Security Aspects in Software Development
KU 2012/2013
"C4 - Registry"

Daniel Hein Johannes Winter Moritz Lipp
Sebastian Ramacher
Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8020 Graz, Austria
sicherheitsaspekte@iaik.tugraz.at

December 29, 2012; Revision: v1.0

Contents

1 Introduction

Assignment title:	The registry (C4)
Group size:	2 students
Start date:	December 5, 2012
Maximum score:	25 points (25% percent of final* mark)
HARD SUBMISSION DEADLINE:	**January 9, 2012 (23:59:59 CET)**

See Section 3 for details on the submission process.
See Section 4 for the subtasks and questions of this assignment.

* Bar the oral exam.

1.1 Motivation

In this assignment the parts from the previous assignments are put together to implement a registry service. The registry will use an SQL database backend (as in assignment WS) and the bpack/bunpack functions from C3 are used to put the communication between a client and a server into practice.

As in assignment WS, the service might be used as a backend for a web application or as a publicly available data service on the Internet. To prevent malicious usage of the registry, it has to be protected against attacks like *SQL injection*, or buffer overflows that might interrupt the service. Furthermore, it is important to protect it against path traversal, and other attacks that could reveal sensitive information.

Since many publicly available services are written in C (e.g. HTTP servers, database servers) it is necessary to deal with this issues in lower level languages as well.

1.2 Objectives

The goal of this exercise is to learn how to

- implement access to an SQL database in C. This implementation must be protected against SQL injections.
- apply HMACs to ensure data integrity and authenticity of messages transmitted over a potentially insecure channel.
- find and fix path traversal attacks.
- find and fix format strings attacks.

1.3 Registry

The registry allows you to store, retrieve and search (key, value) pairs.

The registry is indexed by a (domain, key) pair and holds `int64_ts`, `doubles`, NUL-terminated C-strings and binary blobs.

The registry consists of multiple components: *libregistry*, *server* and *communication*. The user of the registry uses the functions provided by *libregistry* to get and set data from the registry. *server* implements the registry backend. *communication* is used by *libregistry* and *server* to talk to each other.

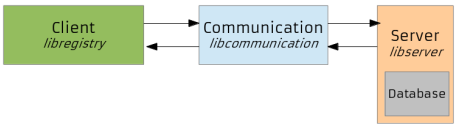

Figure 1: The registry

1.4 Plagiarism

⚠ *Plagiarism will not be tolerated and any contributions containing copied code will automatically receive a negative rating. It is explicitly allowed to use material and code snippets shown during the lecture or the exercises.*

2 Software requirements

The reference platform for this exercise is the VirtualBox[7] image[1], which can be found in the download section of the course website.

This image is based on a x86 platform running a recent 3.2-series Linux kernel. As reference compiler we use GCC 4.7 [2] in combination with a recent version of the GNU C library (`2.16` or newer). The course website contains a 32-bit and a 64-bit version of the reference image.

The GNU debugger `GDB`[3], `valgrind`[6] and a couple of other useful tools are already installed on the reference image.

⚠ *It is mandatory to use a correctly configured GIT installation in this task. In particular, this includes a properly configured name and e-mail address to be used by GIT as commit author information. Refer to the setup instructions of the "Web Security" (WS) task for more details on configuring this information.*

⚠ *It is mandatory to test your programs with `valgrind` in order to eliminate* all *invalid memory accesses (reads/writes) and uses of uninitialized values. Submissions, which contain these kinds of errors will receive a significantly lower rating.*

External libraries For this assignment your submission *MUST NOT* use any external libraries (and code) like string libraries, logging libraries, ... without explicit permissions. It must be possible to successfully compile and run your programs on a system where no libraries apart from the C runtime library are installed.

You must use ısqlite for this assignment and therefore it is excluded from the above mentioned rule.

See Section 2.3 for more details on test code.

C language and coding standards Any source code written for this task *MUST* follow the ISO-C99 [8] standard as close as possible. Your programs *MUST NOT* not use any compiler or runtime library specific features[2].

We do *not* demand use of any particular coding standard; it is your own responsibility to agree on a consistent and readable coding standard within your group. However, we demand that your programs are written and formatted in a consistent way (for example, we do not want one group member to use tabs for indentation while the rest uses spaces, ...).

[1] In Open Virtualization Format, see `http://www.virtualbox.org/manual/ch01.html#ovf`
[2] like, for example the "secure CRT functions" provided by certain version of Microsoft's Visual C++ compiler.

Your application must compile under GCC[2] in pedantic C99-mode[8] with all warnings and minimal optimizations enabled (`-g -std=c99 -pedantic -Wall -Wno-unused-parameter -O1`) without causing any compiler errors or warnings.

32-bit and 64-bit portability We expect your source code to be fully portable between 32-bit and 64-bit environments. The Makefile shipped with this assignment specification is already configured to build a 32-bit *and* and 64-bit variant of your code.

The requirements regarding language standard, warnings and errors mentioned above are valid for 32-bit *and* 64-bit builds.

Memory leaks All memory that has been allocated by your implementation must be freed.

2.1 Source code documentation

It is mandatory that the delivered source code is compatible with the doxygen [1] tool. Each function *SHOULD* have a general description of its purpose, its parameters and its return values. It is not necessary to duplicate the description of a particular function in the source and the header file. Each header and source file *SHOULD* start with a brief comment describing the intended purpose of the file.

2.2 Configuring the build system

To simplify your task when writing your assignment and our task when testing your code, we have prepared a simple Makefile based build system in the `c4` subdirectory of your repositories. This build-system consists of a main Makefile `c4/Makefile`, an auxiliary Makefile `c4/common.mk` and a customizable configuration file called `c4/config.mk`. This Makefile builds the library that we will test.

⚠ *You may change `c4/Makefile` and `c4/common.mk` according to your needs, but your submission* MUST *compile with the original version of the `c4/Makefile` and `c4/common.mk` files on the reference image. When building your submissions we will override `Makefile` and `common.mk` with our own versions, thus any modifications done by you to these files will be ignored.*

If your submission fails to build with the original version of these files, you will receive no points for this task!

Furthermore, we have included the `c4/examples` subdirectory. This subdirectory contains a similar build-system consisting of a Makefile `c4/examples/Makefile`, an auxiliary Makefile `c4/examples/common.mk` and a customizable configuration file `c4/examples/config.mk`. Together with an example source file `c4/example/example.c`, that demonstrates the usage of the registry service, the build-system can be used to develop your own testing application.

Configuring the build environment to your needs is a simple matter of editing the `c4/config.mk` and `c4/example/config.mk` configuration files with a text editor. The configuration file `c4/config.mk` lists a number of make variables which *MUST* be customized to your source code layout. Usually you have to configure two sets of variables:

COMMUNICATION_SOURCES, SERVER_SOURCES and REGISTRY_SOURCES Lists the C files which are part of your implementation of the communication library, the server and the registry library respectively. Files listed in this variables *MUST NOT* contain a `main` function. We will consider anything listed in `COMMUNICATION_SOURCES`, `SERVER_SOURCES` and `REGISTRY_SOURCES` for automated testing.

EXAMPLES_SOURCES Lists all C files which are part of your own test code. One of the files listed in this variable should contain a `main` function. Files which are already listed in `COMMUNICATION_SOURCES`, `SERVER_SOURCES` or `REGISTRY_SOURCES` should not be listed here. We do not consider the code listed in `EXAMPLES_SOURCE` for automated testing.

Furthermore, the following configuration variables are supported in both `config.mk` files:

DEBUG Allows you to select between a debug (`1`) and a release (`0`) build of your application. In debug mode the `DEBUG` preprocessor macro is defined and generation of debugging information (`-g` compiler switch) is enabled. In release mode the `NDEBUG` macro is defined [3] and no debug information is generated.

COVERAGE Allows you to enable `gcov`-based code coverage analysis support. Setting this variable to a non-zero value enables the `-fprofile-arcs` and `-ftest-coverage` compiler options, which are needed to use tools like `gcov` and `lcov`.

VERBOSE Controls verbosity of the makefile output. You can set this variable to a non-zero value to show the actual compiler command-line invocations done by the build system.

TARGET Defines the target platform. You can set it to `x86_64` to build in a 64-bit environment or to `i386` to build in a 32-bit environment.

⚠ *For users of the 64-bit reference image: To build the code in a 32-bit environment `gcc-multilib` has to be installed on the reference image first. The following commands can be used to install `gcc-multilib`.*

```
sudo apt-get update
sudo apt-get install gcc-multilib
```

No other changes are necessary.

For users of the 32-bit reference image: To build the code in a 64-bit environment `gcc-multilib` is required as well. Furthermore, you need `qemu-user` to run the 64-bit executables:

```
sudo apt-get update
sudo apt-get install gcc-multilib qemu-user
```

Now you are able to run 64-bit executables by starting it through `qemu-x86_64`:

```
qemu-x86_64 $EXECUTABLE
```

Additionally `c4/examples/config.mk` contains the following variables that can be changed to fit your needs:

CFLAGS Additional compiler flags used when compiling files listed in `EXAMPLES_SOURCE`. (See Section 2.3)

LDFLAGS Additional linker flags used when linking the test program from source files listed in `EXAMPLES_SOURCE`. (See Section 2.3)

LIBS Additional libraries used when linking the test program from source files listed in `EXAMPLES_SOURCE`. (See Section 2.3)

2.3 Your own test code

You are highly encouraged to write your own test code. Our Makefile based build-system provides facilities to easily integrate with different unit-testing frameworks, by means of the `EXAMPLES_SOURCE`, `CFLAGS`, `LDFLAGS` and `LIBS` configuration variables in `c4/examples/config.mk`

Integration of a unit-testing framework usually requires one or more external libraries to be linked with your application. Test code is exempted from the external library rule discussed in Section 2. You may

[3] As a side-effect all standard C `assert`s are no-operations in release mode.

use the CFLAGS, LDFLAGS and LIBS build variables to configure your makefile for the unit-testing framework of your choice. Note that these settings only affect files listed as test-code in the EXAMPLES_SOURCE variable.

⚠ *Note that the communication, server and registry API must be implemented entirely by source files listed in COMMUNICATION_SOURCES, SERVER_SOURCES and REGISTRY_SOURCES respectively and that the no-external libraries rule applies to these files.*

2.4 Updating your repository

The makefiles and headers for this assignment are distributed as a GIT patch against the ws-baseline tag in your repositories. To integrate the updates for this task into your repository you simply have to download the patch-file[4] and merge it into your local working copy. To merge the changes for this assignment run the following commands in your working directory:

```
# Download the patch-file for task C4 into your home-directory
sase@SASE-reference-image:~$ wget -O ~/c4-patch.txt \
  http://tinyurl.com/chs79u6

# Switch to your local working copy.
sase@SASE-reference-image:~$ cd ~/SASD/

# Merge the patch into your working directory using "git am".
sase@SASE-reference-image:~/SASD$ git am --signoff ~/c4-patch.txt
```

3 Submission

The submission for this tasks consists of a correctly tagged Git commit in the c4 subdirectory of your group's Git repository.

Your submission should at least contain:

- A properly configured c4/config.mk configuration file.
- All C source and header files required to build your submission. These files *MUST* reside below the c4/ directory of your repository.

Each proper submission must be tagged with a Git *tag* called submission-c4. To correctly tag the currently checked out commit and push the new tag to the server run:

```
sase@SASE-reference-image:~/SASD$ git tag submission-c4
sase@SASE-reference-image:~/SASD$ git push
sase@SASE-reference-image:~/SASD$ git push origin refs/tags/submission-c4
```

You can easily verify your submission by cloning a fresh copy of your repository into a new directory and trying to check out the submission-c4 tag. The Git commands used by us to clone your submissions are semantically equivalent to:

```
$ git clone git@teaching.student.iaik.tugraz.at:sase2012gXX.git
$ cd sase2012gXX
$ git checkout refs/tags/submission-c4
```

☞ *You might accidentally tag the wrong commit for submission. To re-submit another commit simply create a new tag named submission-c4-X where X is an increasing number. Assuming that you want re-submit for the first time, run the following commands:*

```
sase@SASE-reference-image:~/SASD$ git tag submission-c4-1
sase@SASE-reference-image:~/SASD$ git push
sase@SASE-reference-image:~/SASD$ git push origin refs/tags/submission-c4-1
```

[4] We have created the TinyURL http://tinyurl.com/chs79u6 shortcut for the patch download URL to save you a little bit of typing. The complete URL is http://www.iaik.tugraz.at/content/teaching/master_courses/sicherheitsaspekte_in_der_softwareentwicklung/practicals/downloads/ku2012/c4-patch.txt

To re-submit again use `submission-c4-2`, `submission-c4-3` and so on.

The tags that qualify for submission, i.e. were created before the deadline, and reference a commit before the deadline, are sorted according to their name, and we will consider the tag that sorts last as your submission.

We will checkout your submissions shortly after the deadline has passed, and we will publish a newsgroup posting in `tu-graz.lv.sicherheitsaspekte` listing the received Git commit IDs that are considered to be your final submissions for this assignment.

⚠ *Any discrepancies between the commit IDs published by us in the newsgroup and the actual tags in your repositories will be resolved in favor of the published commit IDs - Attempts to (re-)tag your submissions after the deadline will be penalized.*

The directory structure of your repository should look similar to:

```
...
 ws/*                    - Results from previous assignment (KEEP THEM!)
 c1/*                    - ...
 c2/*                    - ...
 c3/*                    - ...
...
 c4/                                    - Directory for anything related to task C4
 c4/Doxyfile                            - Doxyfile for building the doxygen documentation
 c4/Makefile                            - Makefile for building the task (no change required)
 c4/README                              - Public README file
 c4/common.mk                           - Common extension for the Makefile
 c4/communication/bpack.h               - Header file for bpack/bunpack (no change allowed)
 c4/communication/channel.h             - Header file for the channel specification (no change allowed)
 c4/communication/channel-hmac.h        - Header file for the HMAC channel (no change allowed)
 c4/communication/channel-with-server.h - Header file for the channel-with-server (no change allowed)
 c4/communication/crypto/hmac.h         - Header file for the HMAC functions (no change allowed)
 c4/communication/crypto/sha1.h         - Header file for the SHA1 hash function (no change allowed)
 c4/config.mk                           - Configuration file for the Makefile
 c4/errors.h                            - Header file for the used error types (no change allowed)
 c4/registry/registry.h                 - Header file for the registry (no change allowed)
 c4/server/database.h                   - Header file for the database interface (no change allowed)
 c4/server/server.h                     - Header file for the server interface (no change allowed)
 c4/sql/create-database.sh              - Shell script to create a database
 c4/sql/database-init.sql               - SQL schema
...
```

⚠ *Keep your repository clean from build artifacts like object files or executables that were generated during the build process. The same applies for generated doxygen documentation and other by-products of the build process. Any leftover files will result in a loss of 10 percent of the available points.*

⚠ *Make sure that your submission builds and works with the original version of the header files.*

3.1 Time schedule and deadlines

HARD SUBMISSION DEADLINE: January 9, 2012 (23:59:59 (CET))

⚠ *We highly recommend to submit your solution at least one or two days before the ultimate submission deadline. Last minute submissions are always risky with respect to unforeseen technical difficulties.*

4 Tasks

4.1 *(5 points)* libregistry

The functions declared in `registry.h` are the public interface to the user. These function are to be implemented according to the their documentation in the header file.

Each of these functions sends the appropriate requests to the server and waits for the response, parses it and returns the values and error codes accordingly.

Questions:

(1.a) ☐ *(1.0 points)* Implement `registry_open`, `registry_close` and `registry_get_channel`

(1.b) ☐ *(1.0 points)* Implement `registry_key_get_value_type` and `registry_enum_keys`.

(1.c) ☐ *(0.75 points)* Implement `registry_get_int64` and `registry_set_int64`.

(1.d) ☐ *(0.75 points)* Implement `registry_get_double` and `registry_set_double`.

(1.e) ☐ *(0.75 points)* Implement `registry_get_string` and `registry_set_string`.

(1.f) ☐ *(0.75 points)* Implement `registry_get_blob` and `registry_set_blob`.

4.2 *(5 points)* communication

communication handles the data exchange between *libregistry* and *server*. Table 1 contains the list of packets sent from *libregistry* to the server and Table 2 contains the list of packets sent from the server as response. Each packet is prefixed by a single byte containing the packet type and is followed by bpacked binary data packed according to the format described in those two tables.

Table 1: Packets sent by the client

Type	Format	Data	Description
PACKET_GET_INT	`ss`	Domain, key	Get integer value associated with domain, key
PACKET_SET_INT	`ssl`	Domain, key, integer value	Set a double value
PACKET_GET_DOUBLE	`ss`	Domain, key	Get a double value associated
			with domain, key
PACKET_SET_DOUBLE	`ssd`	Domain, key, double value	Set an integer value
PACKET_GET_STRING	`ss`	Domain, key	Get a string value associated
			with domain, key
PACKET_SET_STRING	`sss`	Domain, key, string	Set a string
PACKET_GET_BLOB	`ss`	Domain, key	Get a blob associated
			with domain, key
PACKET_SET_BLOB	`ssb`	Domain, key, blob	Set a blob
PACKET_GET_ENUM	`ss`	Domain, key pattern	Get enum keys based on the search pattern associated with the domain
PACKET_GET_VALUE_TYPE	`ss`	Domain, key	Get the type associated with domain, key
PACKET_SHUTDOWN			Shutdown the server. No response packet required.

Table 2: Packets sent by the server

Type	Format	Data	Description
PACKET_OK			Result of PACKET_SET_*
PACKET_ERROR	`l`	Error code	Result if an error occurred
PACKET_INT	`l`	The integer value	Result of PACKET_GET_INT
PACKET_DOUBLE	`d`	The double value	Result of PACKET_GET_DOUBLE
PACKET_STRING	`s`	The string value	Result of PACKET_GET_STRING
PACKET_BLOB	`b`	The blob	Result of PACKET_GET_BLOB
PACKET_ENUM	`lb`	Number of results and the result as blob	Result of PACKET_GET_ENUM
PACKET_TYPE	`l`	Type of domain, key pair	Result of PACKET_GET_VALUE_TYPE

Please note that for `PACKET_ENUM` the blob is only sent if and only if the number of results is non-zero.

Packets will be sent and received via `channel_t` and its methods. A sample channel implementation is given in `channel-with-server.h` and `channel-with-server.c`. This channel starts up its own server and sends the packets directly to the server.

In addition to the existing channel implementation, you have to implement the channel described in `channel-hmac.h`. This channel uses HMAC-SHA1 (as described in RFC2104[4] and RFC3174[5]) to ensure integrity and authenticity of the data packets. The functions declared in `hmac.h` and `sha1.h` have to implemented for this channel as well. You should use the reference implementation of SHA1 and HMAC-SHA1 given in the above mentioned RFCs (Cite the origin of the code in your source correctly).

Questions:

(2.a) ☐ *(2.0 points)* Implement `sha1`, `hmac_encrypt` and `hmac_verify`.

(2.b) ☐ *(3.0 points)* Implement the HMAC-SHA1 channel (`channel_hmac_new` and `channel_hmac_set_key` and all the methods required for the channel, i.e. `client_read_bytes`, `client_write_bytes`, `server_read_bytes`, `server_write_bytes` and `free`).

4.3 *(15 points)* server

The *server* translates the packets received from *libregistry* via the channel into the appropriate database requests. After processing the request, the server sends a packet that contains the result of the request, or an error packet if an error occurred.

The header file `server.h` contains the declaration of all methods that have to be implemented for the server component.

Database You must use ısqlite as a backend for the database. The header file `database.h` contains the declaration of all methods that have to be imlemented for the database sub system. The file `database-init.sql` contains the database schema to be used. Please note that no component is allowed to create the database. You have to initialize the database using the provided shell script.

For integers, doubles and strings the values are directly stored in the database. For blobs some more additional work is needed. Blobs are stored in files somewhere in the blob-path directory (absolute path) found in the database (domain NULL, key blob-path). This directory must exist when opening the database. The path column in ValueBlob is treated as path relative to blob-path. Absolute paths and paths leaving the blob-path directory are invalid.

Questions:

(3.a) ☐ *(1.0 points)* Implement `server_init` and `server_shutdown`.

(3.b) ☐ *(2.0 points)* Implement `server_process`.

(3.c) ☐ *(1.0 points)* Implement `database_open` and `database_close`.

(3.d) ☐ *(2.0 points)* Implement `database_get_type` and `database_enum_keys`.

(3.e) ☐ *(2.0 points)* Implement `database_get_int64` and `database_set_int64`.

(3.f) ☐ *(2.0 points)* Implement `database_get_double` and `database_set_double`.

(3.g) ☐ *(2.0 points)* Implement `database_get_string` and `database_set_string`.

(3.h) ☐ *(3.0 points)* Implement `database_get_blob` and `database_set_blob`.

References

[1] *doxygen*. http://doxygen.org/.

[2] *GCC - the GNU Compiler Collection*. http://gcc.gnu.org/.

[3] *GNU debugger*. http://www.gnu.org/software/gdb/.

[4] *RFC2104*. https://www.ietf.org/rfc/rfc2104.txt.

[5] *RFC3174*. https://www.ietf.org/rfc/rfc3174.txt.

[6] *valgrind*. http://valgrind.org/.

[7] *VirtualBox*. http://www.virtualbox.org/.

[8] *ISO/IEC 9899*, 1999. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n897.pdf.

A Registry API

A.1 c4/registry/registry.h

```
#ifndef REGISTRY_H
#define REGISTRY_H

/** @brief Public API of libregistry.
 *
 *  The error code @a ERROR_REGISTRY_INVALID_STATE is used if and only if the
 *  registry received an error packet and the error code inside is equal to
 *  ERROR_DATABASE_INVALID.
 *
 *  All the handles may not be @a NULL. All domains and keys may not be @a NULL and
 *  have to be non-empty NUL-terminated strings. The target pointers in the
 *  registry_get_* functions and in registry_enum_keys may not be @a NULL. Blobs
 *  and strings may not be @a NULL.
 *
 *  All values are passed unmodified to the server.
 *
 *  @file  registry.h
 */

#include <stdint.h>
#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif // __cplusplus

#include "communication/channel.h"

typedef struct registry_s registry_t;

/**
 * Open a connection to the registry.
 *
 * @param[out] handle Pointer to the registry handle.
 *   The handle will be set to @a NULL on failure.
 * @param[in] identifier The registry identifier. The identifier must not be @a NULL and
 *   describes which registry implementation is used:
 *
 *    * file://<path> - create a channel_with_server instance
 *    * hmac://<key>  - create a channel_hmac instance
 *
 *  They can be seperated by |, e.g. file://<path>|hmac://<key> creates a
 *  channel_with_server instance using the datbase found at path and also
 *  creates a channel_hmac instance with <key> as its' key. The
 *  channel_with_server instance is passed to the channel_hmac as child
 *  argument. Please note that hmac://<key>|file://<path> is not valid since
 *  there is no child channel that could be passed to channel_hmac. hmac:// can
 *  occur multiple times in the identifer:
 *  file://<path>|hmac://<key>|hmac://<key> is valid and creates a chain of one
 *  channel_with_server instance and two channel_hmac instances.
 *
 *  If the identifier is not valid, ERROR_REGISTRY_UNKNOWN_IDENTIFIER is
 *  returned.
 *
 * @param[in] domain The domain identifier. The identifier must be a valid
 *   domain name and can be any arbitrary non-empty non-NULL string.
 *
 * @return @ref ERROR_OK on success,
 * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
 * @return @ref ERROR_REGISTRY_UNKNOWN_IDENTIFIER Specified identifier is not
 *   known
 * @return @ref ERROR_MEMORY Out of memory
 * @return @ref ERROR_UNKNOWN An unspecified error occurred
 */
int registry_open(registry_t** handle, const char* identifier,
    const char* domain);

/**
 * Closes the given connection to registry.
 *
 * @param[in] handle A registry handle. If the handle is @a NULL, this function
 *   is a no-op and returns @ref ERROR_INVALID_ARGUMENTS.
 *
 * @return @ref ERROR_OK on success,
```

```
75   * @return @ref ERROR_UNKNOWN An unspecified error occurred
76   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
77   */
78  int registry_close(registry_t* handle);
79
80  /**
81   * Retrieve a signed 64-bit integer value from the registry.
82   *
83   * @param[in] handle A valid registry handle. %TODO return values
84   * @param[in] key The key name of the value that should to be retrieved.
85   * @param[out] value Pointer to the variable receiving the value.
86   *
87   * @return @ref ERROR_OK on success,
88   * @return @ref ERROR_REGISTRY_NO_SUCH_KEY Given key does not exist
89   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
90   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
91   * @return @ref ERROR_UNKNOWN An unspecified error occurred
92   */
93  int registry_get_int64(registry_t* handle, const char* key, int64_t* value);
94
95  /** Set a signed 64-bit integer value in the registry.
96   *
97   * @param[in] handle A valid registry handle.
98   * @param[in] key The key name of the value that should to be set.
99   * @param[in] value The value.
100  *
101  * @return @ref ERROR_OK on success,
102  * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
103  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
104  * @return @ref ERROR_UNKNOWN An unspecified error occurred
105  */
106 int registry_set_int64(registry_t* handle, const char* key, int64_t value);
107
108 /**
109  * Retrieve a double precision floating point value from the registry.
110  *
111  * @param[in] handle A valid registry handle.
112  * @param[in] key The key name of the value that should to be retrieved.
113  * @param[out] value Pointer to the variable receiving the value.
114  *
115  * @return @ref ERROR_OK on success,
116  * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
117  * @return @ref ERROR_REGISTRY_NO_SUCH_KEY Given key does not exist
118  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
119  * @return @ref ERROR_UNKNOWN An unspecified error occurred
120  */
121 int registry_get_double(registry_t* handle, const char* key, double* value);
122
123 /** Set a double precission floating point value in the registry.
124  *
125  * @param[in] handle A valid registry handle.
126  * @param[in] key The key name of the value that should to be set.
127  * @param[in] value The value.
128  *
129  * @return @ref ERROR_OK on success,
130  * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
131  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
132  * @return @ref ERROR_UNKNOWN An unspecified error occurred
133  */
134 int registry_set_double(registry_t* handle, const char* key, double value);
135
136 /**
137  * Retrieve a NUL-terminanted string from the registry.
138  *
139  * @param[in] handle A valid registry handle.
140  * @param[in] key The key name of the value that should be set.
141  * @param[out] value Pointer to the variable receiving the NULL-terminanted value.
142  *
143  * @return @ref ERROR_OK on success,
144  * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
145  * @return @ref ERROR_REGISTRY_NO_SUCH_KEY Given key does not exist
146  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
147  * @return @ref ERROR_UNKNOWN An unspecified error occurred
148  */
149 int registry_get_string(registry_t* handle, const char* key, char** value);
150
151 /**
152  * Set a NUL-terminanted string in the registry.
153  *
154  * @param[in] handle A valid registry handle.
```

```
155   * @param[in] key The key name of the value that should be set.
156   * @param[in] value A NUL-terminated string.
157   *
158   * @return @ref ERROR_OK on success,
159   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
160   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
161   * @return @ref ERROR_UNKNOWN An unspecified error occurred
162   */
163  int registry_set_string(registry_t* handle, const char* key, const char* value);
164
165  /** Retrieve a blob value from the registry.
166   *
167   * @param[in] handle A valid registry handle.
168   * @param[in] key The key name of the value that should be set.
169   * @param[out] size Pointer to variable receiving the size of the blob.
170   * @param[out] value Pointer to the variable receiving the blob.
171   *
172   * @return @ref ERROR_OK on success,
173   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
174   * @return @ref ERROR_REGISTRY_NO_SUCH_KEY Given key does not exist
175   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
176   * @return @ref ERROR_UNKNOWN An unspecified error occurred
177   */
178  int registry_get_blob(registry_t* handle, const char* key, unsigned char** value, size_t* size);
179
180  /** Set a blob value in the registry.
181   *
182   * @param[in] handle A valid registry handle.
183   * @param[in] key The key name of the value that should be set.
184   * @param[in] size Size of the blob.
185   * @param[in] value The blob of @a size bytes.
186   *
187   * @return @ref ERROR_OK on success,
188   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
189   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
190   * @return @ref ERROR_UNKNOWN An unspecified error occurred
191   */
192  int registry_set_blob(registry_t* handle, const char* key, const unsigned char* value, size_t size);
193
194  /** Enumerate keys according to a pattern.
195   *
196   * The keys are returned in one large string that is separated by @a 0s. For
197   * valid patterns have a look at @ref database_enum_keys. The caller is
198   * responsible the free the memory block referenced by keys.
199   *
200   * @param[in] handle A valid registry handle.
201   * @param[in] pattern The key pattern.
202   * @param[out] count Count of enumerated keys.
203   * @param[out] size Size of the keys.
204   * @param[out] keys The enumerated keys.
205   *
206   * @return @ref ERROR_OK on success,
207   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
208   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
209   * @return @ref ERROR_UNKNOWN An unspecified error occurred
210   */
211  int registry_enum_keys(registry_t* handle, const char* pattern, size_t* count, size_t* size, char** keys);
212
213  /**
214   * Retrieves the type of a key
215   *
216   * @param[in] handle A valid registry handle.
217   * @param[in] key The key name of the value that should be used.
218   * @param[out] type The type of the key
219   *
220   * @return @ref ERROR_OK on success,
221   * @return @ref ERROR_REGISTRY_INVALID_STATE Corrupt database
222   * @return @ref ERROR_REGISTRY_NO_SUCH_KEY Given key does not exist
223   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
224   * @return @ref ERROR_UNKNOWN An unspecified error occurred
225   */
226  int registry_key_get_value_type(registry_t* handle, const char* key, int* type);
227
228  /**
229   * Returns the channel object from the registry handle.
230   *
231   * @param handle A valid registry handle
232   *
233   * @return Reference to the used channel or NULL. The channel is still owned by
234   * the registry.
```

```
 */
channel_t* registry_get_channel(registry_t* handle);

#ifdef __cplusplus
} // extern "C"
#endif // __cplusplus

#endif // REGISTRY_H
```

B Server API

B.1 c4/server/server.h

```
#ifndef SERVER_H
#define SERVER_H

/** @brief Server interface
 *
 * @file server.h
 */

#include <stddef.h>

#include "communication/channel.h"

#ifdef __cplusplus
extern "C" {
#endif // __cplusplus

typedef struct server_s server_t;

/**
 * Initializes a server. The database referenced by @a database is opened.
 *
 * @param[out] server Pointer to the server
 * @param[in] database Path to the sqlite database file
 *
 * @return @ref ERROR_OK on success.
 * @return Any error code that is returned by @ref database_open.
 * @return @ref ERROR_MEMORY Out of memory.
 * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed.
 * @return @ref ERROR_UNKNOWN An unspecified error occurred.
 */
int server_init(server_t** server, const char* database);

/**
 * Processes a packet
 *
 * @param[in] server The server
 * @param[in] data The data
 * @param[in] size The size of data
 * @param[out] response The response data
 * @param[out] response_size The size of the response data
 *
 * @return @ref ERROR_OK on success,
 * @return @ref ERROR_SERVER_SHUTDOWN The server should shutdown
 * @return @ref ERROR_MEMORY Out of memory
 * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
 * @return @ref ERROR_UNKNOWN An unspecified error occurred
 */
int server_process(server_t* server, const unsigned char* data, size_t size, unsigned char** response, size_t* response_size);

/**
 * Closes the server
 *
 * @param server The server. If the server is @a NULL, this function is a no-op.
 *
 * @return @ref ERROR_OK on success,
 * @return @ref ERROR_UNKNOWN An unspecified error occurred
 * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
 */
int server_shutdown(server_t* server);

#ifdef __cplusplus
} // extern "C"
```

```
63 #endif // __cplusplus
64
65 #endif // SERVER_H
```

B.2 c4/server/database.h

```
1  #ifndef DATABASE_H
2  #define DATABASE_H
3
4  /** @brief Database interface
5   *
6   * The database is used as backend to store the data entered into the registry.
7   * It is able to store four different types of data: 64-bit integers, double
8   * precision floating point values, NUL-terminated strings and binary blobs.
9   *
10  * @ref database_open opens an existing database and never creates the database
11  * on its own. It also performs basic sanity checks on the database's scheme. It
12  * must match the scheme defined in sql/database-init.sql. Additional tables and
13  * columns as well as data types may exist. However, if any tables, columns or
14  * data-types are missing, constraints are not set properly (e.g. primary key
15  * and not null), @ref database_open fails with @ref ERROR_DATABASE_INVALID.
16  * Additionally, @ref database_open reads the blob path from the database. The
17  * blob path is stored as string value with domain NULL and key blob-path. The
18  * path stored in this value has to exist and has to be a directory.
19  *
20  * @ref database_get_int64, @ref database_get_double, @ref database_get_string and
21  * @ref database_get_blob as well as database_get_type retrieve the value from
22  * the database and return the associated type respectively. @ref
23  * database_set_int64, @ref database_set_double, @ref database_set_string and @ref
24  * database_set_blob set values. These four functions rollback any changes if
25  * one of the queries fails or, in the case of blobs, any file system operations
26  * fails. They also make sure that the database is kept clean, meaning:
27  *  - If the row in KeyInfo has the id @a a and data type @a b, then there
28  *    exists a row in Value@a b with the exact same key and no other Value table
29  *    contains a row with id @a a.
30  *  - If a row from ValueBlob is removed, the referenced file has to be deleted.
31  *  - If a existing blob is overwritten and the new data is written to a new
32  *    file, the old file has to be deleted.
33  *  Please note that an error while removing an unreferenced blob file is not
34  *  critical and is ignored.
35  *
36  *  Blob files are stored according to the following scheme: @a $blob-path/$path
37  *  where @a $blob-path is extracted from the database in @ref database_open and
38  *  @a $path is stored in the ValueBlob table. Every access of an blob file has
39  *  to make sure that the file is a regular file and that the file is in @a
40  *  $blob-path or any of its subdirectories.
41  *
42  *  Wherever a domain or key is required as argument, they both may not be @a
43  *  NULL or empty strings. All arguments that are used as destination may not be
44  *  @a NULL. Furthermore all @database_handle_t pointers may not be @a NULL. All
45  *  other strings and blobs may not be @a NULL
46  *
47  * All the handles may not be @a NULL. All domains and keys may not be @a NULL
48  * and have to be non-empty NUL-terminated strings. The target pointers in the
49  * database_get_* functions and in database_enum_keys may not be @a NULL. Blobs
50  * and strings may not be @a NULL.
51  *
52  * @file database.h
53  */
54
55 #include <stdint.h>
56 #include <stddef.h>
57
58 #ifdef __cplusplus
59 extern "C" {
60 #endif // __cplusplus
61
62 /**
63  * @a database_handle_s has to be definied in the C file implementing the
64  * database API.
65  */
66 typedef struct database_handle_s database_handle_t;
67
68 typedef enum database_value_type_e
69 {
70   DATABASE_TYPE_INT64 = 0,
71   DATABASE_TYPE_DOUBLE,
```

```
72    DATABASE_TYPE_STRING,
73    DATABASE_TYPE_BLOB
74  } database_value_type_t;
75
76  /**
77   * Open an existing database. The database must exist and be valid. The
78   * function returns an error if this is not the case..
79   *
80   * @param[out] handle Pointer to the database handle that should be used for
81   *   this database connection.
82   * @param[in] path Path to a valid sqlite database.
83   *
84   * @return @ref ERROR_OK on success,
85   * @return @ref ERROR_DATABASE_OPEN The database does not exist or is not a
86   *  regular file.
87   * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e a table does
88   *  not exist or a column doesn't match the specification or the blob-path is
89   *  not an existing directory.
90   * @return @ref ERROR_MEMORY Out of memory.
91   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed.
92   * @return @ref ERROR_UNKNOWN An unspecified error occurred.
93   */
94  int database_open(database_handle_t** handle, const char* path);
95
96  /**
97   * Close database.
98   *
99   * @param[in] handle Database handle to be freed, non-NULL.
100  *
101  * @return @ref ERROR_OK on success.
102  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed.
103  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
104  */
105 int database_close(database_handle_t* handle);
106
107 /**
108  * Query the value's type associated to a domain and key.
109  *
110  * @param[in] handle Database handle
111  * @param[in] domain The domain of the keys
112  * @param[in] key The key
113  * @param[out] type The type of the key
114  *
115  * @return @ref ERROR_OK on success,
116  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
117  *  queries failed.
118  * @return @ref ERROR_DATABASE_TYPE_UNKNOWN The associated type is not Int64,
119  *  Double, String or Blob.
120  * @return @ref ERROR_DATABASE_NO_SUCH_KEY The domain, key pair does not exist.
121  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed.
122  * @return @ref ERROR_MEMORY Out of memory.
123  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
124  */
125 int database_get_type(database_handle_t* handle, const char* domain,
126     const char* key, database_value_type_t* type);
127
128 /** Enumerate keys.
129  *
130  * The keys are returned in one large string that is separated by \c 0s. The SQL
131  * statement @a GLOB is used to enumerate the keys. So @a pattern can be
132  * anything that is valid as argument to @a GLOB. The result is sorted
133  * alphabetically.
134  *
135  * The caller is responsible to free up the memory pointed to by keys.
136  *
137  * @param[in] handle A valid database handle.
138  * @param[in] domain The domain of the keys.
139  * @param[in] pattern The key pattern.
140  * @param[out] count Number of enumerated keys.
141  * @param[out] size Size of keys.
142  * @param[out] keys The enumerated keys.
143  *
144  * @return @ref ERROR_OK on success,
145  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
146  *  queries failed.
147  * @return @ref ERROR_MEMORY Out of memory.
148  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed.
149  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
150  */
151 int database_enum_keys(database_handle_t* handle, const char* domain,
```

```
152     const char* pattern, size_t* count, size_t* size, char** keys);
153
154 /**
155  * Retrieve the value associated to the domain and key.
156  *
157  * @param[in] handle A valid database handle.
158  * @param[in] domain The domain of the keys.
159  * @param[in] key The key.
160  * @param[out] value The value
161  *
162  * @return @ref ERROR_OK on success,
163  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
164  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
165  *  queries failed.
166  * @return @ref ERROR_DATABASE_NO_SUCH_KEY The domain, key pair does not exist.
167  * @return @ref ERROR_DATABASE_TYPE_MISMATCH The value associated to the domain,
168  *  key pair is not of the correct type.
169  * @return @ref ERROR_MEMORY Out of memory.
170  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
171  */
172 int database_get_int64(database_handle_t* handle, const char* domain,
173     const char* key, int64_t* value);
174
175 /**
176  * Set the value associated to the domain and key.
177  *
178  * @param[in] handle A valid database handle.
179  * @param[in] domain The domain of the keys.
180  * @param[in] key The key.
181  * @param[in] value The value
182  *
183  * @return @ref ERROR_OK on success,
184  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
185  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
186  *  queries failed.
187  * @return @ref ERROR_MEMORY Out of memory.
188  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
189  */
190 int database_set_int64(database_handle_t* handle, const char* domain,
191     const char* key, int64_t value);
192
193 /**
194  * Retrieve the value associated to the domain and key.
195  *
196  * @param[in] handle A valid database handle.
197  * @param[in] domain The domain of the keys.
198  * @param[in] key The key.
199  * @param[out] value The value (not NAN)
200  *
201  * @return @ref ERROR_OK on success,
202  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
203  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
204  *  queries failed.
205  * @return @ref ERROR_DATABASE_NO_SUCH_KEY The domain, key pair does not exist.
206  * @return @ref ERROR_DATABASE_TYPE_MISMATCH The value associated to the domain,
207  *  key pair is not of the correct type.
208  * @return @ref ERROR_MEMORY Out of memory.
209  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
210  */
211 int database_get_double(database_handle_t* handle, const char* domain,
212     const char* key, double* value);
213
214 /**
215  * Set the value associated to the domain and key.
216  *
217  * @param[in] handle A valid database handle.
218  * @param[in] domain The domain of the keys.
219  * @param[in] key The key.
220  * @param[in] value The value
221  *
222  * @return @ref ERROR_OK on success,
223  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
224  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
225  *  queries failed.
226  * @return @ref ERROR_MEMORY Out of memory.
227  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
228  */
229 int database_set_double(database_handle_t* handle, const char* domain,
230     const char* key, double value);
231
```

```
232 /**
233  * Retrieve the value associated to the domain and key.
234  *
235  * @param[in] handle A valid database handle.
236  * @param[in] domain The domain of the keys.
237  * @param[in] key The key.
238  * @param[out] value The value
239  *
240  * @return @ref ERROR_OK on success,
241  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
242  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
243  *  queries failed.
244  * @return @ref ERROR_DATABASE_NO_SUCH_KEY The domain, key pair does not exist.
245  * @return @ref ERROR_DATABASE_TYPE_MISMATCH The value associated to the domain,
246  *  key pair is not of the correct type.
247  * @return @ref ERROR_MEMORY Out of memory.
248  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
249  */
250 int database_get_string(database_handle_t* handle, const char* domain,
251     const char* key, char** value);
252
253 /**
254  * Set the value associated to the domain and key.
255  *
256  * @param[in] handle A valid database handle.
257  * @param[in] domain The domain of the keys.
258  * @param[in] key The key.
259  * @param[in] value The value
260  *
261  * @return @ref ERROR_OK on success,
262  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
263  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
264  *  queries failed.
265  * @return @ref ERROR_MEMORY Out of memory.
266  */
267 int database_set_string(database_handle_t* handle, const char* domain,
268     const char* key, const char* value);
269
270 /**
271  * Retrieve the value associated to the domain and key.
272  *
273  * @param[in] handle A valid database handle.
274  * @param[in] domain The domain of the keys.
275  * @param[in] key The key.
276  * @param[out] value The value
277  * @param[out] size The size of value
278  *
279  * @return @ref ERROR_OK on success,
280  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
281  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
282  *  queries failed or the referenced file is not a regular file or doesn't exist
283  *  or isn't located in the blob-path or any of its subdirectories.
284  * @return @ref ERROR_DATABASE_NO_SUCH_KEY The domain, key pair does not exist.
285  * @return @ref ERROR_DATABASE_TYPE_MISMATCH The value associated to the domain,
286  *  key pair is not of the correct type.
287  * @return @ref ERROR_DATABASE_IO Reading from the referenced file failed.
288  * @return @ref ERROR_MEMORY Out of memory.
289  * @return @ref ERROR_UNKNOWN An unspecified error occurred.
290  */
291 int database_get_blob(database_handle_t* handle, const char* domain,
292     const char* key, unsigned char** value, size_t* size);
293
294 /**
295  * Set the value associated to the domain and key.
296  *
297  * @param[in] handle A valid database handle.
298  * @param[in] domain The domain of the keys.
299  * @param[in] key The key.
300  * @param[in] value The value
301  * @param[in] size The size of value
302  *
303  * @return @ref ERROR_OK on success,
304  * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
305  * @return @ref ERROR_DATABASE_INVALID The database is invalid, i.e one of the
306  *  queries failed.
307  * @return @ref ERROR_DATABASE_IO Writing to the blob file failed.
308  * @return @ref ERROR_MEMORY Out of memory.
309  */
310 int database_set_blob(database_handle_t* handle, const char* domain,
311     const char* key, const unsigned char* value, size_t size);
```

```
312
313 #ifdef __cplusplus
314 } // extern "C"
315 #endif // __cplusplus
316
317 #endif // DATABASE_H
```

C Communication API

C.1 c4/communication/channel.h

```
1  #ifndef CHANNEL_H
2  #define CHANNEL_H
3
4  #include <stddef.h>
5
6  /** @brief Message channel
7   *
8   * Message channels are used to exchange messages between two end-points.
9   * Channels have four methods: (client,server)_read_bytes to receive a message
10  * and (client,server_)write_bytes to send a message.
11  *
12  * Channels have a very simple semantic: Let's call the two end-points A and B
13  * for now. A shall be known as the client and B as the server. With this
14  * semantic, A connects to B. A sends a message to B. B receives the message
15  * and sends a response back to A. A can send multiple messages to B and B
16  * shall receive them in the same order.
17  *
18  * Not that read_bytes and write_bytes may fail with ERROR_CHANNEL_BUSY. This
19  * is an indication that their is currently no data to read or that the channel
20  * can not send data right now. The caller should try again later in that case.
21  *
22  * @file channel.h
23  */
24 typedef struct channel_s
25 {
26   /**
27    * Write bytes to the channel and send them to the server.
28    *
29    * @param[in] channel The channel.
30    * @param[in] bytes Bytes to be written.
31    * @param[in] size Number of bytes in bytes.
32    *
33    * @return @ref ERROR_OK on success
34    * @return @ref ERROR_CHANNEL_BUSY The channel is busy
35    * @return @ref ERROR_INVALID_ARGUMENTS If the @a channel is invalid, @bytes
36    *  is NULL or size is 0
37    * @return @ref ERROR_UNKNOWN Any other error occured
38    */
39   int (*client_write_bytes)(struct channel_s* channel, const unsigned char* bytes, size_t size);
40
41   /**
42    * Read bytes from the channel received from the server.
43    *
44    * The caller is responsible to free the memory pointed to by @a bytes.
45    *
46    * @param[in] channel The channel.
47    * @param[out] bytes Pointer to the variable where pointer to the bytes
48    *   should be stored.
49    * @param[out] size Pointer to the variable where the number of bytes should
50    *   be store.
51    *
52    * @return @ref ERROR_OK on success
53    * @return @ref ERROR_CHANNEL_BUSY The channel is busy
54    * @return @ref ERROR_INVALID_ARGUMENTS If the @a channel is invalid, @bytes
55    *  is NULL or size is NULL
56    * @return @ref ERROR_UNKNOWN Any other error occured
57    */
58   int (*client_read_bytes)(struct channel_s* channel, unsigned char** bytes, size_t* size);
59
60   /**
61    * Write bytes to the channel and send them to the client.
62    *
63    * @param[in] channel The channel.
64    * @param[in] bytes Bytes to be written.
```

```
65      * @param[in] size Number of bytes in bytes.
66      *
67      * @return @ref ERROR_OK on success
68      * @return @ref ERROR_CHANNEL_BUSY The channel is busy
69      * @return @ref ERROR_INVALID_ARGUMENTS If the @a channel is invalid, @bytes
70      *  is NULL or size is 0
71      * @return @ref ERROR_UNKNOWN Any other error occured
72      */
73     int (*server_write_bytes)(struct channel_s* channel, const unsigned char* bytes, size_t size);
74
75     /**
76      * Read bytes from the channel received from the client.
77      *
78      * The caller is responsible to free the memory pointed to by @a bytes.
79      *
80      * @param[in] channel The channel.
81      * @param[out] bytes Pointer to the variable where pointer to the bytes
82      *   should be stored.
83      * @param[out] size Pointer to the variable where the number of bytes should
84      *   be store.
85      *
86      * @return @ref ERROR_OK on success
87      * @return @ref ERROR_CHANNEL_BUSY The channel is busy
88      * @return @ref ERROR_INVALID_ARGUMENTS If the @a channel is invalid, @bytes
89      *  is NULL or size is NULL
90      * @return @ref ERROR_UNKNOWN Any other error occured
91      */
92     int (*server_read_bytes)(struct channel_s* channel, unsigned char** bytes, size_t* size);
93
94     /**
95      * Frees the channel and all of its allocated ressources.
96       *
97       * @param[in] channel The channel.
98       *
99       * @return @ref ERROR_OK on success, an error code otherwise.
100      * @return @ref ERROR_INVALID_ARGUMENTS If the channel is invalid
101      * @return @ref ERROR_UNKNOWN Any other error occured
102      */
103    int (*free)(struct channel_s* channel);
104
105    /**
106     * Channel specific data.
107     */
108    void* data;
109 } channel_t;
110
111 /**
112  * Wrapper around the channel's client_read_bytes.
113  *
114  * @see channel_t.client_read_bytes
115  */
116 int channel_client_read_bytes(channel_t* channel, unsigned char** bytes, size_t* size);
117
118 /**
119  * Wrapper around the channel's client_write_bytes.
120  *
121  * @see channel_t.client_write_bytes
122  */
123 int channel_client_write_bytes(channel_t* channel, const unsigned char* bytes, size_t size);
124
125 /** \brief Wrapper around the channel's server_read_bytes.
126  *
127  * @see channel_t.server_read_bytes
128  */
129 int channel_server_read_bytes(channel_t* channel, unsigned char** bytes, size_t* size);
130
131 /**
132  * Wrapper around the channel's server_write_bytes.
133  *
134  * @see channel_t.server_write_bytes
135  */
136 int channel_server_write_bytes(channel_t* channel, const unsigned char* bytes, size_t size);
137
138 /** Wrapper around the channel's frees.
139  *
140  * @see channel_t.free
141  */
142 int channel_free(channel_t* channel);
143
144 #endif
```

C.2 c4/communication/channel-hmac.h

```
#ifndef CHANNEL_HMAC_H
#define CHANNEL_HMAC_H

#include <stddef.h>

/** @brief Message channel with HMAC.
 *
 * The HMAC channel is an implementation of a channel as described in @see
 * channel.h. It's designed as a channel wrapper that uses an existing channel
 * for the real data exchange between different endpoints.
 *
 * The characteristics of an HMAC channel are that it calculates an HMAC for any
 * data that should be send and packs the result at the end of the outgoing
 * message. In addition, it checks every incoming data for a correct appended
 * HMAC which will be stripped out before it is passed to the user.
 *
 * If no key has been set for the HMAC channel the data will be passed further
 * to its child channel without any modifications.
 *
 * @file channel-hmac.h
 */

#include "channel.h"

/**
 * Creates a new HMAC channel
 *
 * @param[out] channel The HMAC channel
 * @param[in] child The child channel. The child channel is owned by the newly
 *  created channel and will be freed automatically.
 *
 * @return @ref ERROR_OK if no error occured
 * @return @ref ERROR_MEMORY Out of memory
 * @return @ref ERROR_INVALID_ARGUMENTS If either the channel or the child is a
 *  @a NULL pointer
 * @return @ref ERROR_UNKNOWN Any unspecified error occured
 */
int channel_hmac_new(channel_t** channel, channel_t* child);

/**
 * Sets the key from the HMAC channel. Iff the specified key is NULL the key of
 * the channel is unset and therefore HMAC is disabled.
 *
 * @param channel The HMAC channel
 * @param key The key that shall be used or NULL if HMAC should be turned off
 * @param len Length of the passed key. Irrelevant if @a key is NULL
 *
 * @return @ref ERROR_OK if no error occured
 * @return @ref ERROR_MEMORY Out of memory
 * @return @ref ERROR_INVALID_ARGUMENTS If no channel has been passed to the
 *  function
 * @return @ref ERROR_UNKNOWN Any unspecified error occured
 */
int channel_hmac_set_key(channel_t* channel, const unsigned char* key, size_t len);

#endif // CHANNEL_HMAC_H
```

C.3 c4/communication/crypto/hmac.h

```
#ifndef HMAC_H
#define HMAC_H

/**
 * @brief HMAC-SHA-1
 *
 * @file hmac.h
 */

#include <stddef.h>

/**
 * This function calculates a hash-based message authentication code for a given
 * message based on the HMAC-SHA-1. The @a key of length @a keysize is used to
 * calculate the HMAC-SHA-1 of @a message of length @a messagesize. The result
 * is saved in the buffer @a hmac that has the size of @ref SHA1_BLOCKSIZE.
```

```
17   *
18   * The function returns @ref ERROR_INVALID_ARGUMENTS if any of the given
19   * parameters is invalid (invalid buffer pointers, a key or message size of
20   * zero). If the function for some reason runs out of memory, @ref ERROR_MEMORY
21   * will be returned. If the calculation of the HMAC succeded, @ref ERROR_OK will
22   * be returned and if the function fails for any other reason, @ref
23   * ERROR_UNKNOWN will be returned.
24   *
25   * See http://www.ietf.org/rfc/rfc2104.txt for the definition and analysis of
26   * the HMAC construction.
27   *
28   * @param[in] key The used key
29   * @param[in] keysize The length of the key
30   * @param[in] message The input message
31   * @param[in] messagesize The length of the input message
32   * @param[out] hmac Location where the calculated HMAC-SHA-1 value is stored.
33   *  The caller has to make sure that the buffer fits at least @ref SHA1_BLOCKSIZE
34   *  bytes
35   *
36   * @return @ref ERROR_OK No error occurred
37   * @return @ref ERROR_MEMORY Out of memory
38   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
39   * @return @ref ERROR_UNKNOWN Any other error occurred
40   */
41  int hmac(const unsigned char* key, size_t keysize,
42      const unsigned char* message, size_t messagesize,
43      unsigned char* hmac);
44
45  /**
46   * This functon is used to verify the given HMAC-SHA-1 @a hmac for the given @a
47   * message of length @a messagesize and the @a key of length @a keysize.
48   *
49   * The function returns @ref ERROR_INVALID_ARGUMENTS if any of the given
50   * parameters is invalid (invalid buffer pointers, a key or message of
51   * length zero). If the given HMAC-SHA-1 value @a hmac is correct, the function
52   * returns @ref ERROR_OK, otherwise @ref ERROR_HMAC_VERIFICATION_FAILED will be
53   * returned. If any other unspecified error occurs, this function returns @ref
54   * ERROR_UNKNOWN.
55   *
56   * @param[in] key The used key
57   * @param[in] keysize The length of the used key
58   * @param[in] message The input message
59   * @param[in] messagesize The length of the input message
60   * @param[in] hmac The calculated HMAC-SHA-1 value
61   *
62   * @return @ref ERROR_OK No error occurred and the HMAC-SHA-1 value is correct
63   * @return @ref ERROR_HMAC_VERIFICATION_FAILED The passed HMAC-SHA-1 value does not
64   *  match for the given input
65   * @return @ref ERROR_INVALID_ARGUMENTS Invalid arguments have been passed
66   * @return @ref ERROR_UNKNOWN Any other error occurred
67   */
68  int hmac_verify(const unsigned char* key, size_t keysize,
69      const unsigned char* message, size_t messagesize,
70      const unsigned char* hmac);
71
72  #endif
```

C.4 c4/communication/crypto/sha1.h

```
1   #ifndef SHA1_H
2   #define SHA1_H
3
4   /**
5    * @brief SHA-1
6    *
7    * @file sha1.h
8    */
9
10  #include <stddef.h>
11  #include <stdint.h>
12
13  /**
14   * Size of the hash in bytes.
15   */
16  #define SHA1_BLOCKSIZE 20
17
18  /**
```

```
19   * This function calculates the SHA-1 hash sum for the given input @a data
20   * and its given length @a len. The 160 bits result will be written
21   * into the first 20 bytes of @a res, a buffer that has to be allocated by the
22   * caller of this function.
23   *
24   * If and only if the calculation of the SHA-1 hash sum succeeded, this function
25   * will return @ref ERROR_OK, otherwise @ref ERROR_UNKNOWN or @ref
26   * ERROR_INVALID_ARGUMENTS (Invalid buffer pointers or a passed length of 0)
27   * will be returned.
28   *
29   * See http://www.ietf.org/rfc/rfc3174.txt for the defined standard and an
30   * example C implementation.
31   *
32   * @param[in] data The input data
33   * @param[in] len The length of the input data @a data
34   * @param[out] res The buffer where the SHA-1 hash sum will be written to
35   *
36   * @return @ref ERROR_OK on success
37   * @return @ref ERROR_UNKNOWN on failure
38   * @return @ref ERROR_INVALID_ARGUMENTS If invalid arguments have been passed
39   */
40  int sha1(const unsigned char* data, size_t len, uint8_t *res);
41
42  #endif
```

D Other

D.1 c4/errors.h

```
1   #ifndef ERRORS_H
2   #define ERRORS_H
3
4   #ifdef __cplusplus
5   extern "C" {
6   #endif // __cplusplus
7
8   enum {
9     ERROR_OK = 0,
10    ERROR_UNKNOWN,
11    ERROR_MEMORY,
12    ERROR_INVALID_ARGUMENTS,
13    ERROR_EOF,
14
15    ERROR_BPACK_INVALID_FORMAT_STRING,
16    ERROR_BPACK_WRITE,
17    ERROR_BPACK_READ,
18    ERROR_BUNPACK_INVALID_DATA,
19
20    ERROR_CHANNEL_BUSY,
21    ERROR_CHANNEL_FAILED,
22
23    ERROR_REGISTRY_NO_SUCH_KEY,
24    ERROR_REGISTRY_UNKNOWN_IDENTIFIER,
25    ERROR_REGISTRY_INVALID_STATE,
26
27    ERROR_DATABASE_OPEN,
28    ERROR_DATABASE_INVALID,
29    ERROR_DATABASE_NO_SUCH_KEY,
30    ERROR_DATABASE_IO,
31    ERROR_DATABASE_TYPE_MISMATCH,
32    ERROR_DATABASE_TYPE_UNKNOWN,
33
34    ERROR_SERVER_INIT,
35    ERROR_SERVER_SHUTDOWN,
36    ERROR_SERVER_PROCESS,
37
38    ERROR_HMAC_VERIFICATION_FAILED
39  };
40
41  typedef enum packet_type_e {
42    PACKET_INVALID,
43    PACKET_OK,
44    PACKET_ERROR,
45    PACKET_INT,
46    PACKET_GET_INT,
```

```
47    PACKET_SET_INT,
48    PACKET_DOUBLE,
49    PACKET_GET_DOUBLE,
50    PACKET_SET_DOUBLE,
51    PACKET_STRING,
52    PACKET_GET_STRING,
53    PACKET_SET_STRING,
54    PACKET_BLOB,
55    PACKET_GET_BLOB,
56    PACKET_SET_BLOB,
57    PACKET_ENUM,
58    PACKET_GET_ENUM,
59    PACKET_TYPE,
60    PACKET_GET_VALUE_TYPE,
61    PACKET_SHUTDOWN
62  } packet_type_t;
63
64  #ifdef __cplusplus
65  } // extern "C"
66  #endif // __cplusplus
67
68  #endif // ERRORS_H
```