Zostera

Release 0.0.1

SpartanerSpaten

Oct 19, 2019

CONTENTS

The simple managing tool for Sockets and Encryption in C++.

CHAPTER

ONE

CONTENT

1.1 Intro

1.1.1 Introduction

This Library manages System Sockets in C++ and extends them with Encryption. This Currently only works for Linux Sockets also when you find in options.h the option for compiling for Windows this is not ported yet.

1.2 Setup

1.2.1 Requirements

This library only supports Linux distributions. This Library is depending on CryptoPP.

1.2.2 Building from Source

```
$ git clone https:/github.com/SpartanerSpaten/Zostera
```

Clones the repository.

You have following Compiling options

- -D_USECRYPTOPP=(ON/OFF)

If this is ON you have to install the cryptopp library. This will provide you with RSA and AES encryption extension for your sockets

- -D_UNITTESTS=(ON/OFF)

Will require the criterion unit testing library will compile unit tests which can be executed with make test or ./test/zostera_unit_tests

- -D_EXAMPLES=(ON/OFF)

Will compile examples in the example directory. Pls check if you want to add some parameters like host and port first.

```
$ mkdir build && cd build
$ cmake ..
$ make
```

1.2.3 Installing

```
$ make install
```

Installs the Library most of the times under /usr/local/lib/.

PEM Extension for CryptoPP

See https://www.cryptopp.com/wiki/PEM_Pack#Downloads for more information

```
$ cmake -D_PEM=ON
```

1.3 Starter

1.3.1 Encrypting Stuff

Example encrypts a std::string with AES.

```
#include <Zostera/aes.h>

int main(){

    auto * aes = new Zostera::AES();
    aes->generateRandomAes(64);

    std::string message = "We will attack at dawn";

    std::string cipher = aes->encrypt(message);

}
```

Example encrypting with RSA.

```
#include <Zostera/rsa.h>

int main(){
    auto * rsa_private = new Zostera::RSA_PRIVATE();
    Zostera::RSA_PUBLIC * rsa_public = rsa_private->generate_private_rsa(2048);

    std::string message = "We will attack at dawn";

    std::string cipher = rsa_public->encrypt(message);
    std::string recovered = rsa_private->decrypt(cipher);

}
```

1.4 Encryption

This Entry describes supported encryption methods. Its is more efficient and easier to use these Models in combination with the keyring and cryptosys class. Because they handle the entire encryption process and make sure the classes always have the right input size.

1.4.1 Key Class

#include <Zostera/Key.h>

CKeyWrapper is essentialy the wrapper class for all encryption methods. It has 2 usefull functions for the user.

class CKeyWrapper
Subclassed by *Zostera::AES*, *Zostera::RSA_PRIVATE*, *Zostera::RSA_PUBLIC*

Public Functions

virtual std::string **exportKey**() = 0
Returns the Key has Hex encoded string.

virtual int **readKey**(**const** char *) = 0
Creates the internal Key structure based on the given file.

Parameters

- `path`: Path to a given valid keyfile.

virtual int **writeKey**(**const** char *) = 0
Saves the internal Key structure into a given file.

Parameters

- `path`: Path (string) where the keyfile should be created / exists.

virtual int **writeKey**(std::ofstream&) = 0
Saves the internal Key structure into a given file.

Parameters

- `path`: iostream which piplines data into the desired file.

virtual int **readKey**(std::istream&) = 0
Creates the internal Key structure based on the given file.

Parameters

- `path`: iostream which provides data of the desired keyfile..

virtual std::string **encrypt**(std::string&) = 0
Encrypts the given string.

Parameters

- `message`: PlainText Message

virtual std::string **decrypt**(std::string&) = 0
Decrypts the given string.

Parameters

- `message`: Cipher Text Message

virtual std::tuple<CryptoPP::byte *, size_t> **encrypt**(std::tuple<CryptoPP::byte *, size_t>) = 0
Encrypts the given data.

Parameters

- `message`: Tuple of a CryptoPP::byte pointer and the corresponding length (Plaintext).

virtual std::tuple<CryptoPP::byte *, size_t> **decrypt**(std::tuple<CryptoPP::byte *, size_t>) = 0
Decrypts the given data.

Parameters

- `message`: Tuple of a CryptoPP::byte pointer and the corresponding length (Ciphertext).

std::tuple<char *, size_t> **encrypt**(std::tuple<char *, size_t> *input*) = 0
Encrypts the given data.

This Function converts your `char *` into the internal used `CryptoPP::byte*` and calls the normal byte encryption method. `char *` is the pointer too your byte array and `size_t` is the corresponding length. So consider you do it manually

Parameters

- `input`: Tuple of a char pointer and the corresponding length (Plaintext).

std::tuple<char *, size_t> **decrypt**(std::tuple<char *, size_t> *input*) = 0
Decrypts the given data.

This Function converts your `char *` into the internal used `CryptoPP::byte*` and calls the normal byte encryption method. `char *` is the pointer too your byte array and `size_t` is the corresponding length. So consider you do it manually

Parameters

- `input`: Tuple of a char pointer and the corresponding length (Ciphertext).

virtual std::tuple<unsigned int, unsigned int, std::string> **expectedInputSize**() = 0
Returns important relevant information about the encryption method.

This Function returns the max input size, blocksize and the typ of encryption method (BLOCK, ASYNC)

Public Members

std::string **str_procedure**
Representing the typ of method.

1.4.2 Block Ciphers

AES

#include <Zostera/aes.h>

AES is a block cipher which has a block size of 16 bytes.

class AES : **public** Zostera::*CKeyWrapper*

Public Functions

int **generateRandomAes**(unsigned long)
Generates Random *AES* Key and Init Vector based on the given key strength;.

Parameters

- `input_key_strength`: `unsigned char` Key strength

int **setKey**(**const** std::string &*istr_key*)
Sets your own key as one randomly generated one pls make sure it is 16 bytes long.

Parameters

- `key`: `std::string` New key

std::string **exportKey**()
Exports the Key as HEX encoded string.

int **writeKey**(**const** char **path*)
Writes your *AES* Key into the file given by the path.

Parameters

- `path`: Path to the file.

int **readKey**(**const** char **path*)
Creates the *AES* Key based on the given path to a file.

Parameters

- `path`: Path to the keyfile

int **writeKey**(std::ofstream &*keyfile*)
Saves the *AES* Key into the file given by the stream.

Parameters

- `path`: ofstream

int **readKey**(std::istream &*keyfile*)
Creates the Private Key based on the given stream.

Parameters

- `path`: ifsteam

std::string **encrypt**(std::string &*message*)
Encrypts the given plain text.

Parameters

- `message`: `std::string` plaintext message

std::string **decrypt**(std::string &*cipher*)
Decrypts the given cipher text.

Parameters

- `message`: ciphertext

std::tuple<CryptoPP::byte *, size_t> **encrypt** (std::tuple<CryptoPP::byte *, size_t> *plaintext*)
Encrypts the given plain text.

Parameters

- `message`: tuple made of `CryptoPP::byte*` pointer and the length of the text.

std::tuple<CryptoPP::byte *, size_t> **decrypt** (std::tuple<CryptoPP::byte *, size_t> *cipher*)
Decrypts the given cipher text.

Parameters

- `message`: tuple made of `CryptoPP::byte*` pointer and the length of the text.

std::tuple<unsigned int, unsigned int, std::string> **expectedInputSize** ()
Returns important relevant information about this *AES* Key.

This Function returns the max input size (1), blocksize (16), and encryption method (BLOCK)

1.4.3 Asyncrone Ciphers

#include <Zostera/rsa.h>

Encrypting / Decrypting with the wrong key will throw you an *ZosteraExcpetion* be aware of that.

RSA Private Key

class RSA_PRIVATE : **public** Zostera::*CKeyWrapper*

Public Functions

Zostera::*RSA_PUBLIC* ***generate_private_rsa** (size_t *intBitLength*)
Generates your private key and a public Key.

Parameters

- `int_bit_length`: Bit key strength

std::string **exportKey** ()
Exports your Private Keys as Hex Encoded string.

int **writeKey** (**const** char **path*)
Writes your Private Key into the file given by the path.

Parameters

- `path`: Path to the file.

int **readKey** (**const** char **path*)
Creates the Private Key based on the given path to a file.

Parameters

- `path`: Path to the keyfile

int **writeKey**(std::ofstream &*outfile*)
Saves the Private Key into the file given by the stream.

Parameters

- `path`: ofstream

int **readKey**(std::istream &*input_file*)
Creates the Private Key based on the given stream.

Parameters

- `path`: ifsteam

std::string **decrypt**(std::string &*cipher*)
Decrypts the given cipher text.

Parameters

- `message`: ciphertext

std::string **encrypt**(std::string&)

Warning DO NOT CALL THIS METHOD IT WILL THROW YOU AN ZOSTERA EXCEPTION: YOU CAN NOT ENCRYPT WITH A PRIVATE KEY

std::tuple<CryptoPP::byte *, size_t> **decrypt**(std::tuple<CryptoPP::byte *, size_t> *cipher*)
Decrypts the given cipher text.

Parameters

- `message`: tuple made of `CryptoPP::byte*` pointer and the length of the text.

std::tuple<CryptoPP::byte *, size_t> **encrypt**(std::tuple<CryptoPP::byte *, size_t>)

Warning DO NOT CALL THIS METHOD IT WILL THROW YOU AN ZOSTERA EXCEPTION: YOU CAN NOT ENCRYPT WITH A PRIVATE KEY

std::tuple<unsigned int, unsigned int, std::string> **expectedInputSize**()
Returns important relevant information about this Private Key.

This Function returns the max input size, blocksize (1), and encryption method (ASYNC)

RSA Public Key

class RSA_PUBLIC : **public** Zostera::*CKeyWrapper*

Public Functions

std::string **exportKey**()
Exports your Public Keys as Hex Encoded string.

int **writeKey**(**const** char **path*)
Writes your Public Key into the file given by the path.

Parameters

- `path`: Path to the file.

int **readKey**(**const** char **path*)
Creates the Public Key based on the given path to a file.

Parameters

- `path`: Path to the keyfile

int **writeKey**(std::ofstream *&outfile*)
Saves the Public Key into the file given by the stream.

Parameters

- `path`: ofstream

int **readKey**(std::istream *&input_file*)
Creates the Public Key based on the given stream.

Parameters

- `path`: ifsteam

std::string **decrypt**(std::string&)

Warning DO NOT CALL THIS METHOD IT WILL THROW YOU AN ZOSTERA EXCEPTION: YOU CAN NOT DECRYPT WITH A PUBLIC KEY

std::string **encrypt**(std::string *&message*)
Encrypts the given plain text.

Parameters

- `message`: `std::string` plaintext message

std::tuple<CryptoPP::byte *, size_t> **decrypt**(std::tuple<CryptoPP::byte *, size_t>)

Warning DO NOT CALL THIS METHOD IT WILL THROW YOU AN ZOSTERA EXCEPTION: YOU CAN NOT DECRYPT WITH A PUBLIC KEY

std::tuple<CryptoPP::byte *, size_t> **encrypt**(std::tuple<CryptoPP::byte *, size_t> *plaintext*)
Encrypts the given plain text.

Parameters

- `message`: tuple made of `CryptoPP::byte*` pointer and the length of the text.

std::tuple<unsigned int, unsigned int, std::string> **expectedInputSize**()
Returns important relevant information about this Public Key.

This Function returns the max input size, blocksize (1), and encryption method (ASYNC)

1.5 Sockets

Sockets currently wrap System Sockets.

1.5.1 Interface Classes

Interface Server

class SecSocket_Server
Subclassed by *Zostera::LinSecSocket_Server*

Public Functions

virtual int **send**(std::string&, int) = 0
Send the given information over the TCP connection with the given socket number.

Parameters

- `message`: `std::string` Message
- `socket`: `int` socket number

virtual int **send**(std::tuple<char *, size_t>, int) = 0
Send the given information over the TCP connection with the given socket number.

Parameters

- `message`: `std::tuple<char*, size_t>` Message
- `socket`: `int` socket number

virtual int **send**(std::shared_ptr<char>, size_t, int) = 0
Send the given bytes to the Sever depending if there is and cryptosys placed also encrypted.

Parameters

- `message`: `std::tuple<std::shared_ptr<char>, size_T>` smart char pointer and the length of the message.

virtual std::string **receiveString**(int) = 0
Receives Information from the TCP connection.

Parameters

- `socket`: `int` socket number

virtual std::tuple<char *, size_t> **receiveRawByte**(int) = 0
Receives Information from the TCP connection.

Parameters

- `socket`: `int` socket number

virtual std::tuple<std::shared_ptr<char>, size_t> **receiveByte**(int) = 0
Receives Information from the TCP connection.

Parameters

- `socket`: `int` socket number

virtual std::tuple<int, int, std::string> **listen_and_accept**() = 0
Will listen for incoming connections.

virtual int **bind**(std::string&, unsigned short) = 0
Will try to bind this socket to the given host and port.

Parameters

- `host`: `std::string`
- `port`: `unsigned short`

virtual void **placeEncryptionSys**(std::shared_ptr<Zostera::*CCryptoSys*>) = 0
Places the *CCryptoSys* which will be used for encryption and decryption of incoming data.

Parameters

- `ptr`: `CCryptoSys*` pointer to an already configured CryptoSys

virtual void **close**(int *socket*) = 0
Closes given socket.

Parameters

- `socket`: Socket that should be closed.

virtual bool **checkValidity**(int *socket*) = 0
Checks for validity of connection.

Parameters

- `socket`: Given socket that should bechecked

Interface Client

class SocketClientInterface
Subclassed by *Zostera::LinSecSocket_Client*

Public Functions

virtual int **send**(std::string&) = 0
Send the given String to the Server depending if there is and cryptosys placed also encrypted.

Parameters

- `message`: `std::string` the given string that should be send.

virtual int **send**(std::tuple<char *, size_t>) = 0
Send the given bytes to the Sever depending if there is and cryptosys placed also encrypted.

Parameters

- `message`: `std::tuple<char*, size_T>` char pointer and the length of the message.

virtual int **send**(std::shared_ptr<char>, size_t) = 0
Send the given bytes to the Sever depending if there is and cryptosys placed also encrypted.

Parameters

- `message`: `std::tuple<std::shared_ptr<char>, size_T>` smart char pointer and the length of the message.

virtual std::string **receiveString**() = 0
Receives Information.

virtual std::tuple<char *, size_t> **receiveRawByte**() = 0
Receives Information.

virtual std::tuple<std::shared_ptr<char>, size_t> **receiveByte**() = 0
Receives Information.

virtual int **connect**(std::string&, unsigned short) = 0
Will try to connect to a other listening socket.

Parameters

- `host`: `std::string`
- `port`: `unsigned short`

virtual void **placeEncryptionSys**(std::shared_ptr<Zostera::*CCryptoSys*>) = 0
Places the *CCryptoSys* which will be used for encryption and decryption of incoming data.

Parameters

- `ptr`: `CCryptoSys*` pointer to an already configured CryptoSys

virtual void **close**() = 0
Closes Connection.

virtual bool **checkValidity**() = 0
Checks for validity of connection.

1.5.2 Inherited Classes

Linux Server

class LinSecSocket_Server : **private** Zostera::*SecSocket_Server*

Linux Client

class LinSecSocket_Client : **private** Zostera::*SocketClientInterface*

1.6 KeyRings

#include<Zostera/key_ring.h>

KeyRing manages all your encryption methodes. It has for Encryption an decryption (asyncron) method you need 2 keyrings one for encryption purporsed and one for decryption

class CKeyRing

Public Functions

void **addKey**(Zostera::*CKeyWrapper* *wrapper)
Adds the given Key to the keylist.

Parameters

- `wrapper`: `CKeyWrapper*` instance of one of the classes that inherits from *`CKeyWrapper`*

void **wipeKeys**()
Deletes all keys that this keyring manages.

Zostera::*CKeyWrapper* ***getKey**(unsigned int)
Returns one of the key based on the given index. If the index is out of bound it will throw a Zostera Exception.

Parameters

- `index`: `unsigned int` index

void **load_and_append**(**const** std::string &*method*, **const** std::string &*encrypt*, **const** std::string &*mode*, **const** std::string &*file*)
Function that is primary used internally.

Parameters

- `method`: (M_AES; M_RSA)
- `encrypt`: (“1”, “0”)
- `mode`: `std::string`
- `file`: `std::string` path to the given file.

void **save**(**const** std::string &*path*, **const** std::string &*pur*)
Saves all keys that this keyring is managing.

Parameters

- `pur`: `std::string` tells if this keyring was used for encryption or decryption purposes.
- `path`: `std::string` directory where all the files should be put.

unsigned int **size**()
Returns how many keys this keyring is managing.

1.7 CryptoSys

#include<Zostera/crypto_sys.h>

CryptoSys is the Class that manages now your keyrings that you can chain varius Encryption metodes to increase complexity.

class CCryptoSys

Public Functions

~CCryptoSys()
Constructor.

Parameters

- `auto_create`: If the Keyrings should be initialized

void **putEncryptionKeyring**(Zostera::*CKeyRing* **kr*)
Adding the keyring for Encryption purposes.

Parameters

- `kr`: the keyring

void **putDecryptionKeyring**(Zostera::*CKeyRing* **kr*)
Adding the keyring for Decryption purposes.

Parameters

- `kr`: the keyring

std::string **encrypt**(std::string &*message*)
Encrypts the string with the given encryption keyring.

Parameters

- `message`: your plain text

std::string **decrypt**(std::string &*message*)
Decrypts the string with the given decryption keyring.

Parameters

- `message`: your cipher text

std::tuple<char *, size_t> **encrypt**(std::tuple<char *, size_t> *input*)
Encryption method for bytes.

Parameters

- `input`: Tuple of a char pointer and the corresponding length

std::tuple<char *, size_t> **decrypt**(std::tuple<char *, size_t> *input*)
Decryption method for bytes.

Parameters

- `input`: Tuple of a char pointer and the corresponding length

int **autoBuild**(std::string &*path*)
Builds the cryptosys with a given directory. It generates the Keyrings and all Keys.

Parameters

- `path`: Directory

int **autoSave**(std::string &*path*)
Saves the entire cryptosys into a given directory.

Parameters

- `path`: Directory

1.8 Development

1.8.1 Bug Reports

Pls Report Bugs here:

https://github.com/SpartanerSpaten/Zostera/issues

1.8.2 Contributing

Just Write me an E-Mail *revol-xut@protonmail.com*

1.8.3 Roadmap

- Fixing Winsock
- Elliptic Curve Cryptography
- Diffi-Hellman

INDEX

C

Z