libfrontg8

Release 0.1.0

October 18, 2016

libfrontg8 is a new foundation for frontg8, the security-centered, opensource communication system. It provides a common infrastructure for server and client implementations and allows you to integrate frontg8 in your own project.

To maximize interoperability, libfrontg8 is designed to expose a straight-forward C API. This allows you to use libfrontg8 from the language of your choice. Currently we officially support the bare C API and a C++ frontend for the library. One of our future goals is to provide language bindings for Python, Java and Rust. If you would like to help out, you can reach us via Github.

CHAPTER 1

User documentation (C API)

1.1 The C API

1.1.1 Error processing functions

`#include <frontg8/error.h>`

Typedefs

typedef `fg8_error_t`

The opaque error pointer type.

This type carries generic error information. The library uses this type throughout the whole codebase to communicate error conditions back to the your code. There is no way to construct objects of this type except indirectly by forcefully causing a function to reach an error condition. See *fg8_error_message* and *fg8_error_destroy* for information on how to make use of this type.

Author Felix Morgner

Since 0.1.0

Functions

char const* **`fg8_error_message`** (*fg8_error_t* const *error*)

Retrieve the error message contained in an *fg8_error_t* object.

Being an opaque type, it is not possible to directly read the content of the error object. You can use this function to extract the error message contained in an *fg8_error_t* object. Calling this function on a `NULL` object will return a `NULL` pointer. The string (`char const *`) returned by the function is a standard `NULL` terminated C string. The memory of the string is **owned by the error object**.

Example

```c
#include <frontg8/error.h>
#include <stdio.h>
// other includes ...

int main()
  {
```

```
  fg8_error_t error;

  // Do something that might produce an error ...

  if(error)
    {
    printf("ERROR: %s", fg8_error_message(error));
    }

  // Do the rest ...
  }
```

Return A pointer to the message contained in the error or `NULL` if there was no error. The memory of the message is managed by the error object and the user code **MUST NEVER** call `free()` or equivalent on it.

Author Felix Morgner

Since 0.1.0

Parameters

- `error` - The error object whose message should be retrieved. Might be `NULL`.

void **`fg8_error_destroy`** (*fg8_error_t* const *error*)

Destroy an error object.

A lot of the libraries functions take a pointer to *fg8_error_t* (see *fg8_protocol_message_encrypted_create* for an example) in order to be able to communicate error conditions back to the calling code. As long as you reuse the same error object, you don't need to worry about memory management, the library will take care of that for you. There are some situations, where **YOU need to take care of the error object**. For example, if you do not want to reuse the same error object or at the end of the relevant scope, **you must use *fg8_error_destroy*** to cleanly destroy the error object and release its memory. Note that you **MUST NEVER** call `free()` or equivalent on an error object.

Example

```
#include <frontg8/error.h>
#include <stdio.h>
// other includes ...

int main()
  {
  fg8_error_t error;

  // Do something that stores an error in error ...

  fg8_error_destroy(error);
  }
```

Author Felix Morgner

Since 0.1.0

Warning Calling destroy multiple times on the same error object will result in undefined behaviour

All pointers to the message formerly contained in the error object may point to invalid memory after destruction.

Parameters

- `error` - The error object to be destroyed

1.1.2 Protocol fuctions

Encrypted messages

`#include <frontg8/protocol/message/encrypted.h>`

Typedefs

typedef `fg8_protocol_message_encrypted_t`

typedef `fg8_protocol_message_encrypted_const_t`

Functions

fg8_protocol_message_encrypted_t **`fg8_protocol_message_encrypted_create`**(char const *const *content*, size_t const *length*, *fg8_error_t* *const *error*)

Create an encrypted message.

Author Felix Morgner

Since 0.1.0

Return An encrypted message if construction succeedes. Otherwise, NULL is returned and if `error` is not NULL, it will be set to a new error object.

Parameters

- `content` - The content of new message. Might be NULL.
- `length` - The length of the data pointed to by `content`. Passing in 0 will result in an empty message.
- `error` - A pointer to an error object. Might be NULL.

fg8_protocol_message_encrypted_t **`fg8_protocol_message_encrypted_copy`**(*fg8_protocol_message_encrypted_const_t* *oth* *fg8_error_t* *const *error*)

Create an encrypted message by copying an existing one.

Author Felix Morgner

Since 0.1.0

Note Passing an NULL object for `other` will result in an error.

Return An encrypted message if copy construction succeedes. Otherwise, NULL is returned and if `error` is not NULL, it will be set to a new error object.

Parameters

- `other` - The source of the copy.

- `error` - A pointer to an error object. Might be NULL.

void **`fg8_protocol_message_encrypted_destroy`** (*fg8_protocol_message_encrypted_t* const *instance*)

Cleanup and destroy an encrypted message.

Author Felix Morgner

Since 0.1.0

Note You must use this function to cleanup messages you no longer need. Accessing an encrypted message object after destruction might lead to undefined behaviour.

Parameters

- `instance` - An existing encrypted message. Might be NULL.

fg8_protocol_message_encrypted_t **`fg8_protocol_message_encrypted_deserialize`** (char const *const *data*, size_t *length*, *fg8_error_t* *const *error*)

Create an encrypted message from serialized data.

Author Felix Morgner

Since 0.1.0

Note Passing in NULL for `content` will result in a default constructed encrypted message being returned.

Return An encrypted message if deserialization succeedes. Otherwise, NULL is returned and if `error` is not NULL, it will be set to a new error object.

Parameters

- `data` - A string pointing to serialized data. Might be NULL.
- `length` - The length of the data pointed to by `data`
- `error` - A pointer to an error object. Might be NULL.

char* **`fg8_protocol_message_encrypted_serialize`** (*fg8_protocol_message_encrypted_const_t* const *instance*, size_t * *length*, *fg8_error_t* *const *error*)

Serialize an encrypted message into a byte array.

Author Felix Morgner

Since 0.1.0

Return A pointer to the first byte of the serialized data. The memory is owned by the client and must be freed appropriately. If serialization fails, a NULL pointer is returned and error is set accordingly if a non NULL value was passed in.

Parameters

- `instance` - The message to be serialized. Must not be NULL.
- `length` - A pointer to a variable in which the size of the returned array will be stored. Might be NULL.
- `error` - A pointer to an error object. Might be NULL.

char const* **`fg8_protocol_message_encrypted_get_content`** (*fg8_protocol_message_encrypted_const_t* const *instance*, size_t *const *length*, *fg8_error_t* *const *error*)

Get the content of an encrypted message.

Author Felix Morgner

Since 0.1.0

Return A string containing the content data of `instance` or NULL if the encrypted message has no content (e.g is default-initialized). The memory is managed by the instance and must not be freed. If `error` is not NULL and an error occurs, it will be set to point to a new error object.

Parameters

- `instance` - An encrypted message. Must not be NULL.
- `length` - A pointer to a variable in which the size of the returned array will be stored. Might be NULL.
- `error` - A pointer to an error object. Might be NULL.

void **`fg8_protocol_message_encrypted_set_content`** (*fg8_protocol_message_encrypted_t* const *instance*, char const *const *content*, size_t const *length*, *fg8_error_t* *const *error*)

Set the content of an encrypted message.

Author Felix Morgner

Since 0.1.0

Note If an error occurs, `instance` will remain unchanged and `error` will be set accordingly if a non NULL value was passed in

Parameters

- `instance` - An encrypted message. Must not be NULL.
- `content` - The new content of the message. Passing in NULL will clear the message content.
- `length` - The length of the data pointed to by `content`.
- `error` - A pointer to an error object. Might be NULL.

bool **`fg8_protocol_message_encrypted_is_valid`** (*fg8_protocol_message_encrypted_const_t* const *instance*)

Check if an encrypted message is in a valid state (e.g has content)

Author Felix Morgner

Since 0.1.0

Return true if the message is valid, false otherwise

Parameters

- `instance` - An encrypted message.

bool **`fg8_protocol_message_encrypted_compare_equal`** (*fg8_protocol_message_encrypted_const_t* const *left*, *fg8_protocol_message_encrypted_const_t* const *right*)

Compare two encrypted messages for equality.

Author Felix Morgner

Since 0.1.0

Note Two messages are considered equal iff they have the same content. NULL values always compare unequal.

Return true if the messages are equal, false otherwise

Parameters

- `left` - An encrypted message (“left-hand side”)
- `right` - An encrypted message (“right-hand side”)