

# ***SElink™***

# ***Data-in-Motion Protection***

## *Technical Documentation*

Release: February 2021





## Proprietary Notice

The present document offers information subject to the terms and conditions described hereinafter. The authors reserve the possibility to change the content and information described in this document and to update such information at any time, without notice. Despite the attention that has been taken in preparing this document, typographical errors or omissions may have occurred.

## Authors

**Matteo FORNERO** (Researcher, CINI Cybersecurity National Lab) [matteo.fornero@consorzio-cini.it](mailto:matteo.fornero@consorzio-cini.it)

**Nicoló MAUNERO** (PhD candidate, Politecnico di Torino) [nicolo.maunero@polito.it](mailto:nicolo.maunero@polito.it)

**Paolo PRINETTO** (Director, CINI Cybersecurity National Lab) [paolo.prinetto@polito.it](mailto:paolo.prinetto@polito.it)

**Gianluca ROASCIO** (PhD candidate, Politecnico di Torino) [gianluca.roascio@polito.it](mailto:gianluca.roascio@polito.it)

**Antonio VARRIALE** (Managing Director, Blu5 Labs Ltd) [av@blu5labs.eu](mailto:av@blu5labs.eu)

## Trademarks

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by Blu5 View Pte Ltd. Other brands and names mentioned herein may be the trademarks of their respective owners. No use of these may be made for any purpose whatsoever without the prior written authorization of the owner company.

## Disclaimer

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN “AS IS” BASIS AND ITS AUTHORS DISCLAIM ALL WARRANTIES, EXPRESS, OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PURPOSE. THE SOFTWARE IS PROVIDED TO YOU “AS IS” AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEREUNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY LIKELIHOOD OF SUCH DAMAGES.





## Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>SEcube™ SDK libraries and dependencies</b>	<b>7</b>
<b>3</b>	<b>SElink™ hardware requirements</b>	<b>8</b>
<b>4</b>	<b>SElink™ architectural requirements</b>	<b>8</b>
<b>5</b>	<b>SElink™ environment setup</b>	<b>8</b>
<b>6</b>	<b>The SElink™ class</b>	<b>8</b>
<b>7</b>	<b>SElink™ APIs</b>	<b>9</b>
<b>8</b>	<b>SElink™ example</b>	<b>10</b>



## 1 Introduction

This user manual describes how to use the **SElink™** library. If you need additional details about the **SElink™** library, please check out the documentation of the **SEcube™** Open Source SDK<sup>1</sup>.

**SElink™** is a library that allows you to easily encrypt and decrypt data with the **SEcube™** without having to worry about low level security details. **SElink™** cannot be used without the **SEcube™** device.

**SElink™** is not a tool that runs on top of preexisting software, it requires custom software to be developed specifically to use the **SElink™** APIs in order to encrypt and decrypt data. **SElink™** is intended to be used for the encryption and decryption of *data in motion*, for the *data at rest* scenario please have a look at **SEfile™**.

**SElink™** grants the confidentiality, integrity and authentication of data using AES256-HMAC-SHA256; moreover, it offers the possibility to serialize and deserialize data before/after their transmission over any network (base64 encoding is used). A simplified overview of how **SElink™** works is shown in Figure 1.

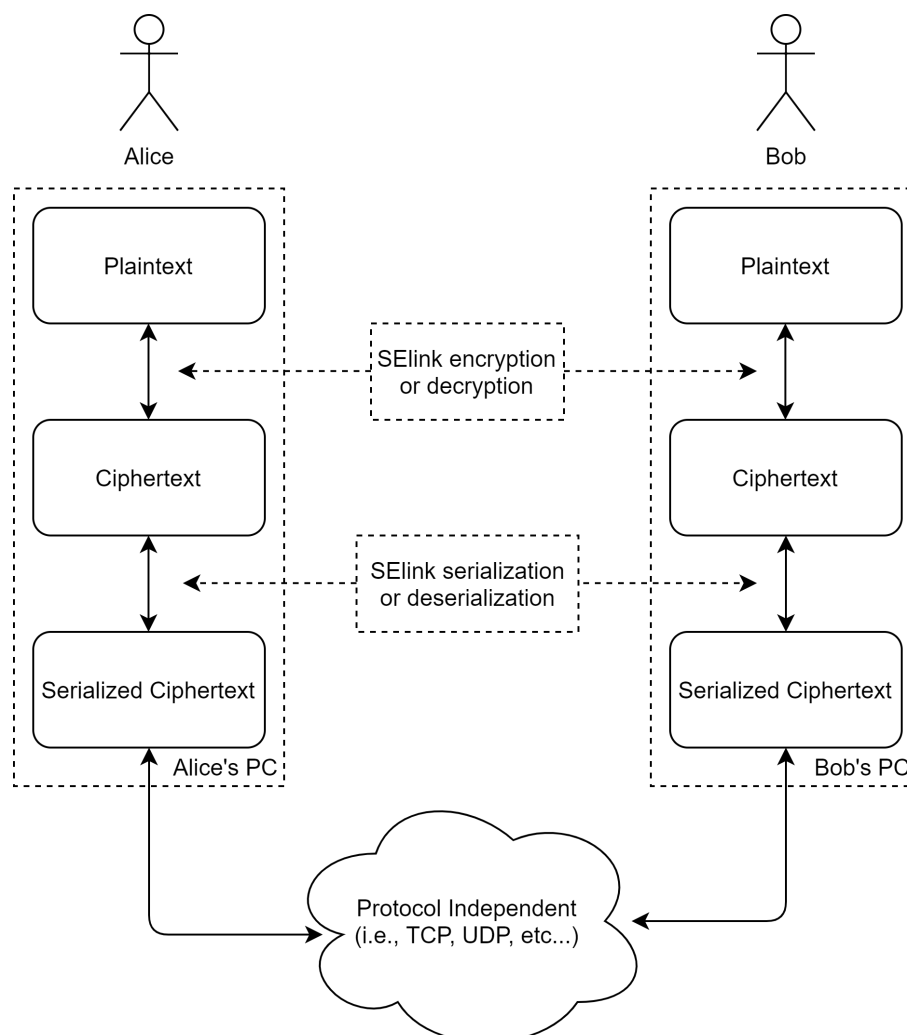


Figure 1: Simplified **SElink™** overview.

<sup>1</sup><https://www.secube.eu/resources/open-sources-sdk/>

## 2 SEcube™ SDK libraries and dependencies

The SEcube™ SDK consists of several libraries that run on a host computer to which the SEcube™ is connected. These libraries have been developed according to a hierarchical structure, therefore there are specific dependencies between them.

Table 1 summarizes the requirements of the SEcube™ libraries (Y required, N not required). Each row identifies a library, each column identifies a dependency from another library. For example, SEkey™ depends on L0, L1, SEfile™ and the Secure Database.

	L0	L1	SEfile	SElink	SEkey	SEcure DB
L0	-	N	N	N	N	N
L1	Y	-	N	N	N	N
SEfile	Y	Y	-	N	optional	optional
SElink	Y	Y	N	-	optional	N
SEkey	Y	Y	Y	N	-	Y
SEcure DB	Y	Y	Y	N	N	-

Table 1: Requirements and dependencies of SEcube libraries.

Notice that there are optional dependencies. In the case of SEfile™, you do not need to include also the SEkey™ library if you do not plan to use the Key Management System; similarly, you do not need to include the Secure Database library if you do not plan to use encrypted SQLite databases. In the case of SElink™, you do not need to include the SEkey™ library if you do not plan to use the Key Management System, resorting instead on manual key management.

In Figure 2 you can see how a SEcube™ project can be structured inside the IDE. This screenshot, in particular, was taken from the project that is actually used to develop the SDK. You can easily recognize, in fact, the folder related to the SDK (named 'sources', containing L0 and L1 APIs), the folder of SEkey™, SElink™, the Secure Database and SEfile™.

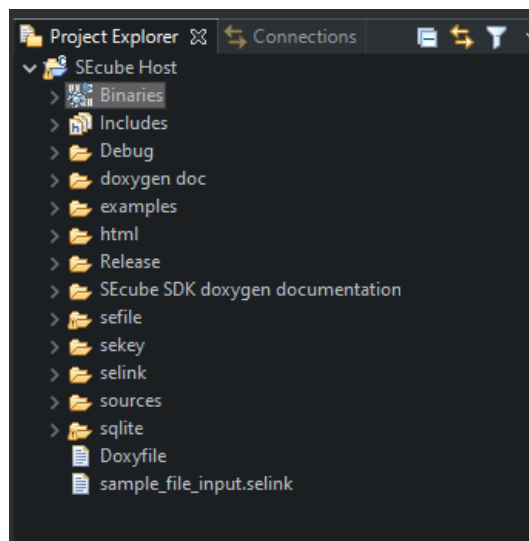


Figure 2: An example of how a SEcube™ project can be structured.

### 3 SElink™ hardware requirements

SElink™ simply requires a SEcube™ device and a host computer. Each person interested in using SElink™ must own a dedicated SEcube™ device.

### 4 SElink™ architectural requirements

SElink™ does not require any specific underlying architecture in order to be used. The library encrypts and decrypts the data before they are used by applications at the higher levels, therefore it is completely independent from the protocol and the infrastructure you want to use to distribute the *data in motion*.

### 5 SElink™ environment setup

SElink™ has been compiled and tested on the following platforms:

- Windows 10 64-bit (10.0.18363 build 18363), Eclipse 2019-12, Mingw-w64 (x86\_64-8.1.0-win32-seh-rt\_v6-rev0)
- Ubuntu 18.04.4 LTS 64-bit, Eclipse 2019-12, Linux GCC/G++

Follow these steps in order to setup and compile SElink™ :

1. go to <https://www.secube.eu/resources/open-sources-sdk/> and download the Open Source SEcube™ SDK;
2. extract the downloaded archive;
3. go to <https://www.secube.eu/resources/open-sources-sdk/> and download the SElink™ SDK;
4. extract the downloaded archive and copy the *selink* folder into the folder where you extracted the Open Source SEcube™ SDK (see Figure 2 as a general example);
5. launch your IDE (i.e., Eclipse) in order to create a dedicated SEcube™ project.

### 6 The SElink™ class

The SElink™ API heavily resorts on the features provided by the L1 layer. In order to manage the data encrypted by SElink™, the API is based on a 'SElink' object that encapsulates a 'SEcube\_ciphertext' object, the same that is used in L1 API. On top of this, SElink™ introduces also a couple of methods to serialize and deserialize the data, simply because SElink™ is focused on *data in motion*. The serialization is done using the base64 encoding.

```
class SElink{
public:
    SEcube_ciphertext ciphertext;
    void serialize(unique_ptr<char[]>& serialized_data, size_t&
        serialized_size);
    void deserialize(unique_ptr<char[]>& buffer_b64, size_t
        buffer_b64_size);
};
```





In conclusion, a 'SElink' object is, by all means, equal to the 'SEcube\_ciphertext' object already used by L1 APIs. The only different is that this new object also embeds the ability to be serialized and deserialized. Moreover, the encryption/decryption functions of SElink™ are based on the 'SElink' object.

## 7 SElink™ APIs

This section provides a brief overview about the SElink™ APIs. For more details about their implementation, please refer to the extensive comments in the source code. You can also process the code through the Doxygen engine in order to generate the documentation.

```
int selink_encrypt_manual(L1 *l1, shared_ptr<uint8_t[]>
    plaintext, size_t plaintext_size, SElink& ciphertext,
    uint32_t key);
```

This function is used to encrypt a plain text with SElink™. The 'manual' in the name of the function means that the caller is supposed to provide a key ID (an unsigned integer on 4 bytes) to the function, if the SEcube™ contains a key with the provided ID, then the plain text is encrypted. If the function succeeds, the 'SElink' object passed as fourth is filled with the cipher text and other attributes. Then, this object can be serialized in order to be sent to the receiver (serialization might not be always required). Finally, notice that the caller is supposed to provide the plain text as an array of bytes, passing also the size of the array.

```
int selink_encrypt_auto(L1 *l1, shared_ptr<uint8_t[]> plaintext,
    size_t plaintext_size, SElink& ciphertext, vector<std::
    string>& recipient);
```

This function is, by all means, equal to the previous one. The only difference is that the user is not supposed to provide the ID of the key to be used for encryption. The user, in fact, should only provide the list of recipients. This list might include one or multiple users from SEkey™, or even an entire group from SEkey™. The list must be filled with the IDs of the recipients. When issued, this function will automatically resort to SEkey™ to find the most suitable key for encryption, then it will fill the SElink object passed as fourth parameter with the encrypted data. As usual, serialization can be performed after encryption.

```
int selink_decrypt(L1 *l1, shared_ptr<uint8_t[]>& plaintext,
    size_t& plaintext_size, SElink& ciphertext);
```

This function is used to decrypt a payload that was encrypted by one of the two functions described above. Notice that the caller must provide the cipher text object, SElink™ will decrypt the cipher text and it will automatically verify the signature, providing therefore integrity and authentication of the data. In case of success, the plain text will be stored in the byte array passed as second parameter. In case of error, for example because the signatures of the data do not match, an error is returned.

```
void serialize(unique_ptr<char[]>& serialized_data, size_t&
    serialized_size);
```

Serialize the encrypted data stored by a SElink object using the base64 encoding. This method belongs to the SElink class. The serialized buffer is stored in the first parameter, the size of the serialized buffer is stored in the second parameter.

```
void deserialize(std::unique_ptr<char[]>& buffer_b64, size_t
    buffer_b64_size);
```



Deserialize a data buffer that was previously serialized with the corresponding method of the SElink class. The first parameter is the buffer containing the serialized data, the second parameter is the size of the serialized data. After completion, this method stores the deserialized data inside the SElink object upon which the deserialize function is called. After deserialization, the 'SElink' object can be passed as input to the selink\_decrypt() function in order to decrypt the data. This method belongs to the SElink class.

## 8 SElink™ example

This is a very basic example about how to use SElink™ to encrypt and decrypt data. A complete example is provided in 'examples' folder of the SEcube™ Open Source SDK<sup>2</sup>; notice that the source code shown below omits many instructions that are needed in order to use the SEcube™ (i.e., login on the device).

```
/* SENDER */
unique_ptr<L1> l1 = make_unique<L1>();
/* L1 object is used to execute the login on the SEcube before
   using any other API */
shared_ptr<uint8_t[]> plaintext_ptr(new uint8_t[plaintext_size])
    ; /* the plaintext must be initialized with the data that
       must be encrypted */
SElink ciphertext;
int rc = selink_encrypt_manual(l1.get(), plaintext_ptr,
    plaintext_size, ciphertext, key); /* encrypt the plaintext */
if(rc != SELINK_OK){
    // error!
}
size_t serialized_size = 0;
unique_ptr<char[]> serialized;
ciphertext.serialize(serialized, serialized_size); /* serialize
    the data */

/* RECEIVER */
unique_ptr<L1> l1 = make_unique<L1>();
/* L1 object is used to execute the login on the SEcube before
   using any other API */
unique_ptr<char[]> deserialized_ptr = make_unique<char[]>(
    serialized_size); /* buffer with the serialized data */
SElink deserialized; /* object containing the ciphertext */
deserialized.deserialize(deserialized_ptr, serialized_size); /*
    deserialize the data */
shared_ptr<uint8_t[]> decrypted;
int rc = selink_decrypt(l1.get(), decrypted, decrypted_size,
    deserialized); /* decrypt the data */
if(rc != SELINK_OK){
    // error!
}
```

<sup>2</sup><https://www.secube.eu/resources/open-sources-sdk/>

