International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307

OPEN ACCESS

Available Online at : www.ijsrcseit.com
doi : https://doi.org/10.32628/CSEIT2410213

Secure Messaging Using Post-Quantum Key Sharing Based on CSIDH and Fujisaki-Okamoto Transform

Rakotondramanana Radiarisainana Sitraka[*1], **Ramafiarisona Hajasoa Malalatiana**[1], **Randrianandrasana Marie Emile**[1], **Henintsoa Stephana Onjaniaiana**[2]

[*1]Telecommunication-Automatic-Signal-Image-Research, Laboratory, Doctoral School in Science and Technology of Engineering and Innovation, University of Antananarivo Antananarivo, Madagascar

[2]Telecommunication, High School Polytechnic of Antananarivo, University of Antananarivo, Madagascar

ARTICLEINFO

Article History:

Accepted: 10 March 2024

Published: 25 March 2024

Publication Issue

Volume 10, Issue 2

March-April-2024

Page Number

208-220

ABSTRACT

Preserving the confidentiality of information exchanges relies fundamentally on an end-to-end encryption system, involving the use of a secret key to secure the entire communication. However, with the imminent emergence of quantum computing, threats to traditional encryption systems are multiplying. This is where post-quantum key sharing, in particular the Commutative Super Isogenies Diffie Hellman (CSIDH) algorithm, comes in. The CSIDH uses the ideal of some class number to a morphism of elliptic curve for calculating the shared key. It offers an innovative solution for secure key generation between two users, while providing a robust defense against potential attacks from quantum computers, whose computing power is redefining the limits of cryptographic security, based on the mathematical foundations of elliptic curves and isogeny. This technological advance represents an essential pillar in preserving the confidentiality of communications, in a context where security challenges are constantly evolving. Combining with Fujisaki-Okamoto transform, socket, and Linux; a chat application over python could be created for transmitting secure messages.

Keywords : CSIDH, Decryption, Encryption, Fujisaki-Okamoto, Messaging, Post-Quantum

I. INTRODUCTION

The need to ensure the security of communications between individuals has always been apparent. Thus, ensuring that exchanged messages are not modified during transit, or worse, intercepted by third parties, is a constant concern that only dissipates with the emergence of enhanced security methods.

Today, with the emergence of powerful computers and the performance of new calculation algorithms, cryptography based on theories number and algorithms of mathematical logic has become vulnerable to attacks. However, attacks occur periodically to undermine user trust. In cryptography, it is necessary to express the objectives and means of the attacker, i.e., what they seek to do and how they act within the system. This is where cryptography shows its limitations in the face of reality.

Furthermore, a new form of information protection is implemented through the exploitation of elliptic curves and isogenies. The use of post-quantum cryptography aims to make information inviolable against attacks from quantum machines while using classical hardware.

II. LITTERATURE REVIEW

Classical cryptography encompasses the traditional methods of securing information through symbols and algorithms. Developed over centuries, these techniques have been crucial for ensuring the confidentiality of sensitive data in various contexts such as military and diplomatic communications. Despite modern advancements, classical cryptography remains significant for understanding the foundational principles of encryption [1-7].

In cryptography, there are primarily two "dual" disciplines:

- Cryptography, which focuses on securing information.
- Cryptanalysis, which aims to attack crypto-system.

Cryptography therefore aims to secure the transmission between the sender and the recipient. Information security is understood at several levels:

- **Confidentiality:** Only the recipient Bob (and possibly the sender Alice) can understand the transmitted information.
- Authentication: Each party is indeed who they claim to be.
- Integrity: The transmitted information has not been altered during its journey.
- Non-Repudiation: The sender cannot deny having sent the information.
- Access Control: Only authorized person(s) can access the information.

A. Beginning of Cryptography

Historical encryption systems, no longer in use due to simplicity of cryptanalysis, highlight the importance of understanding various attacks and the computational effort modern computers would require to break crypto-systems [1].

B. Scytale

The scytale is the oldest known encryption system. It involves wrapping a strip of leather around a wooden rod, writing the message perpendicular to the wrapping, then, unwinding the leather to read the message. To decrypt, the recipient needs a scytale of the -same diameter as the one used for encryption.

- Caesar Cipher

The Caesar cipher is a substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on. This shift value is the key to encrypting and decrypting the message as illustrated on the Table I.

TABLE I
CAESAR CIPHER EXAMPLE

Plain text	**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
Encrypted text	**D E F G H I J K L M N O P Q R S T U V W X Y Z A B C**

- Vigenère Cipher

The Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to shift letters in the plaintext. The keyword is repeated to match the length of the message, and each letter of the keyword determines the amount of shift applied to the corresponding letter in the plaintext as illustrated on the Table II. This creates a more complex cipher than the Caesar cipher, making it more resistant to frequency analysis.

TABLE II
CAESAR CIPHER EXAMPLE

Plain text	UN TEXTE CHIFFRE AVEC VIGENERE
Key	SE CRETS ECRETSE CRET SECRETSE
Encrypted	MR VVBMW GJZJYJI CMIV NMIVRXJI

- Enigma

Enigma was an electromechanical rotor machine used for encryption and decryption. It scrambled messages by passing electrical current through rotors, with settings determining the encryption. Its complexity and numerous configurations made decryption difficult for cryptanalysts.

C. Modern Cryptography

With the emergence of computers, traditional cryptographic methods are no longer sufficient for secure communication. Even complex systems like Enigma could be deciphered with computational power. Cryptanalysis of Enigma involves determining the initial rotor positions, selecting from 5 rotors in 3 different orders, and permuting 20 letters on the front panel [1,2].

- Symmetric Crypto-systems

Symmetric crypto-system uses the same key for both encryption and decryption. The principle of operation involves applying mathematical algorithms to transform plaintext into ciphertext and vice versa using a shared secret key. Common examples include the Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Triple DES (3DES).

- New Paradigm

This section discusses new paradigms in cryptography, which may involve novel approaches or advancements beyond traditional cryptographic methods. It could include topics such as quantum cryptography, homomorphic encryption, or lattice-based cryptography. The principle of operation for each paradigm varies depending on its underlying mathematical principles and cryptographic techniques [2].

- Asymmetric Crypto-systems

In asymmetric cryptography, Alice generates a pair of keys: a public key and a private key. She selects two large prime numbers, p and q, and calculates $N = p.q$. Then, she chooses an integer e coprime with $\varphi(N) = (p - 1)(q - 1)$ and computes the integer d using the extended Euclidean algorithm, such that. $ed = 1 \bmod \varphi(N)$. Alice's public key is (N, e), while her private key is (N, d).

When Bob wants to send a message m to Alice, he calculates $c = m^e \, mod \, N$ and sends it to her. To decrypt the message, Alice uses her private key and computes as follow as the equation (1).

$$c^d = (m^e)^d \, \mathrm{mod} N = m \bmod N \qquad (1)$$

- Hybrid Cryptography

Hybrid cryptography merges symmetric and asymmetric encryption for efficiency and security. It generates a session key using asymmetric encryption, which encrypts the data symmetrically. By encrypting the session key with the recipient's public key, it addresses key distribution challenges and computational overhead, ensuring secure and efficient communication [3].

D. Computational Model

Various complexity notions required to assess the robustness of a crypto-system, will be explored as well as the main underlying problems in modern asymmetric cryptography. Then, the architecture of quantum computing and its implications on this same cryptography will be discussed. [1-7]

- Complexity Notions

Complexity notions refer to various measures used to evaluate the efficiency of algorithms. This includes concepts like Big O notation, which describes the upper bound on the time or space required by an algorithm as a function of the size of its input. Time and space complexity assess the resources needed by an algorithm to execute. Polynomial time algorithms are those whose running time is bounded by a polynomial function, while NP problems are those for which solutions can be verified in polynomial time but may not be found in polynomial time.

- Factorization and the RSA Problem

Factorization involves breaking down a composite number into its prime factors. The RSA encryption scheme relies on the difficulty of factoring large composite numbers, as it's based on the assumption that factoring the product of two large prime numbers is computationally infeasible. The security of RSA encryption hinges on the challenge of factorization, making it resistant to attacks.

- Factorization and the RSA Problem

Factorization involves breaking down a composite number into its prime factors. The RSA encryption scheme relies on the difficulty of factoring large composite numbers, as it's based on the assumption that factoring the product of two large prime numbers is computationally infeasible. The security of RSA encryption hinges on the challenge of factorization, making it resistant to attacks.

- Discrete Logarithm and the Diffie-Hellman Problem

The discrete logarithm problem involves finding the exponent in a given modular equation. The Diffie-Hellman key exchange protocol relies on the difficulty of solving the discrete logarithm problem in a finite field. This means that even if the values exchanged during the protocol are intercepted, it's computationally difficult for an eavesdropper to derive the shared secret key without solving the discrete logarithm problem.

- Quantum Computer and its Algorithms

Quantum computers leverage principles of quantum mechanics such as superposition and entanglement, to perform computations. Shor's algorithm, a quantum algorithm, efficiently factors large integers, posing a threat to RSA and other crypto-systems based on integer factorization.

III. METHODS AND MATERIAL

Post-Quantum Cryptography (PQC) is a rapidly emerging field within the realm of cybersecurity, prompted by the looming threat posed by quantum computers to traditional cryptographic algorithms. Unlike classical computers, quantum computers possess the potential to efficiently solve certain mathematical problems upon which many widely-used encryption schemes rely.

A. Elliptic Curves

An elliptic curve over a field K is a non-singular plane algebraic curve as follows as the equation (2).

$$\boldsymbol{y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_4x} \qquad (2)$$

Where $\boldsymbol{a_i} \in \mathbb{K}$
This form is called the Weierstrass form [8-12].

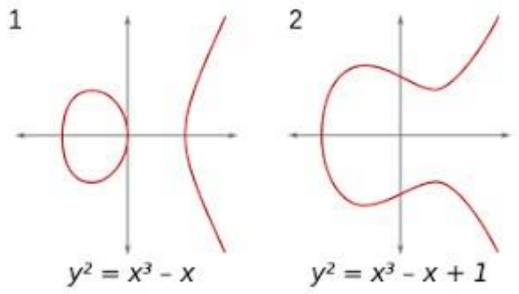


Figure 1. Elliptic curve

B. Isogeny-Based Crypto-system

Isogeny-based cryptography, originating from Couveignes' work in the late 1990s, was made resilient to post-quantum attacks by Jao and De Feo in 2011. It offers security against classical attacks and reasonable data sizes similar to current systems. However, its youth raises concerns about quantum resistance and practical performance compared to other post-quantum schemes. Isogeny graphs consist of nodes representing elliptic curves over F_q , with edges as isogenies of fixed degree. For supersingular elliptic curves, isogeny graphs exhibit the Ramanujan property and have a finite number of isomorphism classes, approximately p/12.There are three methods of cryptography based on isogeny:

- Authentication and Zero-Knowledge Proof of Identity
- Secret Key Establishment
- Encrypted Message Transmission

C. Isogeny-Based Cryptanalysis

Isogeny cryptanalysis is a specialized field of cryptography focused on analysing attacks against cryptographic systems relying on isogenies. Isogenies, mathematical functions preserving elliptic curve group structures, are integral to public-key cryptography, such as Super Isogeny Diffie-Hellman SIDH. Attacks in this area involve two main objectives: finding isogenies between known elliptic curves and exploiting isogeny-related vulnerabilities to break cryptographic systems. [6].

D. Computing security

Computing security is the set of technical, organizational, legal, and human means required to protect computer systems against threats and attacks. It aims to ensure the confidentiality, integrity, and availability of data and digital resources.

- Information systems environment

Statistics indicate that 40% of attacks are caused by users of an Information System (IS) themselves. Despite the array of technologies used to secure enterprise Information Systems (IS), attacks on IS still occur [12].

- Key Management and Security

Key exchange plays a crucial role in information systems, management and security by using CSIDH methods and Fujisaki-Okamoto transform.

E. Network security

Before specifying the models in use on networks, how information is transmitted will be explored briefly. in any case, what principles have been adopted. Network architecture models are composed of layers, relatively independent of each other. The analysis of network will be started by applications (high level) to physical transmission (low level).[13]

- Open System Interconnexion (OSI) model

The OSI model is not a network architecture, as it does not explicitly describe the protocols involved. This model consists of 7 layers:

Physical: Effectively transmits signals in the form of bits between parties.

Data Link: Converts bits into frames.

Network: Manages communication, ensures packet routing and addressing.

Transport: Routes messages.

Session: Synchronizes exchanges between remote tasks.

Presentation: Processes information to make it compatible between different applications.

Application: Interface between the user and the network.

- Transmission Control Protocol / Internet Protocol (TCP/IP)

The TCP/IP model is not significantly different from the OSI model. However, it only has 4 layers, although a hybrid model, separating the physical and Link layers, may be authoritative [13].

Network Access Layer: Ensures physical connection to route data.

Internet Layer: Injects packets into any network.

Transport Layer: Functions similarly to the Transport Layer of the OSI model.

Application Layer: This layer encompasses all high-level protocols (File Transfer Protocol or FTP, Simple Mail Transfer Protocol or SMTP, Hypertext Transfer Protocol or HTTP, Domain Name System, or DNS...).

F. Socket

Socket is used to establish a stream of data (byte) transmission between two machines or applications. Socket manipulation proceeds as follows [14-15]:

- Creation of the socket
- Configuration of the socket
- Closure of the previously established connection.

G. Transmission of message as a String

To enable data transmission, a server program will need to send data, and a client program will need to receive them. For this purpose, three functions will be used [16]:

- Send functions

The send function facilitates the transmission of data over a socket. It requires parameters such as the socket descriptor and a pointer to the data buffer containing the information to be sent, along with the size of the data in bytes and optional flags. Typically, data is sent in the form of a char array or buffer. Upon successful completion, send returns the number of bytes sent. If an error occurs, it returns -1.

- Recv function

Conversely, the recv function is employed to receive data from a socket. It shares similarities with the send function in terms of parameters, including the socket descriptor, a pointer to the buffer where received data will be stored, the maximum number of bytes to receive, and optional flags. Once data is received, it is stored in the specified buffer. The return value of recv indicates the number of bytes received. If the connection has been closed, it returns 0; otherwise, -1 denotes an error.

- Shutdown functions

Finally, the shutdown function serves to disable sending and receiving data on a socket. By providing the socket descriptor and an integer representing the desired shutdown action (such as SHUT_RD for further receives, SHUT_WR for further sends, or SHUT_RDWR for both), it effectively closes the communication channels while keeping the socket open until explicitly closed using the close function.

Upon completion, shutdown returns 0 to signify success, or -1 in case of failure.

H. Implementation of an Encrypted Messaging System

- Linux

Linux is a multitasking and multi-user operating system, meaning it can run multiple programs simultaneously and can be used by multiple users concurrently. It is also a portable operating system, which means it can be installed on a wide variety of hardware.

- Python

Python is chosen because it serves as the foundation for the SIBC library, which provides implementations of several post-quantum algorithms. It was also easy to implement the Fujisaki-Okamoto transformation on CSIDH within SIBC using this language [15].

- SIBC and CSIDH

CSIDH is an isogeny-based protocol that can be used for key exchange and encapsulation, as well as for other advanced protocols and primitives. **Figure 2** illustrates how CSIDH can be executed similarly to Diffie-Hellman to produce a shared secret between Alice and Bob. It's noteworthy that the elliptic curves $\boldsymbol{E_{BA}}$ and $\boldsymbol{E_{AB}}$ calculated by Alice and Bob at the end of the protocol are identical. CSIDH operates over a finite field $\mathbb{F}_{\boldsymbol{p}}$, where p is a prime number of the form [16-21]:

$$p = 4\prod_{1}^{n} \ell_i - 1 \tag{3}$$

with $\boldsymbol{\ell_1}, \dots, \boldsymbol{\ell_n}$ being a set of small odd prime numbers.

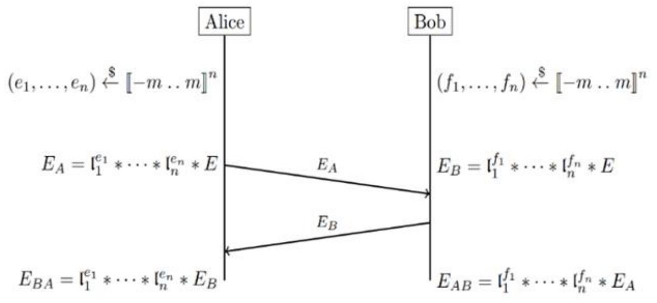

Alice
Bob
$(e_1,\dots,e_n) \xleftarrow{\$} [-m\,..\,m]^n$
$(f_1,\dots,f_n) \xleftarrow{\$} [-m\,..\,m]^n$
$E_A = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E$
E_A
$E_B = \mathfrak{l}_1^{f_1} * \cdots * \mathfrak{l}_n^{f_n} * E$
E_B
$E_{BA} = \mathfrak{l}_1^{e_1} * \cdots * \mathfrak{l}_n^{e_n} * E_B$
$E_{AB} = \mathfrak{l}_1^{f_1} * \cdots * \mathfrak{l}_n^{f_n} * E_A$

Figure 2. CSIDH Key Exchange Protocol

- Fujisaki-Okamoto Transform

The Fujisaki-Okamoto transform is a powerful mathematical tool used in the field of cryptography. This transform, developed by Tatsuaki Okamoto and Shigenori Fujisaki, is primarily employed to secure protocols for transferring sensitive data. It relies on complex mathematical principles to ensure the confidentiality and authenticity of exchanged information [13].

Fujisaki-Okamoto transform provides the security scheme IND-CCA2 provide. By using hybrid encryption scheme $\Pi^{hy} = (K^{hy}, E^{hy}, D^{hy})$, crypto-system will satisfy the desired security properties. This scheme is built from a symmetric decryption scheme and an asymmetric encryption scheme [14].
In this crypto-system hybrid, symmetric and asymmetric cryptography is needed.

let the crypto-system hybrid where :

$\boldsymbol{\Pi^{asym}} = (\mathbf{K}^{\boldsymbol{asym}}, \mathbf{E}^{\boldsymbol{sym}}, \mathbf{D}^{\boldsymbol{sym}}, \mathbf{E}^{\boldsymbol{asym}}, \mathbf{D}^{\boldsymbol{asym}})$.

$\mathbf{K}^{\boldsymbol{asym}}$ is the probabilistic key generator which returns secret key and public key (pk, sk)

$\mathbf{E}^{\boldsymbol{sym}}$ is the symmetric function for encryption

$\mathbf{D}^{\boldsymbol{sym}}$ is the symmetric function for decryption

$\mathbf{E}^{\boldsymbol{asym}}$ is the asymmetric function for encryption

$\mathbf{D}^{\boldsymbol{asym}}$ is the asymmetric function for decryption

$$\forall k \in \mathbb{N} \forall (pk, sk) \text{ de } \mathrm{K}^{asym}(1k)$$

$$\forall x : \mathrm{D}^{asym}(sk, \mathrm{E}^{asym}(pk, \quad x;\ r)) = x \tag{4}$$

$$\Pi^{sym}: \forall k \in \mathbb{N} \forall a$$

$$\forall x : \mathrm{D}^{sym}(a, \mathrm{E}^{sym}(a, x)\ ;\ r) = x. \tag{5}$$

Key Generations: $\boldsymbol{KG(1^k)}$

1. $(pk, sk) := K^{asym}(1^k)$
2. Return (pk, sk)

Encryption: $\boldsymbol{Enc(pk, m)}$

$\boldsymbol{\sigma}$ uniformly samples the asymmetric message space

1. $c := E^{sym}(G(\sigma), m)$
2. $e := E^{asym}(pk, \sigma\ ;\ H(\sigma, c))$
3. Return $e \parallel c$

Decryption: $\boldsymbol{Dec(sk, e \parallel c)}$

```
1. $(e, c) := e \parallel c$
a. if admissible, continue
b. else, return $\varepsilon$
2. $\sigma' := D^{asym}(sk, e)$
a. if admissible, continue
b. else, return $\varepsilon$
3. $a' := G(\sigma')$
4. $h' := H(\sigma', c)$
5. test whether $e == E^{asym}(pk, \sigma'\ ;\ h')$
a. if yes, return $D^{sym}(a', c)$
b. else, on return ε
```

Using Fujisaki-okamoto, attacker could not modify the value of "$\boldsymbol{e \parallel c}$" because he doesn't find the value sk on the formula of e defined by the equation (6).

$$\text{"e} == \mathrm{E}^{asym}(\mathrm{pk}, \mathrm{D}^{asym}(\mathrm{sk}, \mathrm{e});\ \mathrm{H}(\mathrm{D}^{asym}(\mathrm{sk}, \mathrm{e}), \mathrm{c}))\text{"} \tag{6}$$

IV. RESULTS

In this paragraph, functionality, the protocol flow and the interface user for the conception of the application will be presented.

A. Functional diagram

The **Figure 3** represented the functional diagram of our secure messaging application.

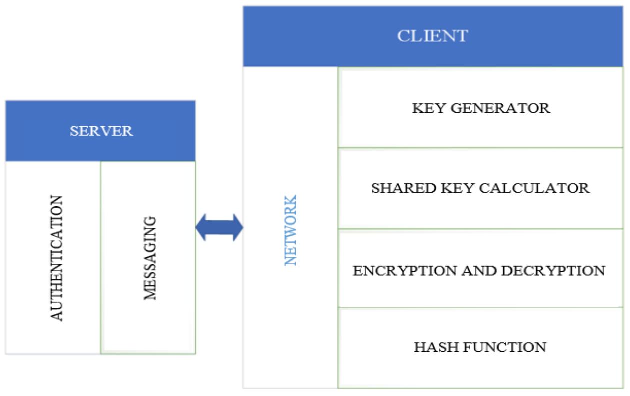


Figure 3. Functional diagram

- **Authentication:** Authentication manages client connections.
- **Messaging:** Messaging handles message transfers between clients.
- **Key Generator:** The key generator generates the private and public keys using the SIBC library with the CSIDH protocol.
- **Shared Key Calculator:** The shared key calculator calculates the shared key from exchanged private and public keys, also using the SIBC library with the CSIDH protocol.
- **Encryption and Decryption:** The encryption and decryption module encrypt or decrypt messages using a modified AES algorithm to support encryption with public and private keys or only with the shared key.
- **Hash Function:** The hash function module implements the Fujisaki-Okamoto (FO) algorithm for encryption or decryption to verify message authenticity.

- **Network:** The network ensures sending requests and messages to the server.

B. Protocol flow of the application

- Client 1 initiates a chat request with client 1, hence client 1 is doing two chats between client 2 and client3.
- Client 1 has issued an end-of-chat request on its chat client 2, which results in these two clients being free for further discussions, and they are receiving the other available clients from the server. Here, client 1 has permanently left this terminal, but client 2 is waiting for a new discussion.
- The server continues to do its job to handle current communications, and also other connection requests from other new users. And the discussion between client 1 and client 3 continues.

The **Figure 4** shows the different stages of operation of the client-server architecture:

- The server works all the time to ensure the connection clients.
- After the clients connect to the server, the server sends the list of available users to each client, and then the client selects another client to chat or waits for another client to start the chat.
- In this **Figure 4**, many functions are represented:
    - Client 1 that uses two terminals for two separate chats.
    - Two other clients, Client 2 and Client 3 respectively.
- Then, client 2 launches a chat request with client 1:
    - First, client 1 and client 2 exchange their public keys.
    - Client 1 and client 2 calculate the private key which must be identified by using CSIDH protocol.
    - Afterwards, they use the private key to encrypt each message.

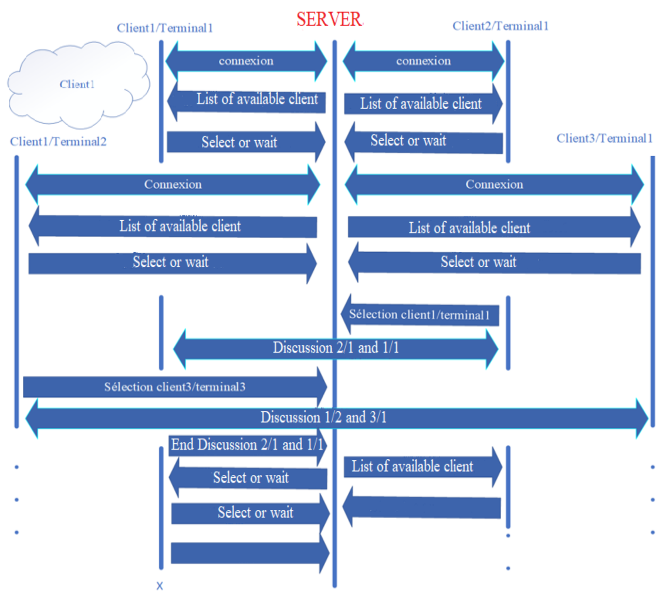
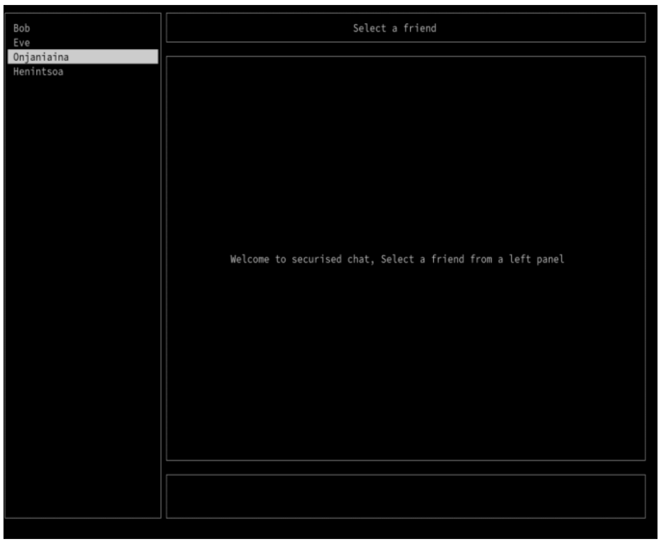


Figure 4. Protocol flow diagram

C. User interface

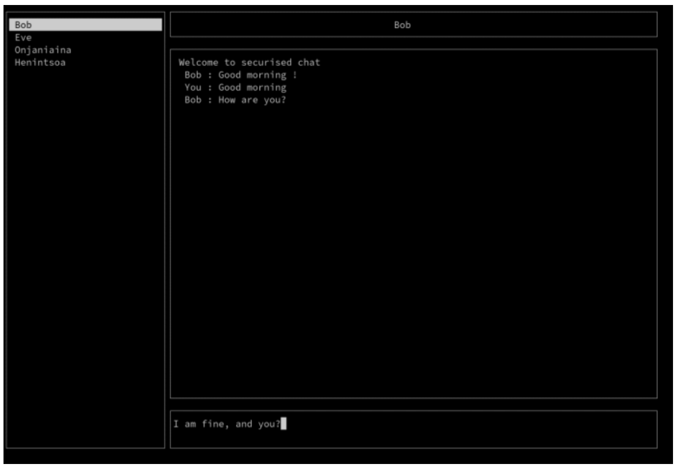


Welcome to securised chat, Select a friend from a left panel

Figure 5. User interface home page

- **Server:** The server receives connection requests from clients and responds to client requests by providing the requested service, which is to provide network connection and manage message exchanges between these users.

- Client: The client program is responsible for displaying active users and handling input and this interface is divided into two categories, one for the client and the other for the server.

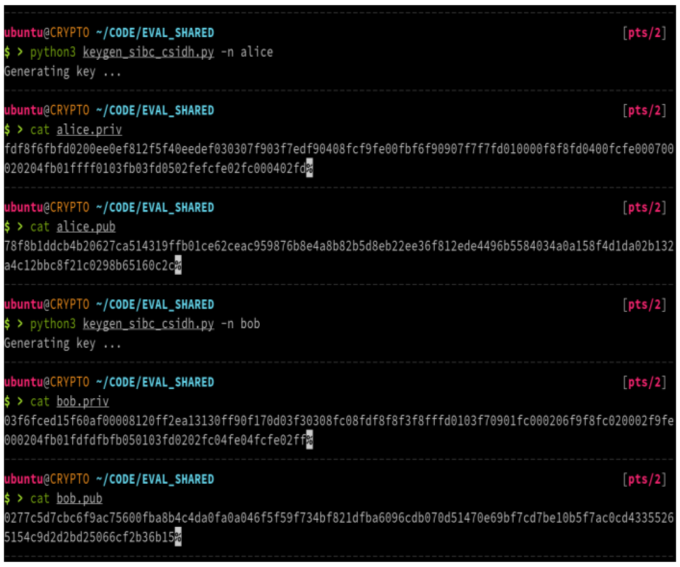


```
Bob            Bob
Eve
Onjaniaina     Welcome to securised chat
Henintsoa       Bob : Good morning !
                You : Good morning
                Bob : How are you?

               I am fine, and you?
```

Figure 6. Message exchange between Alice and Bob

D. Crypto System Kernel

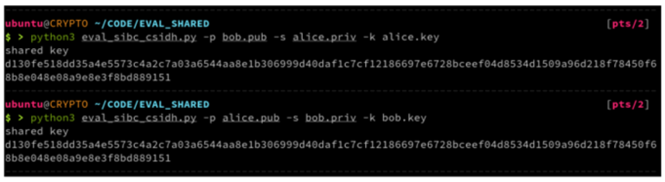
The basis of the key exchange relies on SIBC, a Python library that implements CSIDH, which is used to generate and exchange keys. Here, by using act methods, the private and public keys for Alice and Bob will be generated.



```
ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > python3 keygen_sibc_csidh.py -n alice
Generating key ...

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > cat alice.priv
fdf8f6fbfd0200ee0ef812f5f40eedef030307f903f7edf90408fcf9fe00fbf6f90907f7f7fd010000f8f8fd0400fcfe000700
020204fb01ffff0103fb03fd0502fefcfe02fc000402fd%

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > cat alice.pub
78f8b1ddcb4b20627ca514319ffb01ce62ceac959876b8e4a8b82b5d8eb22ee36f812ede4496b5584034a0a158f4d1da02b132
a4c12bbc8f21c0298b65160c2c%

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > python3 keygen_sibc_csidh.py -n bob
Generating key ...

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > cat bob.priv
03f6fced15f60af00008120ff2ea13130ff90f170d03f30308fc08fdf8f8f3f8fffd0103f70901fc000206f9f8fc020002f9fe
000204fb01fdfdfbfb050103fd0202fc04fe04fcfe02ff%

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > cat bob.pub
0277c5d7cbc6f9ac75600fba8b4c4da0fa0a046f5f59f734bf821dfba6096cdb070d51470e69bf7cd7be10b5f7ac0cd4335526
5154c9d2d2bd25066cf2b36b15%
```

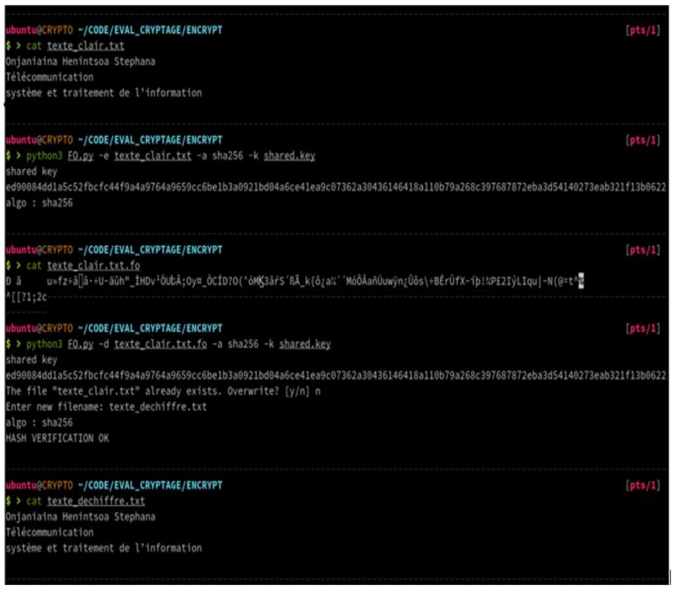
Figure 7. Generation of Private and Public Keys for Alice and Bob

Assuming that Alice and Bob are on two different machines, after generating their respective keys, they exchange their public keys, for example, through the internet. Then, they compute their shared keys, which should be identical.



```
ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > python3 eval_sibc_csidh.py -p bob.pub -s alice.priv -k alice.key
shared key
d130fe518dd35a4e5573c4a2c7a03a6544aa8e1b306999d40daf1c7cf12186697e6728bceef04d8534d1509a96d218f78450f6
8b8e048e08a9e8e3f8bd889151

ubuntu@CRYPTO ~/CODE/EVAL_SHARED                                                    [pts/2]
$ > python3 eval_sibc_csidh.py -p alice.pub -s bob.priv -k bob.key
shared key
d130fe518dd35a4e5573c4a2c7a03a6544aa8e1b306999d40daf1c7cf12186697e6728bceef04d8534d1509a96d218f78450f6
8b8e048e08a9e8e3f8bd889151
```

Figure 8. Calculation of shared key



```
ubuntu@CRYPTO ~/CODE/EVAL_CRYPTAGE/ENCRYPT                                          [pts/1]
$ > cat texte_clair.txt
Onjaniaina Henintsoa Stephana
Télécommunication
système et traitement de l'information

ubuntu@CRYPTO ~/CODE/EVAL_CRYPTAGE/ENCRYPT                                          [pts/1]
$ > python3 FO.py -e texte_clair.txt -a sha256 -k shared.key
shared key
ed90084dd1a5c52fbcfc44f9a4a9764a9659cc6be1b3a0921bd04a6ce41ea9c07362a30436146418a110b79a268c397687872eba3d54140273eab321f13b0622
algo : sha256

ubuntu@CRYPTO ~/CODE/EVAL_CRYPTAGE/ENCRYPT                                          [pts/1]
$ > cat texte_clair.txt.fo
Ð ã     u»fz÷ã[]â·÷U-ãûh"_ÌHDv¹ÖUŁÅ;Oy¤_ÖCÍD?O('óMĶ3ãŕS´ßÂ_k{ô¿a¼´´MóÕÅañÚuwÿn¿Ûôs\÷BÊrÛfX-íþ!¼P£2IýLIqu|-N(@=t^%
^[[?1;2c

ubuntu@CRYPTO ~/CODE/EVAL_CRYPTAGE/ENCRYPT                                          [pts/1]
$ > python3 FO.py -d texte_clair.txt.fo -a sha256 -k shared.key
shared key
ed90084dd1a5c52fbcfc44f9a4a9764a9659cc6be1b3a0921bd04a6ce41ea9c07362a30436146418a110b79a268c397687872eba3d54140273eab321f13b0622
The file "texte_clair.txt" already exists. Overwrite? [y/n] n
Enter new filename: texte_dechiffre.txt
algo : sha256
HASH VERIFICATION OK

ubuntu@CRYPTO ~/CODE/EVAL_CRYPTAGE/ENCRYPT                                          [pts/1]
$ > cat texte_dechiffre.txt
Onjaniaina Henintsoa Stephana
Télécommunication
système et traitement de l'information
```

Figure 9. Encryption and Decryption of a message

And then, proceeding with an example of encryption and decryption according to the following **Figure 9**. In this figure, the message text to be ciphered is represented as "texte_clair.txt" or "plain_text".

After obtaining shared_key by using CSIDH protocol, the function FO permits to cipher the plain text message with authentication, and the encrypted text will be named "texte_clair.txt.fo".

Finally, by deciphering this encrypted text, the origin message will be obtained.

V. RESULTS AND DISCUSSION

The key exchange procedure begins with the generation of the secret key. The secret key generation process is fast, typically taking only a few milliseconds. The key exchange procedure begins with the generation of the secret key. The secret key generation process is swift, typically taking only a few milliseconds, as illustrated in the **Figure 10**. The lowest time could be explained by computation of random number which is very quick using computer.

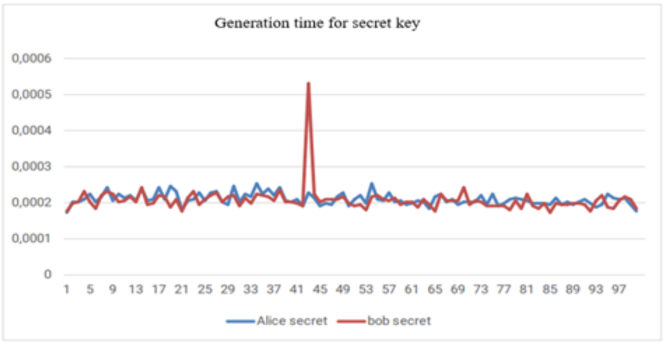


Figure 10. Time taken for secret key generation

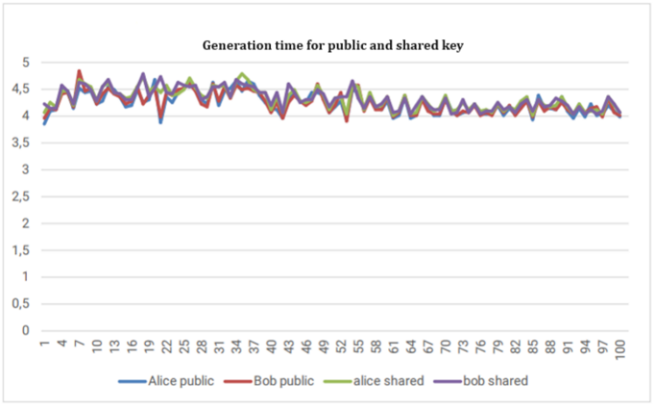


Figure 11. Time taken for public and private key generation

The speed test was conducted over a hundred iterations of key exchanges. Alice and Bob first compute their public keys by using secret key, then they exchange their keys and calculate the shared key. Computing public key and shared key to compute isogeny for the ideal of the class of number by using acting action (symbolized by $*$ on **Figure 2**). The time execution for computing publicand shared key is illustrated on **Figure 11**.

For deep analysis, TABLE III shows the statistic of the time of key exchange between two users before making the chat.

TABLE III
DEEP STATISTICAL ANALYSIS OF TIME TAKEN FOR PUBLIC AND SHARED KEY GENERATION

	Alice public key	Bob public key	Alice shared	Bob shared
Average time	4,23037177s	4,23917467s	4,30588803s	4,31053638s
Minimal time	3,84220457s	3,89435864s	3,98901963s	4,02590895s
Maximal time	4,65548468s	4,82580686s	4,78066635s	4,76524615s

According to this statistic, Alice and Bob generate these keys for a duration of approximately 4 seconds.

They simultaneously generate their public keys for about 4 seconds, then do the key exchange, and then calculate the shared key for about 4 seconds, which on average results in an 8 second key processing period.

Next, the encryption and decryption performance of the system will be tested. For this the graph will be divided into two parts to better visualize the points where the system starts to slow down: small files and big files.

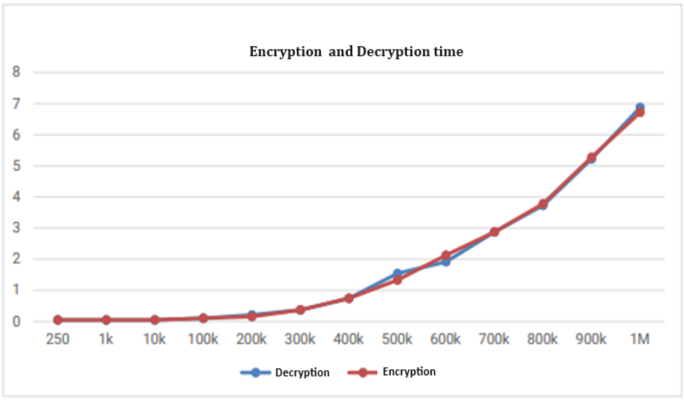


Figure 12. Time taken for encrypting and decrypting small message

The encryption or decryption speed is relatively slow for files smaller than 300 kilobytes, ranging from a few milliseconds to 0.34 seconds. For message data sizes, this is more than sufficient for the system, as for instance, 300 kilobytes of text files can contain up to 1500 paragraphs of text.

The curve increases exponentially, and this increase will degrade the performance of the system if the system will be used the other way to transfer texts ranging from 300kilobytes and more, because in client-server applications which process a lot of data, every millisecond is precious.

For 4-megabyte files, the encryption or decryption process takes 100 seconds, which is no longer tolerable.

To encrypt or decrypt a 10-megabyte file, it would take approximately 9 minutes to do so.

Based on the test, it is better to split the 10-megabyte file into 100 times of 100 kilobyte files, but the encryption of 100 kilobytes only takes 0.056 seconds. Which gives a time of 100files×0.056s/file = 5.6 seconds. 100 kilobytes are the optimal cutting edge, because based on the test, with 200 kilobytes it would have lasted 7.5 seconds, and with 10 kilobytes it would have lasted 15 seconds.

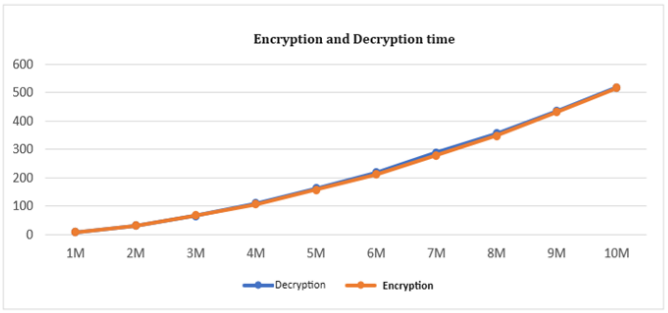


Figure 13. Time taken for encrypting and decrypting big message

VI. CONCLUSION

Post-quantum cryptography is an important response to the challenges posed by the potential advent of quantum computing. It provides solutions to safeguard data confidentiality and integrity in a world where quantum computers could threaten existing security systems. However, it is important to note that post-quantum cryptography is still under development, and it will take time for it to be widely adopted. In the meantime, it is essential to monitor advancements in this field and implement appropriate security measures to protect sensitive data.

VII. REFERENCES

[1] Jean-Christophe Deneuville, "Contributions à la Cryptographie Post-Quantique" , Université de Limoges, 2016.

[2] Whitfield Diffie, Martin E. Hellman, "New directions in cryptography". Information Theory, IEEE Transactions on, 22(6), pp. 644–654, 1976.

[3] Ronald L Rivest, Adi Shamir, Len Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2), pp. 120–126, 1978.

[4] Peter W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM J. Comput., 26(5), pp.1484–1509, 1997.

[5] Lov K. Grover, "A fast quantum mechanical algorithm for database search", In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, pp. 212–219, 1996.

[6] Cheikh Tidiane Mbaye, "Cryptographie post-quantique basé sur les codes correcteurs et isogénies", Aix-Marseille Université, 2018.

[7] Kevin Carrier, "Recherche de presque-collisions pour le décodage et la reconnaissance de codes correcteurs", HAL open science, Sorbonne Université, pp. 52-53 2020.

[8] Vanessa Viste, “Couplages sur courbes elliptiques définies sur des corps finis”, Stage de Master 2, Université Versailles-Saint-Quentin, 2008

[9] Moncef Amara and Amar Siad, “Elliptic Curve Cryptography and its applications”, in IEEE International Workshop on Systems, Signal Processing and their Applications, WOSSPA, 27 june 2011, Tipaza, Algeria, https://doi.org/10.1109/WOSSPA.2011.5931464

[10] Gorantla Naga Manoj,Chowdary, Medapati Phani Sri Rama Lakshmi, Yarababugari Nylu, Botta Deepthi, KV Prasad and Sathish Kumar Kannaiah, “Elliptic Curve Cryptography for Network Security”, in IEEE International Conference on Inventive Computation Technologies (ICICT), 01 June 2023, Lalitpur, Nepal, https://doi.org/10.1109/ICICT57646.2023.10134492

[11] Carlos Andres Lara-Nino, Arturo Diaz-Perez and Miguel Morales-Sandoval, Elliptic Curve Lightweight Cryptography: a Survey, in IEEE Data-report, 17 Mey 20222, https://dx.doi.org/10.21227/bqfj-6c39

[12] Jan L. Camenish, Christian S. Collberg, Neil F. Johnson, Phil Sallee, “Information Hiding”, 8th International Workshop, IH 2006, Alexandria, VA, USA, July 2016, edition Springer

[13] Bruno Saint Pee, « Le Modèle TCP/IP », Lycée Rotrou Dreux.

[14] « Socket », https://projet.eu.org/pedago/sin/term/5-socket.odt , 2023.

[15] Francisco Rodríguez-Henríquez, « SIBC: A Python-3 library for designing and implementing efficient isogeny-based protocols », Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, Computer Science Department, CINVESTAV-IPN, Mexico City, 2021.

[16] W. Castryck, T. Lange, C. Martindale, L. Panny, J. Renes, « CSIDH: an efficient post-quantum commutative group action », Springer, Advances in Cryptology - ASIACRYPT 2018, pp. 395–427, 2018.

[17] D. Cervantes-Vázquez, M. Chenu, J. Chi-Domínguez, L. D. Feo, F. Rodríguez-Henríquez, B. Smith, « Stronger and faster side-channel protections for CSIDH », Springer, Progress in Cryptology - LATINCRYPT 2019, pp. 173–193, 2019

[18] Everett W. Howe, Kristin E. Lauter, Judy L. Walker Editors, Algebraic Geometry for Codong Theory and Cryptography, IPAM, Los Angeles, CA, February 2016, edition Springer

[19] Cyprien Delpech de Saint Guilhem, and Robi Pedersen, “New proof systems and an OPRF from CSIDH”,
COSIC, KU Leuven, Belgium, https://ia.cr/2023/1614

[20] Tomoki Moriya, Hiroshi Onuki and Tsuyoshi Takagi, How to construct CSIDH on Edwards curves, in Finite Fields and Their Applications, volume 92, December 2023, 102310, https://doi.org/10.1016/j.ffa.2023.102310

[21] Mingping Qi, An efficient post-quantum KEM from CSIDH, Mai 2022, in Journal of Mathematical Cryptology, https://doi.org/10.1515/jmc-2022-0007