Exercises on Secret Key Cryptography

Solve the following exercises and submit them until the communicated date. Base **all** of your programs on the *Private key cryptography project* template.

LB-SKC 01.

A symmetrically encrypted communication between two parties, e.g., a client and a server, requires exchanging a symmetric key prior to (secret) messages being sent or received. One possibility to exchange a key securely is the X25519 key exchange protocol. The first step of this protocol requires each of the two parties to generate a secret random number, also referred to as *secret key* (*sk*) in this context. From this secret key, a *public key* (*pk*) is derived in a special way which is hard to reverse. (Note that, despite their names, the public and secret keys **cannot** be used for asymmetric message encryption or decryption.) Write a program which generates a public key and a secret key according to X25519. Use the function `crypto_kx_keypair` (see `https://libsodium.gitbook.io/doc/key_exchange#usage`) to do so. The program must not expect any command line arguments and must print both keys as hexadecimal character strings to `std::cout`, preceded by the text `Public key:` and `Secret key:`, respectively, as well as a delimiting space character.
Hint: You may use the following code snippet to print a "byte" array as a hexadecimal character string to `std::cout`*:*

```
1  #include <iostream>
2  #include <iomanip>
3
4  void PrintKey(const unsigned char key[], const size_t
       ↪ key_length, const std::string &key_name)
5  {
6    std::cout << key_name << ": " << std::hex;
7    for (size_t i = 0; i < key_length; i++)
8      std::cout << std::setw(2) << std::setfill('0') <<
           ↪ static_cast<int>(key[i]);
9    std::cout << std::endl;
10 }
```

Example calls (with command line arguments) for testing:
`./test` must print an output like (note that the key digits will, of course, differ):

```
Public key: 709dd189c8b6bfef2ad9ed6ab4f0f01505868d2fcf6767a25260781cde46b20b
Secret key: 65b646ab71d2a27b44337ff5805ae081276268de02549fbc53a008686b0a3b76
```

LB-SKC 02.

After a public key and a secret key have been generated for both parties participating in the X25519 key exchange protocol, they share their public keys with one another. In the second step of the protocol, each party uses all keys

known to it to independently derive a joint symmetric key based on the special mathematical properties of the secret and public keys.
Write a program which derives a symmetric key according to X25519 from the public and secret key of one party and the public key of the other party. Derive **only one** session key by using the functions `crypto_kx_server_session_keys` and `crypto_kx_client_session_keys`, respectively (see `https://libsodium.gitbook.io/doc/key_exchange`). The program must expect four command line arguments in the following order: the party (`Client` or `Server`), the party's public key, the party's secret key and the counter-party's public key. All keys must be expected as hexadecimal character strings. The derived symmetric key must be printed as a hexadecimal character string to `std::cout`.
Hint: You may use the following code snippet to convert a hexadecimal character string back into a "byte" array:

```
1  #include <sstream>
2  #include <cstring>
3  #include <iomanip>
4
5  bool HexStringToArray(const char * const text, unsigned char
       ↪ array[], const size_t array_size)
6  {
7    if (strlen(text) != 2 * array_size)
8      return false;
9    for (size_t i = 0; i < array_size; i++)
10   {
11     const std::string text_part(text + 2 * i, 2); //Process
           ↪ 2 characters (one byte) at a time
12     std::stringstream s(text_part);
13     s >> std::hex;
14     int value;
15     s >> value;
16     array[i] = value;
17   }
18   return true;
19 }
```

Example calls (with command line arguments) for testing:
`./test Client 709dd189c8b6bfef2ad9ed6ab4f0f01505868d2fcf6767a25260781cde46b20b`
`65b646ab71d2a27b44337ff5805ae081276268de02549fbc53a008686b0a3b76`
`2e4552716b700ca92f982a26a465b91a6c6cbbcd66253ffd539d87528096561c`
must print the output
`1d053900b110fa73a1560515fac208372d366e682a6d3ac2606b772e1495897c`.
`./test Server 2e4552716b700ca92f982a26a465b91a6c6cbbcd66253ffd539d87528096561c`
`8fd8991db6b4b0794c79cdf4bba1bf6d133728b42ecbf6300027b0bb256ec8b3`
`709dd189c8b6bfef2ad9ed6ab4f0f01505868d2fcf6767a25260781cde46b20b`
must print the output
`1d053900b110fa73a1560515fac208372d366e682a6d3ac2606b772e1495897c`.

LB-SKC 03.

After a successful key exchange, both parties can use the derived session key for a symmetric cipher, e.g., AES-256, to encrypt and decrypt messages. In many cases, it makes sense to perform authenticated encryption, i.e., to append a message authentication code (think of it as a checksum) to the encrypted message so that modifications to the encrypted message can be detected.
Write a program which is capable of performing both, authenticated encryption and decryption of a message using AES-256 and a key. Use the functions `crypto_aead_aes256gcm_encrypt` and `crypto_aead_aes256gcm_decrypt` (see `https://libsodium.gitbook.io/doc/secret-key_cryptography/aead/aes-256-gcm`) to do so. The program must expect three command line arguments in the following order: the operation (`Encrypt` or `Decrypt`), the message as a character string (for `Encrypt`) or the ciphertext as a hexadecimal character string (for `Decrypt`) and the symmetric key as a hexadecimal character string. When encrypting, the ciphertext must be printed to `std::cout` as a hexadecimal character string. When decrypting, the decrypted message must be printed as a character string to `std::cout`. For both, encryption and decryption, use `ad = NULL`, `adlen = 0` and a zero nonce as follows:

```
1 const unsigned char nonce[crypto_aead_aes256gcm_NPUBBYTES] =
    ↪ {0};
```

Beware: ***Never*** *use a constant nonce in real-world applications! Here it is only used to simplify the implementation.*

Example calls (with command line arguments) for testing:
`./test Encrypt Hello`
`1d053900b110fa73a1560515fac208372d366e682a6d3ac2606b772e1495897c`
must print the output `000cec41a8cb94ef9f5f52b3b4accdae72ac322d31`.
`./test Decrypt 000cec41a8cb94ef9f5f52b3b4accdae72ac322d31`
`1d053900b110fa73a1560515fac208372d366e682a6d3ac2606b772e1495897c`
must print the output `Hello`.

LB-SKC 04. (<u>not</u> to be submitted)

Form groups of two, where one person takes on the role of the client and the other person takes on the role of the server. Verify your implementations from exercises 01. to 03. individually by performing the following steps:

1. Generate a public key and a secret key (exercise 01.) and mutually share the public keys, e.g., via e-mail.
2. Derive a session key from the available keys (exercise 02.).
3. Encrypt a message (example 03.) and send it to the other party. Conversely, decrypt the message received from the other party (exercise 03.).

Make sure that each person only shares the data required by the respective protocol, i.e., each person keeps their secret keys and the session key secret.