (X) == n means that X->p[.. n] == 0 and X[n-1] != 0;
  axiom MSD_high_order : \forall mpi * X; X->p[X->n-1 .. MSD(X)] == 0;
  axiom MSD_def : \forall mpi * X; MSD(X) > 0 ==> X->p[MSD(X) - 1] != 0;
}
```

This function is used to simplify the justifications below.

9.4.2 mpi_grow function:

The \text{mpi_grow} function is used by MPI in order to dynamically allocate or reallocate memory to store the numbers up to a size of \text{POLARSSL_MPI_MAX_LIMBS} digits which is fixed to 100.

Fixed-size dynamic allocation

Because of limitations in formal verification tools, this study is done for a modified version of the subcomponent where all the dynamically allocated arrays have a fixed size of 101 elements. The \( n \) field still holds the size that would have been really allocated in the original version.

The verification of the MPI sub-component is made under the assumption that the modified \text{mpi_grow} function is used.

This is the modified version of the function used for analyses:

```plaintext
RSA/mpi_grow_malloc.c
int mpi_grow( mpi *X, size_t nblimbs ) {
  t_uint *p;

  if ( nblimbs > POLARSSL_MPI_MAX_LIMBS )
    return ( POLARSSL_ERR_MPI_MALLOC_FAILED );

  if ( X->n < nblimbs ) {
    if ( X->p == NULL ) {
      int nb_alloc = POLARSSL_MPI_MAX_LIMBS + 1;

      if ( ( p = (t_uint *) malloc( nb_alloc * ciL ) ) == NULL )
        return ( POLARSSL_ERR_MPI_MALLOC_FAILED );
    }

    memset ( p, 0, nb_alloc * ciL );
  }
}
```
9. MPI Sub-component Analysis

9.4. MPI Analysis Strategy

Static allocation model

In order to write specifications for the API functions that can be used instead of the source code when analysing
the RSA sub-component, another model has been used for allocation. The idea is to declare three statically allo-
cated arrays, and to return the address of one of them when `mpi_grow` is called. Because of the non deterministic
choice, the content of the arrays stays imprecise so that it doesn’t modify the computation.

```c
#include <frama-c.h>

static t_uint tis_arr1[POLARSSL_MPI_MAX_LIMBS+1];
static t_uint tis_arr2[POLARSSL_MPI_MAX_LIMBS+1];
static t_uint tis_arr3[POLARSSL_MPI_MAX_LIMBS+1];

static const t_uint* tis_p1 = tis_arr1;
static const t_uint* tis_p2 = tis_arr2;
static const t_uint* tis_p3 = tis_arr3;

/*@ assigns \result \from \nothing;  
    ensures gsa_e1: \result == \null || \result == tis_arr1 || \result == tis_arr2 | | \result == tis_arr3; */
t_uint * static_alloc (void) {
    t_uint *p = NULL;
    if (Frama_C_interval(0, 1)) p = tis_arr1;
    if (Frama_C_interval(0, 1)) p = tis_arr2;
    if (Frama_C_interval(0, 1)) p = tis_arr3;
    return p;
}

int mpi_grow( mpi *X, size_t nblimbs )
{
    t_uint *p;

    if( nblimbs > POLARSSL_MPI_MAX_LIMBS )
        return( POLARSSL_ERR_MPI_MALLOC_FAILED );

    if( X->n < nblimbs ) {
        if ( X->p == NULL ) {
            if ( ( p = static_alloc () ) == NULL )
                return( POLARSSL_ERR_MPI_MALLOC_FAILED );

            Frama_C_memset(p, Frama_C_interval(0, 255),
                          (POLARSSL_MPI_MAX_LIMBS + 1) * sizeof p[0]);
            X->p = p;
        }
    }
    X->n = nblimbs;
}
```
9. MPI Sub-component Analysis

9.4. MPI Analysis Strategy

```c
return( 0 );
}

void mpi_free( mpi *X )
{
    if ( X == NULL )
        return;
    //if (X->p != tis_arr1 && X->p != tis_arr2 && X->p != tis_arr3) free (X->p);
    X->s = 1;
    X->n = 0;
    X->p = NULL;
}
```

9.4.3 Error Management: the MPI_CHK macro

Familiarity with the error management system used in MPI is necessary to understand how the properties are reviewed.

To manage the errors, each function declares a `ret` variable, and a `cleanup` label near the end of the function, and each call is embedded in a MPI_CHK macro:

```c
{
    int ret = 0;
    ...
    MPI_CHK (f (...));
    ...
    cleanup:
    ...
    return (ret);
}
```

which expands into:

```c
ret = f (...);
if (ret != 0) {
    goto cleanup;
}
```

Several execution paths reach the `cleanup` label:

- the direct path, where `ret == 0` and no error occurred,
- other indirect paths through `goto` statements created by MPI_CHK, where `ret != 0` is ensured.

9.4.4 make_mpi function

The `make_mpi` function is a helper that has been added in order to build the context for all the analyses below. It allocates a number of a given size, initializes the sign to either -1 or 1, and initializes all the digits to an undetermined value.

```c
RSA/make_mpi.c
int make_mpi(mpi * n, unsigned int s) {
    int ret = 0;
    mpi_init (n);
    return ret;
}
```
9.5. Verification of mpi_add_mpi

9.5.1 mpi_add_mpi Formal Specification

The mpi_add_mpi formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by RSA</td>
<td>§10.3</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§9.5.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§9.5.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§9.5.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§9.5.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

It is defined by:

```acsl
// int mpi_add_mpi( mpi *X, const mpi *A, const mpi *B );
function mpi_add mpi:
contract:
  requires add_rX1: !valid(X) ;
  requires add_rX2: 0 <= X->n <= mpi_max_limbs ;
  requires add_rX3: -1 == X->s || 1 == X->s ;
  requires add_rA4: X->p == 0 || !valid_read(X->p + (0 .. mpi_max_limbs - 1));
  requires add_rA6: A->p == 0 || X->p == tis_arr1 || X->p == tis_ arr2 || X->p == tis_arr3 ; // for RSA
  requires add_rA1: !valid_read(A) ;
  requires add_rA2: 0 <= A->n <= mpi_max_limbs ;
  requires add_rA3: -1 == A->s || 1 == A->s ;
  requires add_rA4: !valid_read(A->p + (0 .. mpi_max_limbs - 1)) ;
  requires add_rA5: !initialized(A->p + (0 .. mpi_max_limbs - 1));
  requires add_rB1: !valid_read(B) ;
  requires add_rB2: 1 <= B->n <= mpi_max_limbs ;
  requires add_rB3: -1 == B->s || 1 == B->s ;
  requires add_rB4: !valid_read(B->p + (0 .. mpi_max_limbs - 1));
  requires add_rB5: !initialized(B->p + (0 .. mpi_max_limbs - 1));
  assigns X->s \from A->s, B->s , // A, A->p, B, B->p, X
      A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
  assigns X->n \from X->n, A->n, B->n, // A, A->p, B, B->p, X
      A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
  assigns X->p \from Frama_C_entropy_source;
```

MCP CHK ( mpi_grow (n, s) );

n->s = Frama_C_nondet (-1, 1);

Frama_C_make unknown(n->p, s * sizeof(t_uint));

cleanup:
  return ret;
}
9. MPI Sub-component Analysis

9.5. Verification of mpi_add_mpi

```c
// tis_p1, tis_p2, tis_p3, // for RSA
A->n, A->s, A->p[0 .. mpi_max_limbs - 1], // A, A->p
B->n, B->s, B->p[0 .. mpi_max_limbs - 1], // B, B->p
X, X->p, X->n;
assigns X->p[0 .. mpi_max_limbs - 1] \from Frama_C_entropy_source, // A, B, X
A->s, A->p[0 .. mpi_max_limbs - 1], // A->p
B->s, B->p[0 .. mpi_max_limbs - 1], // B->p
X->p[0 .. mpi_max_limbs - 1];
assigns \result \from Frama_C_entropy_source, X->n, X->p, // X
A->n, A->s, B->n, B->s, // A, A->p, B, B->p
A->p[0 .. mpi_max_limbs - 1],
B->p[0 .. mpi_max_limbs - 1];

ensures add_e1_val: -1 == X->s || 1 == X->s;
ensures add_e2_val: \result == 0 || \result == tis_POLARSSL_ERR_MPI_MALLOC_FAILED
|| \result == tis_POLARSSL_ERR_MPI_NEGATIVE_VALUE;
ensures add_e3_val: \result == 0 || X->n <= mpi_max_limbs;
ensures add_e4_val: \result == 0 ==> X->n > 0 ==> valid(X->p + (0 .. mpi_max_limbs - 1));
\// ensures add_e6_val: \result == 0 ==> X->p == \valid(tis_arr1 || X->p == tis_arr2 || X->p == tis_arr3; //
\// for RSA
```

9.5.2 mpi_add_mpi Analysis Context

The function takes three arguments:

```c
int mpi_add_mpi( mpi *X, const mpi *A, const mpi *B )
```

X is the result, and A and B the input numbers.

The analysis is done with:

- size of the input numbers between 1 and 100,
- positive or negative sign,
- any possible (initialized) values,
- X may be:
  - initialized, but not pre-allocated,
  - allocated,
  - equal to A or B.

This context is built by the function below. The analysis context is written with help from Frama-C auxiliary functions described in §B.

```c
RSA/mpi_add_mpi.c
```

```c
#include "polarssl/bignum.h"
#include "__fc_builtin.h"
#include "make_mpi.c"

t_uint tis_A [POLARSSL_MPI_MAX_LIMBS];
t_uint tis_B [POLARSSL_MPI_MAX_LIMBS];

int main(){
    int ret = 0;

    mpi a; a.s = Frama_C_nondet (-1, 1); a.p = &tis_A;
a.n = Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS);
Frama_C_make_unknown(tis_A, a.n * sizeof(t_uint));
```
9. MPI Sub-component Analysis

9.5. Verification of mpi_add_mpi

```c
mpi b; b.s = Frama_C_nondet(-1, 1); b.p = &tis_B;
b.n = Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS);
Frama_C_make_unknown(tis_B, b.n * sizeof(t_uint));

if (Frama_C_nondet(0, 1)) {
    mpi x0; mpi_init(&x0);
    mpi_add_mpi(&x0, &a, &b);
} else if (Frama_C_nondet(0, 1)) {
    mpi xa;
    MPI_CHK(make_mpi(&xa, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    mpi_add_mpi(&xa, &xa, &b);
} else if (Frama_C_nondet(0, 1)) {
    mpi xb;
    MPI_CHK(make_mpi(&xb, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    mpi_add_mpi(&xb, &xa, &xb);
} else {
    mpi x1;
    MPI_CHK(make_mpi(&x1, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    mpi_add_mpi(&x1, &a, &b);
}
cleanup:
return ret;
```

In this context, mpi_add_mpi preconditions are all valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rX1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rX2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rX3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rX4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_addmpi</td>
<td>requires add_rA1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rA2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rA3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rA4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rB1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rB2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rB3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>requires add_rB4</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.5.3 mpi_add_mpi Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_hlp</td>
<td>27/27</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>44/44</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>make_mpi</td>
<td>9/9</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>
9. MPI Sub-component Analysis

9.5 Verification of mpi_add_mpi

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_add_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_sub_abs</td>
<td>28/28</td>
<td>100.0%</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>39/39</td>
<td>100.0%</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_grow</td>
<td>15/15</td>
<td>100%</td>
<td>-</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_abs</td>
<td>55/56</td>
<td>96.4%</td>
<td>mpi_grow cannot fail here</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>20/22</td>
<td>90.5%</td>
<td>always different arguments</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_free</td>
<td>8/9</td>
<td>88.9%</td>
<td>non null argument</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_init</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✔</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_add_mpi</td>
<td>ensures add_e1</td>
<td>formal</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>ensures add_e2</td>
<td>formal</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>ensures add_e3</td>
<td>formal</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>ensures add_e4</td>
<td>formal</td>
<td>✔</td>
</tr>
</tbody>
</table>

9.5.4 mpi_add_mpi Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_add_mpi</td>
<td>assigns X-&gt;s</td>
<td>reviewed below</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>assigns X-&gt;n</td>
<td>reviewed below</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>assigns X-&gt;p</td>
<td>reviewed below</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>assigns \result</td>
<td>reviewed below</td>
<td>✔</td>
</tr>
</tbody>
</table>

9.5.5 mpi_add_mpi Assigns Properties

The assigns properties in the specification have been over-approximated to make them better fit the dependencies computed during the analysis. However, indirect dependencies to pointers have to be commented out in the specification to avoid to much imprecision in the current version of the analyzer.

The match between the specified assigns properties and the computed dependencies has been verified.

9.5.6 mpi_add_mpi Reviewed Alarms

No alarm.

9.5.7 mpi_add_mpi Intermediate Annotations

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_hlp</td>
<td>loop invariant subh_l2_1</td>
<td>§A.3.26</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>ensures cmpa_e1</td>
<td>§A.3.1</td>
<td>✔</td>
</tr>
</tbody>
</table>
9.6. Verification of mpi_sub_mpi

9.6.1 mpi_sub_mpi Formal Specification

The mpi_sub_mpi formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by RSA</td>
<td>§10.3</td>
<td></td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§9.6.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§9.6.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§9.6.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§9.6.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

It is defined by:

```plaintext
RSA/mpi_sub_mpi.acsl
// int mpi_sub_mpi( mpi *X, const mpi *A, const mpi *B );
function mpi_sub_mpi:
  contract:
  requires sub_rX1: \valid(X) ;
  requires sub_rX2: 0 <= X->n <= mpi_max_limbs ;
  requires sub_rX3: -1 == X->s || 1 == X->s;
  requires sub_rX4: X->p == 0 || \valid_read(X->p + (0 .. mpi_max_limbs - 1));
  // requires sub_rX6: X->p == 0 || X->p == tis_arr1 || X->p == tis_arr2 || X->p == tis_arr3; // for RSA
  requires sub_rA1: \valid_read(A) ;
  requires sub_rA2: 0 <= A->n <= mpi_max_limbs ;
  requires sub_rA3: -1 == A->s || 1 == A->s;
  requires sub_rA4: \valid_read(A->p + (0 .. mpi_max_limbs - 1));
  requires sub_rA5: \initialized(A->p + (0 .. mpi_max_limbs - 1));
  requires sub_rB1: \valid_read(B) ;
  requires sub_rB2: 1 <= B->n <= mpi_max_limbs ;
  requires sub_rB3: -1 == B->s || 1 == B->s;
  requires sub_rB4: \valid_read(B->p + (0 .. mpi_max_limbs - 1));
  requires sub_rB5: \initialized(B->p + (0 .. mpi_max_limbs - 1));

  assigns X->s \from A->s, B->s, // A, A->p, B, B->p, X
  A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
  assigns X->n \from X->n, A->n, B->n, // A, A->p, B, B->p, X
  A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
  assigns X->p \from Frama_C_entropy_source,
  // tis_p1, tis_p2, tis_p3, // for RSA
  A->n, A->s, A->p[0 .. mpi_max_limbs - 1], // A, A->p
  B->n, B->s, B->p[0 .. mpi_max_limbs - 1], // B, B->p
  X, X->p, X->n;

// mpi_sub_mpi(X, A, B)
```
9.6.2 mpi_sub_mpi Analysis Context

The function takes three arguments:

```c
int mpi_sub_mpi( mpi *X, const mpi *A, const mpi *B )
```

X is the result, and A and B the input numbers.

The analysis is done with:

- size of the input numbers between 1 and 100,
- positive or negative sign,
- any possible (initialized) values,
- X may be:
  - initialized, but not pre-allocated,
  - allocated,
  - equal to A or B.

This context is built by the function below:
In this context, mpi_sub_mpi preconditions are all valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rX1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rX2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rX3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rX4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rB1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rB2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rB3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rB4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>requires sub_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

### 9.6.3 mpi_sub_mpi Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_hlp</td>
<td>27/27</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>44/44</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>make_mpi</td>
<td>9/9</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_abs</td>
<td>28/28</td>
<td>100%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>39/39</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_grow</td>
<td>15/15</td>
<td>100%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_abs</td>
<td>55/56</td>
<td>96.4%</td>
<td>mpi_grow cannot fail here</td>
<td>✓</td>
</tr>
</tbody>
</table>
9. MPI Sub-component Analysis

9.6 Verification of mpi_sub_mpi

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_copy</td>
<td>20/22</td>
<td>90.5%</td>
<td>always different arguments</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_free</td>
<td>8/9</td>
<td>88.9%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_init</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.6.4 mpi_sub_mpi Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_mpi</td>
<td>ensures sub_e1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>ensures sub_e2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>ensures sub_e3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>ensures sub_e4</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.6.5 mpi_sub_mpi Assigns Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_mpi</td>
<td>assigns X-&gt;s</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>assigns X-&gt;n</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>assigns X-&gt;p</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>assigns X-&gt;p[0 .. mpi_max_limbs - 1]</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>assigns\result</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
</tbody>
</table>

The same verification strategy as for mpi_add_mpi (§9.5.5) has been applied. The match between the specified assigns properties and the computed dependencies has been verified.

9.6.6 mpi_sub_mpi Reviewed Alarms

No alarm.

9.6.7 mpi_sub_mpi Intermediate Annotations

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_sub_hlp</td>
<td>loop invariant subh_l2_1</td>
<td>§A.3.26</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>ensures cmpa_e1</td>
<td>§A.3.1</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e3</td>
<td>§A.3.2</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e4</td>
<td>§A.3.3</td>
<td>✓</td>
</tr>
</tbody>
</table>
9.7. Verification of mpi_mul_mpi

9.7.1 mpi_mul_mpi Formal Specification

The mpi_mul_mpi formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by RSA</td>
<td>§10.3</td>
<td>✔</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§9.7.2</td>
<td>✔</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§9.7.2</td>
<td>✔</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§9.7.5</td>
<td>✔</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§9.7.4</td>
<td>✔</td>
</tr>
</tbody>
</table>

It is defined by:

```acsl
// int mpi_mul_mpi( mpi *X, const mpi *A, const mpi *B );
function mpi_mul_mpi:
    contract:
        requires mul_rX1: \valid(X) ;
        requires mul_rX2: 0 <= X->n <= mpi_max_limbs ;
        requires mul_rX3: -1 == X->s || 1 == X->s ;
        requires mul_rX4: X->p == 0 || \valid(X->p + (0 .. mpi_max_limbs - 1));
        // FALSE for mpi_div_mpi: requires mul_rX5: X != A;
        requires mul_rX6: X->p == tis_arr1 || X->p == tis_arr2 || X->p == tis_arr3; // for RSA
        requires mul_rA1: \valid_read(A) ;
        requires mul_rA2: 0 <= A->n <= mpi_max_limbs ;
        requires mul_rA3: -1 == A->s || 1 == A->s ;
        requires mul_rA4: \valid_read(A->p + (0 .. mpi_max_limbs - 1));
        requires mul_rA5: \initialized(A->p + (0 .. mpi_max_limbs - 1));
        requires mul_rB1: \valid_read(B) ;
        requires mul_rB2: 1 <= B->n <= mpi_max_limbs ;
        requires mul_rB3: -1 == B->s || 1 == B->s ;
        requires mul_rB4: \valid_read(B->p + (0 .. B->n - 1));
        requires mul_rB5: \initialized(B->p + (0 .. B->n - 1));

    assigns X->s from A->s, B->s, A->n, B->n, // A, B, X, A->p, B->p, X->p,
        A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
    assigns X->n from A->n, A->p, B->n, // X, A, B, A->p, B->p, X->p,
        A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
    assigns X->p from A->n, B->n, X->n, // X, A, B, A->p, B->p,
        // tis_p1, tis_p2, tis_p3; // for RSA
        A->p[0 .. mpi_max_limbs - 1],
        B->p[0 .. mpi_max_limbs - 1],
        X->p;
    assigns X->p[0 .. mpi_max_limbs - 1] from // X
        A->n, A->p[0 .. mpi_max_limbs - 1], // A, A->p
        B->n, B->p[0 .. mpi_max_limbs - 1]; // B, B->p
    assigns \result from Frama_C_entropy_source; // X, X->p,
        A->n, A->p[0 .. mpi_max_limbs - 1], // A, A->p
        B->n, B->p[0 .. mpi_max_limbs - 1]; // B, B->p
```

56/116 Evaluation version – PolarSSL 1.1.8 verification kit – V1.0
9. MPI Sub-component Analysis

9.7. Verification of mpi_mul_mpi

ensures mul_e1_val:
\[ \text{\texttt{result}} == 0 \] || \[ \text{\texttt{result}} == \text{tis\_POLARSSL:\_ERR\_MPI\_Malloc\_Failed}; \]

ensures mul_e2_val:
\[ \text{\texttt{result}} == 0 \] ==\> \[ 1 <= X->n \] ==\> mpi_max_limbs;

ensures mul_e3_val:
\[ -1 == X->s \] || \[ 1 == X->s; \]
// ensures mul_e4_val:
\[ X->p == \text{tis\_arr1} \] || \[ X->p == \text{tis\_arr2} \] || \[ X->p == \text{tis\_arr3}; \] // for RSA

9.7.2 mpi_mul_mpi Analysis Context

The function takes three arguments:

```c
int mpi_mul_mpi( mpi *X, const mpi *A, const mpi *B )
```

X is the result, and A and B the input numbers.

To study mpi_mul_mpi, it is important to notice that this operation first computes the index of the most significant digit (MSD) of both arguments A and B:

```c
for( i = A->n; i > 0; i-- )
    if( A->p[i - 1] != 0 )
        break;

L_mul_a1 : //@ assert mul_a1: i == TIS_ia;
for( j = B->n; j > 0; j-- )
    if( B->p[j - 1] != 0 )
        break;

L_mul_a2 : //@ assert mul_a2: j == TIS_ib;
```

Separate analyses are done for each possible combination of \( 0 \leq TIS\_ia \leq 100 \) and \( 0 \leq TIS\_ib \leq 100 \), so there are 10201 analyses. The size of the input numbers is set to an undetermined value between the index and 100.

The output number X may be:
- initialized, but not pre-allocated,
- allocated,
- equal to A (the precondition mul_rXB ensures that X is never B).

This context is built by the function below:

```c
#include "polarssl/bignum.h"
#include "__fc_builtin.h"
#include "make_mpi.c"

/* Variables to store the index of the last null element starting from the end
* for A and B.
* \( i == 0 \) means that all the elements are \( 0 \).
* \( i == sz \) means that A[sz-1] != \( 0 \).
*/
t_uint TIS_ia, TIS_ib;

t_uint tis_A [POLARSSL_MPI_MAX_LIMBS];
t_uint tis_B [POLARSSL_MPI_MAX_LIMBS];

//@
ensures init_e1: \( \text{\texttt{result}} == 0 \) ==\> \[ i > 0 \] ==\> \[ p->p[i-1] != 0; \]
9. MPI Sub-component Analysis

9.7. Verification of mpi_mul_mpi

```c
// ensures init_e2: \result == 0 ==> MSD (p) == i;
* /
int init (mpi * p, t_uint * arr, t_uint i) {
    size_t sz = Frama_C_interval (i ? i : 1, POLARSSL_MPI_MAX_LIMBS);
    //@ assert init_a1: (0 <= i <= sz);
    p->s = Frama_C_nondet (-1, 1);
    p->p = arr;
    p->n = sz;

    if (i > 0) { // some non null elements
        Frama_C_make_unknown(p->p + i-1, sizeof(t_uint)); // first non null element
        if ( p->p[i-1] == 0) return -1; // exclude null element here
    }

    if (i < sz) { // null elements between i and sz-1
        Frama_C_memset(p->p+i, 0, (sz-i) * sizeof(t_uint));
    }

    return 0;
}

int main (void) {
    int ret;
    mpi a; mpi_init (&a);
    mpi b; mpi_init (&b);
    TIS_ia = TIS_PARAM_ia; // from 0 to POLARSSL_MPI_MAX_LIMBS included
    TIS_ib = TIS_PARAM_ib; // from 0 to POLARSSL_MPI_MAX_LIMBS included

    MPI_CHK ( init (&a, tis_A, TIS_ia) );
    MPI_CHK ( init (&b, tis_B, TIS_ib) );

    if (Frama_C_nondet (0, 1)) {
        mpi x; mpi_init(&x);
        MPI_CHK ( mpi_mul_mpi (&x, &a, &b) );
    }
    else if (Frama_C_nondet (0, 1)) {
        mpi xa;
        MPI_CHK (make_mpi(&xa, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
        MPI_CHK ( mpi_mul_mpi (&xa, &a, &b) );
    }
    else {
        mpi x;
        MPI_CHK (make_mpi(&x, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
        MPI_CHK ( mpi_mul_mpi (&x, &a, &b) );
    }

    cleanup:
    return ret;
}
```

The parameters TIS_PARAM_ia and TIS_PARAM_ib are inputs from the calling context.

In this context, mpi_mul_mpi preconditions are all valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rX1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rX2</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
9. MPI Sub-component Analysis

9.7. Verification of mpi_mul_mpi

### Justification

#### Validation

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rX3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rX4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rX5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rB1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rB2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rB3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rB4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>requires mul_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

### Coverage Analysis

#### Function

#### # LOC | Coverage | Review | Validation

| mpi_mul_mpi | 404/404 | 100.0% | - | ✓ |
| main        | 35/35   | 100.0% | - | ✓ |
| make_mpi    | 9/9     | 100.0% | - | ✓ |
| mpi_grow    | 15/15   | 100.0% | - | ✓ |
| mpi_mul_mpi | 48/52   | 92.3%  | X != B and lset cannot fail | ✓ |
| mpi_copy    | 21/23   | 91.3%  | always different arguments | ✓ |
| mpi_free    | 8/9     | 88.9%  | non null argument | ✓ |
| mpi_lset    | 9/12    | 75.0%  | always called with z==0 | ✓ |
| mpi_init    | 5/6     | 83.3%  | non null argument | ✓ |

---

### Output Properties

#### Function

#### Properties | Justification | Validation

| mpi_mul_mpi | ensures mul_e1 | formal | ✓ |
| mpi_mul_mpi | ensures mul_e2 | formal | ✓ |
| mpi_mul_mpi | ensures mul_e3 | formal | ✓ |

---

### Assigns Properties

#### Function

#### Property | Justification | Validation

| mpi_mul_mpi | assigns X->s | reviewed below | ✓ |
| mpi_mul_mpi | assigns X->n | reviewed below | ✓ |
| mpi_mul_mpi | assigns X->p | reviewed below | ✓ |
9. MPI Sub-component Analysis

9.8. Verification of mpi_div_mpi

The same verification strategy as for mpi_add_mpi (§9.5.5) has been applied. The match between the specified assigns properties and the computed dependencies has been verified.

9.7.6 mpi_mul_mpi Reviewed Alarms

No alarms.

9.7.7 mpi_mul_mpi Intermediate Annotations

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_mul_mpi</td>
<td>assigns X-&gt;p[0 .. mpi_max_limbs - 1]</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>assigns \result</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.8. Verification of mpi_div_mpi

9.8.1 mpi_div_mpi Formal Specification

The mpi_div_mpi formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by RSA</td>
<td>§10.3</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§9.8.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§9.8.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§9.8.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§9.8.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

It is defined by:

```c
// int mpi_div_mpi( mpi *Q, mpi *R, const mpi *A, const mpi *B );
function mpi_div_mpi:
  contract:
    requires div_rQ1: Q == (mpi*) 0 ;
    requires div_rA1: \valid(R);
    requires div_rA2: \valid(R->p + (0 .. mpi_max_limbs - 1));
  // requires div_rR6: R->p == tis_arr1 || R->p == tis_arr2 || R->p == tis_arr3; // for RSA
    requires div_rA1: \valid_read(A) ;
```
9. MPI Sub-component Analysis

9.8. Verification of `mpi_div_mpi`

```c
requires div_rA2: 1 <= A->n <= mpi_max_limbs ;
requires div_rA3: -1 == A->s || 1 == A->s;
requires div_rA4: valid_read(A->p + (0 .. mpi_max_limbs - 1));
requires div_rA5: initialized(A->p + (0 .. mpi_max_limbs - 1));
requires div_rB1: valid_read(B);
requires div_rB2: 1 <= B->n <= mpi_max_limbs ;
requires div_rB3: -1 == B->s || 1 == B->s;
requires div_rB4: valid_read(B->p + (0 .. mpi_max_limbs - 1));
requires div_rB5: initialized(B->p + (0 .. mpi_max_limbs - 1));
assigns R->s from A->s, B->s, A->n, B->n,
// A, B, R, Q, A->p, B->p, R->p,
// A->p[0 .. mpi_max_limbs - 1], B->p[0 .. mpi_max_limbs - 1];
assigns R->n from A->n, B->n, A->s, B->s,
// A->n, A->s, A->p[0 .. mpi_max_limbs - 1], // A, A->p
// B->n, B->s, B->p[0 .. mpi_max_limbs - 1]; // B, B->p
assigns \result from Frama_C_entropy_source, // A, B, R, Q, A->p, B->p, R->p,
// A->n, A->s, A->p[0 .. mpi_max_limbs - 1], // A, A->p
// B->n, B->s, B->p[0 .. mpi_max_limbs - 1]; // B, B->p
ensures div_e1_val: \result == 0 
  || \result == tis_POLARSSL_ERR_MPI_MALLOC_FAILED 
  || \result == tis_POLARSSL_ERR_MPI_DIVISION_BY_ZERO 
  || \result == tis_POLARSSL_ERR_MPI_NEGATIVE_VALUE;
ensures div_eR1_val: \result == 0 ==> 1 <= R->n <= mpi_max_limbs ;
ensures div_eR2_val: \result == 0 ==> -1 == R->s || 1 == R->s;
// ensures div_eR6_val: \result == 0 ==> R->p == tis_arr1 || R->p == tis_arr2 || R->p == tis_arr3; // for RSA
at L_div_3: assert div_a3_val: 0 <= A->n <= 98 || A->n > 98;
at L_div_2: assert div_a2_rv: MSD (B) == Y.n;
at L_div_1: assert div_a1_rv: X.n >= Y.n;
at L_div_aux1_1: assert div_aux1_1_rv: 0 <= aux1 < mpi_max_limbs;
at loop 1: loop invariant div_l1_aux1_2_rv: aux1 == n - t;
at L_div_aux1_3: assert div_aux1_3_rv: aux1 == n - t;
at L_div_aux1_4: assert div_aux1_4_rv: aux1 == n - t;
at L_div_a5: assert div_a5_rv: 0 < aux2 < mpi_max_limbs;
at L_div_aux2_0: assert div_aux2_0_rv: aux2 == i - t;
at L_div_aux2_1: assert div_aux2_1_sc: aux2 == i - t;
at L_div_aux2_2: assert div_aux2_2_sc: aux2 == i - t;
at L_div_aux2_3: assert div_aux2_3_sc: aux2 == i - t;
at L_div_aux2_4: assert div_aux2_4_sc: aux2 == i - t;
at L_div_aux2_5: assert div_aux2_5_sc: aux2 == i - t;
at L_div_aux2_6: assert div_aux2_6_sc: aux2 == i - t;
at L_div_aux2_7: assert div_aux2_7_sc: aux2 == i - t;
at L_div_aux2_8: assert div_aux2_8_sc: aux2 == i - t;
at L_div_a6: assert div_a6_rv: Y.p[t] != 0;
```

9.8.2 `mpi_div_mpi` Analysis Context

The function takes four arguments:

```c
int mpi_div_mpi( mpi *Q, mpi *R, const mpi *A, const mpi *B )
```
Q and R are the results, and A and B the input numbers.

In the context of rsa_private, mpi_div_mpi is only called from mpi_mod_mpi with Q always NULL and R comes from the result of mpi_sub_mpi so it is already allocated.

So the analysis is done with:

- both input numbers:
  - size between 1 and 100,
  - positive or negative sign,
  - any possible (initialized) values,
- Q is always NULL,
- R is always allocated.

The conditions above match the preconditions.

This context is built by the function below:

```c
#include "polarssl/bignum.h"
#include "__fc_builtin.h"
#include "make_mpi.c"

int main(){
    int ret = 0;
    mpi a; MPI_CHK (make_mpi(&a, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    mpi b; MPI_CHK (make_mpi(&b, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    mpi r; MPI_CHK (make_mpi(&r, Frama_C_interval(1, POLARSSL_MPI_MAX_LIMBS)));
    MPI_CHK (mpi_div_mpi(NULL, &r, &a, &b));
    cleanup:
    return ret;
}
```

In this context, mpi_div_mpi preconditions are all valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rA1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rA2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rA3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rA4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rA5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rB1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rB2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rB3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rB4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rB5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rQ1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rR1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rR2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>requires div_rR6</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.8.3 mpi_div_mpi Coverage Analysis
9.8. Verification of \texttt{mpi\_div\_mpi}

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_mul_hlp</td>
<td>404/404</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_hlp</td>
<td>24/24</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>24/24</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>make_mpi</td>
<td>9/9</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>static_alloc</td>
<td>11/11</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_int</td>
<td>6/6</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>40/40</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_shift_r</td>
<td>38/38</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_shift_l</td>
<td>41/41</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_msb</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_grow</td>
<td>17/17</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_abs</td>
<td>27/28</td>
<td>96.4%</td>
<td>mpi_copy ((X,A)) cannot fail here.</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_mpi</td>
<td>45/47</td>
<td>95.7%</td>
<td>(Y) always positive.</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>21/23</td>
<td>91.3%</td>
<td>never (X == Y)</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>136/149</td>
<td>91.3%</td>
<td>(Q == \null)</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_mpi</td>
<td>41/46</td>
<td>89.1%</td>
<td>never (X == B) and (X) allocated</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_abs</td>
<td>45/52</td>
<td>86.5%</td>
<td>never (X == B) and always (X == A)</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_free</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_init</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_int</td>
<td>8/10</td>
<td>80.0%</td>
<td>always (z == 0)</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_lset</td>
<td>9/12</td>
<td>75.0%</td>
<td>always (z == 0) and (X) allocated</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.8.4 \texttt{mpi\_div\_mpi} Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_div_mpi</td>
<td>ensures (div_e1)</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>ensures (div_eR1)</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>ensures (div_eR2)</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>ensures (div_eR6)</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.8.5 \texttt{mpi\_div\_mpi} Assigns Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_div_mpi</td>
<td>assigns (R-&gt;s)</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assigns (R-&gt;n)</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assigns (R-&gt;p[0 .. mpi_max_limbs - 1])</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assigns (result)</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
</tbody>
</table>

The same verification strategy as for \texttt{mpi\_add\_mpi} (§9.5.5) has been applied. The match between the specified assigns properties and the computed dependencies has been verified.
9.8.6 **mpi_div_mpi** Reviewed Alarms

The statement:

```c
L_div_a6:  r /= Y.p[t];
```

raise the alarm:

```c
assert Value: division_by_zero: (unsigned long long)*(Y.p+t) != 0;
```

This is a false alarm since `div_a6` ensures that `Y.p[t]` != 0 (see §A.3.10).

9.8.7 **mpi_div_mpi** Intermediate Annotations

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_a1</td>
<td>§A.3.7</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_a2</td>
<td>§A.3.8</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_a5</td>
<td>§A.3.9</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_a6</td>
<td>§A.3.10</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_aux2_0</td>
<td>§A.3.4</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_aux1_1</td>
<td>§A.3.5</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_aux1_3</td>
<td>§A.3.6</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>assert div_aux1_4</td>
<td>§A.3.6</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>loop invariant div_l1_aux1_2</td>
<td>§A.3.6</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_mul_hlp</td>
<td>assert mulh_a3</td>
<td>§A.3.22</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_shift_l</td>
<td>assert shl_a2a</td>
<td>§A.3.23</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_shift_l</td>
<td>assert shl_a2b</td>
<td>§A.3.24</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_sub_hlp</td>
<td>loop invariant subh_l1_4</td>
<td>§A.3.25</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_sub_hlp</td>
<td>loop invariant subh_l2_1</td>
<td>§A.3.26</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>ensures cmpa_e1</td>
<td>§A.3.1</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e3</td>
<td>§A.3.2</td>
<td>✔</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e4</td>
<td>§A.3.3</td>
<td>✔</td>
</tr>
</tbody>
</table>

9.9. Verification of **mpi_exp_mod**

9.9.1 **mpi_exp_mod** Formal Specification

The **mpi_exp_mod** formal specification is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by RSA</td>
<td>§10.3</td>
<td>✔</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§9.9.2</td>
<td>✔</td>
</tr>
</tbody>
</table>
It is defined by:

```
RSA/mpi_exp_mod_rsa.acsl

requires exp_rA1: \valid_read(A) ;
requires exp_rA2: 1 <= A->n <= mpi_max_limbs ;
requires exp_rA3: -1 == A->s || 1 == A->s ;
requires exp_rA4: \valid_read(A->p + (0 .. mpi_max_limbs - 1));
requires exp_rA5: \initialized(A->p + (0 .. mpi_max_limbs - 1));

requires exp_rE1: \valid_read(E) ;
requires exp_rE2: E->n == 32 ;
requires exp_rE3: -1 == E->s || 1 == E->s ;
requires exp_rE4: \valid_read(E->p + (0 .. 32 - 1));
requires exp_rE5: \initialized(E->p + (0 .. 32 - 1));

requires exp_rN1: \valid_read(N) ;
requires exp_rN2: N->n == 32 ;
requires exp_rN3: -1 == N->s || 1 == N->s ;
requires exp_rN4: \valid_read(N->p + (0 .. 32 - 1));
requires exp_rN5: \initialized(N->p + (0 .. 32 - 1));

requires exp_rX1_rv: \valid(X) ;
requires exp_rX2: X->p == \null ;
requires exp_rR1_rv: \valid(_RR) ;
requires exp_rR2: _RR->p == \null ;

assigns X->s \from Frama_C_entropy_source,
               A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
               N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
               E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
               X->n, X->p, _RR, _RR->p

assigns X->n \from Frama_C_entropy_source,
               A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
               N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
               E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
               X->n, X->p, _RR, _RR->p

assigns X->p \from Frama_C_entropy_source,
               tis_p1, tis_p2, tis_p3, // for RSA
               A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
               N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
               E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
               X->n, X->p, _RR, _RR->p

assigns X->p[0 .. mpi_max_limbs - 1] \from Frama_C_entropy_source,
               A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
               N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
               E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
               X->n, X->p, _RR, _RR->p

assigns _RR->s \from Frama_C_entropy_source,
               A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
               N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
```

The table for property verification:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§9.9.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§9.9.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§9.9.4</td>
<td>✓</td>
</tr>
</tbody>
</table>
9.9. Verification of mpi_exp_mod

E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
X->n; // X, X->p, RR, RR->p
assigns _RR->n \from Frama_C_entropy_source,
A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
X->n; // X, X->p, RR, RR->p
assigns _RR->p \from
tis_p1, tis_p2, tis_p3, // for RSA
A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
X->n; // X, X->p, RR, RR->p
assigns \result \from Frama_C_entropy_source,
A->p[0 .. mpi_max_limbs - 1], A->s, A->n, // A, A->p
N->p[0 .. mpi_max_limbs - 1], N->s, N->n, // N, N->p
E->p[0 .. mpi_max_limbs - 1], E->s, E->n, // E, E->p
X->n; // X, X->p, RR, RR->p
ensures exp_e_res_val: \result == 0
    // \result == tis_POLARSSL_ERR_MPI_MALLOC_FAILED
    // \result == tis_POLARSSL_ERR_MPI_NEGATIVE_VALUE
    // \result == tis_POLARSSL_ERR_MPI_DIVISION_BY_ZERO
    // \result == tis_POLARSSL_ERR_MPI_BAD_INPUT_DATA;
ensures exp_eX1_rv: \result == 0 ==> 1 <= X->n <= mpi_max_limbs ;
ensures exp_eX2_val: -1 == X->s || 1 == X->s ;
ensures exp_eX6_rv: \result == 0 ==> X->p == tis_arr1 || X->p == tis_arr2 || X->p == tis_arr3; // for RSA
ensures exp_eRR1_rv: \result == 0 ==> 1 <= _RR->n <= mpi_max_limbs ;
ensures exp_eRR2_val: -1 == _RR->s || 1 == _RR->s ;
ensures exp_eRR6_rv: \result == 0 ==> _RR->p == tis_arr1 || _RR->p == tis_arr2 || _RR->p == tis_arr3; // for RSA

9.9.2 mpi_exp_mod Analysis Context

The function takes five arguments:

int mpi_exp_mod( mpi *X, const mpi *A, const mpi *E, const mpi *N, mpi * _RR )

X and _RR are the results, and A, E and N the input numbers.

In the context of rsa_private, mpi_exp_mod is only called in a context where:

- X and _RR are initialized, but not allocated,
- the size of E and N is always 32,
- the size of A is undetermined between 1 and 100.

This context is built by the function below:
In this context, `mpi_exp_mod` preconditions are all valid according to the analysis results:

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rE1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rE2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rE3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rE4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rN1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rN2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rN3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rN4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rN6</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rX1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rX2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rRR1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rRR2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rA1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rA2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rA3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rA4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_ret_0</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>requires exp_rA6</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
9.9.3 mpi_exp_mod Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_montred</td>
<td>7/7</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_montmul</td>
<td>28/28</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_hlp</td>
<td>404/404</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_hlp</td>
<td>24/24</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>20/20</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>static_alloc</td>
<td>11/11</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_mpi</td>
<td>28/28</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_mpi</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_mpi</td>
<td>47/47</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>40/40</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_msbb</td>
<td>18/18</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_grow</td>
<td>17/17</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_abs</td>
<td>27/28</td>
<td>96.4%</td>
<td>mpi_copy cannot fail here</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>178/186</td>
<td>95.6%</td>
<td>RR!!=!0 and some errors cannot occur</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_add_abs</td>
<td>48/52</td>
<td>92.3%</td>
<td>X always allocated and X == A</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_montg_init</td>
<td>11/12</td>
<td>91.7%</td>
<td>int size smaller than 64 bits</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>21/23</td>
<td>91.3%</td>
<td>X != Y</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_lset</td>
<td>10/12</td>
<td>83.3%</td>
<td>always called with z==1</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_free</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_init</td>
<td>5/6</td>
<td>83.3%</td>
<td>non null argument</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_int</td>
<td>8/10</td>
<td>80.0%</td>
<td>always called with z==0</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_shift_l</td>
<td>35/44</td>
<td>75.0%</td>
<td>always called with allocated X and count==2848</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_div_mpi</td>
<td>0/148</td>
<td>0.0%</td>
<td>use the specification.</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.9.4 mpi_exp_mod Output Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_e_res</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eX1</td>
<td>§A.3.11</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eX2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eX6</td>
<td>§A.3.11</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eRR1</td>
<td>§A.3.11</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eRR2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>ensures exp_eRR6</td>
<td>§A.3.11</td>
<td>✓</td>
</tr>
</tbody>
</table>

9.9.5 mpi_exp_mod Assigns Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_exp_mod</td>
<td>assigns X-&gt;s</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns X-&gt;n</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
</tbody>
</table>
9. MPI Sub-component Analysis

9.9. Verification of mpi_exp_mod

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_exp_mod</td>
<td>assigns X-&gt;p</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns X-&gt;p[0 .. mpi_max_limbs - 1]</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns RR-&gt;s</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns RR-&gt;n</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns RR-&gt;p</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns RR-&gt;p[0 .. mpi_max_limbs - 1]</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_exp_mod</td>
<td>assigns \result</td>
<td>reviewed below</td>
<td>✓</td>
</tr>
</tbody>
</table>

The same verification strategy as for mpi_add_mpi (§9.5.5) has been applied. The match between the specified assigns properties and the computed dependencies has been verified.

9.9.6 mpi_exp_mod Reviewed Alarms

The statement:

```c
L_exp_a4: wbits |= {ei <= (wsize - nbits)};
```

raise the alarm:

```c
assert Value: shift:
0 ≤ (unsigned int)(wsize-nbits) && (unsigned int)(wsize-nbits) < 32;
```

This is a false alarm since the two following properties are verified (see §A.3.12):

```c
RSA/mpi_exp_mod_rsa.acsl
```

```c
at L_exp_a4: assert exp_a4a_rv: 0 <= wsize - nbits;
at L_exp_a4: assert exp_a4b_rv: wsize - nbits < 6;
```

9.9.7 mpi_exp_mod Intermediate Annotations

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a4a</td>
<td>§A.3.12</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a4b</td>
<td>§A.3.12</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a5a</td>
<td>§A.3.18</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a5b</td>
<td>§A.3.18</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a8_bufsize_max</td>
<td>§A.3.15</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a8</td>
<td>§A.3.13</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a9_valid</td>
<td>§A.3.16</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a9_wbits</td>
<td>§A.3.17</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a9_wbits_min</td>
<td>§A.3.14</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_exp</td>
<td>assert exp_a9_wbits_max</td>
<td>§A.3.14</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_abs</td>
<td>ensures cmpa_e1</td>
<td>§A.3.1</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mul_hlp</td>
<td>assert mulh_a3</td>
<td>§A.3.22</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_sub_hlp</td>
<td>loop invariant subh_l2_1</td>
<td>§A.3.26</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e3</td>
<td>§A.3.2</td>
<td>✓</td>
</tr>
<tr>
<td>Function</td>
<td>Properties</td>
<td>Justification</td>
<td>Validation</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>mpi_copy</td>
<td>ensures cp_e4</td>
<td>§A.3.3</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_mpi</td>
<td>ensures mod_eR1</td>
<td>§A.3.19</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_mpi</td>
<td>ensures mod_eR2</td>
<td>§A.3.19</td>
<td>✓</td>
</tr>
</tbody>
</table>
10. RSA Sub-component Analysis

10.1. RSA Verification Summary

This section describes the security analyses results for the RSA sub-component deployed in the context of the SSL Server component. In this context, this section states that RSA sub-component is immune to the given list of CWEs, and that the properties used to validate the server component given by the specification are correct. The verification relies on the specifications of some of the functions of the MPI (multi-precision integer) sub-component studied in §9.

<table>
<thead>
<tr>
<th>High level Component</th>
<th>RSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzed API</td>
<td>rsa_private</td>
</tr>
<tr>
<td>Guarantees Perimeter</td>
<td>Used as part of the SSL Server component</td>
</tr>
<tr>
<td>LOC in perimeter/Total LOC</td>
<td>726/1063</td>
</tr>
<tr>
<td>Sub-components</td>
<td>MPI</td>
</tr>
<tr>
<td>Main context size to audit</td>
<td>25</td>
</tr>
<tr>
<td>Total number of analyses</td>
<td>1</td>
</tr>
<tr>
<td>Required properties</td>
<td>35</td>
</tr>
<tr>
<td>Alarms (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Guaranteed properties (FV/V/U)</td>
<td>0/8/0</td>
</tr>
<tr>
<td>Internal properties (V/U)</td>
<td>0/0</td>
</tr>
<tr>
<td>Specified External functions</td>
<td>memset</td>
</tr>
<tr>
<td>Time for analysis</td>
<td>31s</td>
</tr>
<tr>
<td>Global quality</td>
<td>Semi-formal Trust (everything reviewed)</td>
</tr>
</tbody>
</table>

10.2. RSA API

The only function studied separately as representing the RSA sub-component is rsa_private.

10.3. RSA Sub-component Integration

The specifications of the following sub-component functions have been used:

<table>
<thead>
<tr>
<th>Sub-component</th>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Integration Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>mpi_add_mpi</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§9.5.4</td>
<td>✓</td>
</tr>
<tr>
<td>MPI</td>
<td>mpi_sub_mpi</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§9.6.4</td>
<td>✓</td>
</tr>
<tr>
<td>MPI</td>
<td>mpi_mul_mpi</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§9.7.4</td>
<td>✓</td>
</tr>
<tr>
<td>MPI</td>
<td>mpi_div_mpi</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§9.8.4</td>
<td>✓</td>
</tr>
<tr>
<td>MPI</td>
<td>mpi_exp_mod</td>
<td>preconditions</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>postconditions</td>
<td>§9.9.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

Moreover, the specification of the external standard library function has been used:

PolarSSL 1.1.8 verification kit – V1.0 – Evaluation version 71/116
• memset.

### 10.4. Verification of rsa_private

#### 10.4.1 rsa_private Formal Specification

This section presents rsa_private function specification that is composed of three kinds of properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Verification</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>requires</td>
<td>verified by SSL Server</td>
<td>§5.3</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>verified by the context</td>
<td>§10.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>requires</td>
<td>represent the context</td>
<td>§10.4.2</td>
<td>✓</td>
</tr>
<tr>
<td>assigns</td>
<td>match the computed dependencies</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>ensures</td>
<td>verified within the context</td>
<td>§10.4.4</td>
<td>✓</td>
</tr>
</tbody>
</table>

Because only RSA-1024 is considered in the context of the SSL server, all the numbers representing the RSA key are coded with at most 1024 bits. The context field `len` gives the number of bytes of N, which can be assumed to be 128 (1024/8). The input and output buffer sizes must also be large enough to store 1024 bits, so their size is also 128 bytes. The sizes of all the input MPI numbers in the context are given in terms of limbs (t Uint, assumed to be 32-bit), so these sizes are fixed to 32 (1024/32). All the input numbers:

- are positive (eg ctx->N.s == 1),
- have a size of 32 (eg ctx->N.n == 32),
- are allocated (eg \valid (ctx->DQ.p + (0..31))),
- are initialized (eg \initialized (ctx->DQ.p + (0..31))).

---

```c
#include <__fc_builtin.h>
#include "polarssl/rsa.h"
/*@ requires rsa_r1: \valid(ctx);

requires rsa_r2: ctx->ver == 0;
requires rsa_r3: ctx->len == 128;

requires rsa_rN1: ctx->N.s == 1 && ctx->N.n == 32;
requires rsa_rN2: \valid (ctx->N.p + (0..31));
requires rsa_rN3: \initialized (ctx->N.p + (0..31));
requires rsa_rE1: ctx->E.s == 1 && ctx->E.n == 32;
requires rsa_rE2: \valid (ctx->E.p + (0..31));
requires rsa_rE3: \initialized (ctx->E.p + (0..31));
requires rsa_rD1: ctx->D.s == 1 && ctx->D.n == 32;
requires rsa_rD2: \valid (ctx->D.p + (0..31));
requires rsa_rD3: \initialized (ctx->D.p + (0..31));
requires rsa_rP1: ctx->P.s == 1 && ctx->P.n == 32;
requires rsa_rP2: \valid (ctx->P.p + (0..31));
requires rsa_rP3: \initialized (ctx->P.p + (0..31));
requires rsa_rQ1: ctx->Q.s == 1 && ctx->Q.n == 32;
requires rsa_rQ2: \valid (ctx->Q.p + (0..31));
requires rsa_rQ3: \initialized (ctx->Q.p + (0..31));
requires rsa_rDP1: ctx->DP.s == 1 && ctx->DP.n == 32;
requires rsa_rDP2: \valid (ctx->DP.p + (0..31));
requires rsa_rDP3: \initialized (ctx->DP.p + (0..31));
requires rsa_rDQ1: ctx->DQ.s == 1 && ctx->DQ.n == 32;
requires rsa_rDQ2: \valid (ctx->DQ.p + (0..31));
```
10. RSA Sub-component Analysis

10.4. Verification of rsa_private

requires rsa_rDQ3: \texttt{initialized} (ctx->DQ.p + (0..31));
requires rsa_rQP1: ctx->QP.s == 1 && ctx->QP.n == 32;
requires rsa_rQP2: \texttt{valid} (ctx->QP.p + (0..31));
requires rsa_rQP3: \texttt{initialized} (ctx->QP.p + (0..31));
requires rsa_rRN1: ctx->RN.s == 0 && ctx->RN.n == 0 && ctx->RN.p == 0;
requires rsa_rRP1: ctx->RP.s == 0 && ctx->RP.n == 0 && ctx->RP.p == 0;
requires rsa_rRQ1: ctx->RQ.s == 0 && ctx->RQ.n == 0 && ctx->RQ.p == 0;

requires rsa_r4: ctx->hash_id == 0;
requires rsa_r5: ctx->padding == 0;

requires rsa_rI1: \texttt{valid_read}(input+(0..ctx->len-1));
requires rsa_rI2: \texttt{initialized}(input+(0..ctx->len-1));
requires rsa_rO1: \texttt{valid}(output+(0..ctx->len-1));

assigns ctx->RP.s \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx
        input[0..ctx->len-1], // indirect: input
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->RP.s;

assigns ctx->RP.n \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx,
        input[0..ctx->len-1], // indirect: input
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->RP.n;

assigns ctx->RQ.s \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx,
        input[0..ctx->len-1], // indirect: input
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->RQ.s;

assigns ctx->RQ.n \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx,
        input[0..ctx->len-1], // indirect: input
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->RQ.n;

assigns ctx->RP.p \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx,
        input[0..ctx->len-1], // indirect: input,
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->RP.p;

assigns ctx->RQ.p \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx,
        input[0..ctx->len-1], // indirect: input,
        ctx->N.s, ctx->N.n, ctx->N.p[0..ctx->len-1], // indirect: ctx->N
        ctx->P.s, ctx->P.n, ctx->P.p[0..ctx->len-1], // indirect: ctx->P
        ctx->DP.s, ctx->DP.n, ctx->DP.p[0..ctx->len-1], // indirect: ctx->DP
        ctx->0.s, ctx->0.n, ctx->0.p[0..ctx->len-1], // indirect: ctx->0
        ctx->DQ.s, ctx->DQ.n, ctx->DQ.p[0..ctx->len-1], // indirect: ctx->DQ
        ctx->RQ.p;

assigns ctx->RP.p[0..99] \texttt{from} Frama_C_entropy_source, ctx->len, // indirect: ctx
10. RSA Sub-component Analysis

10.4. Verification of rsa_private

```c
int rsa_private( rsa_context *ctx, const unsigned char *input, unsigned char *output );
```

10.4.2 rsa_private Analysis Context

RSA is analyzed in a context built with the following source code. The analysis context is written with help from Frama-C auxiliary functions described in §B.

```c
#include "rsa_spec.h"
#define IN_ALLOC_SIZE 100
#define IN_USED_SIZE 32
static t_uint tis_N[IN_ALLOC_SIZE];
```
10. RSA Sub-component Analysis
10.4. Verification of rsa_private

```c
static t_uint tis_E[IN_ALLOC_SIZE];
static t_uint tis_D[IN_ALLOC_SIZE];
static t_uint tis_P[IN_ALLOC_SIZE];
static t_uint tis_Q[IN_ALLOC_SIZE];
static t_uint tis_DQ[IN_ALLOC_SIZE];
static t_uint tis_DP[IN_ALLOC_SIZE];
static t_uint tis_QP[IN_ALLOC_SIZE];

int main() {
    int ret = 0;
    rsa_context tis_rsa;
    int padding = RSA_PKCS_V15; // Frama_C_nondet (RSA_PKCS_V15, RSA_PKCS_V21);
    rsa_init(&tis_rsa, padding, 0);
    
    tis_rsa.len = 128;

    Frama_C_make_unknown(tis_N, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.N.s = 1; tis_rsa.N.n = IN_USED_SIZE; tis_rsa.N.p = &tis_N;
    Frama_C_make_unknown(tis_E, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.E.s = 1; tis_rsa.E.n = IN_USED_SIZE; tis_rsa.E.p = &tis_E;
    Frama_C_make_unknown(tis_D, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.D.s = 1; tis_rsa.D.n = IN_USED_SIZE; tis_rsa.D.p = &tis_D;
    Frama_C_make_unknown(tis_P, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.P.s = 1; tis_rsa.P.n = IN_USED_SIZE; tis_rsa.P.p = &tis_P;
    Frama_C_make_unknown(tis_Q, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.Q.s = 1; tis_rsa.Q.n = IN_USED_SIZE; tis_rsa.Q.p = &tis_Q;
    Frama_C_make_unknown(tis_DQ, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.DQ.s = 1; tis_rsa.DQ.n = IN_USED_SIZE; tis_rsa.DQ.p = &tis_DQ;
    Frama_C_make_unknown(tis_DP, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.DP.s = 1; tis_rsa.DP.n = IN_USED_SIZE; tis_rsa.DP.p = &tis_DP;
    Frama_C_make_unknown(tis_QP, IN_USED_SIZE * sizeof(t_uint));
    tis_rsa.QP.s = 1; tis_rsa.QP.n = IN_USED_SIZE; tis_rsa.QP.p = &tis_QP;

    unsigned char output[128];
    unsigned char input[128];
    Frama_C_make_unknown(input, 128);
    rsa_private(&tis_rsa, input, output);

    cleanup:
    return ret;
}
```

In this context, all the preconditions of rsa_private are formally verified, and a manual review ensures that all the input contexts defined by the preconditions are covered.

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsa_private</td>
<td>requires rsa_r1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_r2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_r3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_r4</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_r5</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rD1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rD2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rD3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDP1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDP2</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
10. RSA Sub-component Analysis

10.4. Verification of rsa_private Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Properties</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDP3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDQ1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDQ2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rDQ3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rE1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rE2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rE3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rH1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rI1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rI2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rI3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rN1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rN2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rN3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rO1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rP1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rP2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rP3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQ1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQ2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQ3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQP1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQP2</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rQP3</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rRN1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rRP1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>requires rsa_rRQ1</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

10.4.3 rsa_private Coverage Analysis

<table>
<thead>
<tr>
<th>Function</th>
<th># LOC</th>
<th>Coverage</th>
<th>Review</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_size</td>
<td>3/3</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_mod_mpi</td>
<td>28/28</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_mpi</td>
<td>47/47</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_write_binary</td>
<td>17/17</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_msb</td>
<td>20/20</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>main</td>
<td>40/40</td>
<td>100.0%</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>43/44</td>
<td>97.7%</td>
<td>no error in mpi_read_binary</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_grow</td>
<td>17/19</td>
<td>89.5%</td>
<td>never called with nblimbs &gt; 100</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_free</td>
<td>5/6</td>
<td>83.3%</td>
<td>never called with X==0</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_init</td>
<td>5/6</td>
<td>83.3%</td>
<td>never called with X==0</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_cmp_int</td>
<td>8/10</td>
<td>80.0%</td>
<td>always called with z==0</td>
<td>✓</td>
</tr>
<tr>
<td>mpi_read_binary</td>
<td>23/25</td>
<td>92%</td>
<td>no error in the context</td>
<td>✓</td>
</tr>
</tbody>
</table>

10.4.4 rsa_private Output Properties
10.4.5  rsa_private Assigns Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Property</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RP.s</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RP.n</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RP.p</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RQ.s</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RQ.n</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns ctx-&gt;RQ.p</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns output[0..ctx-&gt;len-1]</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
<tr>
<td>rsa_private</td>
<td>assigns result</td>
<td>§10.4.5</td>
<td>✓</td>
</tr>
</tbody>
</table>

The above dependencies given as specification are compared to the dependencies computed automatically for rsa_private.

In the analysis context definition (§10.4.2), tis_rsa is the name of the array the address of which is passed as ctx to rsa_private, and tis_XX are the arrays used to represent the input number (for instance, tis_N is the array for ctx->N input number.

• ctx->RP.s, ctx->RP.len, ctx->RP.s, ctx->RP.len assigns properties come from:

```plaintext
  tis_rsa(.RP{.s; .n}; .RQ{.s; .n});
  FROM indirect: Frama_C_entropy_source; tis_rsa(.len; .N);
  input[0..127]; ctx; input; tis_N[0..31];
  tis_arr1[0..32]; tis_arr2[0..32]; tis_arr3[0..32];
  direct: Frama_C_entropy_source;
  tis_rsa(.len; .P{.s; .n}; .DP{.s; .n}); input[0..127];
  tis_P[0..99]; tis_DP[0..99]; tis_arr1[0..99]; tis_arr2[0..99];
  tis_arr3[0..99] (and SELF)
```

• ctx->RP.p comes from:

```plaintext
  tis_rsa.RP.p
  FROM indirect: Frama_C_entropy_source; tis_rsa(.len; .N);
  input[0..127]; ctx; input; tis_N[0..31];
  tis_arr1[0..32]; tis_arr2[0..32]; tis_arr3[0..32];
  direct: Frama_C_entropy_source; tis_rsa.RP.p;
  input[0..127]; tis_P[0..99]; tis_DP[0..99];
  tis_arr1[0..99]; tis_arr2[0..99]; tis_arr3[0..99];
  tis_p1; tis_p2; tis_p3 (and SELF)
```

• ctx->RQ.p comes from similar dependencies for tis_rsa.RQ.p.

• The tis_arr1, tis_arr2 and tis_arr3 are static arrays used to modelize allocations (see §9.4.2). They are ignored in the right part of the dependencies since they are not considered as inputs (only initialized to 0 before rsa_private call). The dependencies of these arrays can be interpreted as the dependencies of the output arrays ctx->RP.p[...] and ctx->RQ.p[...]:
10. RSA Sub-component Analysis

10.4. Verification of rsa_private

By construction, the three arrays have the same dependencies.

- Frama_C_entropy_source is a way to modelize dependencies on external elements (for instance, does malloc succeed or not). Its dependencies are not meaningful:

```
Frama_C_entropy_source
  FROM indirect: Frama_C_entropy_source; tis_rsa.len; input[0..127]; ctx; input;
  direct: Frama_C_entropy_source (and SELF)
```

- the dependencies of the output buffer are:

```
output[0..127] FROM Frama_C_entropy_source;
  tis_rsa({},.len; .N); (.P; .Q(.s; .n)); .DP(.s; .n);
    .DQ(.s; .n); .QP(.s; .n); .RP(.s; .n); .RQ(.s; .n); );
  input[0..127]; ctx; input; tis_N[0..31]; tis_P[0..99];
  tis_Q[0..99]; tis_QP[0..99]; tis_arr1[0..99]; tis_arr2[0..99];
  tis_arr3[0..99]; tis_p1; tis_p2; tis_p3; direct: Frama_C_entropy_source;
  tis_rsa({.len; .P; .Q{.s; .n}}; {.Q{.s; .n}}; .DQ{.s; .n};
    .DQ{.s; .n}; .QP{.s; .n}; .RP{.s; .n}; .RQ{.s; .n}; );
  input[0..127]; tis_P[0..99]; tis_Q[0..99]; tis_QP[0..99];
  tis_arr1[0..99]; tis_arr2[0..99]; tis_arr3[0..99] (and SELF)
```

- and the last one gives the dependencies of the returned value:

```
\result FROM Frama_C_entropy_source;
  tis_rsa({},.len; .N); (.P; .Q(.s; .n)); .DP(.s; .n); );
  input[0..127]; ctx; input; tis_arr1[0..99]; tis_arr2[0..99];
  tis_arr3[0..99]; tis_p1; tis_p2; tis_p3
```

10.4.6 rsa_private Reviewed Alarms

No alarm.

10.4.7 rsa_private Intermediate Annotations

None.
A. Intellectual Analyses

A.1. Intellectual Analyses Summary

This section contains the intellectual analysis of all the properties on which the security property of PolarSSL is built. In the case of properties that are not formally verified, a natural language justification is provided instead. An informal level of difficulty is provided for these natural-language justifications:

- \( (*) \): simple argument involving local reasoning only,
- \( (**) \): local reasoning over the values of several variables,
- \( (***) \): non-local reasoning,
- \( (****) \) complex argument that may involve global invariants and require familiarity with the algorithm being implemented.

The suffix of the property’s name indicates the method used to verify it. It may be:

- \(_\text{val}\), \(_\text{sc}\) or \(_\text{wp}\) for formally verified properties:
  - \(_\text{val}\) is for the Value plug-in (Abstract Interpretation technique) that verifies properties in the context of each call. A property is marked as verified through the Value plug-in only if it is valid in all the contexts the function is called in.
  - \(_\text{sc}\) refers to the Scope plug-in. This plug-in ensures that the property is identical to an already justified property.
  - \(_\text{wp}\) refers to the WP plug-in (Weakest Precondition technique). The WP plug-in verifies that the property holds as long as the function’s pre-conditions are respected.
- \(_\text{rv}\) for verified properties through intellectual analysis below.

The properties verified by intellectual analysis are:

<table>
<thead>
<tr>
<th>Component</th>
<th>Property</th>
<th>Reference</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL server</td>
<td>spch_a1</td>
<td>§A.2.1</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>recv_r1</td>
<td>§A.2.2</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>mmv_valid_src</td>
<td>§A.2.3</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>rd_r2</td>
<td>§A.2.4</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>rd_r9</td>
<td>§A.2.5</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>wrt_e1</td>
<td>§A.2.6</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>wrt_e2</td>
<td>§A.2.6</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>sha1_update_r_buffer</td>
<td>§A.2.7</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>sha1_finish_r_buffer</td>
<td>§A.2.7</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>md5_update_r_buffer</td>
<td>§A.2.7</td>
<td>*</td>
</tr>
<tr>
<td>SSL server</td>
<td>md5_finish_r_buffer</td>
<td>§A.2.7</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>cmpa_e1</td>
<td>§A.3.1</td>
<td>**</td>
</tr>
<tr>
<td>MPI</td>
<td>cp_e3</td>
<td>§A.3.2</td>
<td>**</td>
</tr>
<tr>
<td>MPI</td>
<td>cp_e4</td>
<td>§A.3.3</td>
<td>**</td>
</tr>
<tr>
<td>MPI</td>
<td>div_aux2_0</td>
<td>§A.3.4</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>div_aux1_1</td>
<td>§A.3.5</td>
<td>**</td>
</tr>
<tr>
<td>MPI</td>
<td>div_l1_aux1_2</td>
<td>§A.3.6</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>div_aux1_3</td>
<td>§A.3.6</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>div_aux1_4</td>
<td>§A.3.6</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>div_a1</td>
<td>§A.3.7</td>
<td>***</td>
</tr>
<tr>
<td>MPI</td>
<td>div_a2</td>
<td>§A.3.8</td>
<td>**</td>
</tr>
<tr>
<td>MPI</td>
<td>div_a5</td>
<td>§A.3.9</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>div_a6</td>
<td>§A.3.10</td>
<td>***</td>
</tr>
<tr>
<td>MPI</td>
<td>exp_eRR1</td>
<td>§A.3.11</td>
<td>*</td>
</tr>
<tr>
<td>MPI</td>
<td>exp_eRR6</td>
<td>§A.3.11</td>
<td>*</td>
</tr>
</tbody>
</table>
A.2. SSL server Intellectual Analyses

A.2.1 Review of spch_a1

Used in:

- SSL Server Intermediate Annotations (§5.7)

Property to check:

```plaintext
server/server.acsl

at L0: assert spch_a1_rv: ciph_len <= n - 6 - chal_len ;
```

Source code:

```plaintext
server/ssl_srv.c

if( n != 6 + ciph_len + sess_len + chal_len )
{
    SSL_DEBUG_MSG( 1, ( "bad_client_hello_message" ) );
    return( POLARSSL_ERR_SSL_BAD_HS_CLIENT_HELLO );
}
```

The L0 label is reached only if the test at line 153 is true, so: n == 6+ciph_len+sess_len+chal_len. The analysis ensures that:
• $n \in [14..1022]$
• $\text{ciph\_len} \in [3..65535]$
• $\text{sess\_len} \in [0..32]$
• $\text{chal\_len} \in [8..32]$

Because computation is done on 32-bit unsigned integers, the sum cannot overflow, and then $n$ is ensured to be larger than any part of the sum, for instance $n \geq 6 + \text{ciph\_len} + \text{chal\_len}$, and then $\text{ciph\_len} \leq n - 6 - \text{chal\_len}$.

So the property $\text{spch\_a1}$ is true.

### A.2.2 Review of $\text{recv\_r1}$

Used in:

• SSL Server Intermediate Annotations (§5.7)

Property to check:

```acsl
server/server.acsl
requires recv_r1_rv: \valid(output+(0..output_len-1));
```

This is a precondition of the function $\text{tis\_recv}$ which is only called through a pointer in $\text{ssl\_fetch\_input}$:

```c
server/ssl_tls.c
L1: ret = ssl->f_recv(ssl->p_recv, ssl->in_hdr + ssl->in_left, len);
```

Since in the call context:

• the second formal parameter $\text{output}$ is $\text{ssl->in\_hdr + ssl->in\_left}$,
• and the third formal parameter $\text{output\_len}$ is $\text{len}$,

then the property is ensured by the formally proved property:

```acsl
server/server.acsl
at L1: assert fetch_a5_wp: \valid(ssl->in_hdr+ssl->in_left+(0 .. len-1));
```

So the property $\text{recv\_r1}$ is true.

### A.2.3 Review of $\text{valid\_src}$

Used in:

• SSL Server Sub-component Integration (§5.3)

Property to check:

```c
libc/string.h
@ requires valid_src: \valid_read((char*)src+(0..n - 1));
```

```c
extern void *memcpy(void *dest, const void *src, size_t n);
```

The only call for which this precondition is not automatically formally verified is in $\text{ssl\_read\_record}$:
Since in the call context:

- the second formal parameter src is ssl->in_msg + ssl->in_hslen,
- and the third formal parameter n is ssl->in_msglen,

then the property is ensured by the formally proved property:

```c
memmove( ssl->in_msg, ssl->in_msg + ssl->in_hslen, ssl->in_msglen );
```

So the property `valid_src` is true.

**A.2.4 Review of `rd_r2`**

Used in:

- SSL Server Intermediate Annotations (§5.7)

Property to check:

```c
assert rdr_a2_wp:
\valid (ssl->in_msg + ssl->in_hslen + (0 .. ssl->in_msglen-1));
```

This property is ensured by two other preconditions that are formally verified:

```c
requires rdr_r2_rv: ssl->in_offt == \null
| | ssl->in_offt <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN ;
```

`rd_r3_wp` can be split in two cases:

- either ssl->in_offt == \null and the first part of the disjunction in `rd_r2_rv` is true.
- or ssl->in_offt + ssl->in_msglen <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN:
  - and ssl->in_offt <= ssl->in_offt + ssl->in_msglen because `rd_r1_val` ensures that 0 <= ssl->in_msglen
  - so ssl->in_offt <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN
  - then the second part of the disjunction in `rd_r2_rv` is true.

So the property `rd_r2` is true in both case.

**A.2.5 Review of `rd_r9`**

Used in:

- SSL Server Intermediate Annotations (§5.7)
A. Intellectual Analyses
A.2. SSL server Intellectual Analyses

Property to check:

```
server/server.acsl
at L1: assert rd_a9_rv:
    ssl->in_offt + ssl->in_msglen <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN ;
```

Source code:

```
server/ssl_tls.c
L4:
    memcpy( buf, ssl->in_offt, n );
size_t tis_old_in_msglen2 = ssl->in_msglen; ssl->in_msglen -= n;
L1:
```

The following property is formally verified at label L4:

```
server/server.acsl
at L4: assert rd_a8_wp:
    ssl->in_offt + ssl->in_msglen <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN ;
```

Because the call to `memcpy` doesn’t modify any data involved in the property, `rd_a8_wp` is still true after the call.

Moreover, the following property is also verified at label L1:

```
server/server.acsl
at L1: assert rd_a1_wp: ssl->in_msglen <= tis_old_in_msglen2 ;
```

It means that `ssl->in_msglen` at label L1 is smaller than at label L4.

Because everything else is not modified:

- `ssl->in_offt + \at(ssl->in_msglen,L1) <= ssl->in_offt + \at(ssl->in_msglen,L4)`
- `ssl->in_offt + \at(ssl->in_msglen,L4) <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN`
- `ensures that at label L1: ssl->in_offt + ssl->in_msglen <= ssl->in_msg+tis_SSL_MAX_CONTENT_LEN`

So the property `rd_r9` is true.

A.2.6 Review of `wrt_e1` and `wrt_e2` *

Property to check:

```
server/server.acsl
ensures wrt_e1 rv: ssl->in_offt == \old (ssl->in_offt);
ensures wrt_e2 rv: ssl->in_msglen == \old (ssl->in_msglen);
```

Because neither `ssl->in_offt`, nor `ssl->in_msglen` appear in the computed dependencies of the `ssl_write` function, and because this analysis gives an over-approximation, it ensure that both `ssl->in_offt` and `ssl->in_msglen` are not modified by `ssl_write`.

So the properties `wrt_e1` and `wrt_e2` are both true.
A. Intellectual Analyses

A.2. SSL server Intellectual Analyses

A.2.7 Review of \texttt{update\_r\_buffer} and \texttt{finish\_r\_buffer} *

The preconditions \texttt{update\_r\_buffer} and \texttt{finish\_r\_buffer} of the functions \texttt{sha1\_update}, \texttt{sha1\_finish}, \texttt{md5\_update} and \texttt{md5\_finish} all have the same formulation:

\begin{verbatim}
initialized((ctx->buffer[0 .. ctx->total[0]%64-1]));
\end{verbatim}

The reachable calls to the four functions are examined below and in each case, the validity of the precondition is justified. When the precondition is not formally verified, the justification always comes from a previous call to \texttt{sha1\_update} or \texttt{md5\_finish} that ensures the postcondition:

\begin{verbatim}
initialized((ctx->buffer[0 .. ctx->total[0]%64-1]));
\end{verbatim}

As one can see, this property is identical to the preconditions to justify.

<table>
<thead>
<tr>
<th>Localisation</th>
<th>Called</th>
<th>Caller</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5.c:354</td>
<td>md5_update</td>
<td>md5_hmac_starts</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5.c:364</td>
<td>md5_update</td>
<td>md5_hmac_update</td>
<td>see 1 below</td>
<td>✓</td>
</tr>
<tr>
<td>md5.c:374</td>
<td>md5_finish</td>
<td>md5_hmac_finish</td>
<td>see 3 below</td>
<td>✓</td>
</tr>
<tr>
<td>md5.c:376</td>
<td>md5_update</td>
<td>md5_hmac_finish</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>md5.c:377</td>
<td>md5_update</td>
<td>md5_hmac_finish</td>
<td>call at line 376</td>
<td>✓</td>
</tr>
<tr>
<td>md5.c:378</td>
<td>md5_finish</td>
<td>md5_hmac_finish</td>
<td>call at line 377</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:389</td>
<td>sha1_update</td>
<td>sha1_hmac_starts</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:399</td>
<td>sha1_update</td>
<td>sha1_hmac_update</td>
<td>see 2 below</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:409</td>
<td>sha1_finish</td>
<td>sha1_hmac_finish</td>
<td>see 4 below</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:411</td>
<td>sha1_update</td>
<td>sha1_hmac_finish</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:412</td>
<td>sha1_update</td>
<td>sha1_hmac_finish</td>
<td>call at line 411</td>
<td>✓</td>
</tr>
<tr>
<td>sha1.c:413</td>
<td>sha1_finish</td>
<td>sha1_hmac_finish</td>
<td>call at line 412</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_srv.c:109</td>
<td>md5_update</td>
<td>ssl_parse_client_hello</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_srv.c:110</td>
<td>sha1_update</td>
<td>ssl_parse_client_hello</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_srv.c:229</td>
<td>md5_update</td>
<td>ssl_parse_client_hello</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_srv.c:230</td>
<td>sha1_update</td>
<td>ssl_parse_client_hello</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:152</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:153</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>call at line 152</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:154</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>call at line 153</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:155</td>
<td>sha1_finish</td>
<td>ssl_derive_keys</td>
<td>call at line 154</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:158</td>
<td>md5_update</td>
<td>ssl_derive_keys</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:159</td>
<td>md5_update</td>
<td>ssl_derive_keys</td>
<td>call at line 158</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:160</td>
<td>md5_finish</td>
<td>ssl_derive_keys</td>
<td>call at line 159</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:199</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:200</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>call at line 199</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:201</td>
<td>sha1_update</td>
<td>ssl_derive_keys</td>
<td>call at line 200</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:202</td>
<td>sha1_finish</td>
<td>ssl_derive_keys</td>
<td>call at line 201</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:205</td>
<td>md5_update</td>
<td>ssl_derive_keys</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:206</td>
<td>md5_update</td>
<td>ssl_derive_keys</td>
<td>call at line 205</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:207</td>
<td>md5_finish</td>
<td>ssl_derive_keys</td>
<td>call at line 206</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:476</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:477</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>call at line 476</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:478</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>call at line 477</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:479</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>call at line 478</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:480</td>
<td>sha1_finish</td>
<td>ssl_mac_sha1</td>
<td>call at line 479</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:484</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>
### A.2. SSL server Intellectual Analyses

#### Localisation

<table>
<thead>
<tr>
<th>Localisation</th>
<th>Called</th>
<th>Caller</th>
<th>Justification</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl_tls.c:485</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>call at line 484</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:486</td>
<td>sha1_update</td>
<td>ssl_mac_sha1</td>
<td>call at line 485</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:487</td>
<td>sha1_finish</td>
<td>ssl_mac_sha1</td>
<td>call at line 486</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1021</td>
<td>md5_update</td>
<td>sl_write_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1022</td>
<td>sha1_update</td>
<td>sl_write_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1095</td>
<td>md5_update</td>
<td>ssl_read_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1096</td>
<td>sha1_update</td>
<td>ssl_read_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1224</td>
<td>md5_update</td>
<td>ssl_read_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1225</td>
<td>sha1_update</td>
<td>ssl_read_record</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1615</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1616</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1617</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1618</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1620</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1621</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1622</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1623</td>
<td>sha1_finish</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1628</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1629</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1630</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1631</td>
<td>md5_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1634</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1635</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1636</td>
<td>sha1_update</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1637</td>
<td>sha1_finish</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1647</td>
<td>md5_finish</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
<tr>
<td>ssl_tls.c:1648</td>
<td>sha1_finish</td>
<td>ssl_calc_finished</td>
<td>formal</td>
<td>✓</td>
</tr>
</tbody>
</table>

1. the md5_hmac_update function only calls md5_update, and is always called after the md5_hmac_starts which always end by calling md5_update. So here again, the precondition is ensured by the postcondition of a previous call.

2. this is exactly the same as 2 for the call to sha1_update in sha1_hmac_update.

3. the precondition of the first call to md5_finish in md5_hmac_finish is ensured because md5_hmac_finish is always called immediately after md5_hmac_update which only calls md5_update. So here again, the precondition is ensured by the postcondition of a previous call.

4. this is exactly the same as 3 for the call to sha1_finish in sha1_hmac_finish.

### A.3. MPI Intellectual Analyses

#### A.3.1 Review of cmpa_e1

Used in:

- **mpi_add mpi** Intermediate Annotations (§9.5.7),
- **mpi_sub mpi** Intermediate Annotations (§9.6.7),
- **mpi_div mpi** Intermediate Annotations (§9.8.7),
- **mpi_exp mpi** Intermediate Annotations (§9.9.7).

Property to check:

---

PolarSSL 1.1.8 verification kit – V1.0 – Evaluation version 85/116
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

RSA/bignum.acsl

\begin{verbatim}
ensures cmpa_e1_rv: \result >= 0 ==> MSD(X) >= MSD(Y);
\end{verbatim}

Source code:

RSA/bignum.c

\begin{verbatim}
for( i = X->n; i > 0; i-- )
  if( X->p[i - 1] != 0 )
    break;
L_cmpa_1:
  for( j = Y->n; j > 0; j-- )
    if( Y->p[j - 1] != 0 )
      break;
L_cmpa_2:
  if( i == 0 && j == 0 )
    return( 0 );
  if( i > j ) return( 1 );
  if( j > i ) return( -1 );
\end{verbatim}

These two loops exactly compute $i = \text{MSD}(X)$ and $j = \text{MSD}(Y)$ (see the definition of MSD §9.4.1).

The property states that $\text{MSD}(X) \geq \text{MSD}(Y)$ each time that the function returns a nonnegative result. This is ensured by:

- at line 660, the test $(i == 0 \&\& j == 0)$ is true, so $\text{MSD}(X) == \text{MSD}(Y) == 0$ and thus $\text{MSD}(X) \geq \text{MSD}(Y)$,
- line 662: the property is ensured when the test is true because $i > j$ ensures that $\text{MSD}(X) > \text{MSD}(X)$,
- line 663 returns a negative number, so the property is true,
- line 664 is reached only when $i == j$, so whatever the return value is, $\text{MSD}(X) == \text{MSD}(Y)$ so the property is ensured.

So the property cmpa_e1 is true.

A.3.2 Review of cp_e3 **

Used in:

- mpi_add_mpi Intermediate Annotations (§9.5.7),
- mpi_sub_mpi Intermediate Annotations (§9.6.7),
- mpi_mul_mpi Intermediate Annotations (§9.7.7),
- mpi_div_mpi Intermediate Annotations (§9.8.7),
- mpi_exp_mpi Intermediate Annotations (§9.9.7).

Property to check:

RSA/bignum.acsl

\begin{verbatim}
ensures cp_e3_rv: \result == 0 == \old (X->p) == 0 == X != Y ==> MSD(X) == X->n | | (MSD(X) == 0 \&\& X->n == 1);
\end{verbatim}

Source code:

RSA/bignum.c

\begin{verbatim}
int mpi_copy( mpi *X, const mpi *Y )
{
  int ret;
\end{verbatim}
A. Intellectual Analyses
A.3. MPI Intellectual Analyses

size_t i;
if ( X == Y )
    return( 0 );
for( i = Y->n - 1; i > 0; i-- )
    if ( Y->p[i] != 0 )
        break;
    i++;
L_cp_a2:
    X->s = Y->s;
MPI_CHK( mpi_grow( X, i ) );

... skip irrelevant statements...

return( ret );

The hypotheses of cp_e3 define a context such that:

• \( X \neq Y \) stating that line 124 is not reached,
• \( \result == 0 \) stating that mpi_grow at line 133 doesn’t fail,
• \( \old (X->p) == 0 \) stating that X was not already allocated.

In this context, the mpi_grow postcondition gm_e6_wp ensures that \( X->n == 1 \) at line 134:

\[
\text{ensures} \quad \text{gm_e6_wp:} \quad \result == 0 \implies \old (X->p) == \null \implies X->n == \text{nblims};
\]

Moreover, another post-condition of mpi_copy gives:

\[
\text{ensures} \quad \text{cp_e4_rv:} \quad \result == 0 \implies \text{MSD}(X) == \text{MSD}(Y);
\]

Now let us consider two cases: either \( \text{MSD}(Y) == 0 \) or \( \text{MSD}(Y) > 0 \):

1. if \( \text{MSD}(Y) == 0 \):
   • after the loop: \( i == 0 \),
   • line 130: \( i == 1 \) (because \( i \) is incremented at line 129)
   • line 134: \( X->n == 1 \) (according to gm_e6_wp)
   • at the end of the function: \( \text{MSD}(X) == 0 \) because:
      - \( \text{MSD}(Y) == 0 \) (hypothesis)
      - \( \text{MSD}(X) == \text{MSD}(Y) \) (by cp_e4 §A.3.3)
   • so the second part of the disjunction in cp_e3 is true.

2. if \( \text{MSD}(Y) > 0 \):
   • after the loop: \( i == \text{MSD}(Y) - 1 \) (according to the definition of MSD §9.4.1)
   • line 130: \( i == \text{MSD}(Y) \) (because \( i \) is incremented at line 129)
   • line 134: \( X->n == \text{MSD}(Y) \) (according to gm_e6_wp)
   • at the end of the function: \( \text{MSD}(X) == \text{MSD}(Y) \) (by cp_e4 §A.3.3)
   • so \( \text{MSD}(X) == X->n \).
   • and the first part of the disjunction in cp_e3 is true.

So the property cp_e3 is true in both cases.
A.3.3 Review of \textit{cp\_e4} 

Used in:

- \texttt{mpi\_add\_mpi} Intermediate Annotations (§9.5.7),
- \texttt{mpi\_sub\_mpi} Intermediate Annotations (§9.6.7),
- \texttt{mpi\_mul\_mpi} Intermediate Annotations (§9.7.7),
- \texttt{mpi\_div\_mpi} Intermediate Annotations (§9.8.7),
- \texttt{mpi\_exp\_mpi} Intermediate Annotations (§9.9.7).

Property to check:

\texttt{RSA/bignum.acsl}

\begin{verbatim}
ensures \textit{cp\_e4\_rv}: \texttt{result == 0 ==> MSD(X) == MSD(Y)};
\end{verbatim}

Source code:

\texttt{RSA/bignum.c}

\begin{verbatim}
L\_cp\_a2:
    X->s = Y->s;
    MPI\_CHK( mpi\_grow( X, i ) );
    memset( X->p, 0, X->n * ciL );
    memcpy( X->p, Y->p, i * ciL );
\end{verbatim}

The property \(\text{MSD}(X) == \text{MSD}(Y)\) is ensured because:

- according to the previous proof §A.3.2, the variable \(i\) at label \texttt{L\_cp\_a2} is computed such that:
  - \(i == 1\) if \(\text{MSD}(Y) == 0\) (case 1),
  - \(i == \text{MSD}(Y)\) if \(\text{MSD}(Y) > 0\) (case 2),
- \texttt{memset} initializes \(X->p[0..X->n - 1]\)
- \texttt{memcpy} copies the \(i\) \textit{digits} of \(Y\) in \(X\),
- so because of the definition of \texttt{MSD} (§9.4.1), all the significant \textit{digits} of \(Y\) are copied in \(X\), which means that \(X\) and \(Y\) represent the same number at the end of the function.

So the property \textit{cp\_e4} is true.

A.3.4 Review of \textit{div\_aux2\_0} 

Used in:

- \texttt{mpi\_div\_mpi} Intermediate Annotations (§9.8.7)

A new local variable \textit{aux2} has been used in \texttt{mpi\_div\_mpi} in order to prove that \((i-t)\) used as index stays in the bounds specified by \texttt{div\_a5}. This change avoids the memory access alarms. Before each modified statement where \((i-t)\) has been replaced by \textit{aux2}, an assertion is used to check the equivalence:

\texttt{RSA/mpi\_div\_mpi.acsl}

\begin{verbatim}
at L\_div\_aux2\_0: assert div\_aux2\_0\_rv: aux2 == i - t;
at L\_div\_aux2\_1: assert div\_aux2\_1\_sc: aux2 == i - t;
at L\_div\_aux2\_2: assert div\_aux2\_2\_sc: aux2 == i - t;
at L\_div\_aux2\_3: assert div\_aux2\_3\_sc: aux2 == i - t;
at L\_div\_aux2\_4: assert div\_aux2\_4\_sc: aux2 == i - t;
at L\_div\_aux2\_5: assert div\_aux2\_5\_sc: aux2 == i - t;
at L\_div\_aux2\_6: assert div\_aux2\_6\_sc: aux2 == i - t;
\end{verbatim}
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

56 at L_div_aux2_7: assert div_aux2_7_sc: aux2 == i - t;
57 at L_div_aux2_8: assert div_aux2_8_sc: aux2 == i - t;

Source code:

```
for( i = n; i > t ; i-- )
{  int aux2 = i - t; L_div_a5: L_div_aux2_0:
  if( X.p[i] >= Y.p[t] )
    Z.p[aux2 - 1] = ~0;
  else
    L_div_aux2_1: Z.p[aux2 - 1] = (t_uint) r;
    do
      { L_div_aux2_3: Z.p[aux2 - 1]--;
        L_div_aux2_4: MPI_CHK( mpi_mul_int( &T1, &T1, Z.p[aux2 - 1] ) );
        while( mpi_cmp_mpi( &T1, &T2 ) > 0 );
        L_div_aux2_5: MPI_CHK( mpi_mul_int( &T1, &T1, Z.p[aux2 - 1] ) );
        L_div_aux2_6: MPI_CHK( mpi_shift_l( &T1, biL * (aux2 - 1) ) );
        L_div_aux2_7: MPI_CHK( mpi_shift_l( &T1, biL * (aux2 - 1) ) );
        MPI_CHK( mpi_add_mpi( &X, &X, &T1 ) );
        L_div_aux2_8: Z.p[aux2 - 1]--;
```

• all the assertions div_aux2_1 .. div_aux2_8 have been formally proved equivalent to div_aux2_0 by
  the analyzer,
• the justification of div_aux2_0 is trivial since the variable has just been assigned.

So all the properties are true.

A.3.5 Review of div_aux1_1"

Used in:

• mpi_div_mpi Intermediate Annotations (§9.8.7)

Property to check:

```
RSA/bignum.c

at L_div_aux1_1: assert div_aux1_1_rv: 0 <= aux1 < mpi_max_limbs;
```

Source code:

```
MPI_CHK( mpi_copy( &X, A ) );
MPI_CHK( mpi_copy( &Y, B ) );
L_div_2: X.s = Y.s = 1;
L_div_1:
... nothing about X->n or Y->n...
n = X.n - 1;
t = Y.n - 1;
int aux1 = n - t; L_div_aux1_1: mpi_shift_l( &Y, biL * aux1 );
```
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

Because:

- line 1092: \( aux1 = n - t \)
- line 1090: \( n = X.n - 1 \)
- line 1091: \( t = Y.n - 1 \)

Then the property \( \text{div}\_aux1\_1 \) is established if: \( 0 \leq (X.n - 1) - (Y.n - 1) < \text{mpi}\_\text{max}\_\text{limbs} \)

This is equivalent to: \( 0 \leq (X.n - Y.n) < \text{mpi}\_\text{max}\_\text{limbs} \)

This is true when reaching line 1090 because:

1. \( 0 \leq X.n - Y.n \) is ensured because of \( \text{div}\_\text{a1} \):

   ```
   RSA/mpi_div_mpi.acsl
   at L_div_1: assert div_al_rv: X.n >= Y.n;
   ```

2. \( X.n - Y.n < \text{mpi}\_\text{max}\_\text{limbs} \) is ensured because of the postcondition \( \text{cp}\_\text{e2} \) of \( \text{mpi}\_\text{copy} \):

   ```
   RSA/bignum.acsl
   ensures cp_e2_wp: \text{result} == 0 ==> 1 <= X->n <= \text{mpi}\_\text{max}\_\text{limbs};
   ```

   - \( 1 \leq X.n \leq \text{mpi}\_\text{max}\_\text{limbs} \) because of the successful call to \( \text{mpi}\_\text{copy} \) at line 1072
   - \( 1 \leq Y.n \leq \text{mpi}\_\text{max}\_\text{limbs} \) by the same reasoning applied to the call to \( \text{mpi}\_\text{copy} \) at line 1073,
   - so \( (X.n - Y.n) \) is at most \( \text{mpi}\_\text{max}\_\text{limbs} - 1 \).

So the property \( \text{div}\_\text{aux1}\_1 \) is true.

A.3.6  Review of \( \text{div}\_\text{l1}\_\text{aux1}\_2, \text{div}\_\text{aux1}\_3 \) and \( \text{div}\_\text{aux1}\_4 \)

Used in:

- \( \text{mpi}\_\text{div}\_\text{mpi} \) Intermediate Annotations (§9.8.7)

A new local variable \( aux1 \) has been used in \( \text{mpi}\_\text{div}\_\text{mpi} \) in order to prove that \( (n-t) \) used as index stays in the bounds specified by \( \text{div}\_\text{aux1}\_1 \). This change avoids the memory access alarms. The assertions ensure that the transformation is valid.

```
RSA/mpi_div_mpi.acsl
at L_div_aux1_3: assert div_aux1_3_rv: aux1 == n - t;
at L_div_aux1_4: assert div_aux1_4_rv: aux1 == n - t;
```

Source code:

```c
RSA/bignum.c
int aux1 = n - t; L_div_aux1_1: mpi_shift_l( &Y, biL * aux1 );
while( mpi_cmp_mpi( &X, &Y ) >= 0 )
{
    L_div_aux1_3: Z.p[aux1]++;
    mpi_sub_mpi( &X, &X, &Y );
}
L_div_aux1_4: mpi_shift_r( &Y, biL * (aux1) );
```

Let us first check this property of the loop:
A.3. MPI Intellectual Analyses

Then, divAux1 and divAux2 are trivially ensured by divAux1.
So the properties divAux1 and divAux2 are both true.

A.3.7 Review of divA1

Used in:
- mpi_div_mpi Intermediate Annotations (§9.8.7)

Property to check:

Source code:

```
if ( mpi_cmp_int( B, 0 ) == 0 )
return( POLARSSL_ERR_MPI_DIVISION_BY_ZERO );

if ( mpi_cmp_abs( A, B ) < 0 )
{
  return( 0 );
}
```

- 1059: line 1061 is reached only if mpi_cmp_int(B,0) != 0
  - so it means that B is not zero, and then MSD(B) != 0 (see the definition of MSD §9.4.1).
- 1065: line 1071 is reached only if mpi_cmp_abs(A,B) >= 0
  - the postcondition cmpa_e1 of mpi_cmp_abs ensures that MSD(A) >= MSD(B)
  - moreover, because B is not zero, MSD(B) > 0 and then MSD(A) > 0.
A.3. MPI Intellectual Analyses

- X was not allocated (X.p == 0),
- then cp_e4 ensures that MSD(X) == MSD(A)
- because MSD(A) > 0, then MSD(X) != 0.
- then cp_e3 ensures that MSD(A) == X.n

1075:

RSA/mpi_div_mpi.acsl

at L_div_2: assert div_a2_rv: MSD (B) == Y.n;

- div_a2 ensures that MSD(B) == Y.n.
- finally:
  - knowing:
    * MSD(A) >= MSD(B),
    * MSD(A) == X->n,
    * MSD(B) == Y->n,
  - ensures that X->n >= Y->n.

So the property div_a1 is true.

A.3.8 Review of div_a2 **

Used in:

- mpi_div_mpi Intermediate Annotations (§9.8.7)

Property to check:

RSA/mpi_div_mpi.acsl

at L_div_2: assert div_a2_rv: MSD (B) == Y.n;

Source code:

RSA/bignum.c

1059  if( mpi_cmp_int( B, 0 ) == 0 )
1060     return( POLARSSL_ERR_MPI_DIVISION_BY_ZERO );
1061  mpi_init( &X ); mpi_init( &Y ); mpi_init( &Z );
1073  MPI_CHK( mpi_copy( &Y, B ) );
1074  L_div_2: X.s = Y.s = 1;

- 1059:
  - line 1061 is reached only if mpi_cmp_int(B,0) != 0
  - so it means that B is not 0, and then MSD(B) != 0. (see the definition of MSD §9.4.1).
- 1073:
  - line 1074 is reached only if mpi_copy (&Y,B) returns 0:
  - Y was not allocated (Y.p == 0) du to the call to mpi_init at line 1062.

RSA/bignum.acsl

- cp_e4 ensures that MSD(Y) == MSD(B)
- MSD(B) != 0
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

– then cp_e3 ensures that \( \text{MSD}(B) = Y.n \).

So the property \( \text{div}_a2 \) is true.

A.3.9 Review of \( \text{div}_a5 \)

Used in:

• mpi_div_mpi Intermediate Annotations (§9.8.7)

Property to check:

```plaintext
RSA/mpi_div_mpi.acsl
at L_div_a5: assert div_a5_rv: 0 < aux2 < mpi_max_limbs;
```

Source code:

```c
RSA/bignum.c

n = X.n - 1;
t = Y.n - 1;
int aux1 = n - t; L_div_aux1_1: mpi_shift_l( 6Y, biL * aux1 );
for( i = n; i > t ; i-- )
{ int aux2 = i - t; L_div_a5: L_div_aux2_0:
}
```

The property \( \text{div}_a5 \) is ensured if \( 0 < i - t < \text{mpi_max_limbs} \) is true at the beginning of each loop iteration:

1. \( 0 < i - t \): ensured by the loop condition \( i > t \).
2. \( i - t < \text{mpi_max_limbs} \):
   • is established, because when entering the loop for the first time:
     - \( i=n \),
     - \( \text{div}_aux1_1 \) ensures that \( n-t < \text{mpi_max_limbs} \):

   ```plaintext
   RSA/mpi_div_mpi.acsl
   at L_div_aux1_1: assert div_aux1_1_rv: 0 <= aux1 < mpi_max_limbs;
   ```
   - so \( i-t < \text{mpi_max_limbs} \) is established.
   • the preservation is ensured because:
     - \( i \) decreases so: \( i-t < n-t \)
     - and \( 0 < i-t \).

So the property \( \text{div}_a5 \) is true.

A.3.10 Review of \( \text{div}_a6 \)

Used in:

• mpi_div_mpi Reviewed Alarms (§9.8.6)
• mpi_div_mpi Intermediate Annotations (§9.8.7)

Property to check:

```plaintext
RSA/mpi_div_mpi.acsl
at L_div_a6: assert div_a6_rv: Y.p[t] != 0;
```
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

Source code:

```c
1074 L_div_2: X.s = Y.s = 1;
1081 k = mpi_msb( &Y ) % biL;
1083 if( k < biL - 1 ) {
1084 k = biL - 1 - k;
1085 MPI_CHK( mpi_shift_l( &X, k ) );
1086 MPI_CHK( mpi_shift_l( &Y, k ) );
1087 } else k = 0;
1091 t = Y.n - 1;
1092 int aux1 = n - t; L_div_aux1_1: mpi_shift_l( &Y, biL * aux1 );
1099 L_div_aux1_4: mpi_shift_r( &Y, biL * (aux1) );
1112 L_div_a6: r /= Y.p[t];
```

- 1074: div_a2 ensures that Y.n == MSD (B)
- MSD (Y) == MSD (B) and MSD(B) != 0 (see §A.3.8).
- 1086: mpi_shift_l( &Y, biL-1-k) where k == mpi_msb(&Y) % biL
  - k gives the number of the most significant bit in the most significant digit (MSD) of Y,
  - shifting by (biL-1-k) pushes the bits left so that the position of the most significant bit in the most significant digit is 31.
  - so MSD(Y) is not changed
- 1091: t = Y.n - 1; so MSD (Y) == t+1
- then Y is shifted left, then right by the same amount:
  - 1092: mpi_shift_l( &Y, biL * aux1 );
  - 1099: mpi_shift_r( &Y, biL * aux1 );
  - so MSD (Y) == t+1 again.
- because MSD (Y) == t+1, then Y.p[t] != 0 by definition of MSD.

So the property div_a6 is true.

A.3.11 Review of exp_eRR1, exp_eRR6, exp_eX1, exp_eX6 *

Used in:
- mpi_exp_mod Output Properties (§9.9.4)

Properties to check:

```c
ensures exp_eX1_rv: \result == 0 ==> 1 <= X->n <= mpi_max_limbs;
ensures exp_eX8_rv: \result == 0 ==> X->p == tis_arr1 || X->p == tis_arr2 || X->p == tis_arr3; // for RSA
ensures exp_eRR1_rv: \result == 0 ==> 1 <= _RR->n <= mpi_max_limbs;
ensures exp_eRR6_rv: \result == 0 ==> _RR->p == tis_arr1 || _RR->p == tis_arr2 || _RR->p == tis_arr3; // for RSA
```

Source code:
Because of MPI error management (§9.4.3), the `cleanup` label can be reached:

1. by the direct path, coming from the label `L_exp_ret_0`, where `exp_ret_0` is formally verified by the analyzer:

   ```
   RSA/mpi_exp_mod_rsa.acsl
   118 at L_exp_ret_0 : assert exp_ret_0_val: ret == 0
   119   && (X->p == tis_arr1 || X->p == tis_arr2 || X ->p == tis_arr3) // for RSA
   120   && 1 <= X->n <= mpi_max_limbs
   121   && (_RR->p == tis_arr1 || _RR->p == tis_arr2 || _RR ->p == tis_arr3) // for RSA
   122   && 1 <= _RR->n <= mpi_max_limbs;
   ```

   • so `ret == 0`,
   • and all the postconditions are ensured.

2. by other indirect paths where `ret != 0`.

So the only path where `ret == 0` is the one coming from the label `L_exp_ret_0` where all the properties are ensured to be true.

So all the properties are true.

**A.3.12 Review of exp_a4a and exp_a4b**

Used in:

- `mpi_exp_mod` Reviewed Alarms (§9.9.6)
- `mpi_exp_mod` Intermediate Annotations (§9.9.7)

Properties to check:

```
RSA/mpi_exp_mod_rsa.acsl
107 at L_exp_a4: assert exp_a4a_rv: 0 <= wsize - nbits;
108 at L_exp_a4: assert exp_a4b_rv: wsize - nbits < 6;
```
... no modification of nbits or wsize...

At each loop iteration,
- at line 1493: $0 \leq \text{nbits} < \text{wsize}$ since:
  - nbits is initialized to 0 at line 1488,
  - nbits is incremented at line 1526,
  - nbits is reset to 0 at line 1543 when it reaches wsize
- at line 1527 (label L_exp_a4):
  - nbits has been incremented at line 1526,
  - so $0 < \text{nbits} \leq \text{wsize}$

So the property exp_a4a is true.

Moreover:
- $\text{wsize} \leq 6$ because:
  - this property is formally verified by the analyzer at label L_exp_a7:

```plaintext
\text{RSA/mpi_exp_mod_rsa.acsl}
133 \text{at L_exp_a7: assert exp}_a7\text{._val.wsige_val:}
134 \text{wsize == 1 | wsize == 3 | wsize == 4 | wsize == 5 | wsize == 6;}
```

- wsize is not modified neither between the label and the loop, nor in the loop.
- $1 \leq \text{nbits}$ at line 1527 as seen above for exp_a4a,
- then $\text{wsize} \leq 6$ and $1 \leq \text{nbits}$ gives: $\text{wsize} - \text{nbits} < 6$

So the property exp_a4b is true.

### A.3.13 Review of exp_a8 *

Used in:
- mpi_exp_mod Intermediate Annotations (§9.9.7)

Property to check:

```plaintext
\text{RSA/mpi_exp_mod_rsa.acsl}
100 \text{at L_exp_a8: assert exp}_a8\text{._rv: 0 <= nblims < E->n;}
```

Source code:

```plaintext
\text{RSA/bignum.c}
1468 \text{L_exp_a7: nblims = E->n;}
1467 \text{buusize = 0;}
1490 \text{while( 1 )}
1491 \text{ { if( buusize == 0 )}
1492 \text{ } \text{if( nblims-- == 0 )}
1493 \text{ break;}
1494 \text{ }}
1517 \text{L_exp_a8:}
1531 \text{ei = (E->p[nblims] >> buusize) & 1;}
```

96/116
The property \( nblimbs < E \cdot n \) at \( L_{exp\_a8} \) because:

1. it is established the first time:
   - line 1486: \( nblimbs \) is initialized to \( E \cdot n \),
   - line 1496: \( bufsize \) is initialized to 0,
   - because \( bufsize == 0 \), \( nblimbs \) is decremented at line 1496,
   - so \( nblimbs == E \cdot n - 1 \) the first time \( L_{exp\_a8} \) is reached.

2. it is preserved:
   - \( nblimbs \) always decreases (decremented at line 1496),
   - the loop exits when \( nblimbs \) reach 0.

So the property \( exp\_a8 \) is true.

### A.3.14 **Review of \( exp\_a_wbits\_min \) and \( exp\_a_wbits\_max \)**

Used in:

- \( mpi\_exp\_mod \) Intermediate Annotations (§9.9.7)

Properties to check:

```plaintext
RSA/mpi_exp_mod_rsa.acsl

104 at L_exp_wbits: assert exp_a_wbits_min_rv: (1 << wsize - 1) <= wbits;
105 at L_exp_wbits: assert exp_a_wbits_max_rv: wbits < (1 << wsize);
```

Source code:

```c
RSA/bignum.c

nbits = 0;
wbits = 0;
state = 0;

while( 1 )
{
    ei = (E->p[nblimbs] >> bufsize) & 1;

    if( ei == 0 && state == 0 )
        continue;
    if( ei == 0 && state == 1 )
        {
            continue;
        }

    state = 2;
    nbits++;
    L_exp_a4: wbits |= (ei << (wsize - nbits));
    L_exp_wbits:
    if( nbits == wsize )
        {
            state--;
            nbits = 0;
            wbits = 0;
        }
}
```

At each loop iteration,
• at line 1493: $0 \leq \text{nbits} < \text{wsize}$ since:
  – \text{nbits} is initialized to 0 at line 1488,
  – \text{nbits} is incremented at line 1526,
  – \text{nbits} is reset to 0 at line 1543 when it reaches \text{wsize}

• while executing the loop, state value is 0, 1 or 2 since:
  – \text{state} is initialized to 0 at line 1490,
  – \text{state} is assigned to 2 at line 1524,
  – \text{state} may be decremented from 2 to 1 at line 1542.

• at line 1493: when \text{state} is 0 or 1, \text{nbits} == 0 and \text{wbits}==0:
  – at loop entry, it is true since \text{nbits}, \text{wbits}=0 and \text{state} are initialized to 0,
  – then:
    * either the continue statements at line 1510 and 1518 are executed, and \text{wbits}, \text{nbits} and \text{state} all keep their values,
    * or line 1524 is reached, and \text{state} is assigned to 2,
      - then either \text{state} stays to 2,
      - or is decremented to 1 at line 1542, but then \text{nbits} and \text{wbits} are reset to 0 at line 1543-1544.

• after line 1504: ei is either 0 or 1,
• before line 1524:
  – either \text{state} == 2,
  – or (\text{state}==0 || \text{state}==1) and then \text{ei} == 1 (because of the continue statements at line 1510 and 1518).

So let us now consider two cases:

1. line 1524 is reached with \text{state} to 0 or 1:
   • \text{nbits} == 0 and \text{wbits}=0 as seen above,
   • \text{nbits} is incremented to 1 at line 1525,
   • then after line 1527, \text{wbits} == (1 << (\text{wsize} - 1)) (since it was 0)
   • both \text{exp_a_wbits_min} and \text{exp_a_wbits_max} are established.

2. line 1524 is reached with \text{state}==2:
   • \text{wbits} has the value it took the previous time line 1527 was executed,
   • one bit may be added to \text{wbits} according to \text{ei},
   • \text{exp_a_wbits_min} is preserved because:
     – \text{wbits} can only grow or stay the same due to the or operation,
     – so $(1 << (\text{wsize} - 1)) \leq \text{wbits}$ is preserved.
   • \text{exp_a_wbits_max} is preserved because:
     – the number of the added bit is $(\text{wsize} - \text{nbits})$
     – because $1 \leq \text{nbits} \leq \text{wsize}$ as seen above, this number is smaller than $(\text{wsize} - 1)$
     – so \text{wbits} stays smaller than $(1 << \text{wsize})$, and \text{wbits} < $(1 << \text{wsize})$ is preserved.

So the properties \text{exp_a_wbits_min} and \text{exp_a_wbits_max} are both true.

A.3.15 Review of \text{exp_a8_bufsize_max} *

Used in:

• mpi_exp_mod Intermediate Annotations (§9.9.7)

Property to check:

```
RSA/mpi_exp_mod_rsa.acsl

142 at L_exp_a8: assert exp_a8_bufsize_max_rv: bufsize < 32;
```

Source code:

```
RSA/bignum.c
```
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

```
bufsize = 0;

while( 1 )
{
    if( bufsize == 0 )
    {
        bufsize = sizeof( t_uint ) << 3;
    }
    bufsize--;
    L_exp_a8:
```

- Line 1487: bufsize is initialized to 0,
- Line 1499: when bufsize reaches 0, it is assigned to 32,
- Line 1502: bufsize is always decremented,
- Line 1503: so bufsize < 32 is always true.

So the property exp_a8_bufsize_max is true.

A.3.16 Review of exp_a9_valid *

Used in:

- mpi_exp_mod Intermediate Annotations (§9.9.7)

Property to check:

```
RSA/mpi_exp_mod_rsa.acsl

at L_exp_a9:
assert exp_a9_valid_rv: \valid (W[(1 << wsize-1)..(1 << wsize)-1].p);
```

Source code:

```
RSA/bignum.c

L_exp_a7: nblims = E->n;
... nothing about W...
while( 1 )
{
    ... nothing about W...
    L_exp_a9: mpi_montmul( X, &W[wbits], N, mm, &T );
    ... nothing about W...
}
```

- the property is established at label L_exp_7 by exp_a7:

```
RSA/mpi_exp_mod_rsa.acsl

at L_exp_a7: assert exp_a7_val: \valid (W[(1 << wsize-1)..(1 << wsize)-1].p);
```

- the property is preserved in the loop since W[..].p is not modified in the loop.

So the property exp_a9_valid is true.
A.3.17  Review of \texttt{exp\_a9\_wbits} *

Used in:

- mpi\_exp\_mod Intermediate Annotations (§9.9.7)

Property to check:

\begin{verbatim}
RSA/mpi_exp_mod_rsa.acsl

at L\_exp\_a9:
assert \texttt{exp\_a9\_wbits\_rv}: (1 \ll \texttt{wsize} - 1) \leq \texttt{wbits} < (1 \ll \texttt{wsize});
\end{verbatim}

Source code:

\begin{verbatim}
RSA/bignum.c

L\_exp\_wbits:
if ( \texttt{nbits} == \texttt{wsize} )
{
... no modification of \texttt{wsize} or \texttt{wbits}...

L\_exp\_a9: mpi\_montmul( \texttt{X}, &\texttt{W[\texttt{wbits}]}, \texttt{N}, mm, &\texttt{T} );
\end{verbatim}

The property is ensured at label \texttt{L\_exp\_wbits} by \texttt{exp\_a\_wbits\_min} and \texttt{exp\_a\_wbits\_max}:

\begin{verbatim}
RSA/mpi_exp_mod_rsa.acsl

at L\_exp\_wbits: assert \texttt{exp\_a\_wbits\_min\_rv}: (1 \ll \texttt{wsize} - 1) \leq \texttt{wbits};
at L\_exp\_wbits: assert \texttt{exp\_a\_wbits\_max\_rv}: \texttt{wbits} < (1 \ll \texttt{wsize});
\end{verbatim}

Since neither \texttt{wbits} nor \texttt{wsize} are not modified between the label \texttt{L\_exp\_wbits} and the label \texttt{L\_exp\_a9}, the properties are still true at label \texttt{L\_exp\_a9}.

So the property \texttt{exp\_a9\_wbits} is true.

A.3.18  Review of \texttt{exp\_a5a} and \texttt{exp\_a5b} ***

Used in:

- mpi\_exp\_mod Intermediate Annotations (§9.9.7)

Properties to check:

\begin{verbatim}
RSA/mpi_exp_mod_rsa.acsl

at L\_exp\_a5: assert \texttt{exp\_a5\_a\_rv}: 0 < \texttt{W[1].n} \leq \texttt{mpi\_max\_limbs};
at L\_exp\_a5: assert \texttt{exp\_a5\_b\_rv}: \texttt{W[1].s} == -1 \lor \texttt{W[1].s} == 1;
\end{verbatim}

Source code:

\begin{verbatim}
RSA/old_bignum.c

L\_exp\_a1:
... nothing about \texttt{W[1]}...

if( mpi\_cmp\_mpi( A, N ) >= 0 )
  mpi\_mod\_mpi( &\texttt{W[1]}, A, N );
else  mpi\_copy( &\texttt{W[1]}, A );
L\_exp\_a5:
\end{verbatim}
The calls to \texttt{mpi\_mod\_mpi} and \texttt{mpi\_copy} may either succeed (return 0) or fail (return something else). Since the error management macro \texttt{MPI\_CHK} is not used here (see \textit{MPI error management} §9.4.3), both cases have to be examined:

1. the calls succeed:
   - for a successful \texttt{mpi\_copy}:
     - \texttt{\texttt{exp\_a5a}} is ensured by the \texttt{mpi\_copy} postcondition:

   \begin{verbatim}
   RSA/bignum.acsl
   ensures cp_e2_wp: \result == 0 \implies 1 <= X->n <= mpi_max_limbs;
   \end{verbatim}

   - and \texttt{exp\_a5b} is ensured by the \texttt{mpi\_copy} postconditions:

   \begin{verbatim}
   RSA/bignum.acsl
   requires cp_rY3: Y->s == 1 || Y->s == -1;
   ensures cp_e5_wp: \result == 0 \implies X->s == Y->s;
   \end{verbatim}

     • for a successful \texttt{mpi\_mod\_mpi}:

   \begin{verbatim}
   RSA/bignum.acsl
   ensures mod_eR1_rv: \result == 0 \implies 1 <= R->n <= mpi_max_limbs;
   ensures mod_eR2_rv: \result == 0 \implies -1 == R->s || 1 == R->s;
   \end{verbatim}

   2. the calls fail:
   - it cannot be the case for \texttt{mpi\_copy} since:

   \begin{verbatim}
   RSA/mpi/mpi_exp_mod_rsa.acsl
   at L_exp_a1: assert exp_a1c_val: W[1].p != \null;
   \end{verbatim}

   - the postcondition \texttt{cp\_e6} then ensures that \texttt{\result == 0}:

   \begin{verbatim}
   RSA/bignum.acsl
   ensures cp_e6_val: \old (X->p) != 0 \implies \result == 0;
   \end{verbatim}

   • for \texttt{mpi\_mod\_mpi}, the possible returned values are given by the postcondition \texttt{mod\_e\_res}:

   \begin{verbatim}
   RSA/bignum.acsl
   ensures mod_e_res_val: \result == 0
   || | \result == tis_POLARSSL_ERR_MPI_MALLOC_FAILED
   || | \result == tis_POLARSSL_ERR_MPI_DIVISION_BY_ZERO
   || | \result == tis_POLARSSL_ERR_MPI_NEGATIVE_VALUE;
   \end{verbatim}

   - \texttt{POLARSSL\_ERR\_MPI\_NEGATIVE\_VALUE}: this error occurs when \texttt{mpi\_mod\_mpi} is called with a negative divisor. In this context, the divisor is the input number \texttt{N} that has been tested by:

   \begin{verbatim}
   RSA/bignum.c
   if( mpi_cmp_int( N, 0 ) < 0 || ( N->p[0] & 1 ) == 0 )
   return( POLARSSL\_ERR\_MPI\_BAD\_INPUT\_DATA );
   \end{verbatim}

   so \texttt{N} is known to be positive (and odd).
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

- **POLARSSL_ERR_MPI_DIVISION_BY_ZERO**: this error occurs when the division is called with a null divisor. Here again, the divisor is the input number N that has been tested by the test above that ensures that N cannot be 0 (otherwise, the second part of the test would have be true).

- **POLARSSL_ERR_MPI_MALLOC_FAILED**: as there are some allocations of local numbers, at least in mpi_div_mpi that is called at the very beginning of mpi_mod_mpi, so it seems that **this error can be raised**!

The code has been patched to protect the call to mpi_mod_mpi:

```c
RSA/bignum.c

1449 if( mpi_cmp_mpi( A, N ) >= 0 )
1450   MPI_CHK( mpi_mod_mpi( &W[1], A, N ) );
1451 else mpi_copy( &W[1], A );
```

Subsequently, according to the above explanations, the properties are then ensured.

The verification of the MPI sub-component is made under the assumption that the call to mpi_mod_mpi is protected by MPI_CHK.

So the properties **exp_a5a** and **exp_a5b** are both true.

**A.3.19 Review of mod_eR1 and mod_eR2**

Used in:

- mpi_exp_mod Intermediate Annotations (§9.9.7)

Properties to check:

```c
RSA/bignum.acsl

66 ensures mod_eR1_rv: \result == 0 ==> 1 <= R->n <= mpi_max_limbs ;
68 ensures mod_eR2_rv: \result == 0 ==> -1 == R->s || 1 == R->s ;
```

Source code:

```c
RSA/bignum.c

L_mod_a1:
cleanup:
  return ( ret );
```

Because of the error management (see §9.4.3), the only path to exit the function with \result == 0 is to come from the label **L_mod_a1** where these properties are ensured by:

```c
RSA/bignum.acsl

69 at L_mod_a1: assert mod_aR1_val: (1 <= R->n <= mpi_max_limbs ) ;
70 at L_mod_a1: assert mod_aR2_val: -1 == R->s || 1 == R->s ;
```

So the properties **mod_eR1** and **mod_eR2** are both true.
A.3.20 Review of \texttt{mul\_a1} and \texttt{mul\_a2} \textsuperscript{*}

Used in:
\begin{itemize}
  \item \texttt{mpi\_mul\_mpi} Intermediate Annotations (§9.7.7)
\end{itemize}

Properties to check on the source code:

\begin{verbatim}
RSA/bignum.c

1011 for( i = A->n; i > 0; i-- )
1012 if( A->p[i - 1] != 0 ) break;
1014 L_mul_a1 : //@ assert mul_a1: i == TIS_ia;
1016 for( j = B->n; j > 0; j-- )
1018 if( B->p[j - 1] != 0 ) break;
1020 L_mul_a2 : //@ assert mul_a2: j == TIS_ib;

\end{verbatim}

The loops compute:
\begin{itemize}
  \item \( i = \text{MSD}(A) \)
  \item \( j = \text{MSD}(B) \)
\end{itemize}
(see the definition of MSD §9.4.1).

The two assertions \texttt{mul\_a1} and \texttt{mul\_a2} are only relevant when the global variables \texttt{TIS\_ia} and \texttt{TIS\_ib} have been set, i.e. in the \texttt{mpi\_mul\_mpi} analysis context. In that case, the context is such that \texttt{TIS\_ia} has been set to \text{MSD}(A) and \texttt{TIS\_ib} has been set to \text{MSD}(B) (see §9.7.2).

So the properties \texttt{mul\_a1} and \texttt{mul\_a2} are both true.

A.3.21 Review of \texttt{mul\_a3} \textsuperscript{*}

Used in:
\begin{itemize}
  \item \texttt{mpi\_mul\_mpi} Intermediate Annotations (§9.7.7)
\end{itemize}

Property to check:

\begin{verbatim}
RSA/bignum.acsl

116 at L_mul_a: assert mul_a3_rv: i+j <= mpi_max_limbs;

\end{verbatim}

Source code:

\begin{verbatim}
RSA/bignum.c

MPI_CHK( mpi_grow( X, i + j ) );
MPI_CHK( mpi_lset( X, 0 ) );
L_mul_a:

\end{verbatim}

This property is ensured by \texttt{mpi\_grow} postcondition:

\begin{verbatim}
RSA/mpi_spec.h

35 ensures ga_e5_wp: nblimbs > mpi_max_limbs ==> \result != 0;

\end{verbatim}

which is equivalent to: \( \text{\texttt{result}} = 0 \Rightarrow \text{nblimbs} \leq \text{mpi\_max\_limbs} \);
because the call to `mpi_grow` is protected by `MPI_CHK` (see MPI error management §9.4.3), the label `L_mul_a` is reached only when `mpi_grow` returns 0, in this call context: `nblims == i+j`.

So the property `mul_a3` is true.

**A.3.22 Review of `mulh_a3` ****

Used in:
- `mpi_div_mpi` Intermediate Annotations (§9.8.7),
- `mpi_exp_mpi` Intermediate Annotations (§9.9.7).

Property to check:

```makefile
RSA/bignum.acsl
233
at L_mulh_a3: assert mulh_a3_rv: \valid (d);
```

Source code:

```c
RSA/bignum.c
L_mulh_a1: ;
do { L_mulh_a3: ; t_uint tis_memo_d = *d;
    *d += c; c = ( *d < c ); L_mulh_a2: d++;
    } while( c != 0 );
}
```

The following property gives the value of `d` at label `L_mulh_a1`, i.e. when entering the loop:

```makefile
RSA/bignum.acsl
232
at L_mulh_a1: assert mulh_a1_wp: d == \at(d,Pre) + \at(i,Pre);
```

The following property ensures that when `*d == 0` at label `L_mulh_a3`, then the loop exits:

```makefile
RSA/bignum.acsl
234
at L_mulh_a2: assert mulh_a2_wp: tis_memo_d == 0 ==> c == 0;
```

So, the property is true if there is an index k ≥ i such that *(d+k) == 0 and d is valid between i and k.

The `mpi_mul_hlp` function is called:
- from `mpi_mul_mpi`,
- and from `mpi_montmul`.

**In the multiplication context**

In the context of the `mpi_mul_mpi` analysis, the property has been formally verified by the analyzer.
In the Montgomery multiplication context

In the context of \texttt{mpi\_montmul}, the source code is:

```c
1336  memset( T->p, 0, T->n * ciL );
1337  d = T->p;
1338  n = N->n;
1339  m = ( B->n < n ) ? B->n : n;
1340  L_montm_a1: ;
1341    for( i = 0; i < n; i++ )
1342      { 
1343        mpi_mul_hlp( m, B->p, d, u0 );
1344        mpi_mul_hlp( n, N->p, d, u1 );
1345        *d++ = u0; d[n + 1] = 0;
1346      } 
1347  mpi_mul_hlp( m, B->p, d, u0 );
1348  mpi_mul_hlp( n, N->p, d, u1 );
1349  *d++ = u0; d[n + 1] = 0;
```

Annotations ensure that:

```plaintext
RSA/bignum.acsl
79  requires montm_rN1: N->n == 32;
80  requires montm_rT1: T->n == 66;
81  requires montm_rT2: \valid(T->p+(0 .. 65));
82  at L_montm_a1: assert montm_a_n_val: n == 32;
83  at L_montm_a1: assert montm_a_m_wp: m <= 32;
84  at loop 1:
85    loop invariant montm_l1_2_val: 0 <= i <= 32;
86    loop invariant montm_l1_1_wp: d == T-> p + i;
```

Remember that the \texttt{mulh\_a3} property is ensured in \texttt{mpi\_mul\_hlp} if there exists an index \( k \geq i \) such that \( *(d+k) == 0 \) and \( \valid(d+(i+1..k)) \), where \( i \) is the first parameter of \texttt{mpi\_mul\_hlp}. Let us call it \( i' \) here to avoid confusion with the local \( i \).

In the context of \texttt{mpi\_montmul}, the condition is satisfied for \( k=33 \) for both calls:

- \( k \geq i' \) since:
  - \( k == 33 \)
  - and \( i' \leq 32 \) because:
    - \( n \) and \( m \) are not modified in the loop so:
      - \( n == 32 \) (preservation of \texttt{montm\_a\_n})
      - \( m \leq 32 \) (preservation of \texttt{montm\_a\_m\_wp})
    - so, for both \texttt{mpi\_mul\_hlp} calls, the first parameter is at most 32.
- \( \valid(d+(i'+1..33)) \), because \( \valid(d+(0..33)) \) and \( i' \leq 32 \) since:
  - \( d == T->p + i \) with \( 0 \leq i < 32 \) so \( T->p \leq d < T->p+32 \)
  - so \( T->p \leq d+(0..33) < T->p+65 \)
  - and \texttt{montm\_rT2} ensures that \( \valid(T->p+(0..65)) \).
- \( *(d+33) == 0 \) since:
  - for the first iteration,
    - \( d == T->p \)
    - \( T->[0..65] == 0 \) because of the call to \texttt{memset} at line 1336,
  - for other iterations:
    - \( d[n+1] \) is set to 0 at the end of the previous iteration (line 1353),
    - \( n==32 \),
A. Intellectual Analyses

A.3. MPI Intellectual Analyses

* so d[33] == 0.

So the property muth_a3 is true for every calls to mpi_mul_hlp.

A.3.23 Review of shl_a2a *

Used in:

- mpi_div_mpi Intermediate Annotations (§9.8.7),

Property to check:

```
RSA/bignum.acsl

at L_shl_2: assert shl_a2a_rv: aux > 0;
```

Source code:

```
RSA/bignum.c

L_shl_1:for( i = X->n; i > v0; i-- )
{
  int aux = i-v0; L_shl_2: L_shl_5: X->p[i - 1] = X->p[aux - 1]; }
```

The property aux>0 is ensured at label L_shl_2 because:

- the loop condition i>v0 is true at label L_shl_2,
- so i - v0 > 0,
- and because aux = i-v0, then aux>0.

So the property shl_a2a is true.

A.3.24 Review of shl_a2b **

Used in:

- mpi_div_mpi Intermediate Annotations (§9.8.7),

Property to check:

```
RSA/bignum.acsl

at L_shl_2: assert shl_a2b_rv: mpi_max_limbs >= aux;
```

Source code:

```
RSA/bignum.c

L_shl_1:for( i = X->n; i > v0; i-- )
{
  int aux = i-v0; L_shl_2: L_shl_5: X->p[i - 1] = X->p[aux - 1]; }
```

1. aux ≤ X->n-v0 because:
   - when entering the loop: i == X->n
   - i decreases (but stays positive), so i ≤ X->n
   - so i - v0 ≤ X->n-v0
   - and aux = i - v0

2. X->n-v0 ≤ mpi_max_limbs because:
   - shl_a6 ensures that X->n ≤ mpi_max_limbs
A.3. MPI Intellectual Analyses

A.3.25 Review of subh_l1_4 **

Used in:

- mpi_div_mpi Intermediate Annotations (§9.8.7),

Property to check:

**RSA/bignum.acsl**

```plaintext
loop invariant subh_l1_4 rv: c == 0 || c == 1;
```

Source code:

**RSA/bignum.c**

```c
for( i = c = 0; i < n; i++, s++, d++ )
{
  ;
L_subh_1: z = (*d < c ); *d -= c;
L_subh_2: c = (*d < *s ) + z; *d -= *s;
}
```

1. the loop invariant is established since c==0.
2. for the preservation, either c==0 or c==1:
   (a) c==0:
      - 785: z==0 (since *d is always positive)
      - 785: z==0 => (c == 0 || c == 1)
      - the property is preserved.
   (b) c==1: at label L_subh_1, either *d >= 1 or *d == 0:
      i. *d >= 1, so again z==0 and the property is preserved,
      ii. *d == 0:
         - 785: *d -= c; with *d == 0 and c==1,
         - 786: *d == 0x00000000 so (*d < *s ) is 0,
         - 786: c == 0 + z with z == 0 || z == 1 so c == 0 || c == 1 and the property is preserved.

So the property subh_l1_4 is true.

A.3.26 Review of subh_l2_1 ****

Used in:

- mpi_add_mpi Intermediate Annotations (§9.5.7),
- mpi_sub_mpi Intermediate Annotations (§9.6.7),
• mpi_div_mpi Intermediate Annotations (§9.8.7),
• mpi_exp_mpi Intermediate Annotations (§9.9.7).

Property to check:

**RSA/bignum.acsl**

```plaintext
at loop 2:
  loop invariant subh_l2_1 rv: i <= mpi_max_limbs;
```

Source code:

**RSA/bignum.c**

```c
static void mpi_sub_hlp( size_t n, t_uint *s, t_uint *d )
{
    size_t i;
    t_uint c, z;
    for( i = c = 0; i < n; i++, s++, d++ )
    {
        L_subh_1: z = ( *d < c ); *d -= c;
        L_subh_2: c = ( *d < *s ) + z; *d -= *s;
    }
    while( c != 0 )
    {
        z = ( *d < c ); *d -= c;
        c = z; i++; d++;
    }
}
```

Informal requirements

Let us consider two MPI numbers B and X, and let’s first assume that:

• X ≥ B,
• s == B->p,
• d == X->p,
• n == MSD(B).

This function computes the subtraction of number B from number X. The first loop processes all the digits of B (0 ≤ i < n) together with the similar elements in X. Then, the second loop propagate the carry in d (X), and one must prove that the carry becomes null before d reaches the X bound.

Establishment of the loop invariant

**RSA/bignum.acsl**

```plaintext
at loop 1:
  loop invariant subh_l1_1 val: 0 <= i <= mpi_max_limbs;
```

The property subh_l1_1 on the loop 1 ensures that subh_l2_1 is established before the loop 2.
Preservation of the loop invariant

```plaintext
RSA/bignum.acsl

at loop 1:

loop invariant subh_l1_2_wp: d == \at(d,Pre) + i;

loop invariant subh_l1_4_rv: c == 0 || c == 1;
```

- At line 788: the loop invariant subh_l1_4 ensures that \( c \) is 0 or 1
  1. if \( c==0 \), the loop invariant holds since the second loop condition is false.
  2. if \( c==1 \), it must be the case that \( X \) has more digits than \( B \), because otherwise, there can’t be any carry since \( X \geq B \). It means that:
     \[
     \exists k, 100 > k \geq n \implies X[k] > 0 \&\& \forall i, k > i \geq n \implies X[i]==0
     \]
     The carry stays at 1 as long as \( X[i]==0 \), but then \( c == 0 \) when \( k \) is reached, and that must happen before mpi_max_limbs since \( X \) is a valid allocated number.

Calling context

Let us now check the informal requirements assumed before. The function \texttt{mpi_sub_hlp} is always called from \texttt{mpi_sub_abs}:

```plaintext
RSA/bignum.c

if( mpi_cmp_abs( A, B ) < 0 )
    return( POLARSSL_ERR_MPI_NEGATIVE_VALUE );

if( X != A )
    MPI_CHK( mpi_copy( X, A ) );

for( n = B->n; n > 0; n-- )
    if( B->p[n - 1] != 0 )
        break;

mpi_sub_hlp( n, B->p, X->p );
```

- 805-806: an error is returned when \( A<\text{B} \), so \( A\geq\text{B} \).
- 816-817: \( A \) is copied into \( X \) (if it is not already in it).
- 927-929: \( n \) is computed such that:
  \[ \forall i, B->n < i \leq n \implies B[i] = 0 \text{ and } B[n-1] != 0 \]
  so \( n = \text{MSD}(B) \) (definition).
- 830: \texttt{mpi_sub_hlp} is called with \( s \) is \( B->p \) and \( d \) is \( X->p \).

Conclusion

- the property is true when some requirements are satisfied,
- the requirements are satisfied at the only call site,

So the property subh_l2_1 is true.
A.4. RSA Intellectual Analyses

A.4.1 Review of rsa_e1 ***

Used in:
- rsa_private Output Properties (§10.4.4).

Property to check:

```c
ensures rsa_e1: \result < 0
|| (\result == 0 && initialized(output+(0..ctx->len-1)));
```

Source code:

```c
MPI_free( &T ); MPI_free( &T1 ); MPI_free( &T2 );
if( ret != 0 )
  return( POLARSSL_ERR_RSA_PRIVATE_FAILED + ret );
return( 0 );
```

Because of MPI error management (§9.4.3), the cleanup label at line 296 can be reached:
- either by indirect paths where \( ret != 0 \),
- or by the direct way (from line 295).

**Case 1**

In this case:
- line 297: \( ret != 0 \)
- line 297: \( ret < 0x4300 \) because of the following property:

```c
assert rsa_a1_val: ret == 0 || ret < 0x4300;
```

- line 300: the test is true since \( ret != 0 \),
- line 301: the function returns:
  - POLARSSL_ERR_RSA_PRIVATE_FAILED + ret
  - which is equal to \( (-0x4300) + ret \)
  - which is strictly negative since \( ret < 0x4300 \).

Then the first part of the disjunction \( \result < 0 \) is true in this case.

**Case 2**

In this case:
- line 297: \( ret == 0 \),
• line 300: the test is false,
• line 303: the function returns 0,

So (\texttt{result == 0}) is ensured.

As already said, the only way to reach the cleanup label with \texttt{ret == 0} is the direct one, i.e. coming from line 295.

\begin{verbatim}
RSA/rsa.c

    olen = ctx->len;
    MPI_CHK( mpi_write_binary( &T, output, olen ) );

cleanup:

\end{verbatim}

Again because of the \texttt{MPI_CHK} macro (see \textit{MPI error management §9.4.3}), the line 295 is reached only if the call to \texttt{mpi_write_binary} returns 0.

The postcondition \texttt{wb_1} of \texttt{mpi_write_binary} states that:

\begin{verbatim}
RSA/bignum.acsl

    ensures wb_1_val: \texttt{result == 0} \implies \texttt{initialized(buf+(0..buflen-1))};

\end{verbatim}

So, in the call context:

• line 295: it ensures \texttt{initialized(output+(0 .. olen-1))},
• and line 293 gives that \texttt{olen == ctx->len},

so \texttt{initialized(output+(0 .. ctx->len-1))} is ensured.

Then the second part of the disjunction is true in this case, because both:

• \texttt{result == 0},
• and \texttt{initialized(output+(0 .. ctx->len-1))}

are verified.

So the property \texttt{rsa_e1} is true in all cases.
B. External Library Functions

PolarSSL relies on functions from the C standard library. The specifications used for these functions during the verification of PolarSSL can be found below. These specifications formalize the natural-language definition found in the C standard.

In addition, the C code written for the purpose of the verification relies on functions specific to the Frama-C framework. The formal specifications of these functions can be found here, together with a summary of their effects.

B.1. Specification of `free`

```
/* frees p;
@ assigns _fc_heap_status from _fc_heap_status;
@ behavior deallocation:
@ assumes p!=null;
@ requires freeable
@ ensures allocable(p);
@ behavior no_deallocation:
@ assumes p==null;
@ assigns nothing;
@ frees nothing;
@ complete behaviors;
@ disjoint behaviors;
*/
void free(void *p);
```

B.2. Specification of `malloc`

```
/* allocates result;
@ assigns _fc_heap_status from size, _fc_heap_status;
@ assigns result from size, _fc_heap_status;
@ behavior allocation:
@ assumes allocable(size);
@ assigns _fc_heap_status from size, _fc_heap_status;
@ assigns result from size, _fc_heap_status;
@ ensures fresh(result,size);
@ behavior no_allocation:
@ assumes !allocable(size);
@ assigns result from nothing;
@ allocates nothing;
@ ensures result==null;
@ complete behaviors;
@ disjoint behaviors;
*/
void *malloc(size_t size);
```
B. External Library Functions

B.3. Specification of `memcmp`

```c
/* requires \valid_read(((char*)s1)+(0..n - 1));
@ requires \valid_read(((char*)s2)+(0..n - 1));
@ assigns \result \from ((char*)s1)[0.. n-1], ((char*)s2)[0.. n-1];
@ ensures \result == memcmp((char*)s1,(char*)s2,n);
*/
extern int memcmp (const void *s1, const void *s2, size_t n);
```

B.4. Specification of `memcpy`

```c
/* requires valid_dst: \valid(((char*)dest)+(0..n - 1));
@ requires valid_src: \valid_read(((char*)src)+(0..n - 1));
@ requires \separated(((char *)dest)+(0..n-1),((char *)src)+(0..n-1));
@ assigns ((char*)dest)[0..n - 1] \from ((char*)src)[0..n-1];
@ assigns \result \from dest;
@ ensures memcmp((char*)dest,(char*)src,n) == 0;
@ ensures \result == dest;
*/
extern void *memcpy(void *restrict dest,
                    const void *restrict src, size_t n);
```

B.5. Specification of `memmove`

```c
/* requires valid_dst: \valid(((char*)dest)+(0..n - 1));
@ requires valid_src: \valid_read(((char*)src)+(0..n - 1));
@ assigns ((char*)dest)[0..n - 1] \from ((char*)src)[0..n-1];
@ assigns \result \from dest;
@ ensures memcmp((char*)dest,(char*)src,n) == 0;
@ ensures \result == dest;
*/
extern void *memmove(void *dest, const void *src, size_t n);
```

B.6. Specification of `memset`

```c
/* requires \valid(((char*)s)+(0..n - 1));
@ assigns ((char*)s)[0..n - 1] \from c;
@ assigns \result \from s;
@ ensures memset((char*)s,c,n);
@ ensures \result == s;
*/
extern void *memset(void *s, int c, size_t n);
```
B. External Library Functions

B.7. Specification of strlen

```
libc/string.h

/*@ requires valid_string_src: valid_string(s);
@ assigns \result \from s[0..];
@ ensures \result == strlen(s);
@Resource
extern size_t strlen (const char *s);
```

B.8. Specification of time

```
libc/time.h

/*@ requires *timeptr, \result \from *timeptr; */
time_t mktime(struct tm *timeptr);
```

B.9. Specification of Frama_C_make_unknown

The function call Frama_C_make_unknown(p, l) puts arbitrary contents in the `l` bytes of memory starting at the address `p`.

```
libc/__fc_builtin.h

/*@ requires \valid(p + (0 .. l-1));
assigns p[0 .. l-1] \from Frama_C_entropy_source;
assigns Frama_C_entropy_source \from Frama_C_entropy_source;
ensures \initialized(p + (0 .. l-1));
@Resource
void Frama_C_make_unknown(char *p, size_t l);
```

B.10. Specification of Frama_C_interval

The function call Frama_C_interval(l, u) returns an arbitrary integer between `l` and `u` inclusive.

```
libc/__fc_builtin.h

/*@ assigns \result \from min, max, Frama_C_entropy_source;
assigns Frama_C_entropy_source \from Frama_C_entropy_source;
ensures min <= \result <= max ;
@Resource
int Frama_C_interval(int min, int max);
```

B.11. Specification of Frama_C_nondet

The function call Frama_C_nondet(a, b) returns an arbitrary choice of `a` or `b`.

```
libc/__fc_builtin.h

/*@ assigns \result \from a, b, Frama_C_entropy_source;
```
B. External Library Functions

B.11. Specification of Frama_C_nondet

```c
# assign Frama_C_entropy_source \from Frama_C_entropy_source;
# ensures \result == a || \result == b ;
/
int Frama_C_nondet(int a, int b);
```
C. Definitions

Alarms: properties emitted by the analyzer that express conditions necessary for the good behavior of the component. Alarms are justified in this document either formally or intellectually, in order to guarantee that none of the software weaknesses listed in §3 is present.

Coverage analysis: review of the structural coverage reported by the analyzer. Uncovered statements are dead code and are reviewed to check that no misconfiguration caused security-related code to be omitted from the scope of the verification.

CWE: The Common Weakness Enumeration is a dictionary of common software weaknesses maintained by the MITRE Corporation.

Formally Verified property: a property that is guaranteed to hold by one of the formal verification tools used.

FV/V/U: in analysis summaries, indicates that the numbers are for Formally Verified properties, Verified properties and Unchecked properties respectively.

Global quality: indicates the level of confidence of part or all of the study:
- Formal Trust: security property formally verified,
- Semi-formal Trust: all alarms are reviewed,
- Basic Trust: at least one alarm within the perimeter is not reviewed.

Guaranteed properties: properties proved by the analysis to hold about the results of the component. These include post-conditions (expressed with the ensures keyword) and dependencies (expressed with the assigns keyword in the specifications of functions in the component’s API).

Guarantees Perimeter: indicates the context in which the analysis is done. The results are only valid for use cases captured by this context.

indirect: in the dependencies computed with option -deps or hand-written as the right-hand-side of an assigns clause, the indirect dependency \( y \text{ \textbackslash from indirect: } x \) indicates that variable \( x \) influences the result through control (e.g. if \( (x) \ y = 3; \)) or through an address computation (e.g. \( y = t[x]; \)), as opposed to a direct dependency \( y = x + 2; \).

Internal properties: properties that refer to the insides of the component, as opposed to the Required properties and Guaranteed properties that define its interfaces. These properties are part of the analysis and, like Alarms, are justified either formally or intellectually.

LOC: unit of measure of code size. Expressed in number of elementary instructions as counted by TrustInSoft Analyzer.

Main context size to audit: size of the main function that was written for the purpose of the analysis. The user of the component should make sure that this function captures the use intended for the component.

Required properties: properties that the user of the sub-component must respect when it invokes it. These are expressed with the requires keyword. The analysis of each sub-component is made under the assumption that these properties hold.

Sub-component: a delimited part of the component that is analyzed independently, with respect to provided specifications.

Unchecked property: a property for which no explanation is provided for why it should hold.

Verified property: a property that is guaranteed to hold by a rigorous argument in natural language.

V/U: in analysis summaries, indicates that the numbers are for Verified properties and Unchecked properties respectively.

http://cwe.mitre.org/