COLLEGE OF COMPUTING AND INFORMATICS

UNIVERSITI TENAGA NASIONAL

IMPLEMENTATION OF HYBRID POST – QUANTUM ENCRYPTION ON COMMUNICATION PLATFORM

FIKRI A

CS01081173

2024

IMPLEMENTATION OF HYBRID POST-QUANTUM ENCRYPTION ON COMMUNICATION PLATFORM

by

AMIRUL FIKRI BIN AZMAN ZAFRI

Project Supervisor: Assoc. Prof. Ts. Dr Asmidar binti Abu Bakar

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE BACHELOR OF COMPUTER SCIENCE (HONS.) (CYBER SECURITY),

COLLEGE OF COMPUTING AND INFORMATICS, UNIVERSITI TENAGA NASIONAL

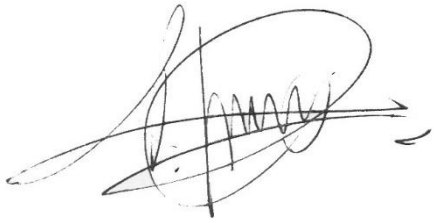
2024

DECLARATION

I hereby declare that this report, submitted to Universiti Tenaga Nasional as a partial fulfilment of the requirements for the Bachelor of Computer Science (Hons.) has not been submitted as an exercise for a degree at any other university. I also certify that the work described here is entirely my own except for excerpts and summaries whose sources are appropriately cited in the references.

This report may be made available within the university library and may be photocopied or loaned to other libraries for the purposes of consultation.

10 July 2024



AMIRUL FIKRI

CS01081173

Digitally signed by Amirul Fikri bin Azman Zafri, .
Date: 2024.07.10 19:10:50 +08'00'

APPROVAL SHEET

This thesis entitled:
"Implementation of Hybrid Post-Quantum Encryption on communication platform"

submitted by:

AMIRUL FIKRI (CS01081173)

In requirement for the degree of Bachelor of Computer Science (Cyber Security) (Hons.), College of Computing and Informatics, Universiti Tenaga Nasional has been accepted.

Supervisor: Assoc. Prof. Ts. Dr Asmidar binti Abu Bakar

Signature: Assoc. Prof. Ts. Dr. Asmidar Binti Abu Bakar
Deputy Dean
Academic & Research
College of Computing and Informatics
Universiti Tenaga Nasional
17/11/23

Date: 18 November 2023

DEDICATION

First, Amirul Fikri would like to thank Allah S.W.T for blessing His servant with an opportunity to live and perform this responsibility. Second, Amirul Fikri would like to thank his family and fellow friends – Hiu Qi Han "*Andrew*," Vyner Nicholas, Lee Xian Yang "*Amous*" and Ammar Haziq "*Marjiqs*" and everyone else that I have met during my time of study, for giving him constant support and love throughout his life. Third, Amirul Fikri would like to thank "Cicada 3301" for giving him the enlightenment from his works "*Liber Primus*" and fuelling his burning passion to cryptography.

Finally, Amirul Fikri would like to thank <3 and dedicate this entire work to her, for giving him a chance to become a better man. *Think of me once in a while. Be the best of yourself. Thank you. I love you. – A*

ACKNOWLEDGEMENTS

Amirul Fikri would like to thank Dr Asmidar as my supervisor for this project on providing guidance and recommendations throughout the entire time of this project, Dr Norziana as my cryptography lecturer and as initiator of the idea of this project during our discussion in class ("*Quantum computers on current encryption*"), Sir Abbas as my examiner for FYP1, and Universiti Tenaga Nasional as a whole, for helping and providing guidance throughout this project.

.

ABSTRACT

The use of current cryptography scheme to protect the confidentiality of message exchange is at risk of theoretical threats from Quantum Computers. The security level of cryptography schemes can be reduced or broken completely with the use of Quantum Computers. In order to protect message confidentiality, new cryptography scheme must be developed. This thesis examines in detail what is the definition and different types of cryptography, and what are the possible attacks that can be performed against current cryptography using conventional and Quantum Computer. This thesis also examines in detail on current mitigation of attacks from Quantum Computer, in which is to integrate one of Post-Quantum Cryptography scheme into existing cryptographic implementations; named Hybrid Post-Quantum Encryption. This thesis also describes in detail on design and implementation of Hybrid Post-Quantum Encryption in a proof-of-concept communication platform. The result of the system proposed in this thesis are successful practical implementation of the Hybrid Post-Quantum Encryption scheme, with feasible performance and network overhead as per project's hardware requirements. Also, the system proposed is truly end-to-end encrypted, and only intended sender and recipient can only exchange messages; and theoretically achieving Post-Quantum Security.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS / TERMINOLOGIES

3-DES	Triple-DES (refer DES abbr. below)
3DH	Triple Diffie-Hellman Key Exchange Protocol
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AK	Authenticated Key Agreement
AKC	Authenticated Key Agreement with Key Confirmation
Ct.	Ciphertext
cols.	columns
CBC	Cipher Block Chaining
CIA	Confidentiality, Integrity, and Authenticity Information Security Triad
CRYSTALS	Cryptographic Suites for Algebraic Lattices
CTR	Counter (mode of operation)
dec.	Decrypt / Decryption process
DES	Data Encryption Standard
DH	Diffie-Hellman Key Exchange Protocol
DL / DLP	Discreet Logarithm (Problem)
DSA / DSS	Digital Signature Algorithm / Scheme
enc.	Encrypt / Encryption process
e.g.,	(*exempli gratia*): for example
et al.	(*et alia*): and others
etc.	(*et cetera*): and the rest
EAX	Encrypt-Authenticate-Translate
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptosystem

ECDL / ECDLP	Elliptic Curve Discreet Logarithm (Problem)
ECDSA	Elliptic Curve Digital Signature Algorithm
EdDSA	Edwards Curve Digital Signature Algorithm
ERD	Entity Relationship Diagram
FIPS	Federal Information Processing Standard
HFE	Hidden Field Equations
IDE	Integrated Development Environment
IND-CCA1/2	Indistinguishability under Chosen Ciphertext Attack / Adaptive Chosen Ciphertext Attack
IND-CPA	Indistinguishability under Chosen Plaintext Attack
init. / IV	Initialization / Initialization Vector
i.e.,	(*id est*): that is
GCHQ	Government Communications Headquarters
JSON	JavaScript Object Notation Format
KDF	Key Derivation Function
KEP	Key Exchange Protocol
KEM	Key Encapsulation Mechanism
LAN	Local Area Network
LWE	Learning With Error
MD	Message Digest Algorithm
ME / MQE	Multivariate-Equation / Multivariate-Quadratic-Equation Cryptosystem
MITM	Man-in-the-middle attack
ML	Module-Lattice-based
ML-LWE	Module-Lattice Learning with Error

NIST	National Institute of Standards and Technology
NTRU	Number Theory Research Unit
NTT	Number-Theoretic Transform
p. / pp.	page number
Pt.	Plaintext
PRNG	Pseudo Random Number Generator
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
PQ	Post-Quantum
PQC / PQE	Post-Quantum Cryptography / Encryption
PQXDH	Post-Quantum Extended Diffie-Hellman Key Exchange Protocol
rnd. / rnds.	Round(s)
RL-LWE	Ring-Lattice Learning with Error
RSA	Rivest-Shamir-Adleman Cryptosystem
RFC	Request For Comment
s.t.	such that
SIS	Short Integer Solution
SHA	Secure Hash Algorithm
SNDL	Store Now, Decrypt Later
SDLC	Software Development Life Cycle
UI	User Interface
UX	User Experience
XEdDSA	Extended Edwards curve Digital Signature Algorithm
X3DH	Extended Triple Diffie Hellman Key Exchange Protocol

---

gcd	greatest common divisor
mod	Modulus Division
GF / $\mathbb{F}$	Galois / Finite number Field
AND / $\wedge$	Bitwise-AND, Conjunction

OR / $\vee$	Bitwise-OR, Disjunction
XOR / $\oplus$	Bitwise-Exclusive OR, Exclusive disjunction
NOT / $\neg$	Bitwise-NOT, Negation
ROTL / <<<	Rotate left no carry
ROTR / >>>	Rotate right no carry
$>$	Larger than
$\geq$	Larger OR Equal than
$<$	Smaller than
$\leq$	Smaller OR Equal than
$\in$	is / are an element of
$\mathcal{O}(n)$	Time-complexity Notation
$\|\|$	Concatenation of element(s)

1 2

[1] All equations written in this thesis should be exempted from the formatting of the document and written in Professional form, Cambria Math font in 12 points, centered with single spacing.
[2] List of Abbreviations and Terminologies maybe incomplete.

CHAPTER 1

INTRODUCTION

1.1 Background

Communication over the Internet have evolved over time. From the first message that was exchanged over ARPANET between two computers using telephone cables, now advanced interconnectivity between continents has been achieved. Nowadays, a person can send a message to anyone in the world, provided that the other person is also on the Internet. Furthermore, instant messaging applications have been developed to make it easier to communicate.

However, with the rise of technology and Internet advancement, so do the threats against it. Threat actors have the capability to read or alter the messages that are in-transit, over the insecure Internet. They may also be able to masquerade as another person in the message exchange.

The system proposed in this project heavily utilizing cryptography to protect the messages exchanged. The system proposed also utilizing cryptography to preserve the authenticity and integrity of messages exchanged and the identity of entities in the message exchange.

In order to secure the confidentiality of message exchange, multiple methods of cryptography are employed, from symmetric and asymmetric cryptography. However, to increase the level of security, it all depends on key length used during encryption. With a longer key length, the ciphertext is harder to crack; to translate into plaintext.

With a longer key length, more computational resources and time are needed in order to complete the operation. This also comes with a higher network overhead since the size of the ciphertext will also increases with the increase of key length. For example, increasing the key length of an asymmetric cipher i.e. RSA.

The focus of this project is to find a balance in performance of cryptography operation while providing the utmost level of security. New cryptographic scheme that is implemented need to be future-proof against both conventional and Quantum computers, while balancing the safeguarding of information security versus requirements of performance and network overhead.

1.2 Problem Statement

Communication between two entities must be secured in order to safeguard the information security. To prevent attacks such as message interception, message manipulation and identity theft, the confidentiality, integrity and authenticity of messages exchanged must be protected. This requires the use of robust and proven cryptographic methods such as symmetric and asymmetric encryption, key exchange protocols and digital signature scheme.

With the advancement of Quantum Computing, there has been a new theoretical threat rising sometime in the future where they could potentially break current conventional encryption scheme; both asymmetric and symmetric with Shor's and Grover's Algorithm [1], [2], once Quantum Computers become more available to the public.

Current cryptographic methods used now have notable limitations and vulnerabilities, in particular from the theoretical threat from Quantum computers, and also other various cryptographic attacks [3], [4]. This project aims to systematically explore these challenges and propose a solution to mitigate the risk arising from the vulnerabilities of current cryptographic methods.

1.3 Objectives

- To explore secure communication methods between two parties using Key Exchange Protocols, Symmetric Encryption, and Digital Signature Schemes.

- To design and develop a proof-of-concept communication platform integrating various cryptographic algorithms.

- To conduct testing to evaluate the functionality and security of the developed system.

1.4 Scope

1.4.1 User Scope

Table 1. 1 User Scope list

User Type	Description
Communicators	The person (“Users or entities”) who exchanges messages to another person in the platform.

Table 1.1 shows the targeted user(s) for the application. Communicators, in the context of the project, are using the application to send and receive messages from and to other people who are also using the same application in the same chat session.

1.4.2 System Scope

Table 1. 2 System Scope list

User Type	Description
Database Administrator	A person of authority, who manages and facilitates the system’s database.

Table 1.3 shows the targeted user(s) for the application, on the system / administration side. Database administrator, in the context of this project, is a person who manages the database’s data and infrastructure. They need to configure the database to be able to receive data from application and store it securely for safekeeping. The data that would be stored in the database are public keys and its fingerprints, signatures and others that are not specified here.

1.5 Expected Outcome

- Two different entities or parties can communicate securely, without the concern of threat posed by conventional and Quantum computers.

- No one will be able to read the messages exchanged between the two parties, only the intended sender and recipient.

CHAPTER 2

LITERATURE REVIEW

This chapter covers the fundamentals and theoretics of cryptography, attacks on cryptography and the threats by conventional and Quantum computers on cryptography.

2.1 Introduction to cryptography

According to Katz and Lindell [5], Cryptography, in modern age, can be defined as the scientific study, research and extensive review of techniques and methodologies implementing complex mathematical theorems for safeguarding data and transactions and enforcing information and cyber security. The word "Cryptography" comes from combination of Ancient Greek words κρυπτός – *kryptós*: "Hidden, secret;" and γράφειν – *graphein*: "To write" or -λογία – *-logia*: "Study" [6], respectively. Phil Zimmerman [7] quotes that "Cryptography is the science of using mathematics to encrypt and decrypt data". Bruce Schneier [8] quotes that "Cryptography is the art and science of keeping message secure". Cryptography is vital because it's being used in securing all communications for governments, militaries, corporates, organisations, and the public

Sensitive information that needs to be exchanged between to government bodies, Operational Intelligence from command post to soldiers in the front line of a battlefield, details of agreement between corporates and even text messages in instant messaging app, is being protected by methodologies implemented by cryptography.

The main objective of cryptography is to protect the Information Security or CIA Triad. Detail of each building block of the triad is emphasized below.

- **Confidentiality**

Confidentiality in the triad means only authorized users or entities should be able to access or modify data, even if the data is transferred between users or entities in an insecure network i.e., over the Internet. In the context of this project, only intended, authorized, and authenticated users or entities shall be able to access or modify data ("Messages") that are exchanged between each other. Furthermore, no one else than the intended sender and receiver; including the service provider and the system itself, should be able to access or modify said data [8], [9].

- **Integrity**

Integrity in the triad means the authorized users or entities have a way to proof that the data is not modified, tampered, or changed partially or as a whole, while the data is in-transit in an exchange. In the context of this project, the intended sender must be able to give proof to the intended recipient of the data that it was not altered during its transit over the insecure Internet. If the proof can be disputed, that means the message has been altered during its transit, and the intended recipient of the data can question the genuineness of the data itself [8], [9].

- **Authenticity**

Authenticity in the triad means the authorized users or entities have a way to proof that they are really the legitimate, intended sender and recipient of the data. In the context of this project, the intended sender must be able to give proof to the intended recipient that he or she can be identified as the legitimate, intended sender of the message, to the intended recipient. If the proof can be disputed, that means the message did not come from the identity of the intended sender, and the intended recipient can question the genuineness of the message originator. This process can also be vice versa, to prove the identity of the message receiver [8], [9].

- **Non-Repudiation**

Non-repudiation in the triad means the authorized users or entities CANNOT have the way to deny the proof that they are really the legitimate, intended sender of the data. Non-repudiation works in conjunction to Authenticity mentioned earlier. In the context of this project, the intended sender CANNOT state the fact that he or she did not send the message to the intended recipient. This will contradict the proof in Authenticity. The intended sender can be held accountable to his / her actions [8], [9].

The Availability triad has been omitted here due to its irrelevancy in cryptography. Availability means the data must be available to access by authorized users or entities at all times required, and it is not within the operating scope of cryptography, but more towards the network infrastructure around the data itself.

Without cryptography in place, there are no secure means of communication over the Internet. Any threat actor can intercept messages that are sent throughout the insecure Internet and perform either passive or active attack. Passive attack means that the threat actor only read the message contents and observe the communication traffic. Active attack means that the threat actor can alter the message content therefore tampering its integrity. The threat actor, in active attack also able to masquerade as the intended sender therefore compromising authenticity of the message [10].

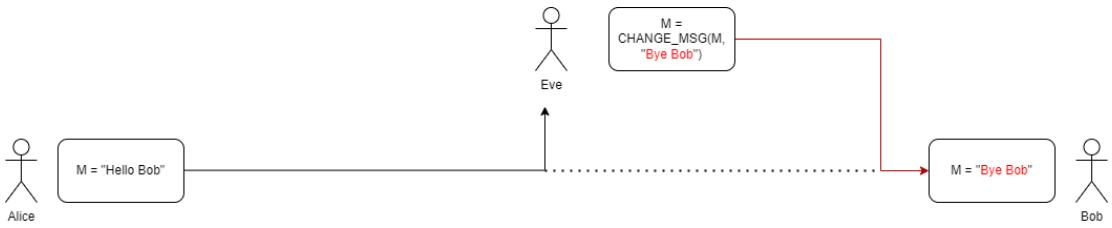


Figure 2. 1 Visualisation of active attack on message exchange

In **Figure 2.1**, Alice tries to send a message with content “Hello Bob” to Bob. But unbeknownst to both party, there is an entity in the middle named “Eve,” where he

intercepted the message from Alice, change the message's content to "Bye Bob" and re-sent the altered message to Bob, seeming as it originates from Alice.

This attack scenario is an example of a critical CIA triad compromise. This is because Eve was able to read Alice's message to Bob, and there are no mechanisms that are applied to prevent other party than Bob; in this case Eve, to read the message content, defeating the Confidentiality rule. Second, Eve was able to tamper with the message content, and there are no mechanisms that are applied to provide proof to Bob that the message is not tampered in-transit; therefore, defeating the Integrity rule. Finally, Eve was able to re-send the tampered message and making Alice as the tampered message sender, and there are no mechanisms that are applied to provide proof to Bob that the message really came from Alice, the genuine intended sender, or Eve, the threat actor in-the-middle; therefore, defeating the Authenticity rule.

Cryptography, in the modern age, should be better suited to be named as cryptology. This is because with new cryptographic algorithms developed over time to be set as world-wide standard, there are a need for proving these algorithms are not vulnerable to exploits, backdoors and, attacks. This task falls under cryptanalysis. Cryptanalysis can be defined as a scientific study and research on breaking techniques and methodologies implemented by cryptography and gaining access to message contents that are protected by said cryptography, while not having any knowledge of cryptographic keys that are used to encrypt and decrypt the messages [7]. The word "Cryptanalysis" comes from combination of Ancient Greek words κρυπτός – *kryptós*: "Hidden, secret;" and αναλύει – *analýei*: "Analyse" [6]. To summarise on this point, cryptography now has become cryptology, where areas of cryptography and cryptanalysis are merged to become as one and complement each other.

Cryptographic algorithm, or cipher, is a method or function that turns intelligible messages that human can make sense of, in this context it is called Plaintext (Pt), into unintelligible Ciphertext that human cannot make sense of (Ct); and vice versa. This function can only work with a correct key that is set and agreed upon by two entities in

communication. Note that with the same cipher and plaintext, changing the key would also change the output of ciphertext [7].

For ciphers to be selected for standardization, cryptographic algorithms need to be proven to be secure enough. There are two cryptographic security schemes that can be based upon in this thesis.

- **Unconditionally secure**

Also called information-theoretic secure, a cipher that conforms to this scheme, should be able to withstand attacks from any adversary with hypothetically speaking, with unlimited time and computer resources in-hand [11]. This is because the required information / data is simply not there in the ciphertext. An example of the cipher that conforms to this scheme would be One-Time Pad or Vernam cipher. Assuming the key length used in the cipher are equal or longer than the plaintext's length, the adversary does not have the intelligence on the key, and they must try possible key combinations, cryptanalysis on any given ciphertext is impossible.

- **Conditionally secure**

Also called computationally secure, a cipher that conforms to this scheme where to break the cipher, the adversary will need a finite time and computer resources, albeit uncomically huge amounts of it, to break the cipher. In other words, it should not be computationally feasible for an adversary to break a cipher, given enough time. [11]. This is because the information encrypted with the cipher will not be useful after it's been "cracked." An example of this would be, consider a hypothetical scenario of a military secret that should be guarded secret for 30 years, and the cipher needs 50 years from now to be broken. After 30 years has passed, the military secret is no longer needed to be bound to the cipher's protection. An adversary finally cracked the secret after 50 years. But to their disappointment, the military secret is irrelevant to that time; therefore, rendering it useless.

Instead of making cipher operations and details hidden or in other words, security through obscurity, a cipher's design and detail must be made available for everyone to access. In essence to the statement, cryptography must follow the Kerckhoff's Principle. These are the first and second, out of the six rules to this principle [12]: –

> *The system* [a cryptographic algorithm in this case] *must be substantially, if not mathematically, undecipherable;*
>
> *The system* [a cryptographic algorithm in this case] *must not require secrecy and can be stolen by the enemy without causing trouble;*

The first rule in the principle can be interlinked with the cryptographic security scheme, and the cipher used should not be able to be broken in the hands of an adversary. The second rule in the principle is important. This rule means that every piece of information regarding the cipher except the key used for the cipher process, should be able to be accessed by anyone, including the threat actor. It is important because this rule contrasts with security through obscurity paradigm, where security through obscurity can give a false sense of security; knowing that no-one else than the cipher users know the inner workings of the cipher. Following this rule, cipher's information should be available to everyone for it to be able to be independently verified by another entity; a standardization body in this case the National Institute of Standards and Technology ("NIST"). Furthermore, unknown weakness and vulnerabilities of the cipher can be discovered if everyone can analyse it.

2.1.1 Symmetric cryptography

Below is a high-level view of how symmetric cryptography works.

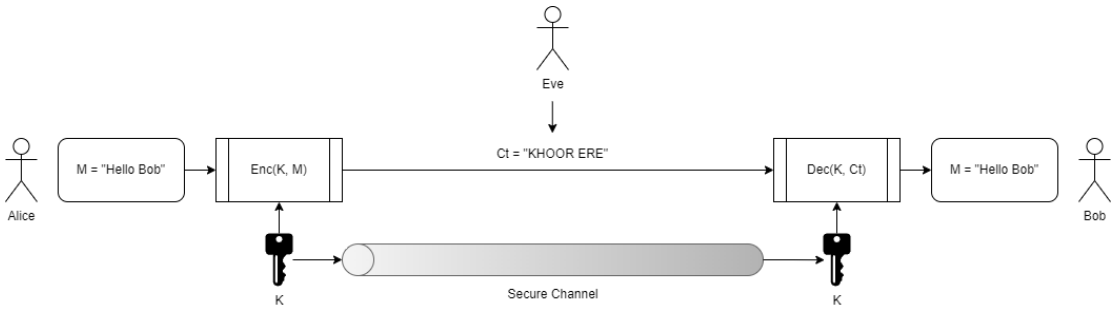


Figure 2. 2 Visualisation of Symmetric cryptography

In **Figure 2.2**, Alice is defined as "intended sender" of a message with content "Hello Bob." Bob would be the "intended receiver" of Alice's message. Eve would be the adversary, trying to perform passive attack on their communication. Recall back from the introduction that passive attack is attempt on reading the content of messages sent between two parties.

Before they can start to use the cipher, both Alice and Bob must agree on a shared secret key, named k. This must be shared on a secure channel because recall on Kerckhoff's principle, everything EXCEPT the key can be made public to everyone.

Then, Alice will use the key to perform encryption on the message. This can be written in an equation: $Ct = Enc(k, M)$. Ct is ciphertext, $Enc()$ is a symmetric encryption algorithm and M is the message that Alice want to send to Bob. Ct is then sent to Bob. Bob will receive the Ct and he can reverse the decryption on the ciphertext by using the same key that Alice used to encrypt the initial message. This can be written in an equation: $M = Dec(k, Ct)$. This time, Dec () would be the symmetric decryption algorithm. It is reversing the process of transforming the message to ciphertext [5], [7], [8].

By expanding upon both equations, the message, M can be also defined as [5], [8]:

$$M = Dec\big(k, Enc(k, M)\big)$$

From **Figure 2.2,** only Alice and Bob can see M, which is "Hello Bob." The adversary, which is Eve, can only see the Ct that are exchanged between both, which is "KHOOR ERE." The ciphertext is obviously unintelligible for human to make sense of, therefore prevent Eve from reading the actual message that is exchanged. Even if Eve knows what cipher algorithm are used in the encryption and decryption process, unless Eve performs brute-force attack on the ciphertext; Eve cannot decrypt the ciphertext because Eve do not hold the key that are used for the encryption and decryption process.

Now, time to get deeper. All ciphers can be categorized into **two types**. First type is a stream cipher, where plaintexts are encrypted bits-by-bits, or by characters in sequential order [13]. Next subtopic will explain how stream cipher works. The second type is a block cipher, where plaintexts will be grouped into a fixed-length block, then encryption process will be done per blocks [13].

Next, All ciphers are built upon a foundation of two building blocks which are: –

- **Substitution**

Substitution works by mapping plaintext elements, characters into ciphertext elements [7], [13]. In other words, a character in plaintext should be mapped to a different character in ciphertext. An example of this building block would be a Caesar cipher. In this thesis, only mono-alphabetic substitution will be covered.

Table 2. 1 Substitution mapping of Caesar cipher

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**
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

Table 2. 2 Character to number conversion

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**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

From **Table 2.1**, the top column of the table is the list of alphabets for plaintext, in lowercase. The bottom column of the table is the list of alphabets for ciphertext, in uppercase. According to Katz and Lindell [5], the standard key for encryption using Caesar cipher would be shifting the plaintext's character 3 to the right. To decrypt using the Caesar cipher would be shifting the ciphertext's character 3 to the left. With this algorithm, equations for encryption and decryption can be defined such that [5]: –

$$Ct = Enc(k, Pt) = (Pt + k)\ mod\ x$$
$$Ct = (Pt + 3)\ mod\ 26$$

$$Pt = Dec(k, Ct) = (Ct - k)\ mod\ x$$
$$Pt = (Ct - 3)\ mod\ 26$$

$$where\ x = no.of\ letters, k = no.of\ shift$$

Since the key, k can be changed for the cipher (the standard for Caesar cipher is shifting right to 3), the equation above can be rearranged to find for k.

$$k = (Ct - Pt)\ mod\ x$$
$$k = (Ct - Pt)\ mod\ 26$$

In **Table 2.2**, each character is mathematically assigned a number, starting with zero at "A." It will be incremented by one for each character, until "Z." Now, with every element of the algorithm are assigned with numbers, the equation can be worked upon. **Figure 2.2** message from earlier is set for example. **Table 2.3** and **Table 2.4** will show the encryption and decryption calculation process by using the equations.

Table 2. 3 Calculation of ciphertext using equation

Pt	h	e	l	l	o	b	o	b
P_i	7	4	11	11	14	1	14	1
K_i	3	3	3	3	3	3	3	3
$P_i + K_i$	10	7	14	14	17	4	17	4
$(Pt + K_i)$ mod 26	10	7	14	14	17	4	17	4
Ct	K	H	O	O	R	E	R	E

Table 2. 4 Calculation of plaintext using equation

Ct	K	H	O	O	R	E	R	E
C_i	10	7	14	14	17	4	17	4
K_i	3	3	3	3	3	3	3	3
$C_i - K_i$	7	4	11	11	14	1	14	1
$(C_i - K_i)$ mod 26	7	4	11	11	14	1	14	1
Pt	h	e	l	l	o	b	o	b

- **Transposition**

Transposition works by moving the position of the plaintext elements, and the result of the movement will become the ciphertext [13]. In transposition, the way of "scrambling" the plaintext is the key for the cipher. An example of this building block would be keyless transposition cipher. Although the name suggests that it is keyless, however, the key for the cipher would be the method of transposing the plaintext characters and reading the ciphertext across a pre-determined two-dimensional matrix.

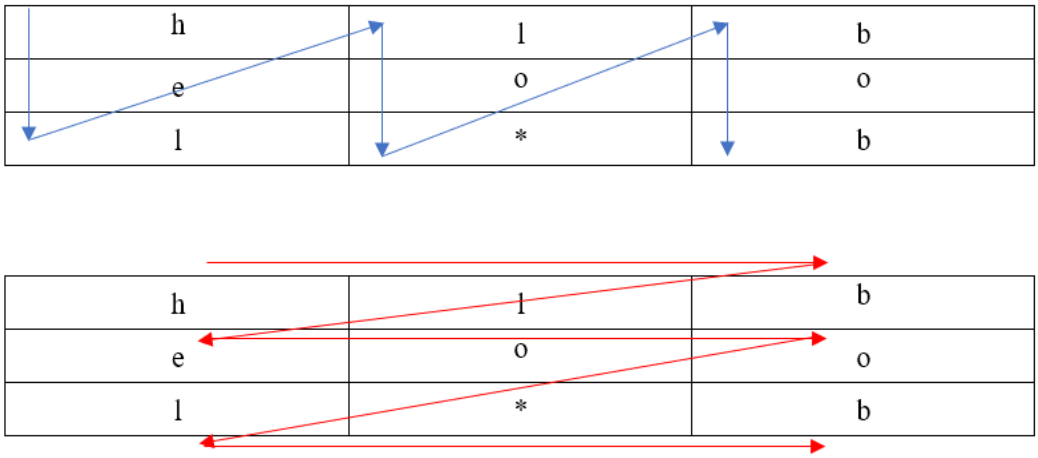


Figure 2. 3 Visualisation of encryption using transposition

Transposition works by arranging the plaintext in a pre-determined matrix size. In **Figure 2.3**, the plaintext is written to a 3x3 matrix; from top-left, going down until it reaches the bottom-right of the matrix. This is visualised in the figure with blue arrows., The ciphertext is read from the top-left, going left until it reach the bottom-right of the matrix to extract the ciphertext. This is visualised in the figure with red arrows. In this example, assume that the character "*" is for space, the ciphertext for the plaintext "hello bob" is "hlb eoo l*b."

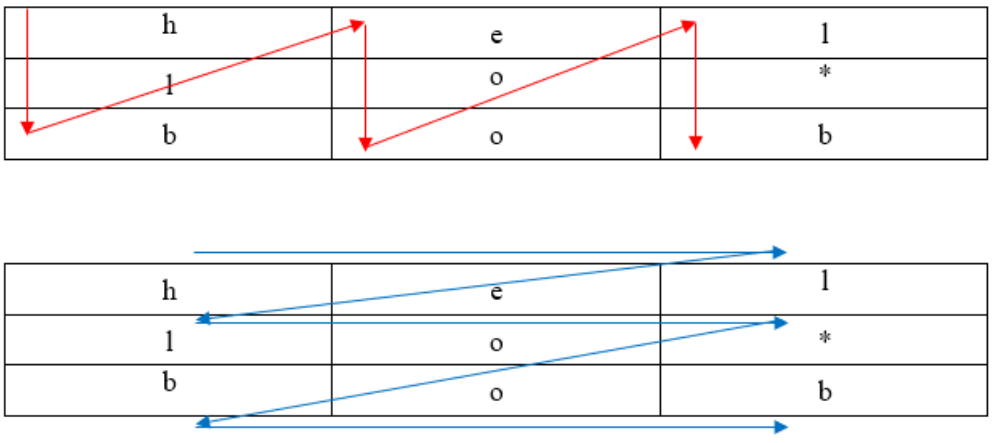


Figure 2. 4 Visualisation of decryption using transposition

To retrieve back the plaintext, the same procedure is reversed by writing the ciphertext like the plaintext is written, from top-left to bottom. The plaintext is read from the top-left to the right. **Figure 2.4** visualises this process.

Claude Shannon in his proposal of a secure cryptographic cipher, asserts that confusion and diffusion properties in a cipher is important [14]. This is because cryptanalysis that is based on statistical analysis by a threat actor, can be prevented or slowed down. Confusion is to make the relationship between the key and the ciphertext as complex as possible. Confusion can be achieved using substitution [13]. While diffusion, it is to increase the redundancy of plaintext by spreading across rows and columns. Diffusion can be achieved using transposition or permutation [13].

a) Advanced Encryption Standard

Advanced Encryption Standard ("AES") is a standard requirement set by National Institute of Standards and Technology ("NIST") for supersede the encryption standard that was set at that time, Data Encryption Standard ("DES") [15]. To simplify, DES and its variant ("3-DES") has become obsolete since its key size was considered short at 56 / 168-bits, and it is no longer computationally secure since modern computational power has proved it to be broken within a feasible time [16].

The program started in 1999, with fifteen symmetric ciphers that were proposed to be standard in Round 1 and Round 2 selection such as MARS, RC6, Rijndael, SAFER+, SERPENT, TWOFISH and others. Cryptanalysis and major flaws found in the design of some proposed ciphers have reduced the total proposed ciphers to five [17], [18], [19]. In Round 3 of the selection, **Rijndael cipher** was announced to be the winner of the selection and to be worked upon to be set as the successor of AES. In 2001, after public request-for-comment ("RFC") and minor iterations of changes made to the original cipher, NIST announced that the last version of the cipher is to be made standard as Federal Information Processing Standard ("FIPS") – FIPS 197 (2001) [15].

AES is a symmetric block cipher that is based upon iterations of Rijndael cipher, which was created by Vincent Rijmen and Joan Daemen [20]. The cipher processes inputs and outputs as a 128-bit block size. It has a variable key size which are 128-bits, 192-bits, and 256-bits long. Depending on the key size, variations of AES are named AES-128, AES-192, and AES-256. Furthermore, for AES-128, the number of rounds that are performed are 10 rounds. For AES-192, it is 12 rounds and for AES-256, it's 14 rounds [21].

To compare with DES and 3-DES, AES's block size is larger than DES's 64-bits block size. AES's key length is longer and much more flexible than DES's key length, which was fixed at 40 or 56-bits long. AES do not use a Feistel structure compared to DES. With AES's longer key length, the number of attempts required to brute-force the key is also increased. **Table 2.5** below shows these comparisons.

Table 2. 5 Comparison between DES, 3-DES, and AES

Criteria/Cipher	**DES**	**3-DES**	**AES**
Primitives	Feistel	Feistel	NOT Feistel
Key size (bits)	56	for k1 ≠ k2 ≠ k3, 168 for k1 = k2, 112	128, 192, 256
Block size (bits)	64	64	128
Possible key attempts	2^{56}	$2^{112}, 2^{168}$	$2^{128}, 2^{192}, 2^{256}$
No. of rnds.	16	16	10, 12, 14
Security	Not secure	Weak	Secure
Speed	Slow	Very slow	Fast

First, inputs from plaintext are converted into hexadecimals (Base16) and separated to 2 bytes each to fill 128-bit blocks. If the final block does not have enough plaintext bits

to fill, padding of the block will be done, depending on the cipher's mode of operation; in this case Cipher Block Chaining ("CBC"). Padding is adding extra bits, typically null (0x00) bits to the end of the plaintext, to fill up the entire block [21]. **Figure 2.5** visualises on how CBC works, compared to normal operation mode, Electronic Code Book ("ECB"). ECB is on the left side, while CBC is on the right side.

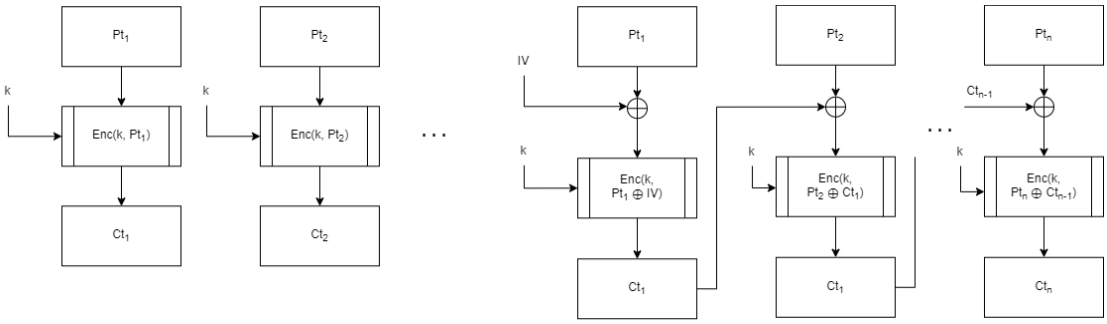


Figure 2. 5 Visualisation of ECB and CBC operation mode

Then, the first block of the formed plaintext will be XORed ("eXclusive OR") with an initial cipher key called Initialisation Vector ("IV"). Next, the main four operations in AES will be done on each block of the formed plaintext. The operations that are done are named *SubBytes (), ShiftRows (), MixColumns (),* and *AddRoundKey ()* [20], [22]. These operations are visualised in **Figure 2.6** below.

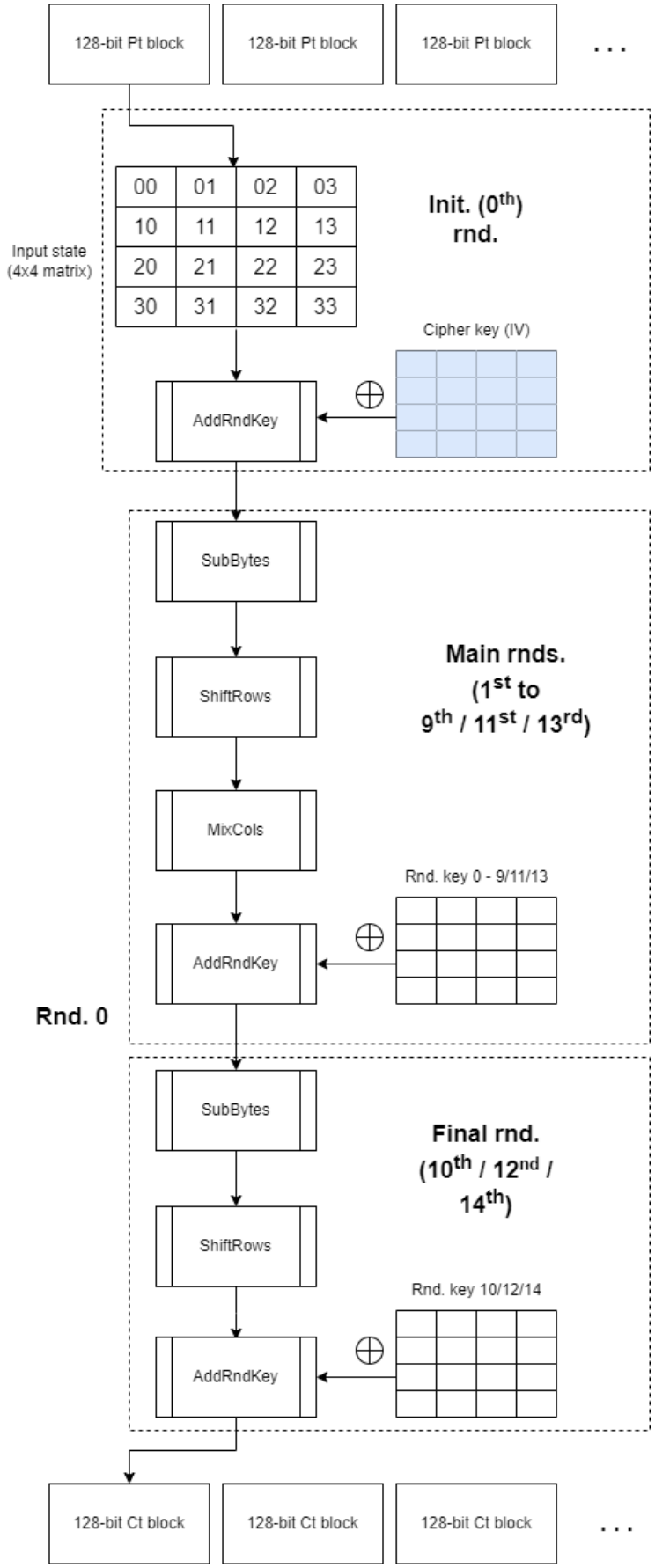


Figure 2. 6 High-level visualisation of AES

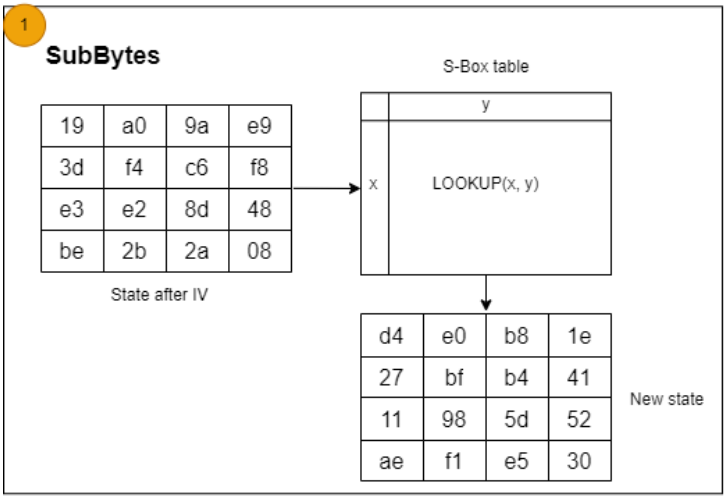


Figure 2. 7 AES's SubBytes operation

Table 2. 6 AES's S-Box table [15]

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

In **Figure 2.7**, the *SubBytes* function is visualised. Each element of the matrix will be performed a substitution by using a table named AES S-Box table, in **Table 2.6**. Take for example the first element in the state which is 19. A lookup of the corresponding value in the S-Box table is performed by assigning the element into (x, y). In this case, it would be $x - axis = 1,\ y - axis = 9; (1, 9)$. The value on the intersection of these axis is extracted, which is $d4$. This operation mimics the substitution, recalled back on cipher's building block [20], [22].

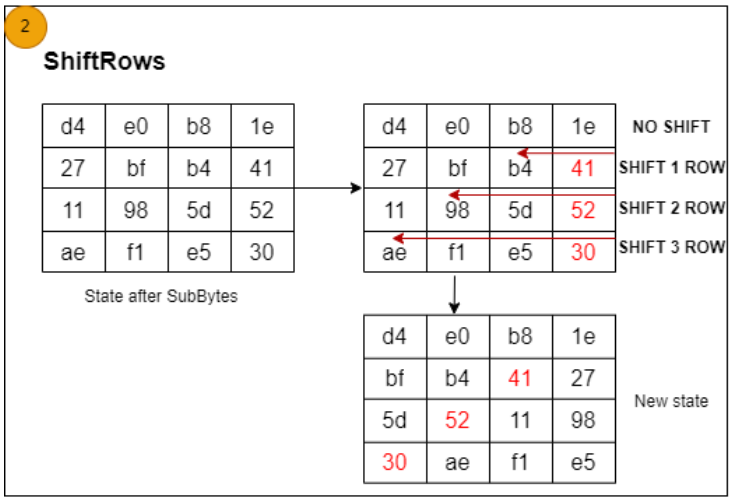


Figure 2. 8 AES's ShiftRows operation

In **Figure 2.8**, the *ShiftRows* operation is visualised. The elements of rightest column of the state will change its position, moving to the left; and incrementing the position by one, per rows. The first row's element will not shift. The second row's element will be shifted one column to left. The third row's element will be shifted two columns to the left. The last row's element will be shifted three columns to the left. Do note that the remaining elements that will be affected by the shift, will be rotated back into the row, from the left; in a queue. This operation mimics the transposition, recalled back on cipher's building block [20], [22].

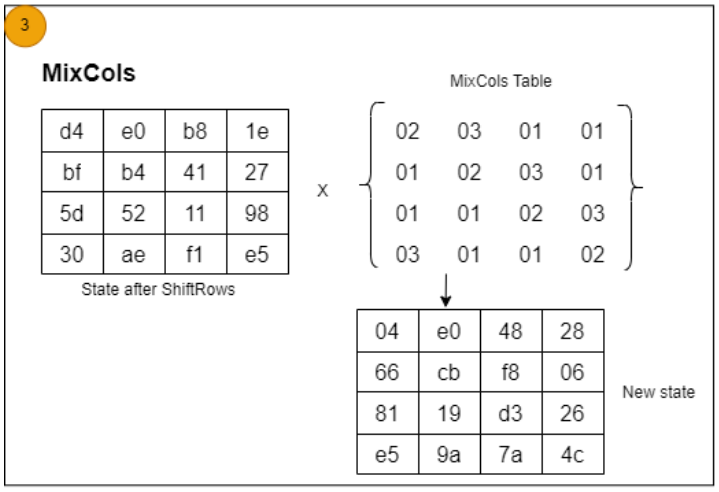


Figure 2. 9 AES's MixCols operation

In **Figure 2.9**, the *MixCols* operation is visualised. All elements per column are performed a matrix modulus multiplication with values in MixCols table. MixCols table is a Rijndael's Galois field – $GF\ /\ \mathbb{F}(2^8)$, by a given matrix, in this case a 4x4 matrix [20], [22]. This operation is vital to increase the diffusion properties of the cipher. Take for example the first column from the left, assuming b_n is the column in the state before the operation and d_n is the column in the state after the operation, the equation would be:

$$b_0 = d4_{16} = 11010100_2\ ;\ GF(2^8)\ poly. = x^7 + x^6 + x^4 + x^2$$
$$b_1 = bf_{16} = 10111111_2\ ;\ GF(2^8)\ poly. = x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$$
$$b_2 = 5d_{16} = 01011101_2\ ;\ GF(2^8)\ poly. = x^6 + x^4 + x^3 + x^2 + 1$$
$$b_3 = 30_{16} = 00110000_2\ ;\ GF(2^8)\ poly. = x^5 + x^4$$

$$2_{10} = 00000010_2 = x\ \ ;\ \ 3_{10} = 00000011_2 = x + 1\ \ ;\ \ x^8 = x^4 + x^3 + x + 1$$

$$\begin{matrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{matrix} \cdot \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{matrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{matrix} = \begin{matrix} b_0 \cdot 2 \oplus b_1 \cdot 3 \oplus b_2 \cdot 1 \oplus b_3 \cdot 1 \\ b_0 \cdot 1 \oplus b_1 \cdot 2 \oplus b_2 \cdot 3 \oplus b_3 \cdot 1 \\ b_0 \cdot 1 \oplus b_1 \cdot 1 \oplus b_2 \cdot 2 \oplus b_3 \cdot 3 \\ b_0 \cdot 3 \oplus b_1 \cdot 1 \oplus b_2 \cdot 1 \oplus b_3 \cdot 2 \end{matrix}$$

The equation above is solved with the polynomial notation of b_n. Note that for every x^8 variables, it will be substituted with the value of the $GF\ /\ \mathbb{F}(2^3)$, x^8 irreducible polynomial theorem.

$$\begin{aligned} &Solve\ for\ [d_0]: \\ &b_0 \cdot 2 = x(x^7 + x^6 + x^4 + x^2) \\ &\quad = \cancel{x^8} + x^7 + x^5 + x^3 \\ &\quad = x^4 + \cancel{x^3} + x + 1 + x^7 + x^5 + \cancel{x^3} \\ &\quad = x^7 + x^5 + x^4 + x + 1 = 10110011_2 \end{aligned}$$

$$\begin{aligned} &b_1 \cdot 3 = x + 1(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \\ &\quad = (\cancel{x^8} + x^6 + \cancel{x^5 + x^4 + x^3 + x^2 + x}) + x^7 + \cancel{x^5 + x^4 + x^3 + x^2 + x} + 1 \\ &\quad = x^4 + x^3 + x + \cancel{1} + x^6 + x^7 + \cancel{1} \\ &\quad = x^7 + x^6 + x^4 + x^3 + x = 11011010_2 \end{aligned}$$

$$b_2 \cdot 1 = 01011101_2$$
$$b_3 \cdot 1 = 00110000_2$$

$$d_0 = 10110011_2 \oplus 11011010_2 \oplus 01011101_2 \oplus 00110000_2$$
$$= 00000100_2 = \mathbf{04_{16}}$$

Solve for $[d_1]$:

$b_0 \cdot 1 = 11010100_2$

$b_1 \cdot 2 = x(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1)$
$= (\cancel{x^8} + x^6 + x^5 + x^4 + x^3 + x^2 + x)$
$= \cancel{x^4 + x^3 + x} + 1 + x^6 + x^5 + \cancel{x^4 + x^3} + x^2 + \cancel{x}$
$= x^6 + x^5 + x^2 + 1 = 01100101_2$

$b_2 \cdot 3 = x + 1(x^6 + x^4 + x^3 + x^2 + 1)$
$= (x^7 + x^5 + \cancel{x^4 + x^3} + x) + x^6 + \cancel{x^4 + x^3} + x^2 + 1$
$= x^7 + x^6 + x^5 + x^2 + x + 1 = 11100111_2$

$b_3 \cdot 1 = 00110000_2$

$$d_0 = 11010100_2 \oplus 01100101_2 \oplus 11100111_2 \oplus 00110000_2$$
$$= 01100110_2 = \mathbf{66_{16}}$$

Solve for $[d_2]$:

$b_0 \cdot 1 = 11010100_2$
$b_1 \cdot 1 = 10111111_2$

$b_2 \cdot 2 = x(x^6 + x^4 + x^3 + x^2 + 1)$
$= (x^7 + x^5 + x^4 + x^3 + x) = 10111010_2$

$b_3 \cdot 3 = x + 1\,(x^5 + x^4)$
$= (x^6 + \cancel{x^5}) + \cancel{x^5} + x^4$
$= x^6 + x^4 = 01010000_2$

$$d_0 = 11010100_2 \oplus 10111111_2 \oplus 10111010_2 \oplus 01010000_2$$
$$= 10000001_2 = \mathbf{81_{16}}$$

Solve for $[d_3]$:

$b_0 \cdot 3 = x + 1(x^7 + x^6 + x^4 + x^2)$
$= (\cancel{x^8} + \cancel{x^7} + x^5 + x^3) + \cancel{x^7} + x^6 + x^4 + x^2$
$= \cancel{x^4} + \cancel{x^3} + x + 1 + x^5 + \cancel{x^3} + x^6 + \cancel{x^4} + x^2$
$= x^6 + x^5 + x^2 + x + 1 = 01100111_2$

$b_1 \cdot 1 = 10111111_2$
$b_2 \cdot 1 = 01011101_2$

$b_3 \cdot 2 = x(x^5 + x^4)$
$= x^6 + x^5 = 01100000_2$

$$d_3 = 01100111_2 \oplus 10111111_2 \oplus 01011101_2 \oplus 01100000_2$$
$$= 11100101_2 = \boldsymbol{e5_{16}}$$

$$\therefore \begin{matrix} d4 \\ bf \\ 5d \\ 30 \end{matrix} \cdot \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{matrix} 04 \\ 66 \\ 81 \\ e5 \end{matrix} \blacksquare$$

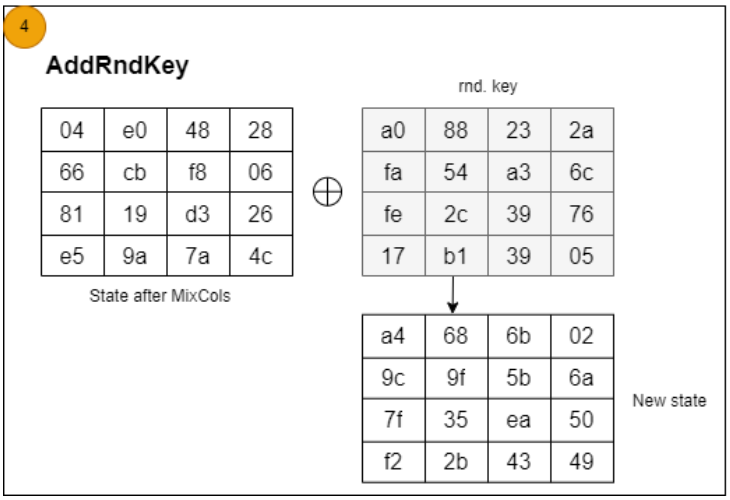


Figure 2. 10 AES's AddRoundKey operation

In **Figure 2.10**, the AddRoundKey operation is visualised. Each column from the state is XORed with columns from current round key that was generated after IV [20], [22]. AES key generation operation for each round is not discussed in this thesis.

Going back to **Figure 2.6**, these four main operations will be repeated until the second last round of the AES, depending on the variation. AES-128 will repeat these operations for 9 times, AES-192 for 11 times and AES-256 for 13 times.

On the last round of AES, which is the 10^{th}, 12^{nd} and 14^{th} round, only SubBytes, ShiftRows and AddRoundKey will be performed. The MixCols operation will be omitted. After this last round has completed, the final state will be the ciphertext for the corresponding plaintext block. Continuing **Figure 2.7** on CBC, the ciphertext block of the final round will be XORed with the next plaintext block instead of the IV and the operations will repeat again to encrypt the subsequent plaintext blocks [20], [22].

To conclude on symmetric cipher, it provides a robust method implementing multiple iterations of substitution and transposition while increasing its confusion and diffusion properties, to protect the confidentiality of message.

2.1.2 Asymmetric cryptography

Symmetric cryptography can only provide data confidentiality, but not data integrity and data authenticity. Below are some of the limitations and disadvantages of symmetric cryptography:

a) Key Proliferation / Exhaustion

With symmetric ciphers, each pair of users will need a separate key. This means that for an $n - user$ system requires $n\left(\frac{(n-1)}{2}\right)$ keys. Furthermore, each user must remember different keys for every distinct channel that used to communicate with other users to communicate. As for the system's side, with the increase of number of users, the number of keys will also increase rapidly [23].

b) Key Distribution

From **Figure 2.2**, for both parties to start using a symmetric cipher, they must have to agree upon a shared secret key. Both parties must use a secure means of communication (secure channel) to exchange the keys. Now, assume that there is no "secure channel" for both parties to exchange the keys. Keys cannot be shared on an insecure channel e.g., over the Internet in plaintext because any adversaries can intercept the key and subsequently decrypt all future communications in future using the key; referring to Kerckhoff's Principle [23].

c) Entity Authentication and Non-Repudiation

Symmetric cryptography in its essence, only provides protection of the message exchange, and do not have the methods to provide the proof of authenticity of the message [23].

Below is a high-level view of how asymmetric cryptography solves some of the problems encountered with symmetric cryptography.

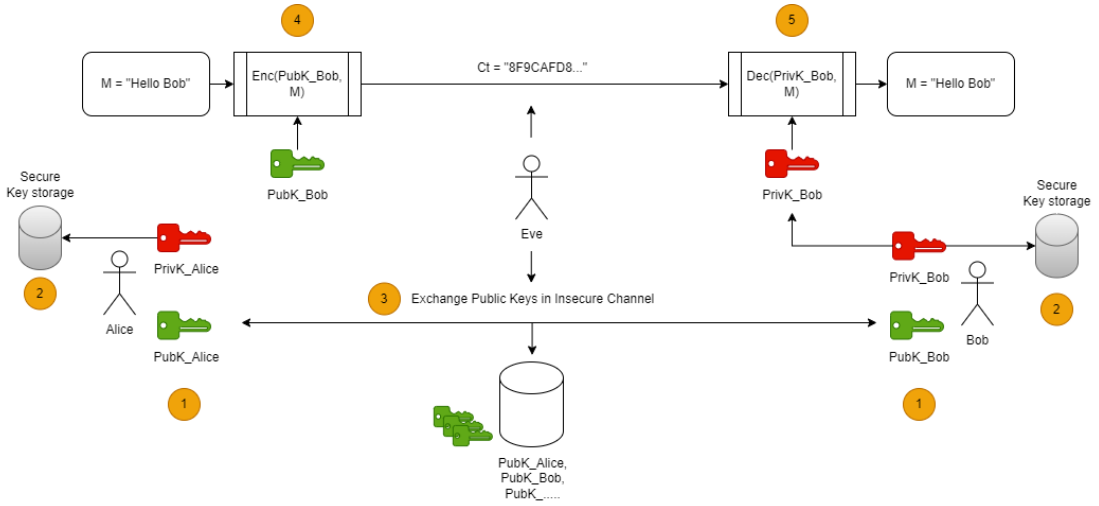


Figure 2. 11 Visualisation of Asymmetric cryptography

In **Figure 2.11**, in step 1, both Alice and Bob now generating a pair of keys, which is a Public and Private key. These two keys can be called $PubK_{user}$ and $PrivK_{user}$ respectively, with $user$ being Alice or Bob. In step 2, both of the $PrivK_{user}$ are stored in a secure location, locally for each party.

For step 3, the simplest key exchange method happens here, where both parties will exchange their $PubK_{user}$, over the insecure Internet. For easier management of the keys, there can be a separate trusted entity that will store all public keys for users in a server.

For step 4, Alice will either obtain Bob's Public key, $PubK_{Bob}$ from the exchange or fetch it from the server of the trusted entity. $PubK_{Bob}$ is then used for the key for the encryption of the message. Alice then send the ciphertext to Bob.

Finally, in step 5, Bob will decrypt the ciphertext using his private key, $PrivK_{Bob}$.

In this thesis, two most renowned asymmetric cryptosystem that have been developed are reviewed, which are Rivest-Shamir-Adleman ("RSA") Cryptosystem, and Elliptic Curve Cryptosystem ("ECC").

a) Rivest-Shamir-Adleman Cryptosystem

Rivest-Shamir-Adleman Cryptosystem ("RSA") is a first public-key cryptography proposed and patented in the US in 1977 by Rivest et. al., at Massachusetts Institute of Technology (MIT); hence the name taken from their initials [7], [24], [25]. Although they were the first one to propose and patent the method publicly, it could also be traced back to a secret program, now declassified, by the Government Communications Headquarters ("GCHQ") back in 1973 by Clifford Cocks [26].

Like AES, RSA also has variable key sizes; 1024-bits, 2048-bits, and 4096-bits. RSA's variation would be named RSA-1024, RSA-2048 and RSA-4096. RSA-1024 is considered insecure and many proposed to use more larger key sizes for RSA; as long as 15360-bits long.

Because of the huge performance overhead of RSA, it is used for performing encryption and decryption for symmetric key exchanges. RSA is also used for digital signatures and digital certificates.

Below is a graphical representation of RSA's algorithm on the practical implementation. **Figure 2.12** shows how RSA works in detail.

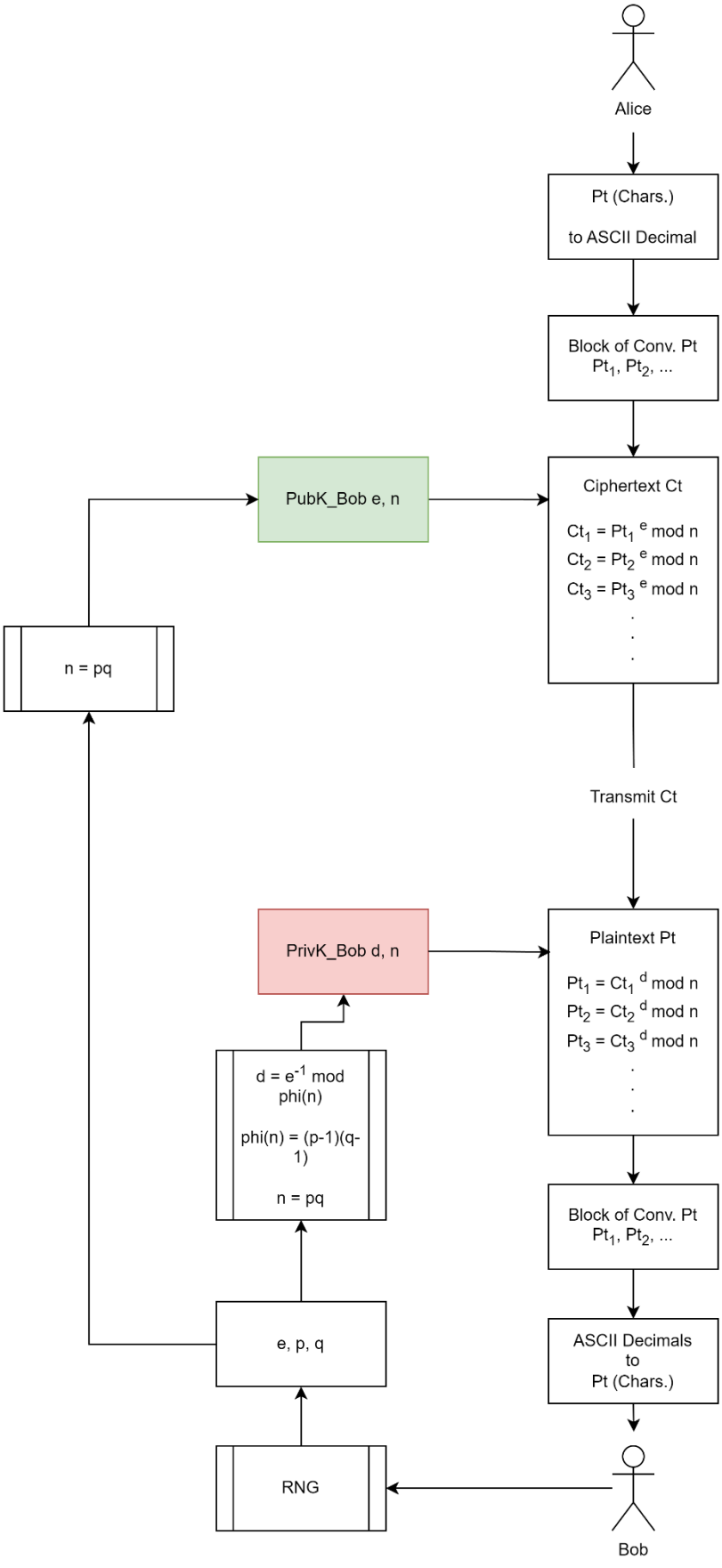


Figure 2. 12 High-level visualisation of RSA

First, the receiver side would have to perform key generation. In this case, Bob would have to generate a public and private key pair.

Using RSA, he would have to choose two prime numbers p, q where $p \neq q$. Next, he would perform multiplication of these two numbers, $n = p * q$. Then, he would perform Euler's second totient function, $\varphi(n) = (p-1)(q-1)$. After calculating all these values, he would proceed to choose the public number e and compute the private number d [24], [27].

For the last step (choosing e and compute d) because the process can be done the other way around (choose d and compute e), it has been set upon on Public-Key Cryptography Standard (PKCS #1); where the earlier was favoured [28]. The equations for the entire process are noted below.

$$p, q \leftarrow p \; and \; q \; prime, \;\; p \neq q$$

$$n = p * q$$

$$\varphi(n) = (p-1)(q-1)$$

$$e \rightarrow \gcd(\varphi(n), e) = 1 \; ; 1 < e < \varphi(n)$$

$$d \rightarrow d \equiv e^{-1}\big(mod \; \varphi(n)\big)$$

$$1 = de \; mod \; \varphi(n)$$

Equations above are Euler's Totient function, $\varphi(n)$. Mathematically, it is supposed to be a Carmicheal's Totient function, $\lambda(n)$ but since $\varphi(n)$ can be divisible by $\lambda(n)$, the Euler's was chosen. In RSA implementation, the second and third Euler's totient function is used.

$$\varphi(1) = 0$$

$$\varphi(p) = p - 1 \; ; p \; is \; prime$$

$$\varphi(m * n) = \varphi(m) * \varphi(n)\ ; m, n\ is\ prime$$

$$\varphi(p^e) = p^e - p^{e-1}\ ; p\ is\ prime$$

Now, the equation is solved with a pre-set number for p, q. While the actual p, q length in RSA would be at least 100-digits long [24], here a small prime no. value is used. In the real implementation of RSA also, the value for p, q must be generated from a secure Pseudorandom Number Generator ("PRNG") to furthermore make cryptanalysis more difficult.

$$p, q = 61, 53$$

$$n = p * q$$
$$n = 61 * 53 = 3233$$

$$\begin{aligned}\varphi(n) &= (p-1)(q-1)\\ &= (61-1)(53-1)\\ &= 60 * 52\\ &= 3120\end{aligned}$$

$$e \rightarrow \gcd(\varphi(n), e) = 1\ ; 1 < e < \varphi(n)$$
$$e = \gcd(3120, e) = 1$$
$$e = 17$$

$$\begin{aligned}d \rightarrow d &\equiv e^{-1}(mod\ \varphi(n))\\ &= de\ mod\ \varphi(n) = 1\\ 1 &= d(17)\ mod\ 3120\\ d &= 2753\end{aligned}$$

Note that the values that are needed to keep private are $p, q, \varphi(n), d$. So, the private key pair will be $PrivK_{Bob} = \{d, n\}$. The public key pair will be $PubK_{Bob} = \{e, n\}$.

With the key pair generated for Bob, Alice would go on to perform encryption of the message, assuming she have already obtained $PubK_{Bob}$ from a key exchange with Bob.

In message encryption of RSA, first the Pt characters are converted to Base10 notation, using the ASCII table. Recall back on Caesar cipher's alphanumeric table assignment

in **Table 2.2**, this operation is the same, just with a different value. **Table 2.7** shows the values of converted ASCII character to Base10 notation (Note it is in lowercase). Actual implementations of RSA would use randomised padding on Pt to furthermore prevent against attacks on otherwise on plain RSA [28].

Table 2. 7 ASCII table of lowercase characters

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**
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	9	9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2
7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2

With Alice's plaintext "hello bob" converted with the ASCII table to Base10 notation, it will become: 104 # 101 # 108 # 108 # 111 # 032 # 098 # 111 # 098. Note that # 032 # is space (Not shown in the table).

Now, Alice can encrypt the message using the value converted from the plaintext characters. The equation for encryption and decryption of RSA are noted below [24], [27]. The value of Pt/Ct will be risen to the power of e/d, depending on encryption or decryption process. The exponent value will then be performed modulus division to find the remainder, and the final value of the operation will become the Pt/Ct. This operation is called modular exponentiation.

$$Ct = Pt^e \ mod \ n$$
$$Pt = Ct^d \ mod \ n$$

The equation could also be re-written with just Pt variable, by multiplying e, d.

$$Pt = (Pt^e)^d \ mod \ n$$
$$Pt = Pt^{ed} \ mod \ n$$

The values converted earlier is used to solve the equation, and to encrypt the Pt.

$$Ct_1 = 104^{17}\ mod\ 3323 = 2170$$
$$Ct_2 = 101^{17}\ mod\ 3323 = 1313$$
$$Ct_3 = 108^{17}\ mod\ 3323 = 0745$$
$$Ct_4 = 108^{17}\ mod\ 3323 = 0745$$
$$Ct_5 = 111^{17}\ mod\ 3323 = 2185$$
$$Ct_6 = 32^{17}\ mod\ 3323 = 1992$$
$$Ct_7 = 98^{17}\ mod\ 3323 = 2570$$
$$Ct_8 = 111^{17}\ mod\ 3323 = 2185$$
$$Ct_9 = 98^{17}\ mod\ 3323 = 2570\ \blacksquare$$

These values now can be considered as Ct, and can be transferred to Bob. To decrypt the Ct from Alice, he would then use his $PrivK_{Bob}$ to perform decryption. Proof using the decryption equation are noted below.

$$Pt_1 = 2170^{2753}\ mod\ 3323 = 104$$
$$Pt_2 = 1313^{2753}\ mod\ 3323 = 101$$
$$Pt_3 = 0745^{2753}\ mod\ 3323 = 108$$
$$Pt_4 = 0745^{2753}\ mod\ 3323 = 108$$
$$Pt_5 = 2185^{2753}\ mod\ 3323 = 111$$
$$Pt_6 = 1992^{2753}\ mod\ 3323 = 32$$
$$Pt_7 = 2570^{2753}\ mod\ 3323 = 98$$
$$Pt_8 = 2185^{2753}\ mod\ 3323 = 111$$
$$Pt_9 = 2570^{2753}\ mod\ 3323 = 98\ \blacksquare$$

Bob would then do the reverse process of character conversion, from Base10 to ASCII characters, “hello bob”.

RSA is secure because of the computational hardness assumption of infeasibility of current computational power to reverse integer factorization or prime decomposition. This is also called RSA problem. Below statements are the security requirements of the RSA cryptosystem:

- It’s feasible to find values of $\{e, d, n\}$, such that $Pt^{ed}\ mod\ n = Pt\ ; Pt < n$
- It’s feasible to calculate the $Pt^e\ mod\ n\ , Ct^d\ mod\ n\ ; Pt < n$

- It's infeasible to determine d given $\{e, n\}$, which is the public elements of the key.

With current computational power, no known algorithm can solve RSA problem in polynomial time $\mathcal{O}(n^k)$, where n here is specified length or bits of a number, and k is a constant (to the power of 2, 3, …). However, there is one algorithm that can solve the problem in polynomial time, utilizing the power of Quantum computers. **Chapter 2.5.2** of this thesis will cover this topic.

b) Elliptic Curve Cryptosystem

Elliptic Curve Cryptography ("ECC") is another solution for Public-Key Cryptography. ECC relies on algebraic structure of elliptic curves. Elliptic curve use in cryptography was independently proposed by Koblitz N. and Miller V. in 1985 [29], [30]. Since its introduction, ECC has been showing up in standardisation efforts in IEEE P1363 Standard for PKC [31] and endorsed by NIST for public use around 2005 [32], [33]. Most importantly, ECC can provide similar security while being more efficient in implementation compared to RSA and AES, for smaller key size.

ECC's key sizes are usually 160, 224, 256, 384 and 521-bits long. Initially it was developed to be a Key Exchange Protocol, but the feasibility of the algorithm to be used as a cryptographic primitive makes it suitable for asymmetric cryptography usage [34].

To compare ECC with RSA and AES, **Table 2.8** shows the comparison between all the ciphers mentioned with security relative to current requirement by key size.

Table 2. 8 Comparison between AES, RSA and ECC

Security/Cipher	**AES key size (bits)**	**RSA key size (bits)**	**ECC key size (bits)**
Insecure	80	1024	160
Weak	112	2048	224
Adequate / Baseline for AES	128	3072	256
Good	192	7680	384
Most secure	256	15360	512

ECC relies on multiple addition on elliptic curve point. To define an elliptic curve, a simplified *Weierstrass* equation below is noted; assuming the curve is plotted over a Galois Field or real number $\mathbb{R}$; $a, b \in \mathbb{R}$, along a distinguished point at ∞:

$$y^2 = x^3 + ax + b$$

Visualisation of the equation above is noted below in **Figure 2.13**, with values of $a = -3, b = 5$.

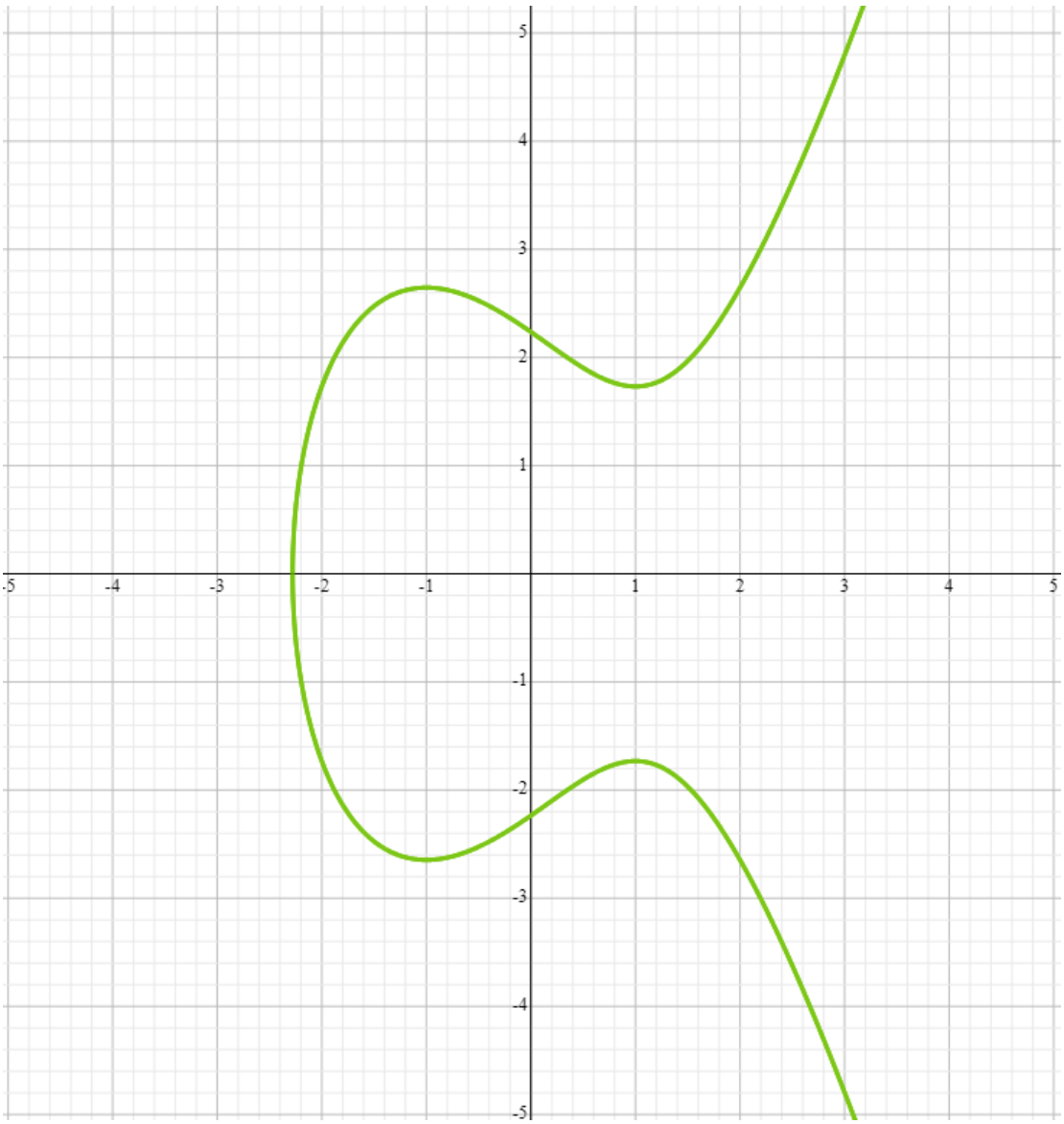


Figure 2. 13 Elliptic Prime Curve on graph

For users to use ECC, they must agree upon a curve to use. There are multiple curves on Galois Field, $\mathbb{F}$ to choose from, but NIST recommended on three types of curves to use in FIPS 186-4 [33]:

- 5 prime fields, $\mathbb{F}_p$ for primes $p = 192, 224, 256, 384, 521$-bits. Each p has one recommended elliptic curve.

- 5 binary fields, $\mathbb{F}_{2^m}$ for m is degree of the field, $m = 163, 233, 283, 409, 571$-bits. Each m has one recommended elliptic curve.

- From $\mathbb{F}_{2^m}$, 5 Koblitz curve is also selected for recommendation.

For curves, E over $\mathbb{F}_p$, the following equation for the curve is noted.

$$E : y^2 \equiv x^3 + ax + b \ (mod\ p)$$

For curves, E over $\mathbb{F}_{2^m}$, the following equation for the curve is noted.

$$E : y^2 + xy = x^3 + ax^2 + b$$

For curves, E_a over $\mathbb{F}_{2^m}$ for Koblitz curve, the following equation for the curve is noted.

$$E_a : y^2 + xy = x^3 + ax^2 + b \ ; a = 0 \ or \ 1$$

There are also few more other curves that users can choose upon outside NIST recommendation. These curves are for example Edwards curve and Montgomery curve. Both curves are birationally equivalent to each other. The visualisation of these two curves are in **Figure 2.14** and **Figure 2.15**.

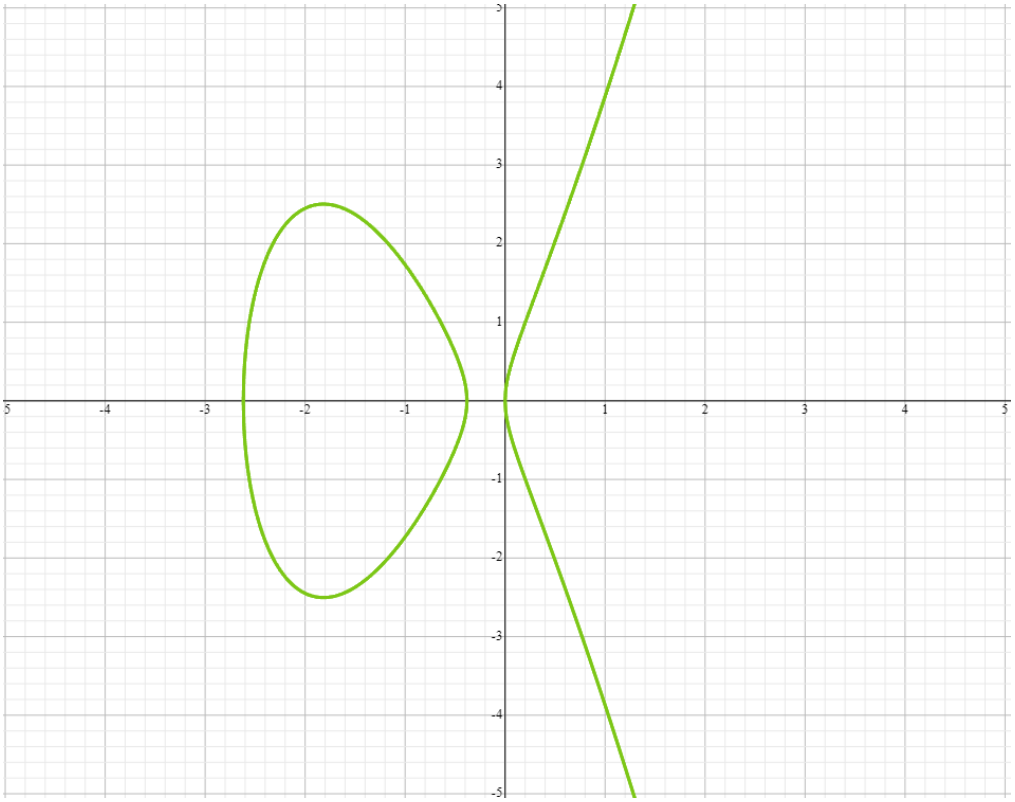


Figure 2. 14 Elliptic Montgomery curve on graph

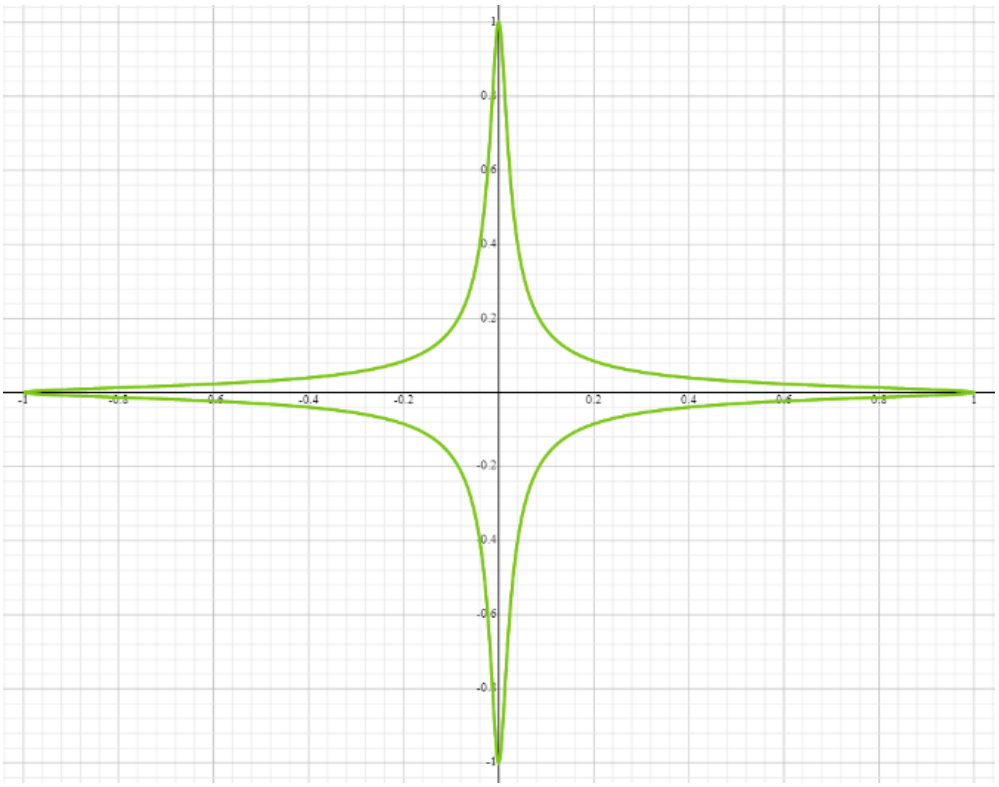


Figure 2. 15 Elliptic Edwards curve on graph

In **Figure 2.14** and **Figure 2.15**, the graph of both curves mentioned earlier is visualised. The Montgomery curve equation is noted here with a value example: $by^2 = x^3 + ax^2 + x\,; b = \frac{1}{4}\,, a = 3$. The Edwards curve equation is noted here with a value example: $x^2 + y^2 = 1 + dx^2y^2\,; d = -3301$

The Montgomery curve over $\mathbb{F}_p$ $y^2 = x^3 + ax^2 + x \; mod \; p$ with $a = 486662, p = 2^{255} - 19$ is notably used among most modern application nowadays. This curve is called *Curve25519*. *Curve25519* was independently proposed by Bernstein D. and are not subject to scrutiny because it's not under NIST or NSA influence [34], [35].

There is a reason why Montgomery and Edwards Curves exists for ECC. ECC had gained a quite bad reputation since its malicious usage by NSA, part of their decryption program named BULLRUN, to implement a backdoor on Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG) as a standard by NIST in SP800-90A, where the PRNG uses ECC for its implementation [36]. Its backdoor was discovered by independent cryptanalysts and deduced that only the designers of the algorithm; only NSA could confirm the existence of the backdoor [37]. It was since disapproved by NIST to become a standard, and it put a lot of distrust on the entire ECC algorithm in general [38]. This is also why widely used ECC is not from NIST, since the fear of NSA's backdoor implementation in ECC and overall distrust on NIST over the ECC is still high.

After selecting the curve to be agreed upon by both entities, first the person must select a distinct point on the chosen elliptic curve, noted p.

Now, with the point p, point addition can be performed on the point with respect to the chosen elliptic curve. Point addition in elliptic curve arithmetic is adding the point to itself to get to another new coordinate [39]; the equation would be noted as:

$$Q = kP$$

Q would be the new coordinate on the elliptic curve after the addition operation. k would be how many times of operation (or hops) it takes to satisfy given point Q [39]. When reduced, it could also be noted in a logarithm form. This equation, could also be written as:

$$Q = P * P * P * \dots$$

$$Q = P^k = \{P * P * P \dots\} : k\ factors$$

$$k = \log_p Q$$

Figure 2.16 visualise on how Point addition are done on the elliptic curve; denoted with R. Also note that two points are used for addition, where $P = \{x_1, y_1\}, Q = \{x_2, y_2\}, R = \{x_3, -y_3\}, -R = \{x_3, y_3\}$.

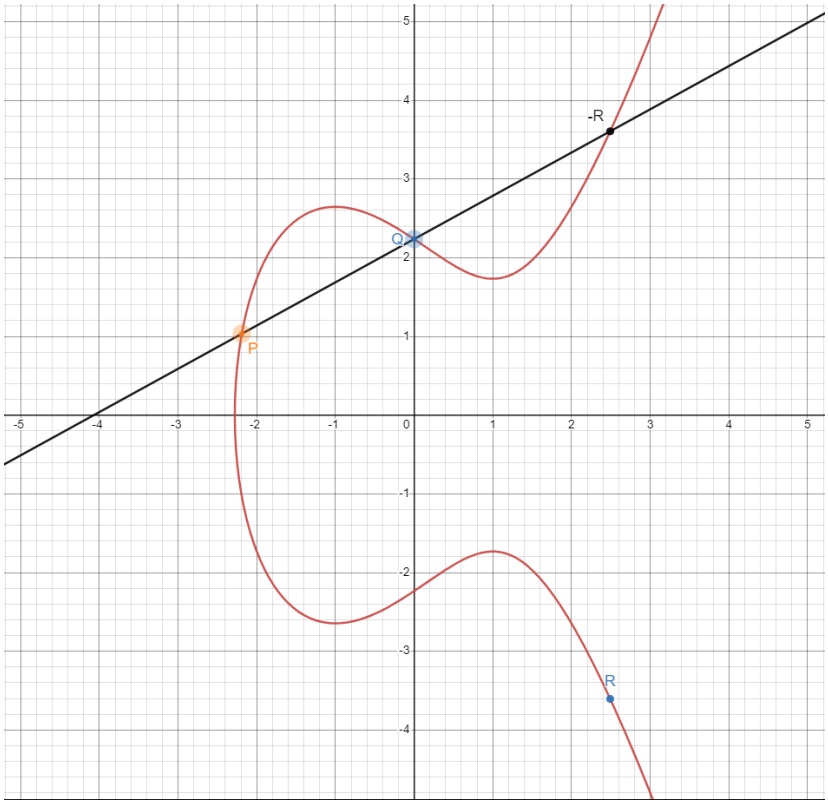


Figure 2. 16 Visualisation of point addition in ECC [40]

With these values computed, the public key would be the starting point and the end point of the above equation with respect to the chosen elliptic curve; $PubK_{user} = \{P, Q\}$. The private key would be the number of point addition operation (or hops) it takes from P to satisfy Q; $PrivK_{user} = \{k\}$. **Figure 2.17** visualise on how ECC works.

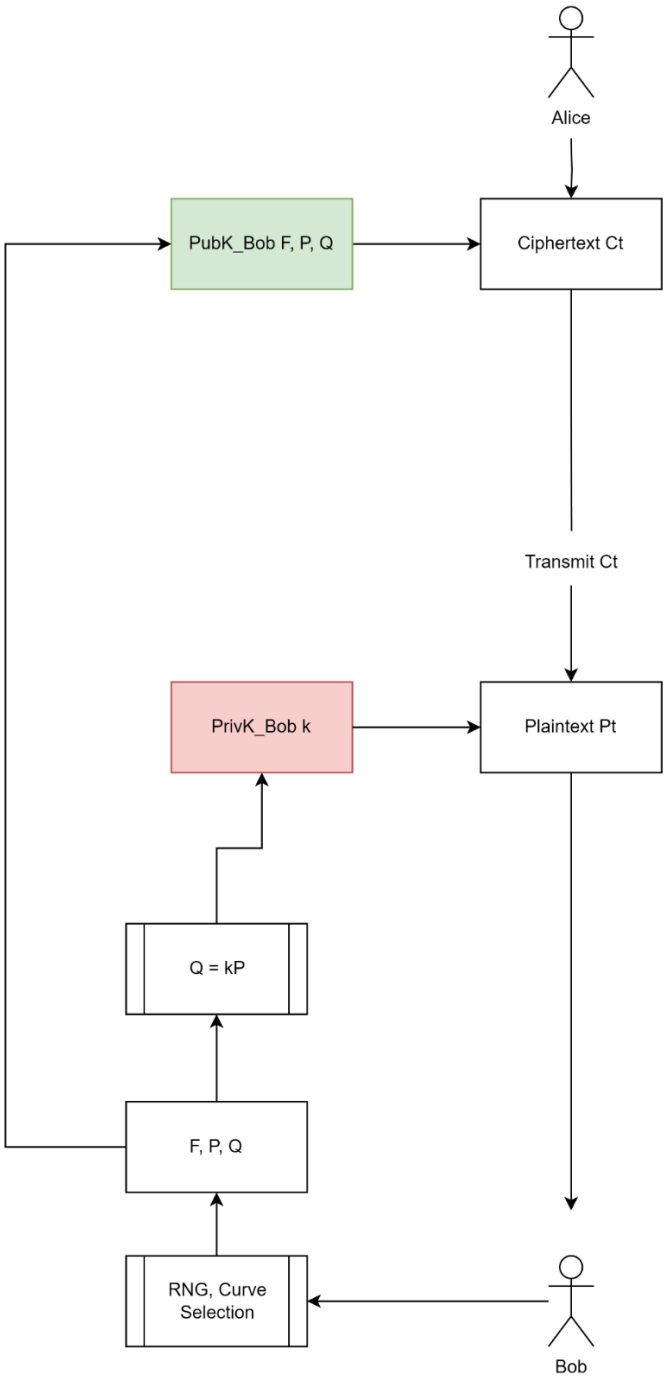


Figure 2. 17 High-level visualisation of ECC

ECC is considered secure because of the computational hardness assumption of infeasibility of current computational power to solve the elliptic curve logarithm problem. This problem is like discreet logarithm, just that it is applied to an elliptic curve. Below statements are the security requirements of ECC:

- It is feasible to find Q, given k, P to satisfy that $Q = kP$.

- It is infeasible to find k, given P, Q to satisfy that $Q = kP$.

Till date, there is no available algorithms that can compute discreet logarithm in polynomial time using current computational power. The general solution for solving discreet logarithm is trial multiplication, where from $k = \log_p Q$, P is raised to larger and larger powers of k until Q is found to satisfy earlier equation. This solution will take exponential time, $\mathcal{O}(2^n)$ and only applicable for small values of P, Q. Although, there are few algorithms that exists, inspired from integer factorization. To name a few, Pohlig-Hellman attack, and Pollard-Rho attack [39].

To conclude on Asymmetric cryptography, it provides a way to provide confidentiality and authenticity by using a different key to encrypt and decrypt messages, which a key pair belongs to a distinct user.

2.1.3 Post-Quantum cryptography

Despite the name “Post-Quantum” or “Quantum-resistant” cryptography suggests, it does not use a quantum computer to perform the algorithm. That term for the latter would be “Quantum Cryptography.” Current asymmetric cryptography primitives are based on integer factorization; RSA problem, discreet logarithm, and elliptic curve discreet logarithm; ECDL problem. It is just a matter of finding another “hard problem” that cannot be solved both in conventional and quantum computers. Post-quantum algorithms implementing the “hard problem” must be able to run in an acceptable manner in a conventional computer.

Current post-quantum cryptography research and development focuses on five different methodologies, excluding post-quantum symmetric cryptography. All post-quantum cryptography proposal now focuses on asymmetric cryptography and in more particular, key exchange protocols or now are named key encapsulation mechanisms (“KEM") [41].

Since the research of these new emerging cryptographic primitives that employs the use of a new “hard problem,” thorough security assessments and testing must be done in order to validate them to be really secure against both conventional and Quantum computer attacks. Currently, there is one public institution that encouraging the research and development of Post-Quantum cryptography, which is PQ-CRYPTO [42].

In this thesis, three selected Post-Quantum cryptosystem that have been developed are reviewed, which are Code-based Cryptosystem, Multivariate-Quadratic-Equation based (“MQE”) Cryptosystem , and Lattice-based Cryptosystem.

a) Code-based Cryptosystem

In this thesis, the **McEliece cryptosystem** is reviewed. McEliece cryptosystem is an asymmetric cipher developed by McEliece R. in 1978, around the same time where RSA

and the idea of asymmetric cryptography was popularized [41], [43], [44]. But since RSA was chosen for the standard for conventional asymmetric cryptography, code-based cryptosystem did not gain a lot of attention at that time. However, it has been realised in today's time where it could withstand attacks from Quantum computers due to the "hard problem" that is the foundation of the cryptosystem. In round 4 of current NIST post-quantum standardization program, almost all proposals used code-based cryptographic primitives, derived from the original McEliece cryptosystem.

Figure 2.18 visualises on how McEliece cryptosystem works.

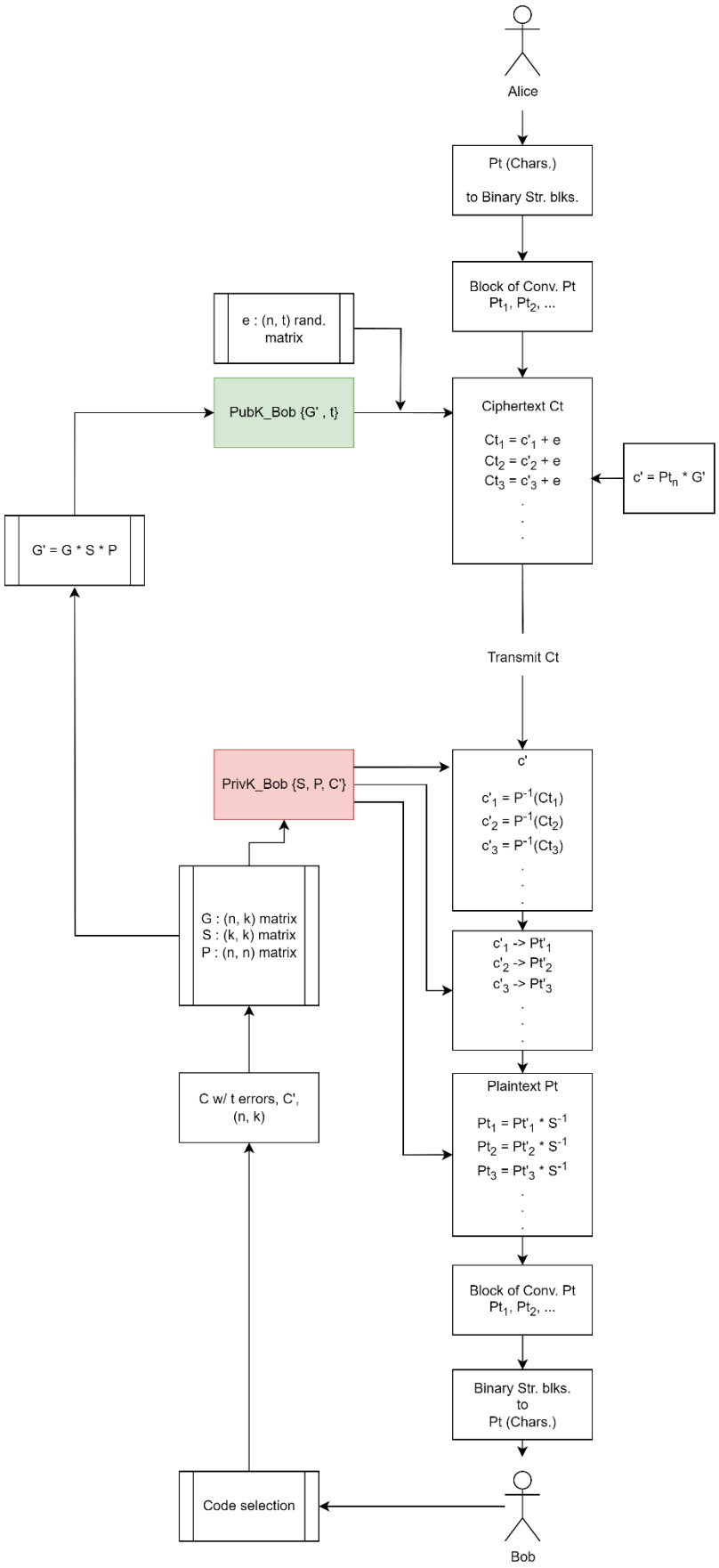


Figure 2. 18 High-level visualisation of McEliece cryptosystem

In key generation, Alice and Bob have to choose an error-correcting code; by default it's a binary Goppa code $\mathcal{C}$; which is known to be efficient to correct an amount of t errors. Since $\mathcal{C}$ is an encoding code, there also must be an efficient decoding algorithm. Assume that $\mathcal{C}'$ is that latter. Also, $\mathcal{G}$ would be any generator matrix for $\mathcal{C}$, which it must satisfy that $\mathcal{G} \in GF/\mathbb{F}_{2^{n*k}}$, where (n, k) is the matrix's size [41], [43].

Next, Alice and Bob have to select a random invertible or non-singular matrix $\mathcal{S}$, where it must satisfy that $\mathcal{S} \in GF/\mathbb{F}_{2^{k*k}}$, where the matrix is (k, k) [41], [43].

Then, Alice and Bob have to select a random permutation matrix $\mathcal{P}$, where it must satisfy that $\mathcal{P} \in GF/\mathbb{F}_{2^{n*n}}$, where the matrix is (n, n) [41], [43].

Finally, Alice and Bob will compute the final matrix of (k, n) from the three matrix they selected earlier, where $\mathcal{G}' = \mathcal{G} * \mathcal{S} * \mathcal{P}$. Now, the public key of the users will be $PubK_{user} = \{\mathcal{G}', t\}$, and the private key of the users will be $PrivK_{user} = \{\mathcal{S}, \mathcal{P}, \mathcal{C}'\}$ [41], [43].

To encrypt messages, Alice would convert the message m, into binary string blocks with k-length. Next, Alice computes a vector $c' = m\mathcal{G}'$. Then, Alice generates a random n-bit vector e, where $e \in GF/\mathbb{F}_{2^n}$ and is a matrix of length n and weight t. Finally, Alice can compute the ciphertext $Ct = c' + e$. This equation could also be expanded to $Ct = m\mathcal{G}' + e$ [41], [43].

To decrypt messages; given that Bob also did the same key generation, Bob would compute the inverse of $\mathcal{P}$ and compute c', i.e., $\mathcal{P}^{-1}(Ct)$. This equation could also be expanded upon as $\mathcal{G} * \mathcal{S} * m + \mathcal{P}^{-1} * e = \mathcal{G}(\mathcal{S} * m) + e'$. Next, he would use the decoding algorithm $\mathcal{C}'$ to decode $c' \rightarrow m'$. Finally, Bob computes the message $m = m' * S^{-1}$ [41], [43].

McEliece cryptosystem's security relies on the computational hardness assumption of infeasibility of current and Quantum computational power to decode a general linear code [45]. Till date, there are no existing efficient and proved algorithms that employs the use of Quantum computer even; that can break McEliece cryptosystem. Below statement is the security requirement for McElliese cryptosystem:

- It is infeasible to compute the individual matrix from $\mathcal{G}'$, where $\mathcal{G}' \rightarrow \{\mathcal{G}, \mathcal{S}, \mathcal{P}\}$ and therefore, compute the code used for the generator matrix $\mathcal{C}$.

In 2008, Bernstein et al. published a practical attack on original implementation of the McEliece cryptosystem, where they have chosen one decoding algorithm $\mathcal{C}'$ with any linear code and going through each codeword of the code. This method is known as information set decoding [46].

There is one large obstacle in implementing McEliece cryptosystem in real-world applications. In McElliese initial proposal, the size of $\{n, k, t\}$ is $1024, 524$ and 50 respectively [43]. With these values, it will result in a public key size of $524 * (1024 - 524) \approx 262000$ bits or 32.5 Kilobytes (kB). According to Bernstein and Lange, the suggested value of $\{n, k, t\}$ is $2048, 1751$ and 27 respectively; in order for the cryptosystem have the same level of protection as 80-bits security when using standard algebraic decoding. These values will result in the public key size of around 65kB [41]. However, in cryptosystem's submission to the round 3 of the NIST Post-Quantum upcoming standardization, the suggested, lowest value for $\{n, k, t\}$ is at least $6688, 128$ and 13 respectively; that brings the public key size of around 104kB [47].

To compare with other asymmetric cryptosystem, RSA's public key size is at most 15360-bits or 1.9kB, while ECC's public key size is at most 512-bits or 0.06kB.

b) Multivariate-quadratic-equation-based Cryptosystem

In this thesis, Hidden Field Equations ("HFE") will be reviewed for Multivariate cryptosystem. But to give a few histories background on it, Matsumoto and Imai published the first multivariate-quadratic-equation asymmetric cipher and digital signature proposal named "C*" back in 1988 [41], [48]. The original C* implementation was cracked and proven to be insecure by Patarin in 1995 [41], [49]. However, the strength of MQE in cipher has not gone unnoticed; few other derivatives were inspired by the general principles from C*, that is, Hidden Field Equations ("HFE"). HFE was developed by Patarin in 1996 and continues to be the prominent MQE scheme till today [41], [44], [50].

Here, the mathematical preliminary of MQE in HFE is explained. MQE's cryptographic primitive is based on multivariate polynomials over a Galois or finite field $GF\ /\ \mathbb{F}$. The quadratic use in MQE is because polynomials with degree 2, is large and secure enough for use; increasing the degree of polynomials will significantly increase the key size and decrease efficiency. Now, given two extension field over a base finite field from $\mathbb{F}_q$; $\mathbb{F}_{q^m}$, $\mathbb{F}_{q^n}$; $q = 2$, a set of m-length polynomials in n variables can be mapped as a function where $\mathbb{F}_{q^n} \rightarrow \mathbb{F}_{q^m}$ [50], [51].

Assume there are a base finite field $\mathbb{F}_q$ and an extension field from the base field K. Also assume that there are a random element $u \in \mathbb{F}_{q^n}$, and h, where $0 < h < q^n\ s.t.\ h = q^\theta + 1$ for some θ and $\gcd(h,\ q^n - 1) = 1$. The gcd of h here is used with u for mapping where $u \rightarrow u^h$; one-to-one relationship, and the inverse of the map is $u \rightarrow u^{h'}$; where h' is the multiplicative inverse of $h\ mod\ q^n - 1$. Now, another element is defined where $w \in \mathbb{F}_{q^n}$ i.e. [50], [51]:

$$w = u^h = u^{q^\theta} u$$

Now, assume that $\beta_1, \dots, \beta_n$ is the basis of K defined earlier as a $\mathbb{F}_q$ in a vector space. u and w now will be represented as $u = (u_1, \dots, u_n), w = (w_1, \dots, w_n)$, with respect to the basis β_n. Also assume that $A^k = a_{ij}^k$ is the matrix of the mapping earlier $u \to u^{q^k}$, with respect to the $\beta_n$basis such that [50], [51]:

$$\beta_i^{q^k} = \sum_{j=1}^{n} \alpha_{ij}^k * \beta_j \; ; \; \alpha_{ij}^k \in \mathbb{F}_q \; ; 1 \le i, k \le n$$

Now, all products of the basis elements can be written i.e. [50], [51]:

$$\beta_i \, \beta_j = \sum_{l=1}^{n} m_{ijl} * \beta_j \;\; ; \;\; m_{ijl} \in \mathbb{F}_q \;\; ; \;\; 1 \le i, j \le n$$

This would be the central map $\mathcal{Q}$, where the base field is mapped to the extension field, $\mathbb{F}_q \to K$.

Next, two invertible (n, n) matrices are chosen for affine transformation of the central map $\mathcal{Q}$, noted with $\mathcal{S}, \; \mathcal{T}$; such that $M_{\mathcal{S}} = \{\mathcal{S}_{ij}\}, M_{\mathcal{T}} = \{\mathcal{T}_{ij}\}$. Two n-length vectors $v_{\mathcal{S}}, v_{\mathcal{T}}$ over $\mathbb{F}_q$ are also chosen. Now, x, y can be defined such that [50], [51]:

$$u = \; \mathcal{S}_x = M_{\mathcal{S}^x} + v_{\mathcal{S}} \;\; ; \;\; w = \; \mathcal{T}_y = M_{\mathcal{T}^y} + v_{\mathcal{T}}$$

By using above equation to replace u_j, w_i with x_k, y_l, the central map $\mathcal{Q}$ is linear in y_l and of degree of 2 in x_k. By applying linear algebra, it will return n explicit equations; for each y_l as polynomials of degree 2 in x_k.

In key generation, from the mathematical explanation earlier, the public key of the users $PubK_{user}$ will be the polynomials that have been transformed and appear random i.e.

$(p_1, \dots, p_n)$. These polynomials could also be represented as $\{Q * S * T\}$. The private key of the users $PrivK_{user}$ will be the inverse of the transformation that are done on the polynomials i.e. $\{Q', S', T'\}$.

To encrypt messages M, it must be first represented as a vector; assume that M would be $(x_1, \dots, x_n) \in \mathbb{F}_q^n$. Every x_n will be iterated with the polynomials, $(p_1, \dots p_n)$ from the $PubK_{user}$, such that $\big(p_1(x_1, \dots, x_n), p_2(x_1, \dots, x_n), \dots, p_n(x_1, \dots, x_n)\big) \in \mathbb{F}_q^n$. This would be our ciphertext Ct [50], [52].

To decrypt Ct, the inverse of the polynomial's transformation is applied to reverse the transformation of the polynomials. Ct will be applied with $\mathcal{S}$ first, denoting its result as x'. x' is then mapped as such $F_{q^n} \rightarrow F_{q^n}$ so that the private polynomials $\mathcal{Q}$ can be applied, denoting its results as y'. y' is then transferred to a vector as such $(y'_1, \dots, y'_n) \in \mathbb{F}_q^n$, and $\mathcal{T}$ will be applied to the vector earlier to get the final output $y \in \mathbb{F}_{q^n}$, from $(y_1, \dots, y_n) \in \mathbb{F}_q^n$ [50], [52]. It must be noted that in the decryption process, the most key step is to find the relationship between $x'$and y'as such $\mathcal{P}(x') = y'$.

Figure 2.19 below visualise the process of using $\{\mathcal{Q}, \mathcal{S}, \mathcal{T}\}$ in message encryption and decryption.

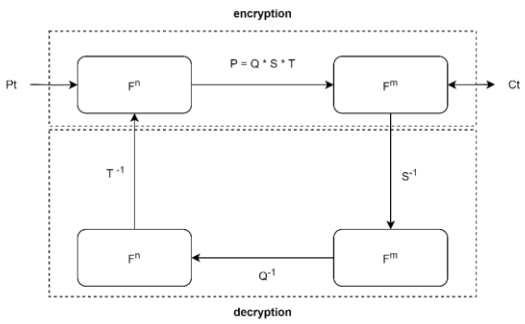


Figure 2. 19 Visualisation of relation between transformations in HFE

The encryption and decryption operation could also be re-written as such; this is what is called the MQ-trapdoor function [51]:

$$x = (x_1, \dots x_n) \ \rightarrow^{\mathcal{S}} \ x' \ \rightarrow^{\mathcal{Q}} \ y' \ \rightarrow^{\mathcal{T}} \ y$$

MQE cryptosystem's security relies on the computational hardness assumption of infeasibility of current and Quantum computational power to solve multivariate polynomials over a finite field. Till date, there are no existing efficient and proven algorithm that employs the use of Quantum computers; that can break MQE cryptosystem.

However, MQE also suffers from the same obstacle from code-based cryptosystem which is large public key size.

c) Lattice-based Cryptosystem

In this thesis, NTRU, specifically NTRUEncrypt and Kyber will be reviewed for Lattice-based cryptosystem, but Kyber will be covered in **Chapter 2.3.2**. But to give some historical background on lattice-based cryptography, in 1998, Hoffstein et. al. proposed a new public-key cryptography method employing the use of ring learning with error ("RLWE") lattice named NTRU [41], [44], [53]. NTRU was then divided into two different algorithms; named NTRUEncrypt and NTRUSign (The name describes itself on its objectives if it's not obvious enough). Essentially, NTRU is similar to that MQE's properties; where it also employs the use of polynomials, but it is based on shortest vector problem in a lattice. In 1997, Hoffstein et. al. patented NTRU [54].

Here, the mathematical preliminary of NTRU is explained. NTRU depends on three parameters $\{N, p, q\}$ and four sets of polynomials with degree $N-1$, $\{\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_\phi, \mathcal{L}_m\}$. Also note that N must be a prime, p, q must be co-prime, and it must adhere as such

$\gcd(p, q) = 1 \; ; q > p$. Assume there is a truncated polynomial ring $R = \mathbb{Z}\frac{[X]}{X^N - 1}$. Now, an element $F \in R$ can be written in polynomials; or vectors, as such [53], [55]:

$$F = \sum_{i=0}^{N-1} F_i x^i = [F_0, F_1, \dots, F_{N-1}]$$

The polynomials can be written as such:

$$\mathrm{a} = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1}$$

Figure 2.20 visualise the construction of RL-LWE.

$$\begin{pmatrix} a_{0,0} & -a_{n,0} & -a_{n-1,0} & \cdots & -a_{1,0} \\ a_{1,0} & a_{0,0} & -a_{n,0} & \cdots & \\ a_{2,0} & a_{1,0} & a_{0,0} & \cdots & \\ a_{3,0} & a_{2,0} & a_{1,0} & \cdots & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & & & & \\ a_{n,0} & & & \cdots & a_{0,0} \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ \\ \vdots \\ \\ s_n \end{pmatrix} + \begin{pmatrix} e_0 \\ e_1 \\ \\ \vdots \\ \\ e_n \end{pmatrix} = \begin{pmatrix} t_0 \\ t_1 \\ \\ \vdots \\ \\ t_n \end{pmatrix}$$

Figure 2. 20 Structure of RL-LWE

Another element G can be added, such that $G \in R$ so that multiplication with F can be performed, as such [53], [55]: (Note that $*$ here is multiplication in R, as a cyclic convolution product)

$$F * G = H \;\; s.t.$$

$$H_k = \sum_{i=0}^{k} F_i G_{k-i} + \sum_{i=k+1}^{N-1} F_i G_{N+k-i}$$

$$H_k = \sum_{i+j \equiv k \,(mod\ N)} F_i G_j$$

Figure 2.21 below visualise on how NTRU works.

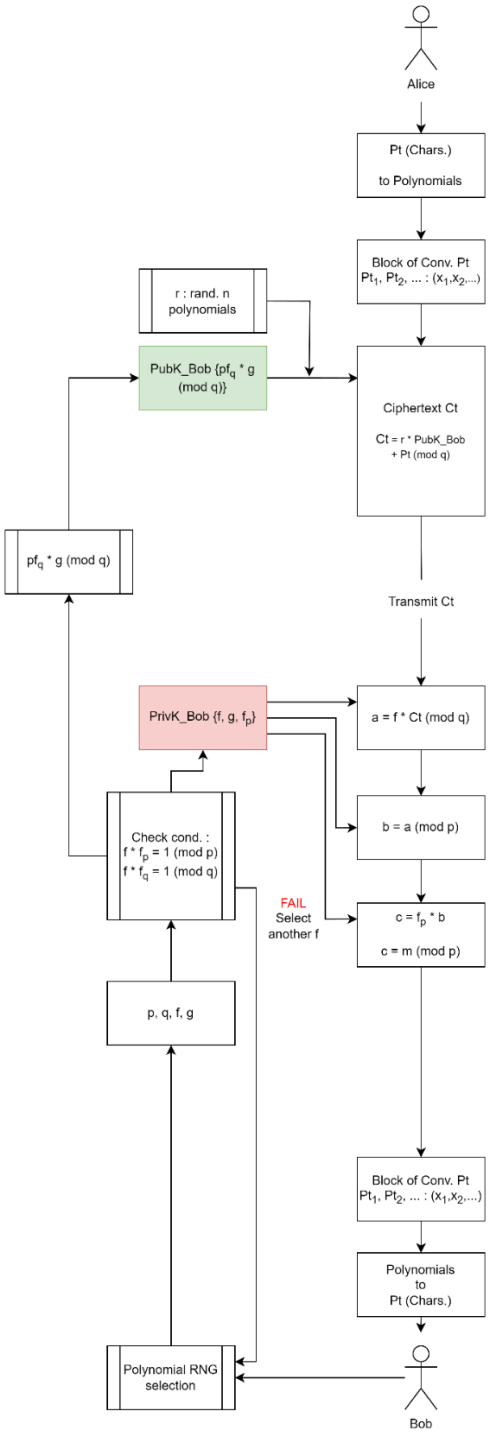


Figure 2. 21 High-level visualisation of NTRU

In key generation, first the user choose two polynomials; assume it's $f, g \in \mathcal{L}_g$ here with degree $N-1$ and with coefficient $\{-1, 0, 1\}$, and it must satisfy that $f \cdot f_p = 1\ (mod\ p)$ and $f \cdot f_q = 1\ (mod\ q)$. Essentially, f must be able to prove that the inverse modulo of p, q exists; using the Euclidean algorithm [53].

The public key of the user $PubK_{user}$ would be noted in the equation below [53], [56]:

$$PubK_{user} = pf_q \cdot g\ (mod\ q)$$

The private key of the user $PrivK_{user}$ would be $\{f, g, f_p\}$ [53], [56].

To encrypt messages, Alice converts the message M into polynomial form; assume it would be $M = (x_1, \dots, x_N)$. M would have the coefficient with respect to p such that $\left\{-\frac{p}{2}, \frac{p}{2}\right\}$. Next, Alice would have to choose another random polynomial, noted here $r = (x_1, \dots, x_N)$; to obscure the message. Assume that Bob has already performed his key generation, the equation for encrypting M is noted below [53], [56]:

$$Ct = r \cdot PubK_{Bob} + M\ (mod\ q)$$

To decrypt messages, Bob will multiply the Ct with his f; part of his private key; such that [53], [56]:

$$a = f \cdot Ct\ (mod\ q)$$

The equation above could be expanded as noted below [53], [56]:

$$a = f \cdot Ct\ (mod\ q)$$

$$a = f \cdot (r \cdot PubK_{Bob} + M) \ (mod\ q)$$

$$a = f \cdot \left(r \cdot pf_q \cdot g + M\right) (mod\ q)$$

$$a = pr \cdot g + f \cdot M \ (mod\ q)$$

The next step is to calculate $a \ mod\ p$, as such [53], [56]:

$$b = a \ (mod\ p) = f \cdot M \ (mod\ p) \leftarrow pr \cdot g \ (mod\ p) = 0$$

Finally, Bob can use his other part of $PrivK_{Bob}$, f_p to recover the message from the ciphertext as such [53], [56]:

$$c = f_p \cdot b = f_p \cdot f \cdot m \ (mod\ p) \leftarrow f \cdot f_p = 1 \ (mod\ p)$$

$$c = m \ (mod\ p)$$

Lattice-based cryptosystem's security is based on the computational hardness of infeasibility of current and Quantum computational power to factoring polynomials in a truncated polynomial ring into a quotient of two polynomials that have small coefficients. In order to solve this problem, it is also highly related to the algorithmic problem of lattice reduction named Closest / Shortest Vector Problem; although it is not equivalent. This is why NTRU belongs under lattice-based cryptography, although it actually does not employ the use of lattices.

According to Hoffstein et. al., NTRU's operation can be related back to McEliece and RSA cryptosystem. This is because in NTRU, the f, g multiplication in the ring R could also be viewed as multiplication of matrices in the key generation of McEliece cryptosystem. However, the real difference lies on the trapdoor function of both cryptosystem in decryption; where McEliece matrix are using an error-correcting (Goppa) code, and decryption is worked upon the number of "errors" in the code. For

NTRU, the matrix is circular, and decryption is worked upon decomposing of the matrix into a product of two matrices; from $mod\ q$ to $mod\ p$ [53].

Going back to post-quantum cryptosystem's disadvantage, which is large key size; **Table 2.9** shows the difference between key sizes of all the post-quantum cryptosystems, including non-post-quantum cryptosystems.

Table 2. 9 Comparison between key sizes of PQC and non-PQC ciphers

Cipher	Type	Public Key size (kB)	Private Key size (kB)
Goppa-based McEliece	Code-based	104	11.5
QUARTZ (HFE deriv.)	MQE	71	3
15360-bit Discreet Log	**Non-PQC**	1.92	0.04
NTRUEncrypt	Lattice	0.095	0.105
512-bit Elliptic Curve	**Non-PQC**	0.064	0.064

As at time of writing, NIST has already approved few lattice-based primitives in round 3, and with round 4, testing of code-based primitives are in the work [57].

To conclude on Post-Quantum cryptography, new hard problems for cryptographic primitives must be researched and assessed thoroughly in order for it to be secure in the future where Quantum computers are getting more accessible and feasible for operation.

2.2 Introduction to Digital Signature

Digital signature can be defined as an electronic "fingerprint" of the content of a message that is encrypted (in this case signed) by a key that is only on the hands of the signer (in this case a private key $PrivK_{user}$) [5], [13], [24], [27].

In asymmetric cryptography, the message sender will encrypt a message with a public key of the receiver, and the receiver will decrypt the message with his / her private key. This process can be done another way around; encrypting with a private key (in this case it is called signing), then decrypting with a public key (in this case it is called verifying). This inversed process is what's called a Digital Signature.

Figure 2.22 shows the digital signature process.

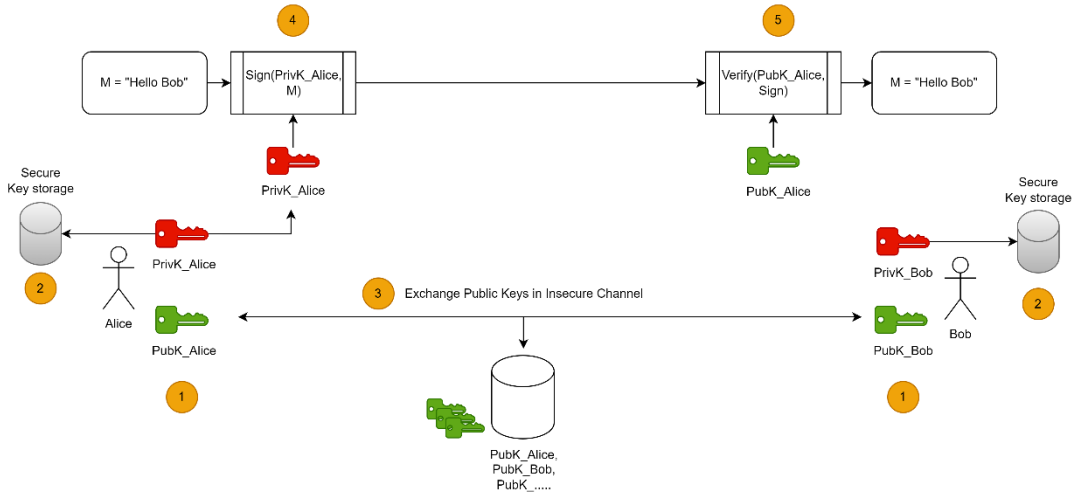


Figure 2. 22 Visualisation of Digital Signature

In **Figure 2.22**, Alice wants to send a message "hello bob" to Bob.

First, Alice would sign the message with her private key $PrivK_{Alice}$. Then she would send the ciphertext of the process, which is now called signature, to Bob. Finally, Bob would verify the message comes from Alice by using Alice's public key $PubK_{Alice}$.

Vitally, this operation adheres to the principle of asymmetric cryptography, where only the correct private key can decrypt the message encrypted with its corresponding public key. Inversely, only the correct public key can verify the message signed with its corresponding private key [5], [13], [24], [27].

However, there are some limitations with performing digital signatures with pure asymmetric cryptography primitives. As noted in RSA's limitations, asymmetric ciphers' operation is more taxing and if pure RSA were to be used for digital signature with the entirety of message content, the data that is needed to transmit are huge. This is because the signature is performed on the whole message. This method is infeasible if the message that to be signed is huge; there will be too much overhead. Apart from the space-complexity problem, there is also security concerns that a threat actor may be able to perform forgery on the signature while being valid [5], [13].

The solution for the limitations noted above is to integrate a hash function to produce a fixed-length fingerprint of the message's content. With this mechanism implemented before the signing operation, any arbitrary original message length can be used because hash function will always generate a fixed-length output. **Chapter 2.2.1** will explain in detail on hash function.

2.2.1 Introduction to cryptographic hash function

Hash function provides the protection for Integrity triad. Hash function can be defined as a function that maps a variable-length data as input into a fixed-length output [5], [13]. Hash function works the same way as encryption, but vitally, its process is not reversible. **Figure 2.22** below shows how hash functions operation differs from encryption operation.

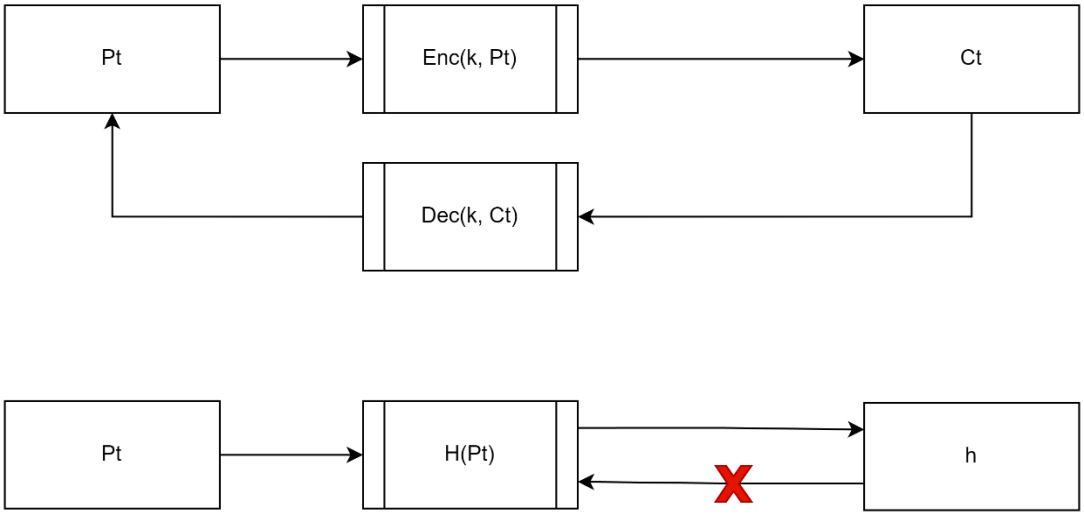


Figure 2. 23 Visualisation of difference between encryption and hashing

From **Figure 2.23**, the operation of a hash function is visualised. A non-fixed length input Pt, going into a hash function $H()$ to produce an output what is called a hash digest h. h is a fixed-length Base16 string, which its size depends on the variation of the hash function.

Below statements are the security requirements for a cryptographic hash function to be secure:

- **Variable Input size**, where a hash function $H()$ must be able to be applied to a block of data of any size, m [13].

- **Fixed Output / Digest size**, where a hash function $H()$ must be able to produce a fixed-length output or hash digest, h [13].

- **Efficiency**, where a hash function $H(m)$ must be easy to compute for any given m [13].

- **Pre – image resistant**, where given h, it's computationally infeasible to solve for m to satisfy that $h = H(m)$. This property is also known as one-wayness [13].

- **Second pre – image resistant**, where given h, m_1, it's computationally infeasible to solve for another distinct m_2 to satisfy that $h = H(m_1) = H(m_2)$. In other words, given an input into the hash function, it should be extremely hard to find another input that has an identical or similar hash digest. This property is called weak collision resistant [13].

- **Collision resistance**, where given h, it's computationally infeasible to solve for distinct m_1, m_2 to satisfy that $h = H(m_1) = H(m_2)$. In other words, it should be extremely hard to find two distinct inputs that will produce an identical or similar hash digest. This property is called string collision resistant [13].

In this thesis, two versions of SHA which is SHA-2 and SHA-3 is reviewed for cryptographic hash function.

Secure Hash Algorithm ("SHA") is a hash function developed by NIST in 1993 and revised in 1995 to be published as part of NIST's FIPS180-1 standard. SHA-1 was initially based on MD4 hash function design. However, in 2002 NIST re-iterated the hash function which become new standard FIPS180-2 that defined three new versions of SHA; called SHA-2 [58]. While previous SHA-1 only produces 160-bit digest, SHA-2 has a variety of hash digest-length which is 256, 284, and 512-bit. These variety are named after the hash digest-length; SHA-256, SHA-384 and SHA-512. To note, SHA-1 has been found to be insecure for usage after collision in the hash function was found, and NIST has formally deprecated the use of SHA-1 in 2017 and advising anyone who were using SHA-1 to urgently migrate to SHA-2 [59][60].

Figure 2.24 below shows the process for SHA-512.

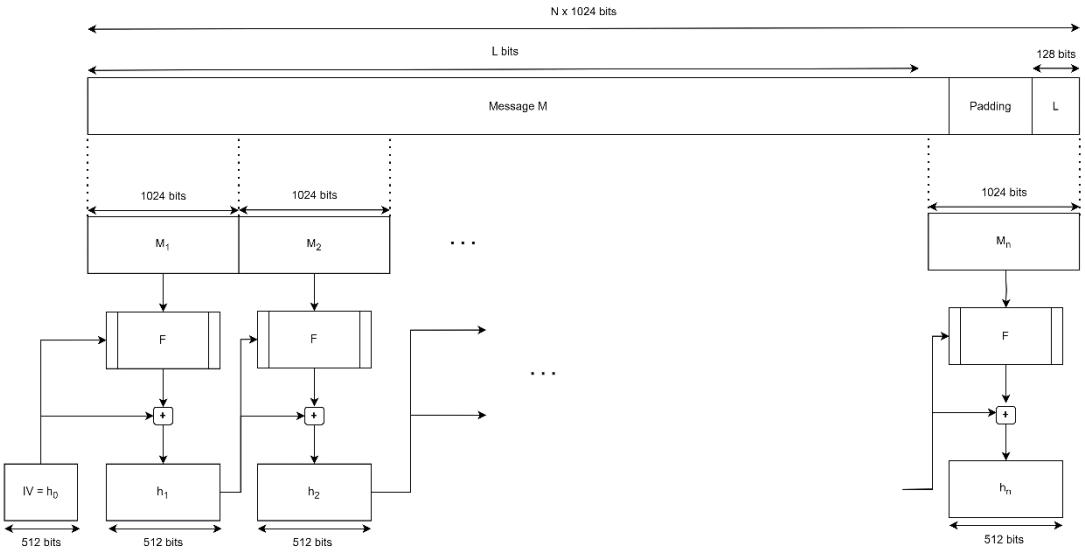


Figure 2. 24 Visualisation of SHA-512 process

First, the message will be padded with padding bits at the last block if the message length is not a multiple of 1024 bits. Second, the message separated into 1024-bit blocks to make sure that the block length is a multiple of 1024 bits. These message blocks are noted with $M_1, M_2, \dots, M_n; n = no.of\ blocks$. Next, the hash buffers are initialized and to be used for the round function, F. Then, the output from the round function is then performed modular addition, word-by-word, with an IV. After this operation, the output in the hash buffers h_1, which is now 512 bits long, will be used for the round function and addition on the next message block M_2. This process is repeated until it reaches the last message block M_n, where the final state of the hash buffer h_n will be taken as the hash digest h for the original message M [58].

Figure 2.25 below shows the round function of SHA-512 in detail.

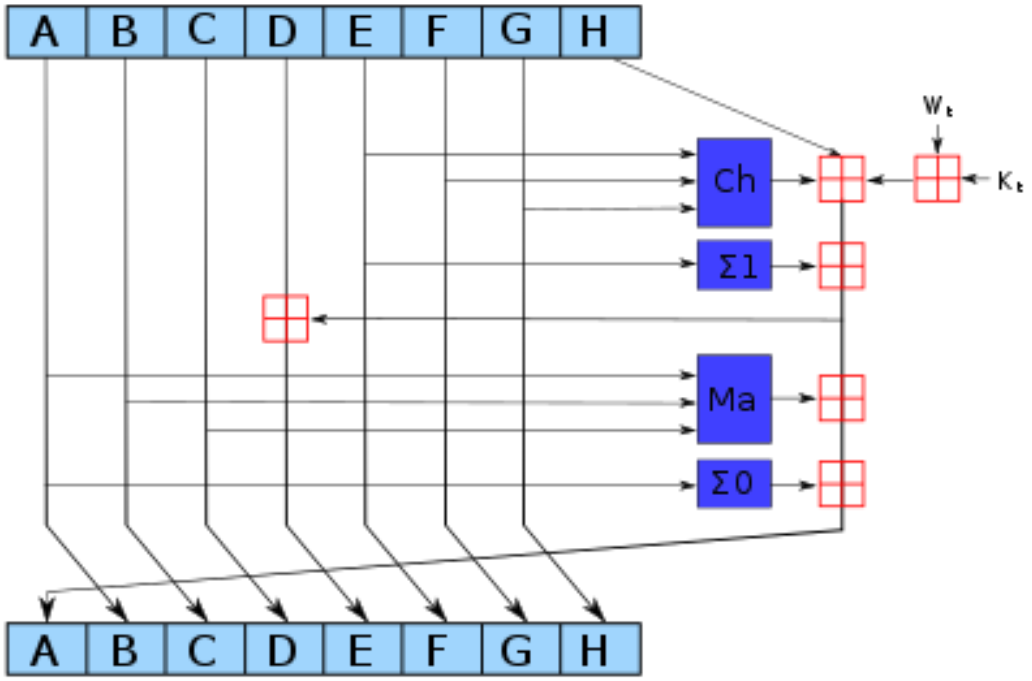


Figure 2. 25 Visualisation of SHA-512's round function [61]

In **Figure 2.25**, the round function for SHA-512 is visualised. (a, b, c, d, e, f, g, h) is the hash buffer used as input and output. The (a, b, c, d, e, f, g) buffer in input state will be transposed one position to the right to fill the (b, c, d, e, f, g, h) buffer in output state. Do note that the red + squares are modulo addition operation. There are a few non-linear functions that are in the round function which are noted below [58]:

$$Ch(E, F, G) = (E \hat{} F) \oplus (\neg E \hat{} G)$$

$$Ma(A, B, C) = (A \hat{} B) \oplus (A \hat{} C) \oplus (B \hat{} C)$$

$$\sum_{0}(A) = (A \ggg 28) \oplus (A \ggg 34) \oplus (A \ggg 39)$$

$$\sum_{1}(A) = (E \ggg 14) \oplus (E \ggg 18) \oplus (E \ggg 25)$$

There are a few more variables for the round function where W_t is the input message block with 64-bit long, and K_t is the round constant. These two are first performed

modulo addition and will be modulo addition again with output of H counter and $Ch()$ [58].

However, NIST did not only stop at SHA-2. To ensure the long-term security and improved efficiency on current implementation of hash functions, NIST has decided to review and select a new hash algorithm standard, SHA-3 which is specified in FIPS202 [62]. SHA-3 was released by NIST in 2015. SHA-3 design was not based on MD4 or MD5 design, but a different and new algorithm named KECCAK. KECCAK algorithm was designed and developed by Bertoni G. et. al. and was based upon an earlier hash functions PANAMA and RadioGatún [63]. The simplicity and security provided by the algorithm makes it the best candidate for hash function to use in current time and in future.

Like SHA-2, SHA-3 also have four variations which are SHA3-224, SHA3-256, SHA3-384, and SHA3-512. Also, all hash digest length depends on the hash function's variation [62].

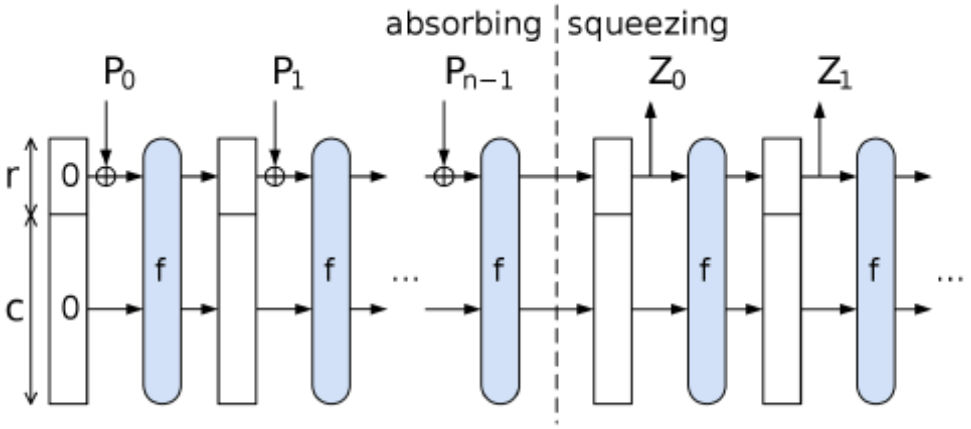


Figure 2. 26 Visualisation of SHA-3's KECCAK Algorithm [64]

In **Figure 2.26**, the KECCAK algorithm is visualised. The algorithm's structure is quite different from the round function used in SHA-2. SHA-3 also do not use modular addition, instead using XOR.

To compare between SHA-1 to SHA-2 and SHA-3, SHA-2 and SHA-3 is not found to be vulnerable for collision attack yet. SHA-2 and SHA-3 have a variety of hash digest length available, unlike SHA-1 which is only 160-bits. **Table 2.10** shows the comparison between these three hash functions.

Table 2. 10 Comparison between hash functions

Hash Function / Criteria		Block size (bits)	Rnds.	Operations	Hash digest size (bits)
SHA-1		512	80	AND, XOR, OR, ROT, Add (mod 2^{32})	160
SHA-2	**SHA-224**	512	64	AND, XOR, OR, ROT, Add (mod 2^{32})	224
	SHA-256	512	64		256
	SHA-384	1024	80	AND, XOR, OR, ROT, Add (mod 2^{64})	384
	SHA-512	1024	80		512
SHA-3	**SHA3-224**	1152	24	AND, XOR, ROT, NOT	224
	SHA3-256	1088	24		256
	SHA3-384	832	24		384
	SHA3-512	576	24		512

Figure 2.27 visualises the process of digital signatures with hash function integration.

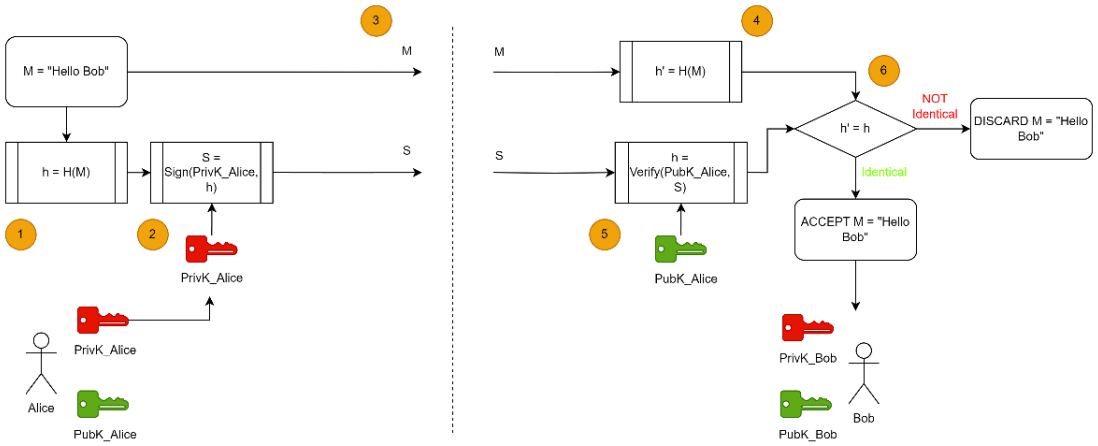


Figure 2. 27 Visualisation of Digital Signature w/ Hash Function

First for the sending side (Alice), the message m is inputted into a hash function $H(m)$ to produce a hash digest h. Then, h will be signed with Alice's private key $PrivK_{Alice}$, to produce a signature S. Next, the message concatenated together with the signature, (M, S) is sent to Bob [5], [13].

For the receiving side (Bob), the received message is inputted into the same hash function used in the signing process to produce another hash digest h'. Next, the signature received from Alice is then verified using Alice's public key $PubK_{Alice}$, to acquire the hash digest calculated by Alice, h. Finally, h' and h is compared; to satisfy that $h' \equiv h$. If h' is identical to h ; $h' \equiv h$, then Bob can be assured that the message is verified to be coming from Alice. If h' is not identical to h ; $h' \not\equiv h$, then Bob can conclude that the message is not coming from Alice, or tampered in-transit, and can safely discard the message [5], [13].

It is important to note that together with the implementation of hash function, the security or susceptibility of digital signatures schemes will depend on the strength of the hash function. In FIPS-180-4, SHA-2 is considered to be secure to be used as the hash function for digital signature schemes [33].

2.2.2 Conventional digital signature

In this thesis, two digital signatures schemes that are RSA Digital Signature Scheme and Elliptic Curve or Edwards Curve Digital Signature Scheme are reviewed for conventional digital signatures.

a) RSA Digital Signature Scheme

RSA Digital Signature scheme employs the use of RSA's asymmetric cryptographic primitive in order to verify the authenticity of messages. **Figure 2.28** visualise on how RSA is implemented in the scheme.

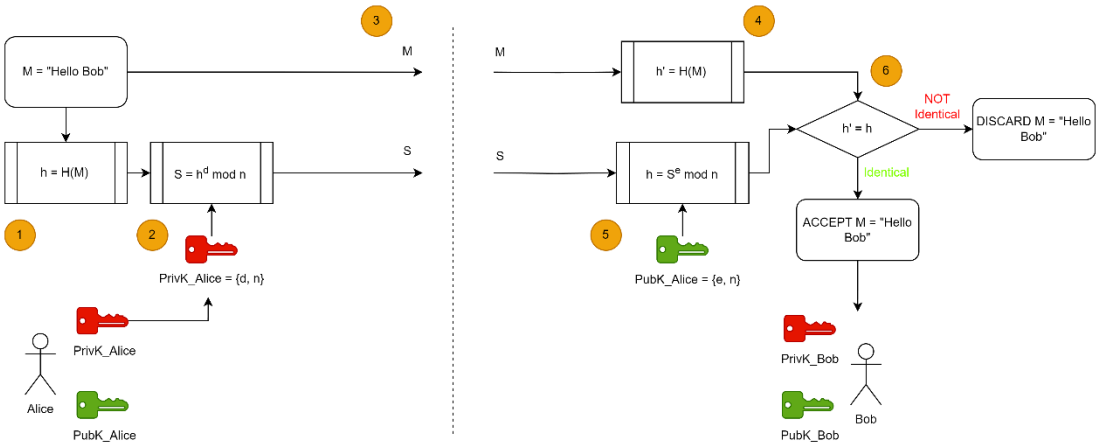


Figure 2. 28 Visualisation of RSA Digital Signature Scheme

Note that this example is with hash function implemented on the plaintext itself, in the digital signature process. First, the message is hashed as such $h = H(M)$. Then, h is signed using $PrivK_{Alice}$ with $S = h^d \ mod \ n$. Message and the signature $\{M, S\}$ is then sent to Bob [5], [13].

When Bob receives the message and signature pair, he first hash the incoming message to get a separate hash digest as such $h' = H(M)$. Then, he will retrieve back the original hash digest from Alice's signature as such $h = S^e \ mod \ n$. These two hash digests are

then compared; if it's identical then Bob can receive the message safely knowing that it's not tampered in-transit. Else, Bob can discard the message; there's a possibility that the message was tampered in-transit [5], [13].

b) Elliptic Curve / Edwards Curve Digital Signature Algorithm

Elliptic Curve Digital Signature Algorithm ("ECDSA") and Edwards Curve Digital Signature Algorithm ("EdDSA") employs the use of elliptic curve with the Digital Signature Algorithm ("DSA"). As the name suggests, ECDSA uses one among three elliptic curves that have been reviewed in **Chapter 2.1.2**; which is either Prime, Binary or Koblitz curve. EdDSA specifically uses the Edwards curve for the operation; refer to **Figure 2.15** [33], [65], [66]. **Figure 2.29** visualises on how ECC is implemented in the scheme.

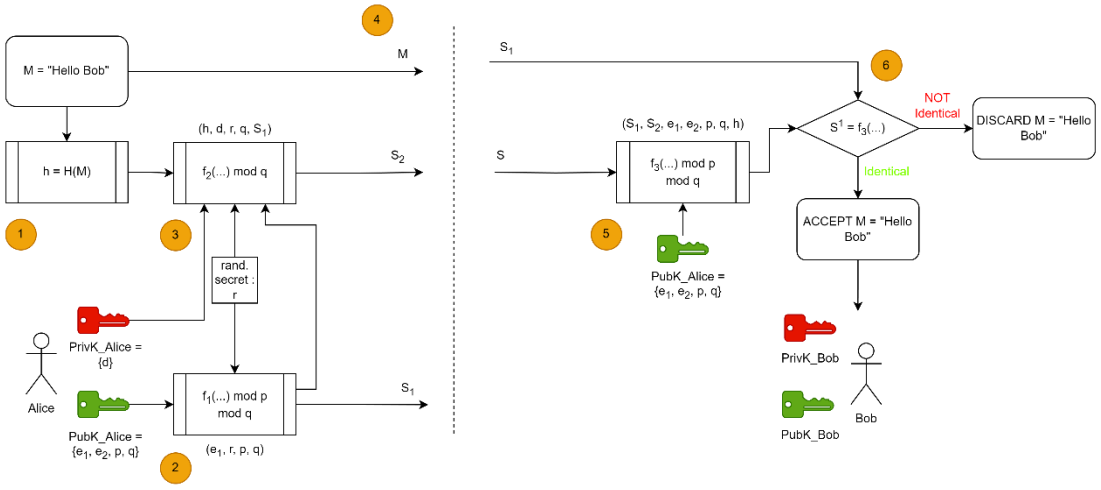


Figure 2. 29 Visualisation of ECDSA / EdDSA

First, the message is hashed as such $h = H(M)$. Next, a secure random number is selected, denoted with r. r, together with part of Alice's public key e_1, p, q will be used for calculation to produce the first part of signature S_1, as such $S_1 = (e_1, r, p, q) \rightarrow f_1(\dots)\ mod\ p\ mod\ q$. S_1 and Alice's private key are then used to calculate the second part of the signature S_2; as such $S_2 = (h, d, r, q, S_1) \rightarrow f_2(\dots)\ mod\ q$. Message and the signatures $\{M, S_1, S_2\}$ is then sent to Bob [33], [65].

When Bob receives the message and the signatures, all he has to do is combining both the signatures and Alice's public key to perform verification; as such $(S_1, S_2, e_1, e_2, p, q, h) \rightarrow f_3(\dots)\ mod\ p\ mod\ q$. Finally, the output from the operation earlier is compared with S_1. If it's identical, Bob can accept the message; if it's not identical, Bob can discard the message [33], [65].

2.2.3 Post-Quantum digital signature

In this thesis, one digital signature scheme that is CRYSTALS-Dilithium will be reviewed for Post-Quantum Digital Signature.

a) CRYSTALS-Dilithium Digital Signature Scheme

Dilithium was developed by CRYSTALS team – Bai et.al. in 2017. It had been in the NIST's Post-Quantum Cryptography Standard round submission and has passed to become the finalist of round 3 and is set to become the prominent post-quantum digital signature scheme in the future [67]. Dilithium is based on Module-Lattice ("ML"), Learning with Errors ("LWE") and Short Integer Solution ("SIS"). After a few iterations, Dilithium uses Fiat-Shamir with aborts to optimize and decrease its public key size [68], [69], [70].

Dilithium has three different variations for various security levels set by the NIST, named Dilithium2, Dilithium3 and Dilithium5. Since NIST require Dilithium to implement AES instead of SHAKE (SHA-3 variant) in the key generation, there are also additional three variants using AES; named Dilithium2-AES, Dilithium3-AES, and Dilithium5-AES [67], [68].

Starting with Dilithium key generation first, a (k, l) matrix denoted A is generated, with respect to a polynomial ring R_q as such:

$$A \leftarrow R_q^{k*l} = \frac{\mathbb{Z}_q[X]}{X^N - 1} \;;\; q = 2^{23} - 2^{13} + 1, n = 256$$

Next, two random short, secret vectors are selected, denoted s_1, s_2; such that $(s_1, s_2) \leftarrow S_n^l \cdot S_n^k$.

Finally, assume a variable t to calculate the values of multiplication of A and s_1, s_2 as such $t = A \cdot s_1 + s_2$. At this point, the $PubK_{user}$ will be $\{A, t\}$ and the $PrivK_{user}$ will be $\{A, t, s_1 s_2\}$ [68]. Do note that the explanation in this thesis is merely a template for Dilithium; over-simplified.

Figure 2.30 below visualises how Dilithium is implemented in the digital signature process.

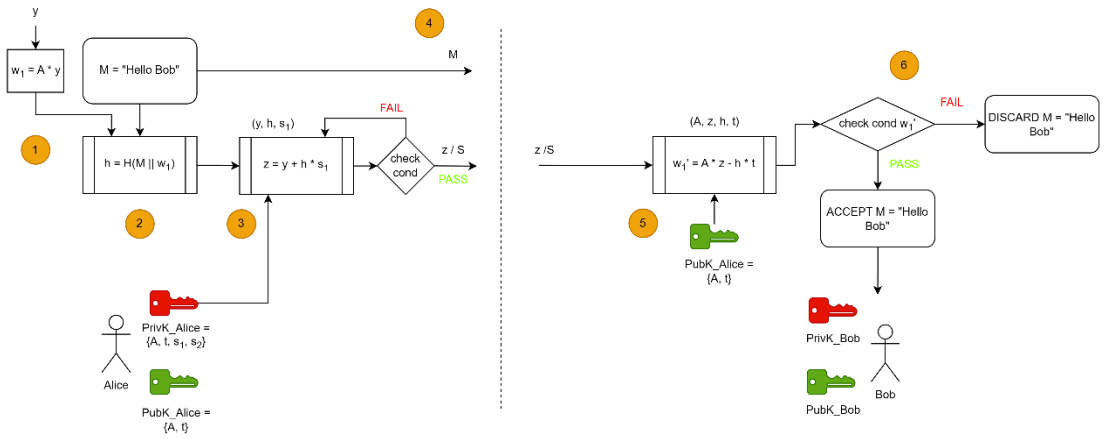


Figure 2. 30 Visualisation of CRYSTALS-Dilithium

To sign messages, Alice first generates a masking vector of polynomials y with coefficients $< \gamma_1$; as such $y \leftarrow S_{\gamma_1 - 1}^l$. Alice then computes the multiplication of y and A, as such $A \cdot y$; then sets the "high-order" bits of the coefficients in the vector, denoted with w_1. It could also be canonically expressed as $w = w_1 \cdot 2_{\gamma_2} + w_0 \;;\; |w_0| \leq \gamma_2$. Next, the message M together with w_1 is inputted into a hash function to produce a hash

digest as such $h = H(M \,||\, w_1)$. Finally, the potential signature is computed as such $z = y + h \cdot s_1$ [68], [70].

However at this point, Z is an insecure signature since the secret key s_1 could be leaked. To mitigate this issue, the Z operation will be checked against few conditions. The conditions are to check whether if coefficient of $z > \gamma_1 - \beta$ or any coefficient of the "low-order" bits of $A \cdot z - h \cdot t > \gamma_2 - \beta$. If one of the conditions are true, then z are re-computed again, whole conditions' statements are in a loop; else if both are false, z will become the final signature [68], [70].

To verify messages, Bob will compute $w_1{}'$ to be the "high-order" bits of $A \cdot z - h \cdot t$. Finally, Bob will accept the message from Alice if it meets the earlier conditions; if coefficients of $z < \gamma_1 - \beta$ and h is hash digest of $M, w_1{}'$. The verification process can also be expressed as such: (Assume $HighBits()$ is the "high-order" bits of variables) [68], [70]:

$$HighBits(A \cdot y,\ 2_{\gamma_2}) = HighBits(A \cdot y - h \cdot s_2\ , 2_{\gamma_2})$$

Going back to PQC's disadvantage of large key sizes, **Table 2.11** shows the comparison between non-PQC digital signature schemes and PQC digital signature schemes; and also the size of signatures after operation.

Table 2. 11 Comparison between non-PQC and PQC DSA

Dig. Sign. Scheme	Type	Public key size (kB)	Signature size (kB)
CRYSTALS-Dilithium	Lattice	2.592	4.595
512-bit ECDSA	**non-PQC**	0.064	0.0065
15360-bit RSADSS	**non-PQC**	1.92	0.0096

To conclude on digital signatures and hash functions, both of these methodologies are combined together to provide protection of integrity and authenticity of messages; sent and received by entities.

2.3 Introduction to Key Exchange Protocol

As in asymmetric cryptography, there are two keys that are used; that are public and private keys. These two keys can be "mixed" to generate a new key. Now, the "mixed" key become a shared secret key, and this shared secret key is equal on both parties. This shared secret key can be used for subsequent symmetric encryption and decryption. Remember the RSA's limitation and application on sending messages on pure RSA; RSA is just used to encrypt secret key used for symmetric encryptions or be a part of a key exchange protocol.

Figure 2.31 visualises the process of key exchange protocol.

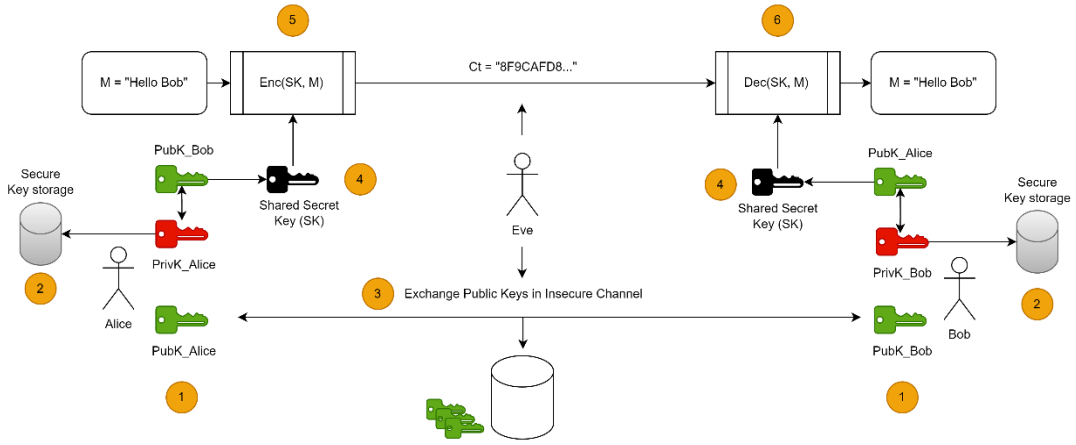


Figure 2. 31 Visualisation of Key Exchange Protocol

Assuming Alice and Bob has already performed key generation, $Alice = \{PubK_{Alice}, PrivK_{Alice}\}$, $Bob = \{PubK_{Bob}, PrivK_{Bob}\}$, both parties will fetch each other's public key. They then calculate the value for the secret shared key $SK = PubK_{user} * PrivK_{user}$ (multiplication here is just oversimplified in this visualisation). This SK value should be identical for all parties that calculates it. Finally, Alice and Bob can use this SK to encrypt their messages using symmetric cipher.

2.3.1 Conventional Key Exchange Protocol

In this thesis, two protocols that are Diffie-Hellman Key Exchange Protocol, and Extended Triple Diffie-Hellman Key Exchange Protocol are reviewed for conventional key exchange protocols.

a) Diffie-Hellman Key Exchange Protocol

The history of Diffie-Hellman Key Exchange Protocol ("DH KEP") can be traced back to RSA. In fact, Diffie-Hellman KEP was the foundation for RSA's asymmetric cipher operation [27]. DH was originally conceptualized by Ralph Merkle and worked upon Whitfield Diffie and Martin Hellman in 1976. It was filed for patent in 1977 [71].

Even though the popularized protocol name is Diffie-Hellman, the name did not include Ralph Merkle as the inventor of the protocol. Martin Hellman suggested the protocol name to be changed to Diffie-Hellman-Merkle Key Exchange Protocol to give credit to Merkle where it's due on the development of public-key cryptography, in his quote [27]:

The system [in this case DH KEP]*...has since become known as Diffie–Hellman key exchange. While that system was first described in a paper by Diffie and me, it is a public key distribution system, a concept developed by Merkle, and hence should be called 'Diffie–Hellman–Merkle key exchange'* [in this case it becomes DHM KEP] *if names are to be associated with it. I hope this small pulpit might help in that endeavour to recognize Merkle's equal contribution to the invention of public key cryptography.*"

Formalized in RFC 7919 from its implementation [72], [73], DH uses two numbers from the finite, Galois Field $\mathbb{F}$, which are defined as p where p is a prime number, and g, which is the primitive root of p. Note that p, g are values that are publicly shared and agreed upon by both parties.

Next, each party chooses a secret number which is not related to p, g (it can be any number regardless of prime or not prime) $a, b \in \mathbb{F}$, and use below equation to compute their respective public key. (Note A is Alice's public key, B is Bob's public key)

$$A = g^a \ mod \ p$$

$$B = g^b \ mod \ p$$

Then, each party will calculate the shared secret key value, by using the public key exchanged, $\{A, B\}$ with their secret number value, $\{a, b\}$. The equation for this process is noted below.

$$s = B^a \ mod \ p$$

$$s = A^b \ mod \ p$$

Now, both Alice and Bob's s value will be equal on both sides. The equation to prove the statement earlier is noted below.

$$A^b \ mod \ p = g^{ab} \ mod \ p = g^{ba} \ mod \ p = B^a \ mod \ p$$

$$(g^a \ mod \ p)^b \ mod \ p = (g^b \ mod \ p)^a \ mod \ p$$

Now, the equation earlier is proved by substituting values into it. The value $p = 7$ and $g = 3$ are used; which satisfies that p is a prime, and g is a primitive root of p.

$$p, g = 7, 3 \ \leftarrow p \ is \ prime, g \ is \ primitive \ root \ p$$

$$a = 5 \ \leftarrow a < p$$
$$b = 6 \ \leftarrow b < p$$

$$A = g^a \ mod \ p$$
$$= 3^5 \ mod \ 7$$

$$= 5$$

$$B = g^b \ mod \ p$$
$$= 3^6 \ mod \ 7$$
$$= 1$$

$$s = B^a \ mod \ p$$
$$s = 1^5 \ mod \ 7$$
$$= 1$$

$$s = A^b \ mod \ p$$
$$s = 5^6 \ mod \ 7$$
$$= 1$$

$$\therefore s = A^b \ mod \ p = B^a \ mod \ p \ \blacksquare$$

Only values of the secret number are made secret which is $\{a, b\}$. Everything else, $\{p, g, A, B\}$ (after calculation) can be sent in an insecure channel.

Note that in the proof here, p, g is a small value for example. In real-world implementations, p value must be much larger, at least 600 digits. However, g value does not need to be as large as p since it's a base for the exponentiation. Usually g value in practice, is a small integer like $2, 3, ...$

b) Extended Triple Diffie-Hellman Key Exchange Protocol

The original DH KEP algorithm has some flaws regarding entity authentication; mainly how do two entities know that only they can compute the secret key and no one else? This is also known as authenticated key agreement ("AK") problem. New methodology were proposed to mitigate the issue that is to have explicit authentication during the KEP process; named authenticated key agreement with key confirmation ("AKC") [74].

The idea of extending the DH is to perform few separate KEPs for different properties i.e. one KEP for long-term identity and one KEP for one-time use; and combining all of the keys into one final KEP to establish a more secure and authenticated agreement such

that the final KEP was tied to both of entities' identities. More KEPs can be established before the final KEP is performed to achieve security objectives i.e. forward secrecy.

Here, the Extended Triple Diffie-Hellman Key Exchange Protocol ("X3DH KEP") is reviewed. The first iteration of Triple Diffie-Hellman ("3DH") was first proposed by Blake-Wilson et. al. in 1997; hence the AK and AKC idea was proposed in their work [74]. 3DH was then improved upon by Kudla and Paterson in 2005 [75].

3DH is then expanded upon to become few other protocols e.g. X3DH, Double Ratchet Algorithm and Signal Protocol.

Figure 2.32 visualises on how X3DH works. In this example, Signal's implementation of X3DH is reviewed.

First, assume that either one ECC's *curve25519* or *curve448* is used, for the ECDH KEP. Also assume that SHA-256 or SHA-512 is used to hash the keys, for generating signatures of key using XEdDSA (XEdDSA is derivative of EdDSA). Assume Key Derivation Function ("KDF") is 32-bytes output from a HKDF function s.t. [76]:

$$KDF(KM) = KDF(F||KM_{HKDF}, salt_{HKDF}, info_{HKDF})$$

where KM is secret key material, $F = 32$ 0xFF bytes sequence if *curve25519* or 57 0xFF bytes sequence if *curve448* , $salt = n$-length 0x00 bytes with $n =$ length of hash digest, $info =$ a string identifying the application e.g. "Protocol". Also assume that *EncodeEC()* function is to encode a *curve25519* or *curve448* public keys into a byte sequence [77].

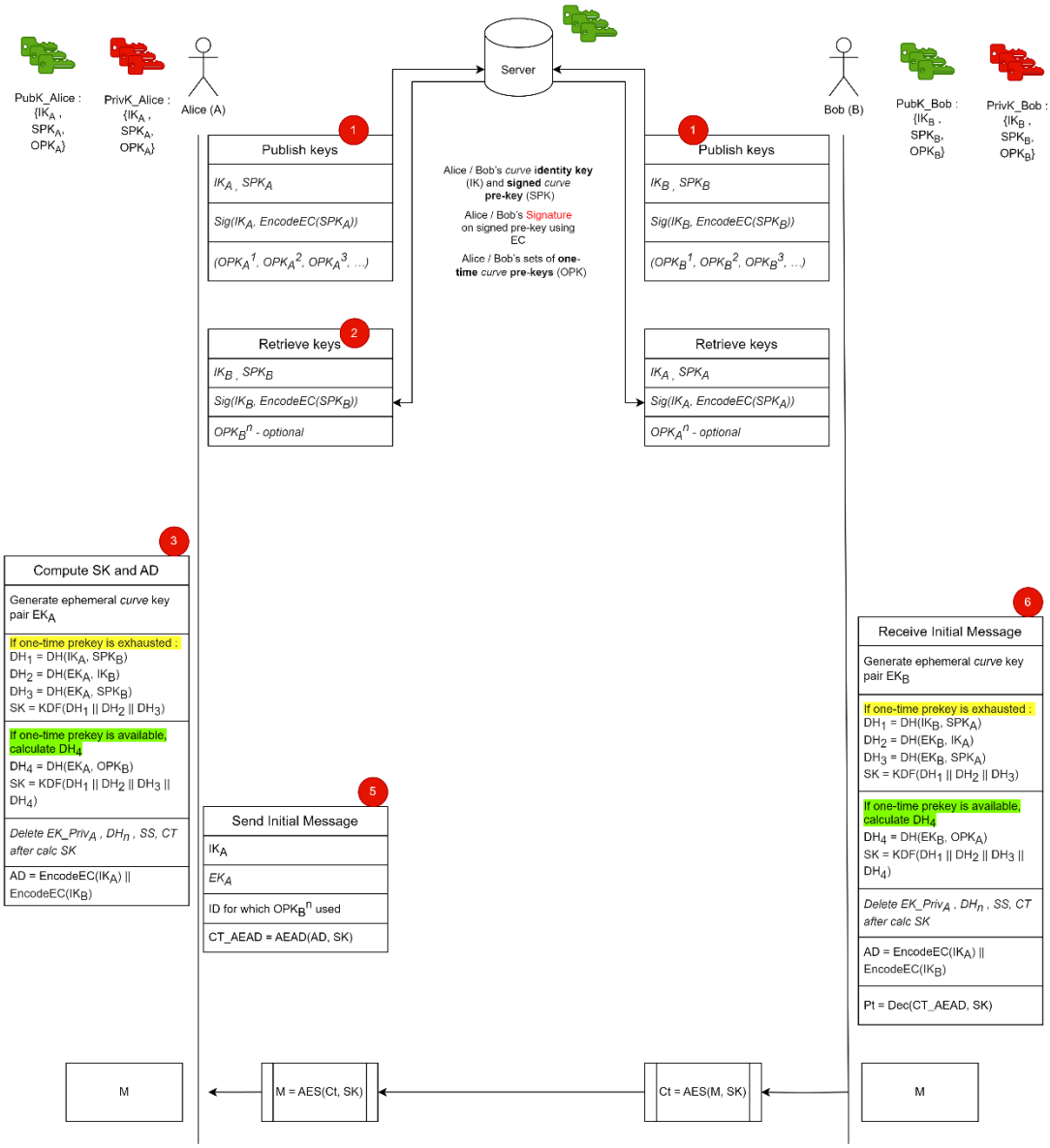


Figure 2. 32 Visualisation of X3DH

First, both parties, Alice and Bob generate and publish a set of keys and their signature. Identity Key, which is the long-term key for identification IK_{user}, periodic signed pre-key SPK_{user} and a set of one-time pre-keys $\{OPK_{user^1}, \dots, OPK_{user^n}\}$. A signature for the signed pre-key is also generated and published as such $Sig\big(IK_{user}, Encode(SPK_{user})\big)$. Signed pre-keys are changed periodically e.g. once a month, while one-time pre-keys are used only once in a run of the X3DH protocol; and will be replaced and replenished by users if its quantity is low [77].

Next, Alice will be retrieving Bob's set of pre-keys; named "prekey bundle", $\{IK_{Bob}, SPK_{Bob}, Sig(IK_{Bob}, Encode(SPK_{Bob})), OPK_{Bob^n}\}$. OPK_{Bob^n} is optional; if the server have it in storage i.e. Bob has it generated and published beforehand, then Alice will fetch the key, and the server will delete the one-time key; else it will just fetch the first three elements only [77].

Alice will be verifying the $Sig()$ of the SPK_{Bob}; if verification is successful, the protocol is a go; else Alice will abort the protocol. Then if the protocol proceeds, Alice will generate a *curve* ephemeral key pair EK_{Alice} and compute the secret key SK following:

$$DH_1 = DH(IK_{Alice}, SPK_{Bob})$$
$$DH_2 = DH(EK_{Alice}, IK_{Bob})$$
$$DH_3 = DH(EK_{Alice}, SPK_{Bob})$$

$$SK = KDF(DH_1 \,||\, DH_2 \,||\, DH_3)$$

The equations for SK above is applied to the condition where Alice do not have Bob's OPK_{Bob^n}. If Alice do have Bob's OPK_{Bob^n}, the equation for SK is noted below:

$$DH_1 = DH(IK_{Alice}, SPK_{Bob})$$
$$DH_2 = DH(EK_{Alice}, IK_{Bob})$$
$$DH_3 = DH(EK_{Alice}, SPK_{Bob})$$
$$DH_4 = DH(EK_{Alice}, OPK_{Bob})$$

$$SK = KDF(DH_1 \,||\, DH_2 \,||\, DH_3 \,||\, DH_4)$$

In essence, DH_1, DH_2 provides mutual authentication while DH_3, DH_4 provides forward secrecy. Alice will then delete her ephemeral's private key and all of DH_n values [77]. Next, Alice will compute an "associated data" byte sequence noted AD that is derived from the identity keys of both parties; s.t. : $AD = Encode(IK_{Alice}) \,||\, Encode(IK_{Bob})$. AD might also contain additional information e.g. certificates, usernames etc [77].

Alice will encrypt an initial ciphertext Ct with an AEAD scheme, with AD as associated data input and SK as its key [78]. In essence, the SK and AD will be used for the receiving side for authentication through encryption. With Ct, Alice will send an initial message together with her identity key IK_{Alice}, ephemeral key EK_{Alice}, and some sort of identifier on which one of Bob's one-time pre-keys $n \leftarrow OPK_{Bob^n}$ are used by Alice during the protocol's run [77].

Upon receiving the message, Bob retrieve Alice's IK_{Alice}, EK_{Alice} from the message and prepare to use his private key pairs of IK_{Bob}, SPK_{Bob} and OPK_{Bob^n} (if used by Alice). Bob will repeat the DH and KDF computation earlier using his retrieved key bundle to derive the SK. Bob will also compute his own AD as earlier.

Finally, he decrypt the initial ciphertext Ct using his derived SK and AD. If the decryption works, the protocol has complete, and SK can be used for subsequent encryption of message i.e. symmetric encryption after the X3DH protocol; else the protocol aborts and SK is deleted.

2.3.2 Post-Quantum Key Exchange Protocol

Primitives used in X3DH mainly ECC will be vulnerable to attacks by Quantum computer. Post-Quantum Key Exchange Protocol in the meantime, integrating current post-quantum primitives into X3DH. This will be called Hybrid Post-Quantum Cryptography since both conventional asymmetric and post-quantum cryptosystem are used in order to perform the key agreement. In this thesis, two key exchange protocols that are CRYSTALS-Kyber and Post-Quantum Extended Diffie-Hellman are reviewed for Post-Quantum Key Exchange Protocol.

In the context of this project, PQ KEP is used to exchange a symmetric key for subsequent message encryption using symmetric cipher.

a) CRYSTALS-Kyber Key Encapsulation Mechanism

Kyber was developed by CRYSTALS team – Avanzi et. al. in 2017 where it's first submitted to NIST and presented in IEEE's *EuroS&P* in 2018 [79]. After few iterations to improve upon Kyber's design, Kyber was chosen for the winner of round 3 NIST's Post-Quantum Cryptography Standardization [57]. Kyber is also set to become the standard for lattice-based cryptography in future as set in FIPS203 [80].

Kyber is a key encapsulation mechanism ("KEM") where it's proven secure from Indistinguishability under chosen or adaptive chosen ciphertext attack ("IND-CCA1/2"). Kyber's security is based on Module Lattices ("ML") with Learning with Errors ("LWE"). Kyber has three different variants, named Kyber-512, Kyber-768 and Kyber-1024. All three variants correspond to NIST's security level 1, 3 and 5. Like its sibling Dilithium, Kyber has another variants that use AES and SHA-2 instead of SHAKE and SHA-3 in all operations; named "90s" – Kyber-512-90s, Kyber-768-90s and Kyber-1024-90s [81]. **Figure 2.33** below shows the construction of ML-LWE.

$$\begin{pmatrix} A_{0,0}(X) & \cdots & A_{0,k}(X) \\ \vdots & \ddots & \vdots \\ A_{k,0}(X) & \cdots & A_{k,k}(X) \end{pmatrix} \cdot \begin{pmatrix} s_0(X) \\ \vdots \\ s_k(X) \end{pmatrix} + \begin{pmatrix} e_0(X) \\ \vdots \\ e_k(X) \end{pmatrix} = \begin{pmatrix} t_0(X) \\ \vdots \\ t_k(X) \end{pmatrix}$$

Figure 2. 33 Structure of ML-LWE

Kyber's construction consists of two main function, *KYBER.CPAPKE*, and *KYBER.CCAKEM*. *KYBER.CPAPKE* is an Indistinguishability under Chosen Plaintext Attack Public Key Encryption ("IND-CPA PKE") to encrypt messages of a fixed length; 32-bytes. *KYBER.CCAKEM* is the second function that will take outputs from *KYBER.CPAPKE* earlier to perform Fujisaki–Okamoto transform on the ciphertext and construct a secure IND-CCA1/2 KEM [81], [82].

Here, the mathematical preliminary of Kyber is explained. Assume R, R_q is a polynomial ring s.t.:

$$R = \mathbb{Z}[X]/(X^n + 1)\ ,\ R_q = (\mathbb{Z}_q\,[X])/(X^n + 1).$$

Assume a matrix is $\boldsymbol{A}$ and a vector with coefficient is $\boldsymbol{v}$, with $\boldsymbol{v^T}$(or $\boldsymbol{A^T}$) noted as transpose of $\boldsymbol{v}$ (or $\boldsymbol{A}$). Also assume that three variables used for the polynomials are fixed s.t. $n = 256,\ n' = 9,\ q = 3329$ [81], [83].

Also assume that for a set $\mathcal{S}$, s is an element which was randomly and uniformly chosen s.t. $s \leftarrow \mathcal{S}$. Assume $Var(\mathcal{S})$ is variance of the distribution, if $\mathcal{S}$ is a probability distribution. Assume x is a bit string and when inputted into $\mathcal{S}$, an output denoted y will be the sampling of x with respect to $\mathcal{S}$ s.t. $y \sim \mathcal{S} = Sam(x)$. Assume that β_n is a central binomial distribution over $\mathbb{Z}$ s.t. $\beta_n = B(2n, 0.5) - n$ [81], [83].

Here, the initial function in Kyber, *Compress()* and *Decompress()* are explained. These two functions' objective is to remove some "low-order" bits in a ciphertext, thus reducing its size. These two functions are noted in equation below [81], [83]:

$$x' = Decompress_q\big(Compress_q(x, d), d\big)\ ;\ x, d \in \mathbb{Z}_q\ , d < [\log_2 q]$$

$$Compress_q(x, d) = \left[\left(\frac{2^d}{q}\right) \cdot x\right] mod\ 2^d\ ; 2^d < q$$

$$Decompress_q(x, d) = \left[\left(\frac{q}{2^d}\right) \cdot x\right]\ ; 2^d < q$$

Starting in *KYBER.CPAPKE* key generation first, assume that three integers for some k, d_u, d_v have been initiated. Assume $\mathcal{M}_{2,n} = \{0,1\}^n$ is the message space s.t. an element $m \in \mathcal{M}_{2,n}$ can be viewed as a polynomial in R with coefficients $\{0, 1\}$. First,

two values ρ, σ from the $\mathcal{M}_{2,n}$ are generated s.t. $\rho, \sigma \leftarrow \{0,1\}^{256}$. Next, a matrix $\boldsymbol{A}$ is generated by taking a sampling of ρ with respect to R s.t.:

$$\boldsymbol{A} \sim R_q^{k*k} = Sam(\rho).$$

A sampling of σ is taken but this time with respect to β_n s.t.

$$(\boldsymbol{s}, \boldsymbol{e}) \sim \beta_{n1}^k * \beta_{n1}^k = Sam(\sigma).$$

Then, a vector $\boldsymbol{t}$ is constructed s.t. $\boldsymbol{t} = \boldsymbol{A} * \boldsymbol{s} + \boldsymbol{e}$. Finally, $PubK_{CPA_{user}}$ will be $\{\boldsymbol{t}, \rho\}$ and $Priv_{CPA_{user}}$ will be $\{\boldsymbol{s}\}$ [81], [83].

In *KYBER.CPAPKE* encryption, first a value r is generated from the $\mathcal{M}_{2,n}$ s.t. $r \leftarrow \{0,1\}^{256}$. Next, a matrix $\boldsymbol{A}$ is generated by taking a sampling of ρ with respect to R s.t. [81], [83]:

$$\boldsymbol{A} \sim R_q^{k*k} = Sam(\rho).$$

Next, a sampling of $\boldsymbol{r}$, $\boldsymbol{e_1}$ and e_2 is taken with respect to β_n s.t. [81], [83]:

$$(\boldsymbol{r}, \boldsymbol{e_1}, e_2) \sim \beta_{n1}^k * \beta_{n2}^k * \beta_{n2} = Sam(r)$$

Next, a vector $\boldsymbol{u}$ is generated and subsequently value v is generated s.t. [81], [83]:

$$\boldsymbol{u} = Compress_q(\boldsymbol{A}^T \boldsymbol{r} + \boldsymbol{e_1}, d_u)$$

$$v = Compress_q\left(\boldsymbol{t}^T \boldsymbol{r} + e_2 + \left\lceil \frac{q}{2} \right\rceil \cdot m, d_v\right).$$

Finally, the ciphertext Ct will be $Ct = \{\boldsymbol{u}, v\}$ [81], [83].

In *KYBER.CPAPKE* decryption, the ciphertext $\{\boldsymbol{u}, v\}$ is decompressed s.t.:

$$\boldsymbol{u} = Decompress_q(\boldsymbol{u}, d_u)$$

$$v = Decompress_q(v, d_v)$$

Finally, the message m can be retrieved as such:

$$m = Compress_q(v - \boldsymbol{s}^T\boldsymbol{u}, 1)$$

In *KYBER.CCAKEM* key generation, a value z is generated s.t. $z \leftarrow \{0,1\}^{256}$. Using key pairs generated from *KYBER.CPAPKE*, *KYBER.CCAKEM*'s private key can be computed as such [81]: (Note that $PubK_{KEM_{user}}$ is also from $PubK_{CPA_{user}}$)

$$PrivK_{KEM_{user}} = (PrivK_{CPA_{user}} \,||\, PubK_{CPA_{user}} \,||\, H\big(PubK_{CPA_{user}}\big)||\, z)$$

In *KYBER.CCAKEM* encapsulation, a message from the message space is generated s.t $m \leftarrow \{0,1\}^{256}$. m will be hashed as such $m \leftarrow H(m)$. Note that $H()$ here is SHA-3-256 or SHA-2-256. Next, values for $\overline{K}, r$ are generated by hashing m and the public key of *KYBER.CCAKEM* s.t. $(\overline{K}, r) = G(m||H(PubK_{KEM_{user}}))$. Note that G here is SHA-3-512 or SHA-2-512. Then, using *KYBER.CPAPKE*'s encryption method, the public key, message and r is encrypted s.t. [81]:

$$Ct = KYBER.CPAPKE\_Enc\big(PubK_{KEM_{user}}, m, r\big)$$

Finally, the shared secret key K is computed, from $\overline{K}$ and hash of Ct s.t. [81]:

$$K = KDF\left(\overline{K} \,||\, H(Ct)\right)$$

In *KYBER.CCAKEM* decapsulation, first the value for $PubK_{KEM_{user}}$, h and z is inititated s.t [81].

$$PubK_{KEM_{user}} = PrivK_{KEM_{user}} + 12 \cdot k \cdot n \,/\, 8$$

$$h = PrivK_{KEM_{user}} + 12 \cdot k \cdot n \,/\, 8 + 32 \in \{0,1\}^{256}$$

$$z = PrivK_{KEM_{user}} + 12 \cdot k \cdot n \,/\, 8 + 64$$

The message m from Ct is decrypted, denoted with m' s.t. [81]:

$$m' = KYBER.CPAPKE\_Dec\left(PrivK_{KEM_{user}}, Ct\right)$$

Next, values for $\overline{K}', r'$ are generated s.t. $(\overline{K}', r') = G(m'||h)$. Then, values for Ct' is generated s.t. [81]:

$$Ct' = KYBER.CPAPKE\_Enc\left(PubK_{KEM_{user}}, m', r'\right)$$

Ct and Ct' are checked if they are identical. If they are identical, then the shared secret key K will be computed as such (Note KDF here is SHAKE-256 or SHA-256) [81]:

$$K = KDF\left(\overline{K}' \,||\, H(Ct)\right)$$

If not, K will be computed as such:

$$K = KDF\left(z \,||\, H(Ct)\right)$$

Figure 2.34 below shows how *KYBER.CCAKEM* works for KEP. Note that *KYBER.CPAPKE* is embedded inside "Encaps" and "Decaps" process as well.

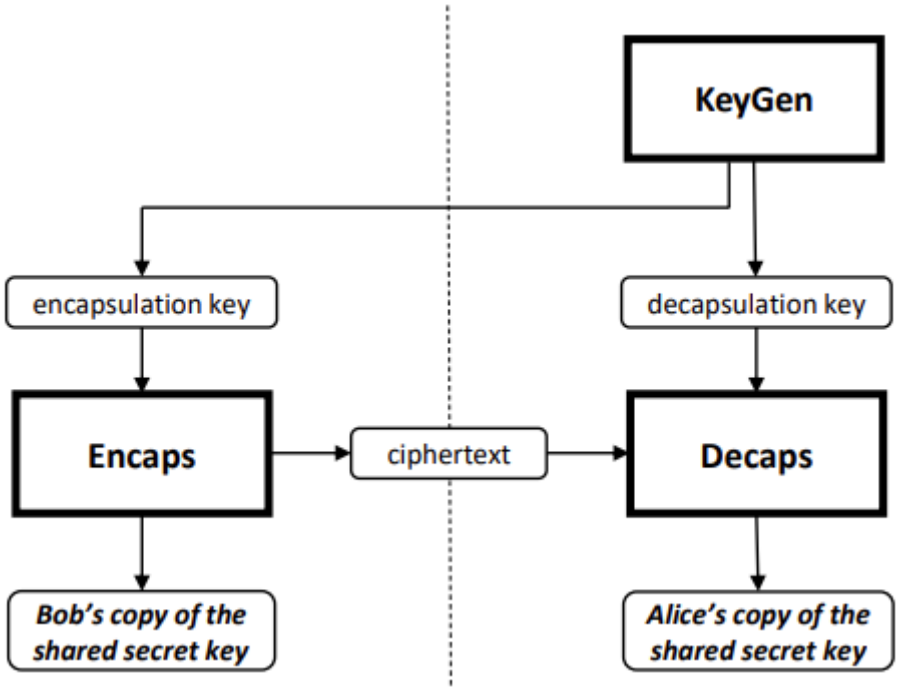


Figure 2. 34 Visualisation of KEP using KEM [80]

Table 2.12 shows the parameter sets for all Kyber's variants.

Table 2. 12 Comparison on parameter values for Kyber's variants [81]

Variant	n	k	q	n_1	n_2	(d_u, d_v)
Kyber-512	256	2	3329	3	2	(10, 4)
Kyber-768	256	3	3329	2	2	(10, 4)
Kyber-1024	256	4	3329	2	2	(11, 5)

Compared to NTRU, Kyber is almost the same since both use LWE. The only difference is where NTRU employs Ring-LWE, and Kyber employs Module-LWE. Polynomials defined in NTRU earlier can be visualised as lattices as such e.g.: a $= a_0 + a_1X + a_2X^2 + \cdots + a_{N-2}X^{N-2} + a_{N-1}X^{N-1}$; where values of a_{N-1} will be columns for lattices when visualised. Now, assume that there are two base vector sets named "Good" and "Bad." A "good" vector set should have a big angle between both; almost orthogonal. A "Bad" vector set should have a very small angle between both; almost parallel to each other. **Figure 2.35** visualises these two base vector sets. (Note that "good" vector set is highlighted in green, and "bad" in red).

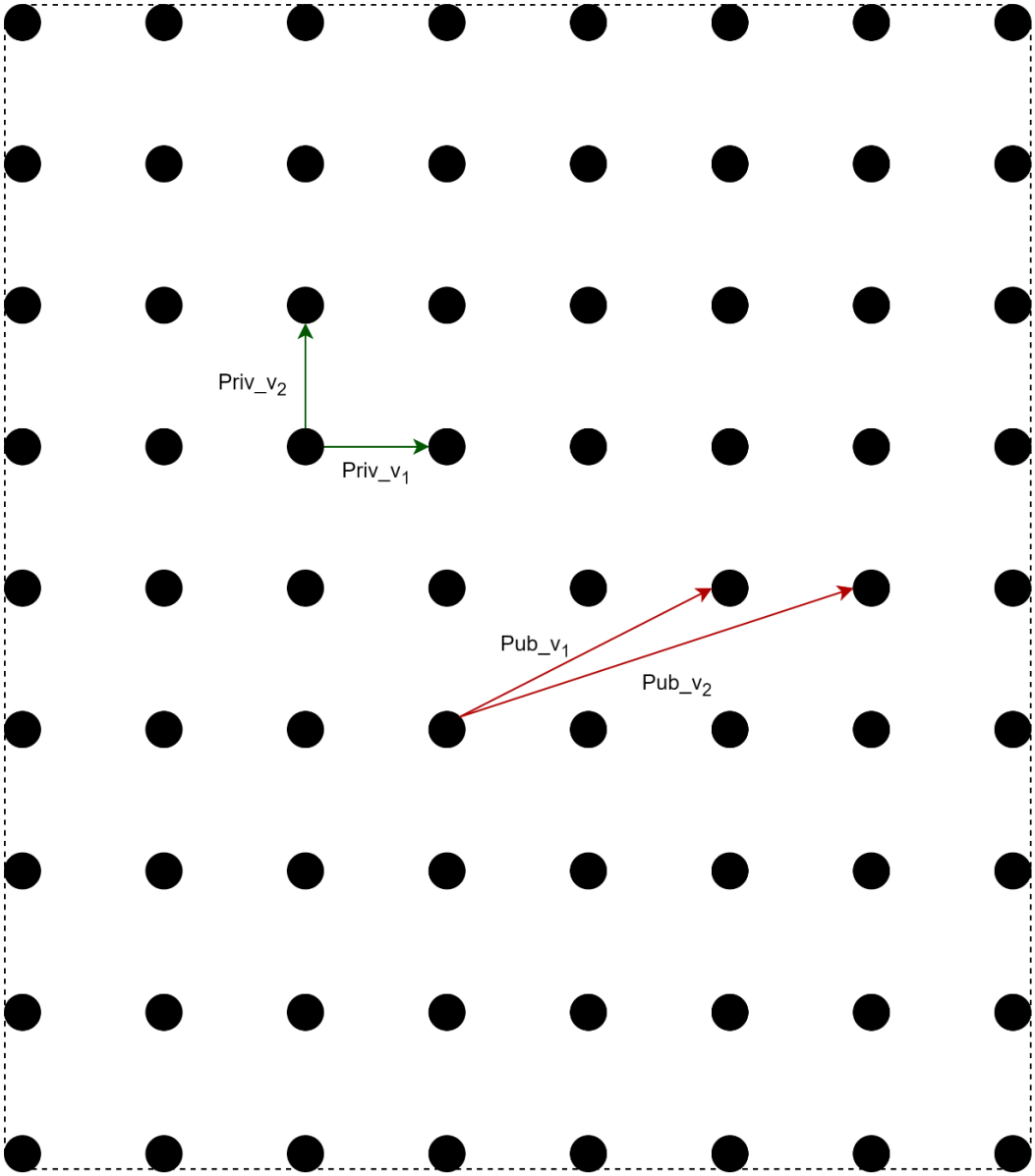


Figure 2. 35 Visualisation of base vector sets in lattice

Now, assume there is a point in the lattice space to traverse to. The objective is to find a way to the closest point, given these two base vector sets. For "good" set vector, this is easy because the way it's mapped to the lattice space makes it efficient to traverse toward the point. However for "bad" set vector, this is very hard since the way it's mapped to the lattice space requires it to traverse extra points (much longer than "good" set vector) to traverse toward the point. **Figure 2.36** visualise how these two vectors solve the objective above. Also note that v^{-1} is the inverse of the vectors.

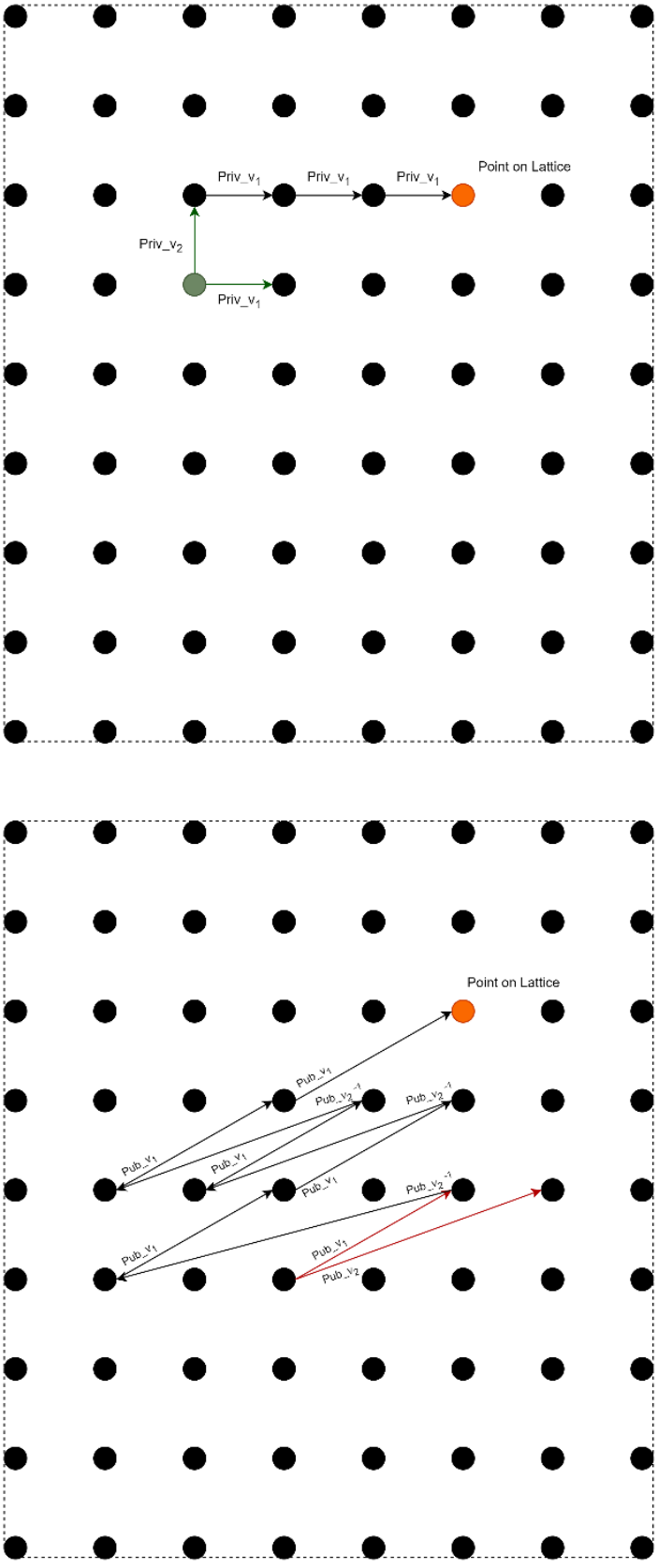


Figure 2. 36 Visualisation of vector's traversal to a point in lattice

However, this problem can be easily solved in two-dimensional space shown in **Figure 2.36**. For a PQC employing lattice-based primitives, it must have use at least 250-dimensional space for it to be secure. Proposed lattice-based PQC will have at least 1000-dimensional lattice space. With the lattice space's uncomically large dimensions, the problem is proven to be very hard for both conventional and quantum computers to solve. **Figure 2.37** sorts of visualise the effects of increasing the dimensions of the lattice; although it's still in three-dimensional.

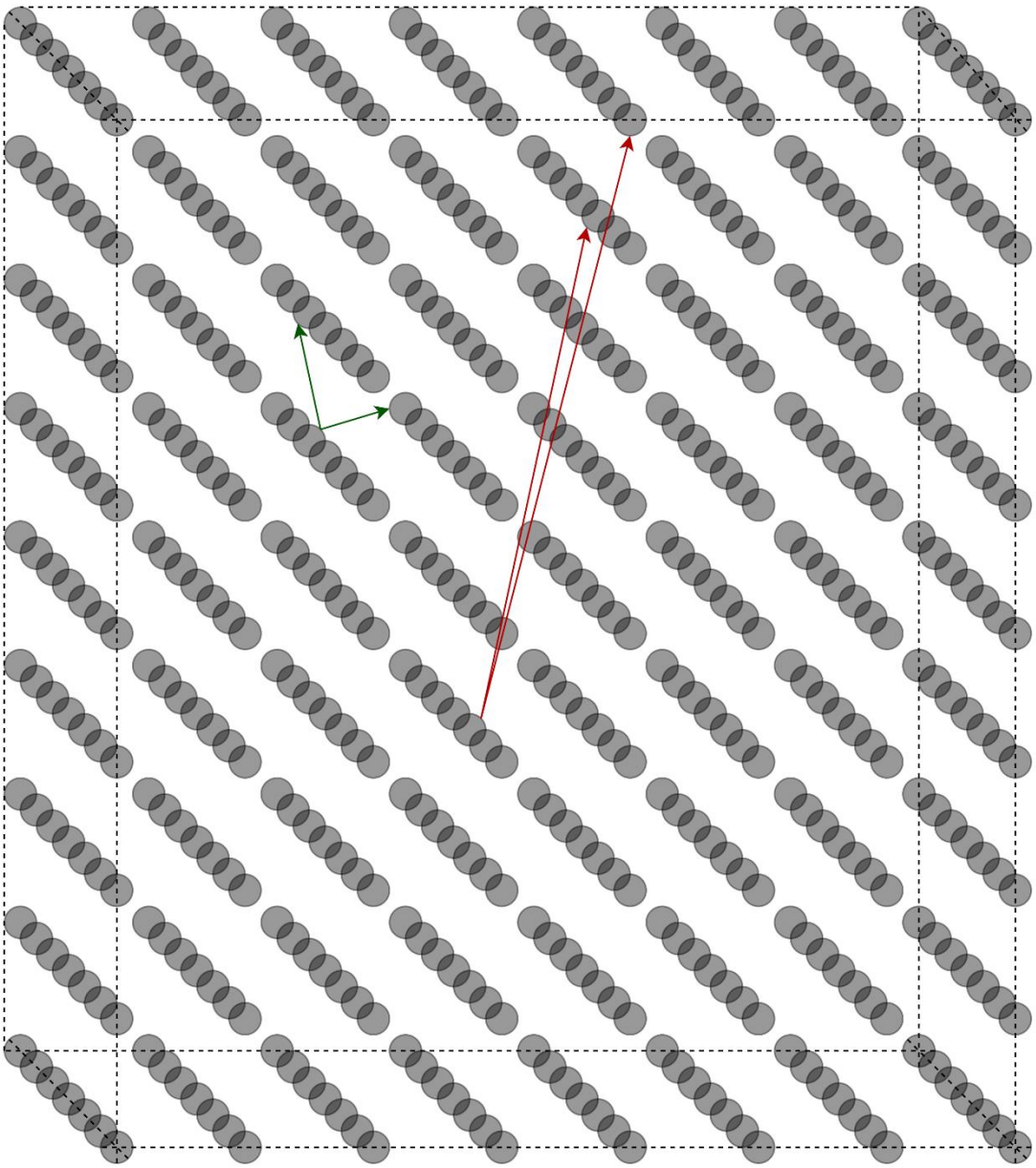


Figure 2. 37 Visualisation of multi-dimensional lattice space with vector sets

b) Post-Quantum Extended Diffie-Hellman Key Exchange Protocol

Post-Quantum Extended Diffie-Hellman Key Exchange Protocol ("PQXDH KEP") is just a derivative of X3DH to implement CRYSTALS-Kyber in the protocol, with some additional keys, signatures and random values that are used in the protocol.

Figure 2.38 visualise on how PQXDH works in conjunction with Kyber integrated into the protocol. All of X3DH's preliminaries are applied here, with few additions. Assume $pqkem$ is a post-quantum key encapsulation mechanism i.e. CRYSTALS-Kyber-1024. Also assume that $EncodeEC()$ is a function that encodes a *curve25519* or *curve448* public keys into a byte sequence; $DecodeEC()$ is the inverse of $EncodeEC()$ s.t. the function converts a byte sequence into a *curve25519* or *curve448* public keys. Similarly, $EncodeKEM()$ is a function that encodes a $pqkem$ into a byte sequence; $DecodeKEM()$ is the inverse of $EncodeKEM()$ [84].

Also assume that $(Ct, SS) = PQKEM\_Enc(PK)$ is where a public key PK is encrypted using a $pqkem$ encryption method to produce a Ct, where a shared secret SS is encapsulated inside it. Assume that $PQKEM\_Dec(PK, Ct)$ is the inverse of the process earlier; where a $pqkem$ decryption method is used to decapsulate SS from Ct using a corresponding private key of PK used during Ct's encryption [84].

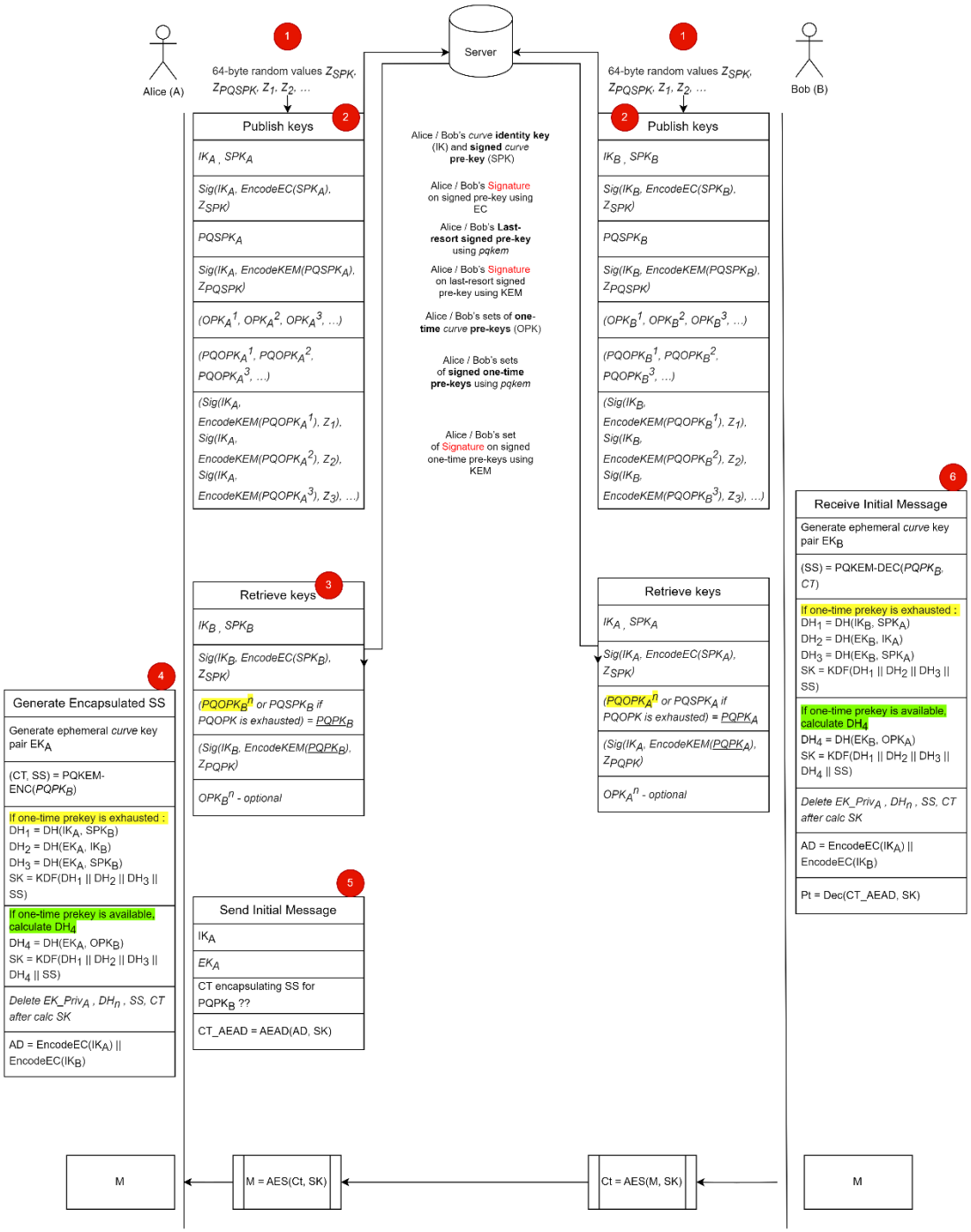


Figure 2. 38 Visualisation of PQXDH with Kyber integration

First, both parties, Alice and Bob will generate a 64-byte random values to be used for signing pre-keys and one-time keys; noted with $Z_{SPK}, Z_{PQSPK}, Z_1, \dots, Z_n$. This step is not in X3DH and only for PQXDH [84].

Alice and Bob then generate and publish a set of keys and their signatures. Like X3DH, the keys that are produced are $\{IK_{user}, SPK_{user}, OPK_{user^n}\}$. However in PQXDH, few additional keys and signatures are generated as well as such: $\{PQSPK_{user}, PQOPK_{user^n}\}$ [84]. Furthermore, signature generation in PQXDH is different from X3DH, using the random values they generated earlier Z as such:

$$Sig(IK_{user}, EncodeEC(SPK_{user}), Z_{SPK})$$

$$Sig\big(IK_{user}, EncodeKEM(PQSPK_{user}), Z_{PQSPK}\big)$$

$$Sig(IK_{user}, EncodeKEM(PQOPK_{user^n}), Z_n)$$

where first signature is for signed pre-key using *curve* with Z_{SPK} random value also added during signing process. Second signature is for signed pre-key using Kyber with Z_{PQSPK} random value also added during signing process. Third signature is for one-time pre-keys using Kyber with Z_n random value also added during signing process [84]. The rule on generating signed pre-key and one-time pre-keys from X3DH also applied here.

Next, Alice will be retrieving Bob's set of pre-keys; named "prekey bundle", $\{IK_{Bob}, SPK_{Bob}, Sig(IK_{Bob}, EncodeEC(SPK_{Bob}), Z_{SPK}),\ PQOPK_{Bob^n} \mid PQSPK_{Bob}\ ,$ $Sig\big(IK_{Bob}, EncodeKEM(PQPK_{Bob}), Z_{PQPK}\big), OPK_{Bob^n}\}$. Note that $PQPK_{Bob}$ here is either one of $PQOPK_{Bob^n}$ or $PQSPK_{Bob}$, depending on the availability of $PQOPK_{Bob^n}$ in the server's storage. If $PQOPK_{Bob^n}$ is depleted, $PQSPK_{Bob}$ is fetched instead [84]. The rule on handling one-time pre-keys from X3DH also applied here.

Alice will be verifying all of the $Sig()$ of the pre-keys. If all verification is successful, the protocol is a go; else Alice will abort the protocol. Then if the protocol proceeds, Alice will generate a *curve* ephemeral key pair EK_{Alice} and compute an Encapsulated Shared Secret s.t. [79], [84]:

$$(Ct, SS) = PQKEM\_Enc(PQPK_{Bob})$$

She will then proceed to compute the secret key SK, together with SS value following [84]:

$$DH_1 = DH(IK_{Alice}, SPK_{Bob})$$
$$DH_2 = DH(EK_{Alice}, IK_{Bob})$$
$$DH_3 = DH(EK_{Alice}, SPK_{Bob})$$

$$SK = KDF(DH_1 \,||\, DH_2 \,||\, DH_3 \,||\, SS)$$

The equations for SK above is applied to the condition where Alice do not have Bob's *curve* OPK_{Bob^n}. If Alice do have Bob's OPK_{Bob^n}, the equation for SK is noted below [84]:

$$DH_1 = DH(IK_{Alice}, SPK_{Bob})$$
$$DH_2 = DH(EK_{Alice}, IK_{Bob})$$
$$DH_3 = DH(EK_{Alice}, SPK_{Bob})$$
$$DH_4 = DH(EK_{Alice}, OPK_{Bob})$$

$$SK = KDF(DH_1 \,||\, DH_2 \,||\, DH_3 \,||\, DH_4 \,||\, SS)$$

Next, Alice will compute an "associated data" byte sequence noted AD that is derived from the identity keys of both parties; s.t.: $AD = EncodeEC(IK_{Alice}) \,||\, EncodeEC(IK_{Bob})$. AD might also contain additional information e.g. certificates, usernames etc. Furthermore, if $PQKEM$ does not include $PQPK_{Bob}$ into Ct, Alice must append $EncodeKEM(PQPK_{Bob})$ to AD; s.t.: $AD = EncodeEC(IK_{Alice}) \,||\, EncodeEC(IK_{Bob}) \,||\, EncodeKEM(PQPK_{Bob})$ [84].

Alice will encrypt an initial ciphertext Ct_{aead} with an AEAD scheme; denoted in X3DH explanation. AD will be inputted into the AEAD scheme as associated data input and SK as its key [78]. With Ct_{aead}, Alice will send an initial message together with Ct, her identity key IK_{Alice}, *curve* ephemeral public key EK_{Alice}, and some sort of identifier on

which one of Bob's one-time pre-keys $n \leftarrow OPK_{Bob^n}$, $PQOPK_{Bob^n}$ are used by Alice during the protocol's run [84].

Upon receiving the message, Bob retrieve Alice's IK_{Alice}, EK_{Alice} from the message and prepare to use his private key pairs of IK_{Bob}, SPK_{Bob}, OPK_{Bob^n}, $PQSPK_{Bob}, PQOPK_{Bob^n}$. Depending on which $pqkem$ key Alice used during her protocol's run, Bob will first compute the decryption of Ct, to decapsulate the SS from inside Ct; s.t. [79], [84]:

$$SS = PQKEM\_Dec(PQPK_{Bob}, Ct)$$

Using the derived SS, Bob will repeat the DH and KDF computation earlier using his retrieved key bundle to derive SK. Bob will also compute his own AD as earlier.

Finally, he will decrypt the initial ciphertext from the AEAD scheme Ct_{aead} using his derived SK and AD. If the decryption works, the protocol has complete, and SK can be used for subsequent encryption of message i.e. symmetric encryption after the PQXDH protocol; else the protocol aborts and SK is deleted [84].

To conclude on key exchange protocol, it provides a secure way of exchanging a shared secret key to be used for symmetric encryption of message by using asymmetric cryptographic primitives.

2.4 Introduction to Quantum computers and its threats to cryptography

Quantum computers for all practical intents and purposes, is just like another normal computer where its task is to receive input, process input and produce output. But Quantum computers have something special in their construction in which they implement some quantum mechanics' phenomena in input and process.

To explain furthermore on last point, a conventional computer uses a bit to represent a data. Bit is a binary and only can be at two states at a time which is $\{0,1\}_2$. For a Quantum computer, data is represented as a Quantum Bit or Qubit. A Qubit is not restricted to two states like a bit; instead, it can be between these two states at a time. This could be represented as such [85], [86]:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

where $\{|0\rangle, |1\rangle\}$ is a two-dimensional linear vector space of a qubit, and ψ is the result of the superposition of qubit states when measured by adding together the value of probability of $\{|0\rangle, |1\rangle\}$ by α, β. Possible ψ states can also be represented as a Bloch sphere. **Figure 2.39** below visualise ψ in a Bloch sphere [85], [86]: (Note that φ, θ are derived from α, β computation of Hopf coordinates)

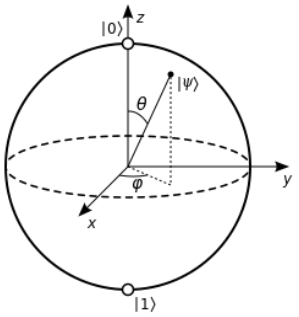


Figure 2. 39 Visualisation of a Qubit in Bloch Sphere [87]

A normal two-bits will be only at four possible state at a time which is $\{\{0,0\}_2, \{0,1\}_2, \{1,0\}_2, \{1,1\}_2\}$. But for two qubits, the number of possible states can rise as such [85], [86]:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

In essence, conventional computer with 2^n-bits can be represented in a quantum computer with a smaller number of Qubits, which is only n-qubits. More information can be stored in one time in a state with Qubits, compared to bits. This means that with enough number of Qubits, a very large number of different states can be represented and perform a large number of computation simultaneously at a time. With just 300 Qubits, a total of 2^{300} different states can be represented, which is an uncomically large number just from a small number of Qubits [85], [86].

However, there is one large obstacle in using Quantum computer. When measurement on ψ is performed, only one qubit state can be observed at random, and all other qubit states will be lost. All but only one information of superpositions can be measured. According to DiVicenzo, there are also other large obstacles on harnessing the power of Quantum computer, which is difficulty of increasing the number of Qubits, the quality of Qubits themselves, corruption of superposition of states in Qubits by decoherence; introducing noise into computation [88].

Sundar Pichai, CEO of Google quoted in [89]:

You know in a five-to-ten-year timeframe, Quantum Computing will break encryption [in this case conventional symmetric and asymmetric cryptosystem] *as we know it today.*

and asserts that quantum computers are nearer to be accessible at a certain point in the future; also be able to break conventional cryptosystem.

This highly probable hypothetical scenario is already become a threat to current cryptography implementation. This is because a threat actor could intercept current ciphertexts encrypted with conventional cryptosystems, and hope that in the future where Quantum computers will become more feasible, they will be able to decrypt the ciphertext. This technique is called “Store Now, Decrypt Later” (“SNDL”) [90].

Relating back to our hypothetical scenario in computationally secure cryptographic scheme (pp. 10), assume that the scenario still holds, but this time the threat actor now have access to a Quantum Computer. If the threat actor could have access to a Quantum Computer before the secret is no longer bound to the cipher’s protection; before the 30 years passes after encryption, the threat actor may be able to decrypt the secret and are able to defeat the Confidentiality triad.

2.5 Attacks on cryptography

There are few ways for a cryptanalyst to break a cryptosystem. Types of attacks on a cryptosystem are noted below:

- **Brute-Force Attack**, where a cryptanalyst tries every single key of a cipher in order to decrypt the Ct, until an intelligible Pt is obtained [5], [8].

- **Ciphertext only**, where a cryptanalyst only have intelligence on a cryptosystem used and only the Ct. Brute-Force Attack may fall under this type [5], [8].

- **Known Plaintext**, where a cryptanalyst have intelligence on a cryptosystem used, the Ct, and one or more Pt that corresponds to Ct as such it is paired as Pt-Ct; formed during encryption with the key [5], [8].

- **Chosen Plaintext**, where a cryptanalyst have intelligence on a cryptosystem used, the Ct, and Pt chosen by a cryptanalyst together with its corresponding Ct' generated with the key [5], [8].

- **Chosen Ciphertext**, where a cryptanalyst have intelligence on a cryptosystem used, the Ct, and Ct' chosen by a cryptanalyst together with its corresponding decrypted Pt generated with the key [5], [8].

- **Chosen Text**, where a cryptanalyst have intelligence on a cryptosystem used, the Ct, a Pt chosen by a cryptanalyst together with its corresponding Ct' generated with the key, and a Ct' chosen by a cryptanalyst together with its corresponding decrypted Pt generated with the key [5], [8].

Note that brute-force attack, ciphertext only and known plaintext attack can be categorized as passive attack, where a cryptanalyst only works on incoming Ct. In chosen plaintext, chosen ciphertext and chosen text, they can be categorized as active

attack since a cryptanalyst is making use of the encryption algorithm to produce Pt'or Ct' using the cryptosystem used for some key, and comparing them with incoming Ct [5].

2.5.1 Attacks on current cryptography with conventional computer

In order to brute-force an encryption method of a cryptosystem, one must have to try at least 2^n attempts in order to break the encryption. n here is the key length, and generally; the longer the key length, more possible key can be used to perform encryption and more attempts are needed to break the encryption.

In 2009, Biryukov et. al. published a key-recovery attack on slightly tweaked AES variant where two related keys are used to recover a complete 256-bit key of 9 rnd. with 2^{39} time. They also managed to recover the key of 10 rnd. with 2^{45} time and 2^{70} time with 11 rnd. However, AES-256 uses a 14 rnds. for its operation; so they concluded the attack is not feasible against current AES's implementation [91].

In 2011, Bogdanov et. al. published another attack on full AES implementation named "biclique attack." Biclique attack are able to reduce time required to break AES, compared to brute force; by a factor of four. The best result from the attack conclude that $2^{126.0}$, $2^{189.9}$ and $2^{254.3}$ time are required to recover the key of full AES-128, AES-192 and AES-256 respectively [92]. Although it is a small step upward, an AES-128 key would still take a very, very long time to compute to recover the key.

Till date, there is no other known successful, practical attack against AES where without the knowledge of the key used during the operation, ciphertext from AES cannot be decrypted.

According to Boneh and Venkatesan [93], even though proofing on breaking RSA with low-exponent RSA is different from factoring integers. The most efficient way known

to solve RSA problem, which is general number field sieve, is by first factoring n. The RSA key generation process turns e with prime factorization, into d. The same algorithm allows any threat actor who compute the factors of n to obtain d; subsequently able to decrypt Ct using d. The only obstacle for this method is it will be infeasible if n is large enough, which current RSA's implementation is using. Finding d is indeed just as hard on computing the factors of n [13].

In 2019, Boudot et. al. and Kleinjung et. al. were able to factor a 795-bit and 768-bit numbers respectively (in RSA term – RSA-240), utilizing about 900 core-years of computing power. Boudot et. al. estimated that RSA-1024, lowest number length used, will take 500 times of the computing power just to factor n [94], [95].

In 2000 to 2004, Harley et. al. and Monico et.al. solved an elliptic Prime curve over $\mathbb{F}_p$; $p =$ 109-bit, $\mathbb{F}_K$; $K =$ 108-bits Koblitz curve and F_{2^m} ; 2^{109} Binary curve using Pollard-Rho attack. It took more than a year to complete the computation, and that was using parallelisation of the algorithm [96].

To look at most recent computational power against the ECDLP, in 2020, Zieniewicz and Pons were able to solve an elliptic curve 114-bit *secp256k1* with their own proprietary Polland-Rho's implementation in their software, based on Pollard Kangaroo. The software ran on 256 NVIDIA Tesla V100 GPUs and using parallelisation as well. They were able to solve the problem in just 13 days. They were also able to solve the problem with a different 109-bit elliptic curve in just 3 days, with the same setup [97].

However, in respect to ECC's real-world implementation which uses a key length of at least 160-bit minimum, it's still computationally infeasible for current computational power to solve ECDLP with that level of security.

In hash functions, it is considered broken if two hash digest that are generated for a hash function are almost identical; contradicting hash requirements (from pp. 62). This is called collision, and in order to find collision; at least $2^{n/2}$ attempts are needed to find a collision within a hash function. Similar to brute-force from encryption, in order to find the pre-image for the hash digest; the message inputted into the hash function, 2^n attempts are needed to find the pre-image.

In 2011, Bogdanov et. al. published the same "biclique attack" method used in their AES attack onto SHA-2. They were able to break pre-image resistance for 57 out of 80 rnds. for SHA-512 and 52 out of 64 rnds. for SHA-256. The attacks took them $2^{511.5}$ and $2^{255.5}$ time to break the pre-image resistance of SHA-2 [98].

In 2013, Mendel et. al. published a new attacks on reduced SHA-256 to find local collisions. The work implements a differential attack to find collisions within the hash function. They concluded that to find collision in SHA-256, in 31 out of 64 rnds.; it would take $2^{65.5}$ time [99]. With respect to SHA-2's full implementation, it would take 2^{128}, 2^{192} and 2^{256} time (from collision equation above) to find collision for SHA-256, SHA-384 and SHA-512, respectively.

However, till date, there is no full collision found on SHA-2 yet.

2.5.2 Attacks on current cryptography with Quantum computer

With the advancement of Quantum computers in recent time, few cryptographers had proposed few algorithms that would break completely or decrease the security level of all cryptosystems that are implemented currently. Especially on asymmetric cryptography, all of the widely used cryptosystems can be broken once Quantum computers are accessible using Shor's Algorithm. This fact is also true for key exchange protocols since KEPs are using asymmetric cryptographic primitives in their operation.

However, for symmetric cryptography and hash functions, the use of Quantum computers cannot break them completely, however their security level will be greatly reduced.

Table 2.13 below shows the comparison between pre-quantum and post-quantum cryptography security level.

Table 2. 13 Comparison on pre-quantum and post-quantum security [41]

Cipher/Hash Function	Type	Pre-Quantum security level	Post-Quantum security level
AES-128	Symm. cipher	128 (1)	64 (Grover)
AES-192	Symm. cipher	192 (3)	96 (Grover)
AES-256	Symm. cipher	256 (5)	128 (Grover)
SHA2-224 / SHA3-224	Hash funct.	224 (<1)	112 (Grover)
SHA2-256 / SHA-3-256	Hash funct.	256 (1)	128 (Grover)
SHA2-384 / SHA3-384	Hash funct.	384 (3)	192 (Grover)
SHA2-512 / SHA3-512	Hash funct.	512 (5)	256 (Grover)
RSA-3072	Asymm. cipher , Signature	128 (1)	**Broken (Shor)**
DH-3072	KEP	128 (1)	**Broken (Shor)**
256-bit ECDH	KEP	128 (1)	**Broken (Shor)**
256-bit ECDSA	Signature	128 (1)	**Broken (Shor)**

a) Shor's Algorithm

Shor in 1997, published an algorithm that could potentially solve integer factorization and discreet logarithmic problems with the help of quantum computers [1]. Crucially, Shor's algorithm is able to take n as input and factorize it into two integers p, q; basically solving the RSA problem. Furthermore, this algorithm can be run in polynomial time. However, the original Shor's algorithm is far from perfect, and

research has been done to optimize it where number of qubits required and qubits' operation are reduced.

In Beauregard's work, a variant of Shor's algorithm was proposed, where $2n + 3$ qubits are required and it can run in $\mathcal{O}(n^3 \log n)$ operations; assuming if N can fit into n-bits [100].

Shor's algorithm applies a periodic function on a superposition of inputs, then perform a Quantum Fourier Transform in order to get the approximate superposition of periods of the function; then measure the superposition to find a random period. The periodic function is denoted with $e \rightarrow a^e \ mod \ N$; where a is a random number, co-prime to N [1], [4].

In Shor's algorithm also he proposed that it was also able to find periods of another periodic function s.t. $e, f \rightarrow g^e h^f \ mod \ p$. This periodic function looks similar to that of DLP where $h = g^k \ mod \ p$; and when $mod \ p$ is substituted with point addition or multiplication on an elliptic curve, it could also be used for solving the ECDLP [1], [3].

b) Grover's Algorithm

Grover in 1996, published an algorithm that could perform searching in an unordered N-sized database with $\sqrt{N}$ queries using Quantum computers [2], [101]. Although it may seem innocuous at first but referring back to the cryptanalysis of AES using conventional computer, this algorithm can be used to effectively search and recover keys used by full AES operation.

In essence, Grover's algorithm can be described as such it solves a function inversion. Assume that there is a function $y = f(x)$ that can be applied to a Quantum computer, x can be computed with the inverse of the function, given y; s.t. $f^{-1}(y) = x$. Now,

assume that there are two 128-bit Pt that is encrypted using AES-128 (Also assume that k is 128), and producing two 256-bit Ct s.t. [41], [44], [102]:

$$Ct = \left(AES_k(Pt_1), AES_k(Pt_2)\right).$$

the function earlier can be used s.t.:

$$f(x) = \left(AES_k(Pt_1), AES_k(Pt_2)\right) - Ct.$$

This function can be applied to a quantum computers where it will find the root of f, where in this case it will take 2^{64} evaluations of f over 3000 qubits [41], [44], [102].

The impact of Grover's algorithm is less significant compared to Shor's algorithm. This is because Grover's algorithm is just a searching algorithm, and from its purpose, it could help us reduce the time required to brute-force a symmetric key; and it does not actually actively exploit symmetric encryption weakness or solving the "hard problems" underlying the symmetric cipher, to recover the key; unlike Shor's algorithm.

2.6 Current advancement on Quantum computers

Although the proposition of using qubits for a Quantum computer is interesting, the qubits that exist today is not a “perfect” Qubit. Qubits in the current time consists of multiple imperfect qubits; as redundant information storage.

As time progresses, more qubits are perfected; thus a smaller number of qubits are required to be used during computation. As time progresses too, more companies and entities are racing toward building a more powerful Quantum computer by increasing their number of perfect qubits – a race for “Quantum Supremacy” [103].

In around 2000 to 2010, the number of perfect qubits are less than ten. However advancement of Quantum computers skyrocketed around 2017, with multiple tech companies were able to create approximately 50 qubits. In 2019, Google claimed that they have achieved quantum supremacy with their Sycamore quantum computer which have 54 qubits [104]. At the same time in 2018, IBM was also developing their own Quantum computer named “Q” which have 16 qubits [105].

At time of writing in 2023, IBM have already developed the first 1121-qubit Quantum processor dubbed “Condor”; however with this large number of qubits, error are more prone to encounter. To tackle this problem, IBM also released another 133-qubit Quantum processor dubbed “Heron”; where it is more error-resistant compared to its bigger brother [106].

To conclude on Quantum computers, it provides a powerful tool in a cryptanalyst’s arsenal to delve into untapped cryptanalysis potentials on currently implemented symmetric and asymmetric cryptosystems using Quantum algorithms.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter covers the research and development methodology and phases of the proposed system, the software and hardware requirements of the proposed system, and the proposed system's features, functional and non-functional requirements and constraints.

3.2 Research Methodology

According to Blackstone [107], qualitative methods in research can be defined as the way of collecting data that are of qualitative values i.e. words or pictures. Quantitative methods in research can be defined as the way of collecting data that are of quantitative values i.e. numbers and statistics. Although these two methods differ in type of data collected, they both complement each other in a research. With that, the primary knowledge source that author used for knowledge gathering is by reading references papers, journal articles, reports etc. on the Internet. Studies, comparison, and analysis on the subjects are also gathered.

Based on Goundar [108], research methodology differs from research methods. In fact, research methodology can be defined as a systematic way to solve a problem. Given the knowledge gathered using research methods defined above, the researchers have to describe and explain upon how to they solve the problem in their works.

In **Figure 3.1** below, the project research methodology together with project scope is specified.

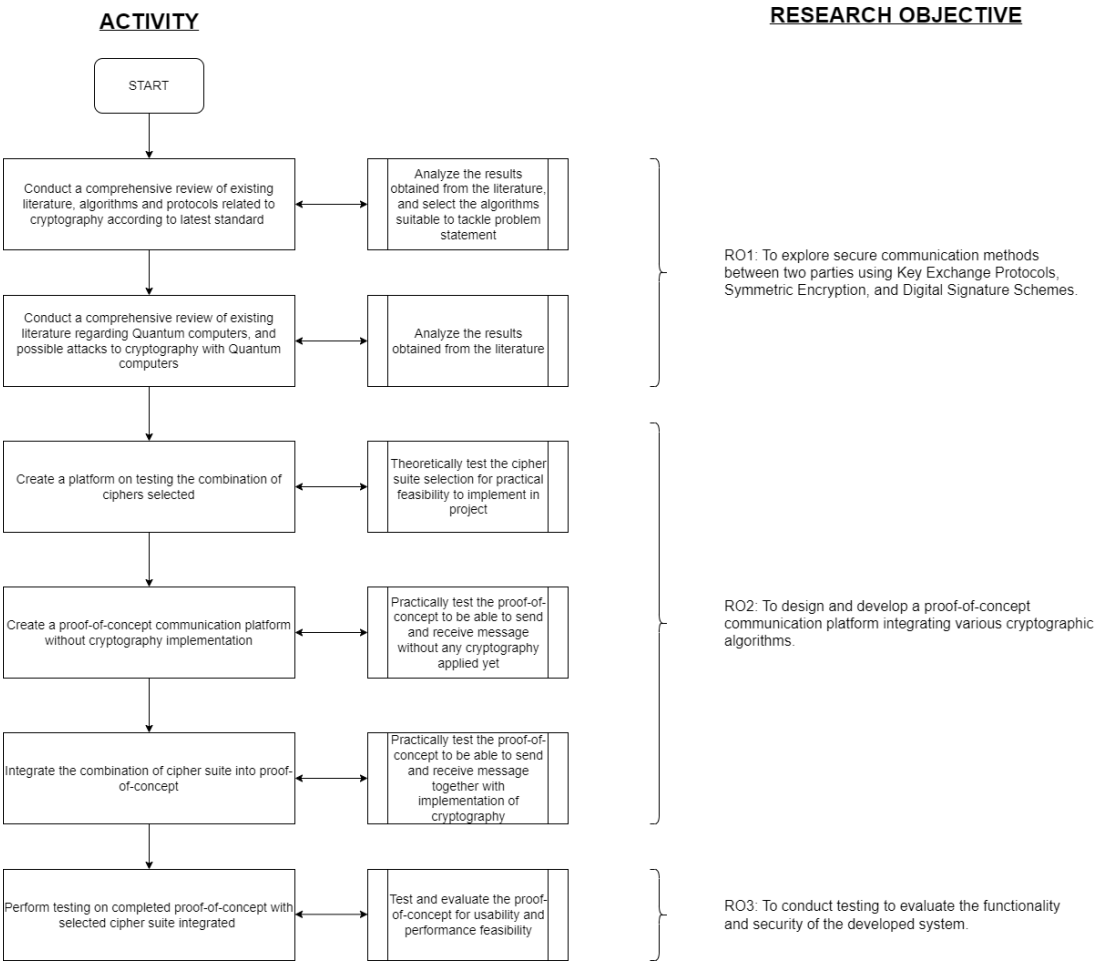


Figure 3. 1 Project Research Methodology

With the research methods and literature review considered upon, below is the cipher suite selection that is used in the proposed system.

- **Symmetric Encryption**

For symmetric encryption, AES has been chosen. This is because from the qualitative and quantitative examples earlier, AES is the most solid candidate for a symmetric cipher.

- **Asymmetric Encryption**

For asymmetric encryption, ECC has been chosen. This is because implementing RSA in a secure way is quite difficult compared to ECC [109]. Also, ECC may have a faster runtime compared to RSA. Moreover, the hardness of ECDLP is much more convincing in provably-secure cryptographic security scheme compared to RSA's problem which rely on integer factorization.

- **Post-Quantum Encryption**

The proposed system would not actually use any Post-Quantum Encryption in this project. This is because after thoughtful consideration, it might be more feasible and secure to integrate Post-Quantum encryption into conventional Key Exchange Protocol, which could reduce time and performance overhead; compared to pure Post-Quantum encryption implementation in the system. So, CRYSTALS-Kyber was chosen to become the Post-Quantum Encryption, but this algorithm would be implemented under KEP.

CRYSTALS-Kyber was chosen because comparing all Post-Quantum encryption's problem or hardness, it is concluded that module-lattice based is the most secure yet the most feasible to implement in the system. To compare with code-based and multivariate-based encryption, lattice-based encryption have a lower key sizes, and run much more efficiently. Kyber has also been peer-reviewed and tested by experts and found that it would be the most suitable candidate for post-quantum encryption standard in the future; with NIST endorsing the motion.

- **Digital Signature**

For digital signature scheme, either CRYSTALS-Dilithium or ECDSA / EdDSA will be chosen by the author in this thesis. Two options were considered because based on the project's time constraints during development, the author may choose to fall back on conventional digital signature scheme. This is also because time is needed to research on how to implement Post-Quantum digital signature scheme in the system.

- **Hash Function**

For Hash function, the author have chosen SHA-3 or SHA-2 variants as the primary hash function to be used in the system. However, depending on cryptographic library used e.g. Python's Argon2 library, the hash function that's used which is BLAKE2 – a derivative of ChaCha stream cipher; the usage of different hash function specified by author is permitted.

- **Key Exchange Protocol**

For Key Exchange Protocol, the author have chosen PQXDH. This is because this protocol would serve as the backbone for post-quantum security in the system's implementation and to solve the problem stated in this thesis. Although the protocol has not become a standard yet, it has been reviewed and tested in Signal's implementation [110].

3.3 Development Methodology

Development methodology in the project context, is the way in which how the deliverable of the project is built in a systematic way – also defined as "Software Engineering Principles." According to Shylesh [111], **Software Development Life Cycle** ("SDLC") can be defined as a methodology that can be used to design and develop software with high standards of quality while being delivered within the project's timeframe. According to Sommerville [112], SDLC can be defined as a set of discipline that are applied on all components of a software production based on engineering. SDLC have multiple methodologies or models that can be used to help software developers build their deliverables for the project, but in this thesis, two popular models of SDLC will be reviewed.

Here, the main phases of software engineering are defined. First, a proposed software for a project must have a thoroughly-defined main objectives or specifications, which determines the functionality and constraints of the software. This phase is called Software Specification [112].

Second, Software development is performed. In order to meet the objectives or specifications of the project, the software must be produced in accordance with the set specifications [112].

Third, Software validation is performed. This is to ensure that the software that is produced is indeed meets the requirements and specifications defined in the first phase [112].

Finally, Software evolution is performed. This is to ensure that the software that is produced are able to keep up with the changing requirements of the project in future [112].

Do note that in between the main phases of software engineering, there are multiple sub-phases and processes such as requirement elicitation, analysis, feasibility study, etc. that must be done in order to move towards the next phase.

3.3.1 Waterfall model

The first SDLC model is called the waterfall model. This model is the earliest model for software engineering and has been based upon for the subsequent SDLC models. Most crucially, the waterfall model's progression is linear and its approach to software engineering is plan-driven. This is because the phases in the model are separate and distinct of each other. Once the developer moves toward another phase using this model, it is very hard to reverse back to previous phases [112]. **Figure 3.2** visualise the waterfall SDLC model.

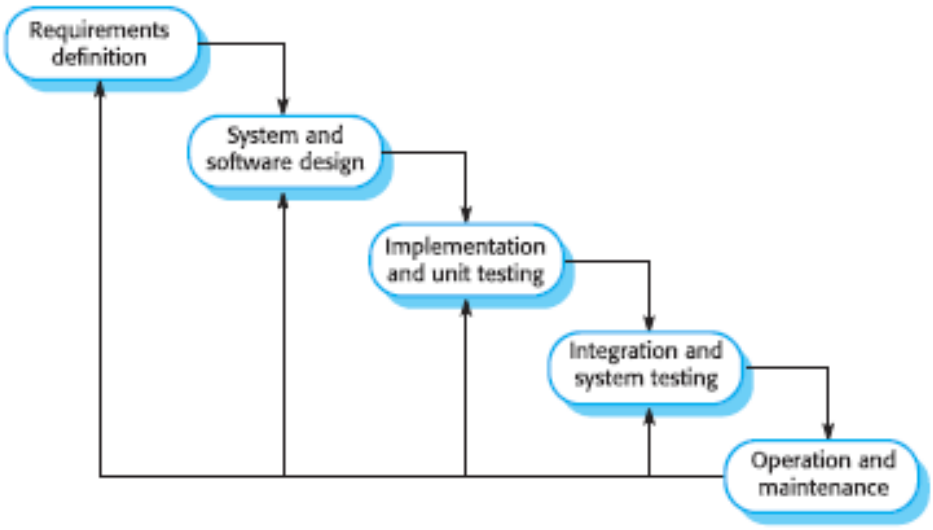


Figure 3. 2 Visualisation of waterfall SDLC model [112]

In the waterfall model, it starts with **requirements definition**, where relating back to the main phases of software engineering, it is under Software Specification. Here, the software's objectives and constraints are formulated. These two elements are then described in detail and will become the software's specification [112].

Second, **System and software design** is performed, where relating back to the main phases of software engineering, it is under the early stages of Software Development. This process will calculate the hardware and software resources required to build the software. It will also establish an overall system architecture [112].

Third, **Implementation and unit testing** is performed, where relating back to the main phases of software engineering, it is under the Software Development and Software Validation. Here, the proposed software is built. While it is built, unit test cases are created and evaluated on the initial version of the software to verify that the software meets its specifications [112].

Forth, **Integration and system testing** is performed, where relating back to the main phases of software engineering, it is under the late stages of Software Development and

Software Validation. The software is compiled and packed when it meets all the specifications set on the first process. If the software comprises of multiple program units, then in this process, all of the program units are integrated into one package to be compiled into one package. After successful testing with the complete compiled software, the compiled software now can be distributed to users [112].

Finally, **Operation and maintenance** is performed, where relating back to the main phases of software engineering, it is under the Software Evolution. With the complete software in practical use by users, maintenance must be performed in order to fix unnoticed errors during its development, improving existing specifications or adding new specifications as user's needs change over time [112].

3.3.2 Incremental model

The second SDLC model is called the **incremental model**. This model is almost similar to that of waterfall model, but rather moving in a linear processes, almost all of the software process or phase in the model are intertwined between each other. This means that if a developer would like to reverse back to a previous phase during the development, it is possible. Furthermore, most of the software process will be performed concurrently and will overlap between each other. The result with using incremental model would be multiple iterations of the software and each iteration is improved upon in development over time [112]. **Figure 3.3** visualise the incremental SDLC model.

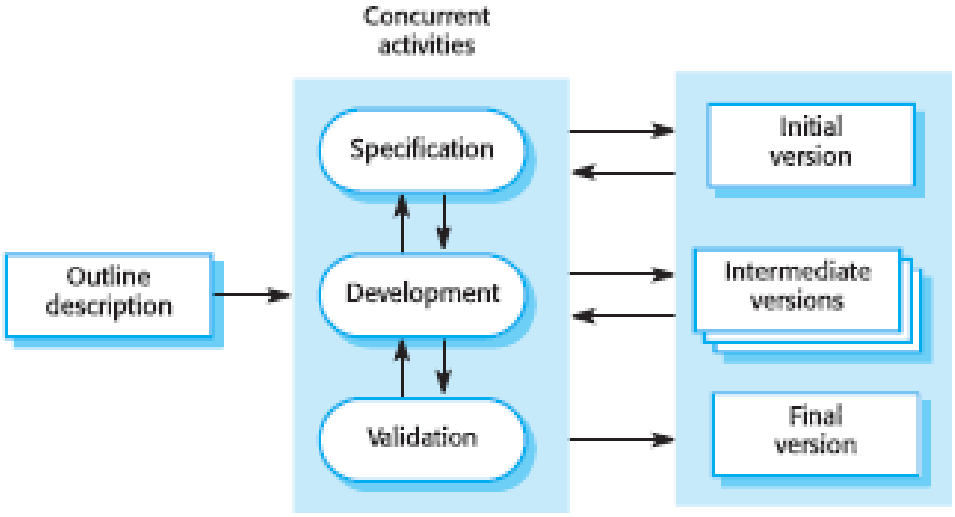


Figure 3. 3 Visualisation of Incremental SDLC model [112]

Incremental SDLC model employs the use of agile-driven software development. Agile approach of software development focuses more on producing solutions for users faster compared to plan-driven i.e. waterfall model. Agile approach also values more from users' feedback during the software's initial testing. Consequently, this allows for a faster re-iteration of the software; the software is able to evolve much faster than software developed with waterfall model [112].

A manifesto by Beck et. al. furthermore supports the values and importance of Agile approach on software development. Below are three out of twelve principles of Agile Manifesto that can be related to this project [113]:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

3.3.3 Comparison between the two models

Starting with waterfall model, its advantage is clear and separate phases during software development. This will also reduce the complexity of the project over time since each phases do not overlap between each other [112].

However, its disadvantage is it's difficult to accommodate requirements or specifications changes request by users once the software is developed. This is because going back to the advantage and the waterfall model fundamentals, once a software development phase is completed, the developers cannot reverse to previous phase, instead only able to move towards next phase. Another disadvantage for waterfall model is limited user feedback where interactions with users are only made in the beginning and at the end of the development [112].

Moving towards the incremental model, its advantage is overall lower cost. This is because changes and fixes to the software can be made in another iteration in the development; compared to waterfall model where this process can only take place at the end of development, therefore increasing cost. Another advantage is earlier user feedback. This is because interaction with users are made throughout the entire timeframe of the phases of the development. Other than that, with concurrent software process in the development, the software can be delivered much faster compared to waterfall model [112].

Incremental model's disadvantage is since its phases are interlinked with each other; the phases become invincible since there are no separation. Also, since software that are developed with this model have a tighter timeframe compared to waterfall model, a working software is prioritized in any way than proper documentations. This results to a messy source code of the software and attempts to refactor the code – fixing or

improving it, will be much more difficult in the future unless it's being done regularly [112].

According to Mishra and Dubey [114], if the requirements are well-defined before the development starts and the developers want full control of the project at all times, then the waterfall model is suitable for the case. Incremental model at the other hand, is suitable for projects that require changes to specifications overtime, with its cyclic interactions in between different iterations of the software. **Table 3.1** shows the comparison between waterfall and incremental model.

Table 3. 1 Comparison between waterfall and incremental model [114]

Features	**Waterfall**	**Incremental**
Requirement specifications	Beginning	Beginning
Cost	Low	Low
Simplicity	Simple	Intermediate
Risk involvement	High	Easily manageable
Expertise	High	High
Flexibility to change	Difficult	Easy
User involvement	Only at beginning	Intermediate
Flexibility	Rigid	Less Flexible

Maintenance	Least	Promotes maintainability
Duration	Long	Very long

In the context of the project, the project use a mixture of both models; taking advantage of each models. This is because well-defined requirements are set in the beginning of development, that waterfall model can be used; but allowing changes and re-iterations during the development, as in the incremental model.

3.4 Project Phases

- **Requirement definition**

In this phase, user requirements and constraints for the project's deliverable are being set. **Chapter 3.5** defined the proposed system's requirements.

- **System and software design**

In this phase, the system's architecture and resources needed to proceed with development are identified. The proposed architecture and resources used for the project is in the next sub-topic, *Hardware and Software resources used for development*. Also, use case diagram, initial GUI prototype and database design diagrams would be specified in Chapter 4.

After this phase, the subsequent phases are cyclic and concurrent, in order to allow re-iteration in the development.

- **Software Development**

In this phase, the software is programmed according to the design set in the previous phase. In this phase also, changes and fixes are made to produce an improved iteration of the software; with errors found during Software Testing.

- **Software Testing**

In this phase, the current iteration of the software is tested with multiple test cases in order to find bugs and errors. If bugs or errors are found, or the software does not pass the test cases, Software Development phase is performed again in order to create another iteration of the software that are improved upon.

- **Software Implementation**

In this phase, the software is tested with near real-world case to ensure that it can run practically in users' environment. If this phase fails, then Software Testing and Software Development phase is performed again in order to identify any errors or bugs encountered during the phase.

- **Delivery and Maintenance**

In this phase, the software is at the final iteration, and ready to be used by users. If any changes need to be made in the future, the phase can be restarted to make new iterations of the software in the future.

3.5 Software Requirements

From Sommerville, Software Requirements can be defined as a detailed explanation of the services that a system should serve and its constraints during operation. Software requirements can be separated to two requirements which is user requirements and system requirements. User requirements can be defined as statements, in words, either broad or precise, of what services that a system should serve to the users. System requirements can be defined as a more detailed description of the software system's functions, services, and constraints during operation [112].

Most crucially, user requirements are more focused on the front-end development and what is happening on the foreground of the system. System requirements are more focused on the back-end development and what is happening on the background of the system.

Also, both these requirements serve different users where user requirements serve the system's users, and system requirements serve the developers and administrators of the system, while also act as support to user requirements.

Table 3. 2 Main features of the application for users

Feature ID	User function	Description / Requirement
UF-001	Register account	Users can register for an account in the app.
UF-002	Login account	Users can log into their account in the app.
UF-003	Search users	Users can search for another user to communicate with, in the platform.
UF-004	Exchange messages	Users can exchange messages with another user as a method of communication, in the platform.
UF-005	Verify users	Users can verify each other with another user to ensure that they are the intended sender or recipient of the messages exchanged, in the platform.

Table 3.2 lists out the main features of the application within user scope. Each feature have their own ID, starting with UF-001 where UF is "User Feature". User function column is the feature for the specific ID and description column is to describe what the feature should do as a requirement.

Table 3. 3 Main features of the application for system

Feature ID	System function	Description / Requirement
SF-001	Store key pairs	System must be able to store all users' asymmetric public key pairs that are needed for communication, in a server; as part of a Public-Key Infrastructure (PKI)
SF-002	Initiate connection	System must be able to establish connection between two users.
SF-003	Perform Key Exchange Protocol	System must be able to initiate Key Exchange Protocol (KEP) in the connection that is established between two users.
SF-004	Perform Message Encryption	System must be able to encrypt and decrypt messages that are sent or receive **BEFORE** leaving the device, using the key derived from the KEP process.
SF-005	Relay messages	System must be able to relay **Ciphertext** (messages that are encrypted) that are exchanged between two users; without the capability to decrypt them in-transit.
SF-006	Verify users	System must be able to show proof of verification on integrity of chat session.

Table 3.3 lists out the main features of the application within system scope. The explanation for **Table 3.3** is the same as **Table 3.2** except here, the ID starts with SF which means "System Function".

3.5.1 Functional Software Requirements

According to Sommerville, Functional Software Requirements refers to description of services of the system should serve, how the system should interact with different inputs, and the system should behave in different situations [112].

Table 3. 4 Functional Requirements of application for users

Feature ID	Req. ID	User function	Description / Requirement
UF-001	UF-001-001	Enter username in register	Users can enter their username in the register page.
	UF-001-002	Enter password in register	Users can enter their password in the register page.
UF-002	UF-002-001	Enter username in login	Users can enter their username in the login page.
	UF-002-002	Enter password in login	Users can enter their password in the login page.
UF-003	UF-003-001	Type username	Users can type in the username of the person they want to message.
UF-004	UF-004-001	Type messages	Users can type in their messages in a chat box.
	UF-004-002	Send messages	Users can send messages by clicking the send button.
	UF-004-003	Receive messages	Users can receive messages from another user.
UF-005	UF-005-001	Sign messages	Users can sign each messages sent from their behalf.
	UF-005-002	Verify messages	Users can verify each messages that they receive from another user.

Table 3.4 shows the Functional Requirements of the application within user scope. From the main user's software requirements, it is broken down into separate actions that fulfil the main requirements; under "Feature ID" column. Each of these separate actions are given an ID which is under "Requirement ID" column; starting with UF-001-001.

Table 3. 5 Functional Requirements of application for system

Feature ID	Req. ID	System function	Description / Requirement
SF-001	SF-001-001	Receive public keys	The system (server) must be able to receive public keys sent by users (client).
	SF-001-002	Receive signatures of keys	The system must be able to receive signatures sent by users (client).
	SF-001-003	Store keys in database	The system must be able to insert new entry into the database, containing the keys, signatures, and other user details.
SF-002	SF-002-001	Connect with clients	The system (server) must be able to accept connection coming from users (client).
SF-003	SF-003-001	Generate key pairs	The users (client) must be able to generate key pairs.
	SF-003-002	Generate signatures	The users (client) must be able to generate signatures of the key pairs.

	SF-003-003	Relay public keys	The system (server) must be able to relay public keys of user to another user in the session.
	SF-003-004	Generate shared secret key	The users (client) must be able to derive a shared secret key using the public keys received from server, with user's private keys.
SF-004	SF-004-001	Encrypt messages	The users (client) must be able to encrypt messages sent to another user using the shared secret key derived from the KEP.
	SF-004-002	Decrypt messages	The users (client) must be able to decrypt messages received from another user using the shared secret key derived from the KEP.
SF-005	SF-005-001	Broadcast messages	The system (server) must be able to broadcast the messages from another user to all users (clients) in the session.
SF-006	SF-006-001	Verify Sender	The system (server) must be able to verify sender using sender's public key.
	SF-006-002	Verify Receiver	The system (server) must be able to verify receiver using receiver's public key.

Table 3.5 shows the Functional Requirements of the application within system scope. From the main system's software requirements, it is also being broken down into

separate actions that fulfil the main requirements; under “Feature ID” column. Each of these separate actions are given a Requirement ID, starting with SF-001-001.

3.5.2 Non-Functional Software Requirements

From Sommerville, Non-Functional Software Requirements can be defined as constraints on the services served by the system [112]. Non-functional software requirements specify the requirements or constraints of the system as a whole. If a non-functional software requirement is not met, the whole system would be unusable or will not operate correctly. Additionally, non-functional software requirements usually uses quantitative metrics in order to specify constraints especially on system’s resources.

Non-functional requirements can be categorized into three main requirements which are Product requirements, Organizational requirements, and External requirements [112].

Product requirements focus on runtime behaviour of the software. Under this category, there are sub-categories of requirements e.g. performance, space, security etc. [112]

Organizational requirements are derived from policies, procedures and standards in the user’s or developer’s organizations. Few example of this requirements would be how the system is intended to use, what programming languages or technology architecture should be used in development, and others [112].

External requirements focus on external governing bodies e.g. regulatory requirements, law requirements and ethical requirements [112]. **Figure 3.4** visualises the full diagram on categorization of non-functional requirements.

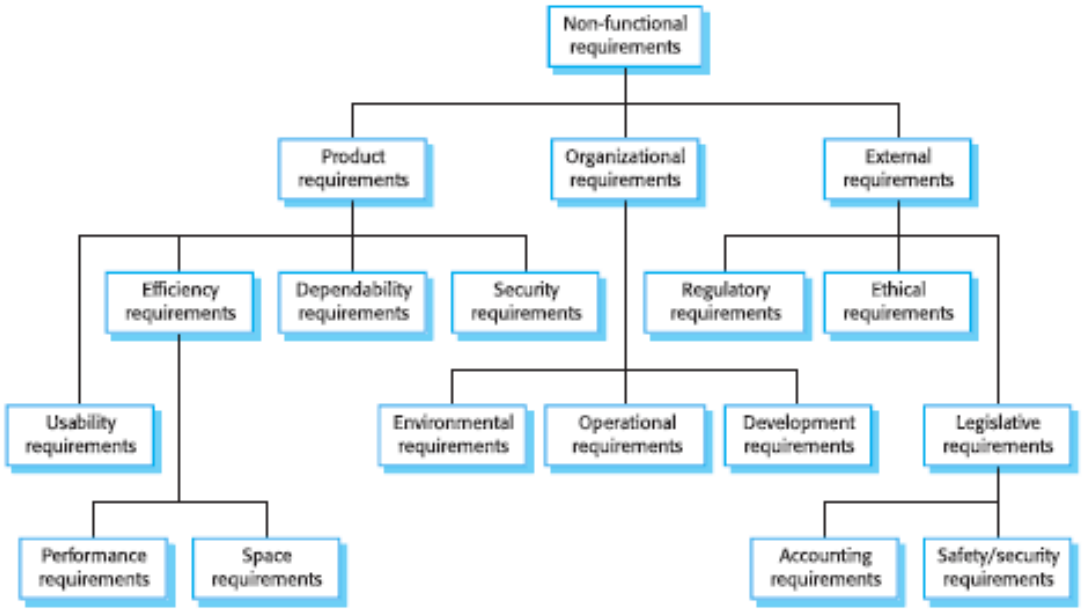


Figure 3. 4 Categories of Non-Functional Requirements [112]

Table 3. 6 Non-Functional Requirements of application

Req. ID	Non-Functional Classification	Description / Requirement
NF-001	Usability requirements	The system must be feasible to be used by anyone with a modern computer.
NF-002	Performance requirements	The system must be able to run on low to medium-spec modern computers.
NF-003	Performance requirements	The system must NOT require a downtime more than 10 seconds.
NF-004	Performance requirements	The system must be able to respond to a user input within 5 seconds.
NF-005	Space requirements	The system must NOT require more than 2GB of RAM and 100MB of hard disk space.

NF-006	Security requirements	The system must be able to perform all cryptographic algorithms that are programmed successfully.
NF-007	Operational requirements	The system must have Internet connectivity in order to function.
NF-008	Operational requirements	The system must have at least ONE server and ONE client running in order to function.
NF-009	Operational requirements	The system (both client AND server) must NOT require more than 5 seconds to perform a cryptographic algorithm.
NF-010	Development requirements	The system must be programmed in Python OR Java.
NF-011	Regulatory requirements	The system must disclose to all users that it implements a cryptographic software; which may be illegal to use depending on where user resides.
NF-012	Regulatory requirements	The system must remind all users that the use of the system is only permitted to only actions that are legal under the law.

Table 3.5 shows the Non-Functional Requirements of the application. Each non-functional requirements are given an ID under “Requirement ID” column, starting with NF-001. “NF” here stands for “Non-Functional”. “Non-Functional Classification”

refers to the requirement's category under the non-functional requirement's category; refer to **Figure 3.3**. "Description / Requirement" column describes each requirement's specification.

3.6 Hardware and Software resources used for development

In this sub-chapter, are listed among the hardware resources and software resources such as application or architecture used to develop and implement the deliverable of this project. Do note that some of the resources listed below may not be used in the development. Also note that there may be resources used in the development that are not listed below.

3.6.1 Hardware resources

- A personal laptop with Windows Operating System, 16GB RAM, 500GB SSD storage device with 56GB of available storage space; to become the platform to design and program the software physically and also to store all files and documentations regarding the software. This hardware will also be the platform to host the server.

  If required during testing, TWO physically separate personal laptop with similar specifications will be used, mimicking the scenario of TWO clients connecting to the server.

- Internet connectivity, to allow software in different machines to connect between each other. For the purposes of testing, the University's Local Area Network ("LAN") is used and only devices in the network can access the server, since it is not hosted on the Internet.

3.6.2 Software resources

- Microsoft Visual Studio Code (“VSC”) OR Apache NetBeans Integrated Development Environment (“IDE”) to design and program the software virtually in the laptop. [115], [116]

- Java OR Python as main programming language for the software. [117], [118]

- MySQL server specifically Python sqlite3 library to store asymmetric key pairs and signatures of these keys of users. For the purpose of testing, the database is hosted locally and stored on-disk. [119], [120]

- Java’s javax.crypto class OR Python pycryptodome library, for handling all operations regarding symmetric encryption – AES and hashing – SHA-3. Pycryptodome is also used for asymmetric encryption – ECC. [121]

- Python Argon2 library for storing passwords in the database in a secure manner; implementing the use of salt and slow hashing in the password hashing operation. [122]

- Python py2crypt library for handling all operations regarding conventional Key Exchange Protocol – ECDH. [123] This library can also be substituted for pycryptodome library for symmetric encryptions.

- Python kyber-py library for handling all operations regarding Post-Quantum Encryption and as a component for PQXDH implementation in the system.[124]

- Python dilithium-py library for handling all operations regarding Post-Quantum digital signature scheme.[125]

If CRYSTALS-Dilithium cannot be implemented in the system, then Python pycryptodome library will be used as a fallback, implementing Deterministic EdDSA. [121]

To conclude on this chapter, by using knowledge gathered in Chapter 2 and implementing Software Engineering Principles and resources for the project, the author believes that the project's problem statements can be solved practically within software development.

CHAPTER 4

DESIGN

4.1 Introduction

This chapter covers the overview of proposed design and flow of the developed system for both system flow and the cryptographic suite selection implementation.

4.2 System's flowchart

Figure 4.1 below visualises the system's intended behaviour based on user input, laid out on a flowchart. Note that this flowchart only represents the foreground's behaviour.

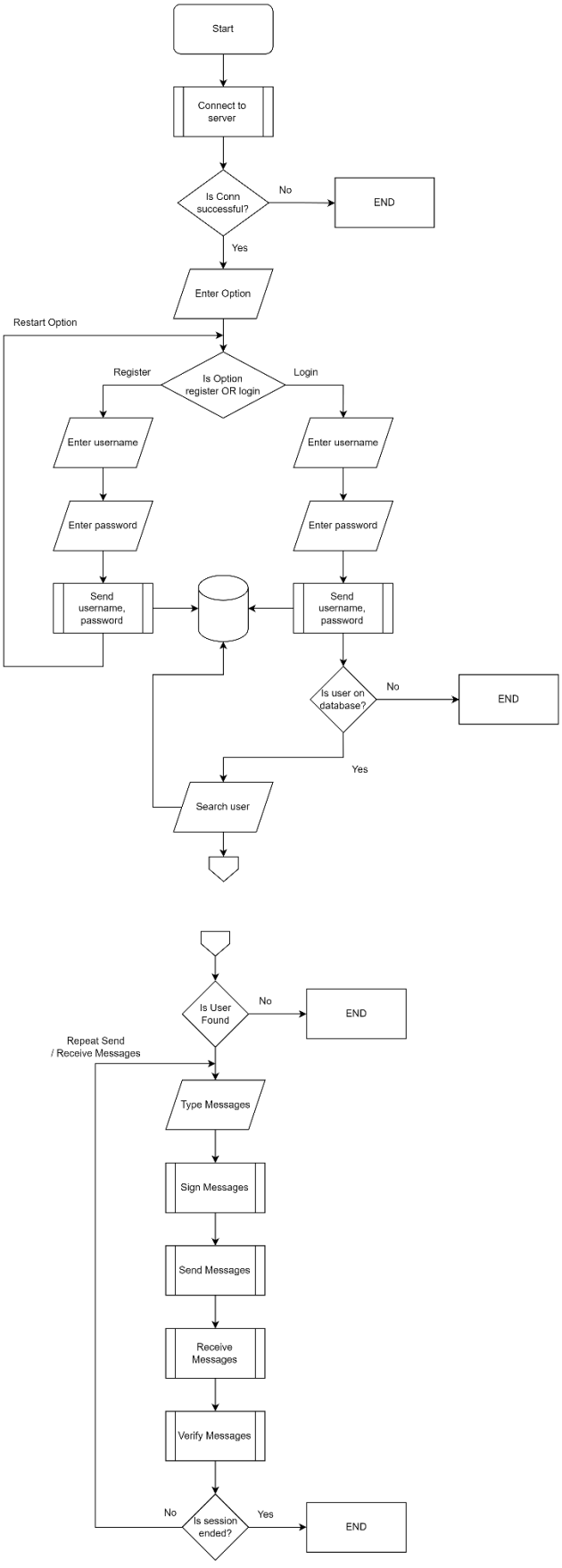


Figure 4. 1 Flowchart of system (client)

Figure 4.2 below visualises the system's intended behaviour on the back-end, which is the server.

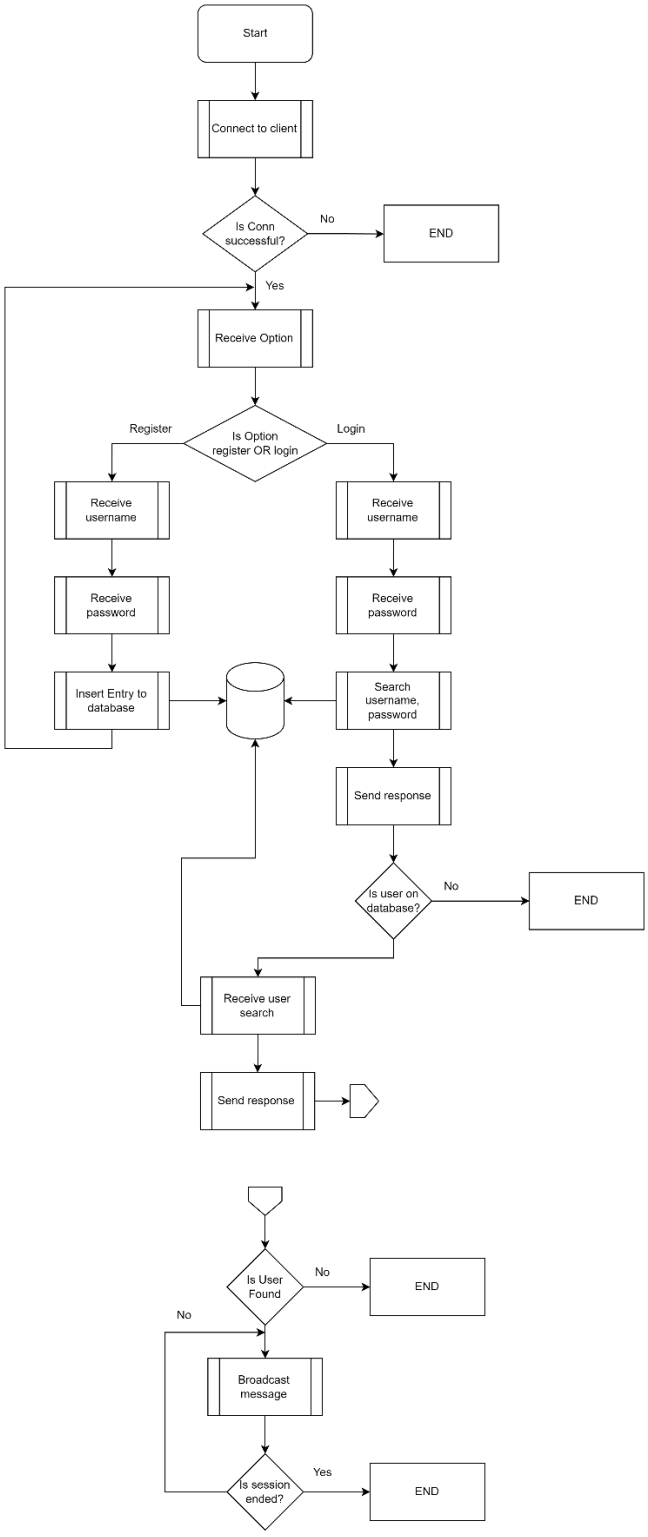


Figure 4. 2 Flowchart of system (server)

Figure 4.3 below visualises the proposed system's simplified view on all cryptographic algorithms used for each major process. Note that this flow is not final and may change during development.

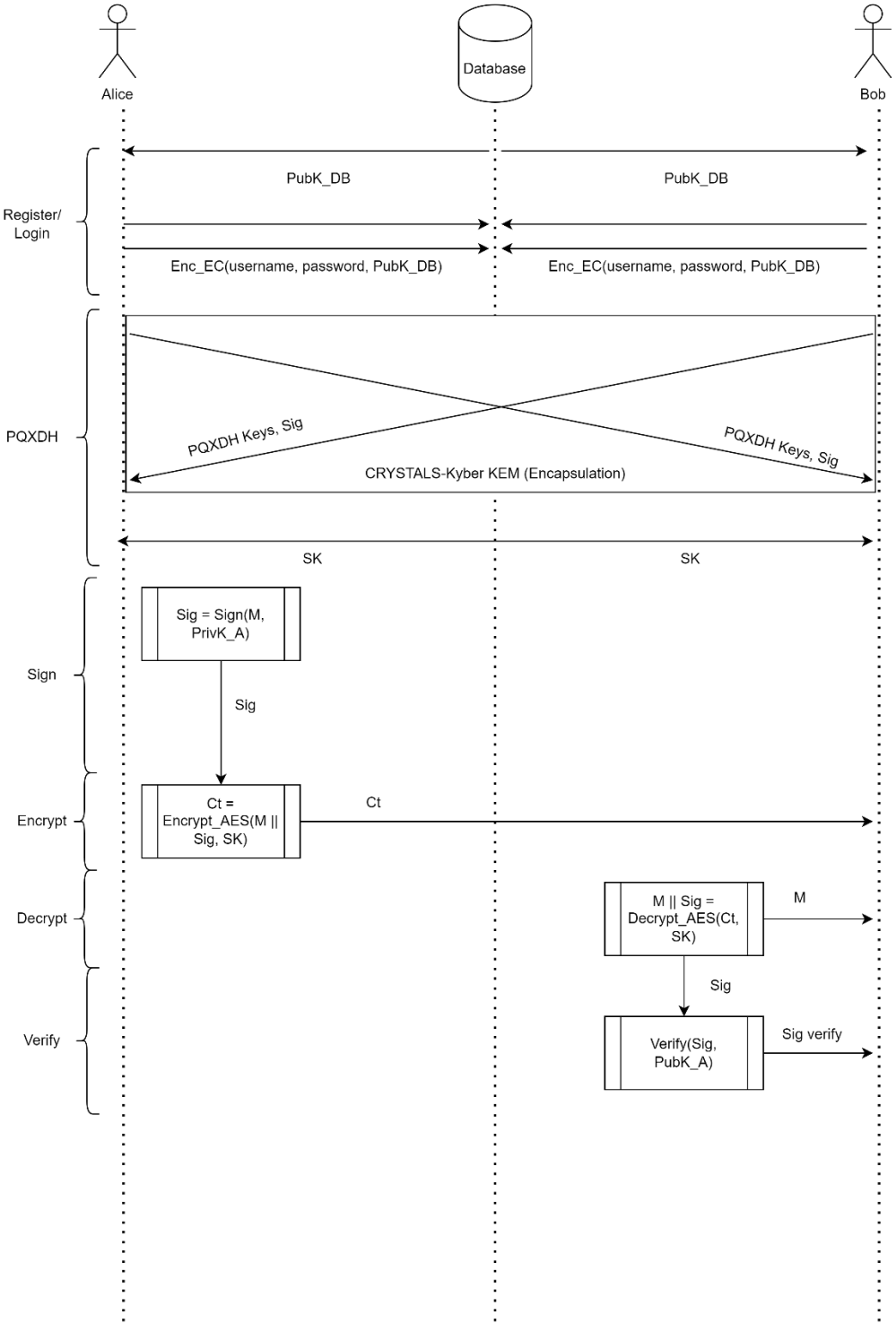


Figure 4. 3 Simplified flow on system's cryptographic algorithms

4.3 System's Entity Relationship Diagram

Figure 4.4 below visualises the proposed system's database design in an Entity Relationship Diagram ("ERD"). Note that the secondary keys or columns in the tables are not final and subjected to change.

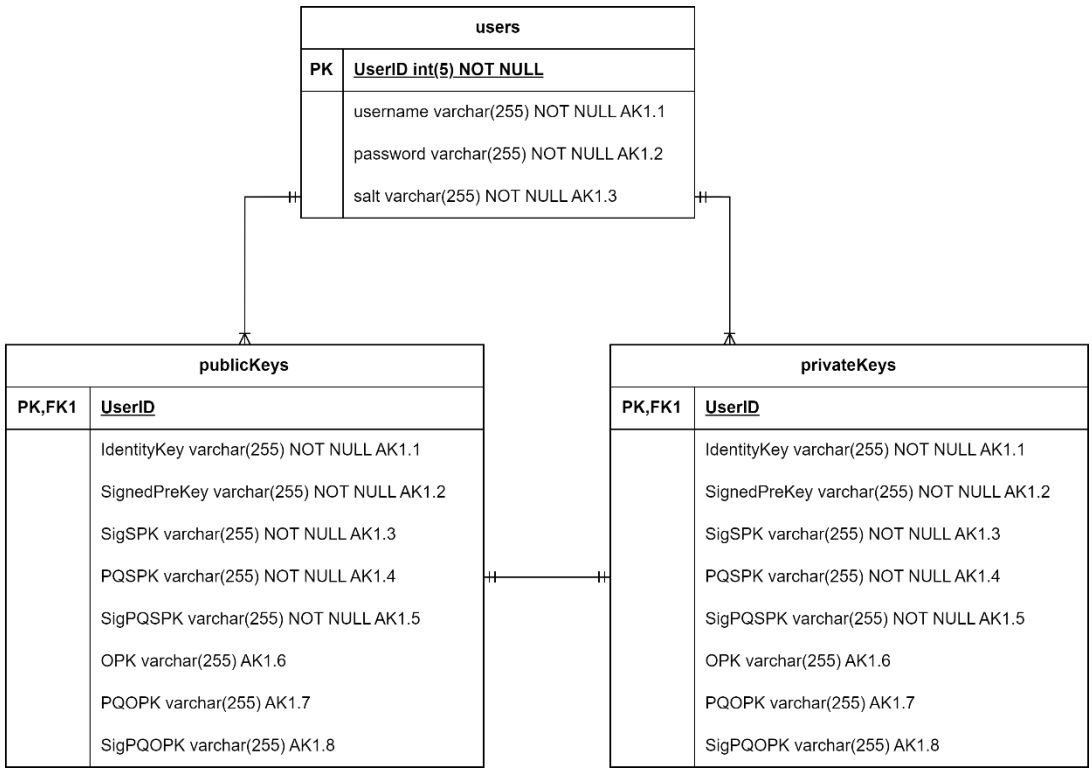


Figure 4. 4 System's Entity Relationship Diagram

The system's database consists of three tables named $users, publicKeys, privateKeys$. The $users$ table will store the username and password (in hash digest format) as well as the salt to be used to derive the hash of the password. The $publicKeys$ table contains all the public keys and signatures necessary to perform the KEP process; refer back to PQXDH's key generation. In this table also, a foreign key is used which is the $UserID$ from the users table to link all the keys to the owner of the key. Similarly, $privateKeys$ table will store all private keys of the key pair generated from the KEP process. It will also be tied to the owner of the key using the $UserID$ values from $users$ table.

It is important to note that all values to be entered into $privateKeys$ table must be encrypted and can only be decrypted using the user's password. Another consideration of security is to that fully eliminate the use of $privateKeys$ table and require all private key pairs to be stored locally in user's device. However, this could add space constraints to user side.

4.4 System's design in Graphical User Interface

In this sub-chapter, the initial wireframe design or sketches of prototype that are imagined or visualized by the author are placed. User Interface ("UI") and User Experience ("UX") design in the prototype are largely influenced by current existing instant messaging applications such as WhatsApp and Telegram [126], [127]. Also, some UI designs are also inspired by Tox [128]. All figures of prototypes in this sub-chapter are built using ProtoPie [129].

4.4.1 Register user

Figure 4.5 below visualises the Home Page, where it is the page that the user will first see upon launching the application. The Home Page will feature a welcome message, a system's description and two boxes to either register as a new user or login.

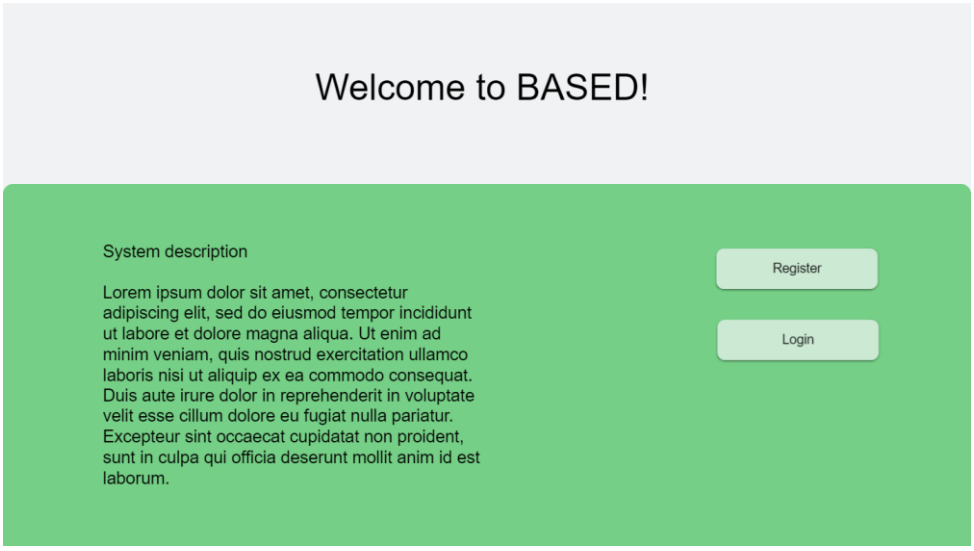


Figure 4. 5 Home Page of prototype

Next, if the user proceeds to register as a new user, a Register page will appear upon clicking the “Register” box in the homepage. **Figure 4.6** visualises the Register page.

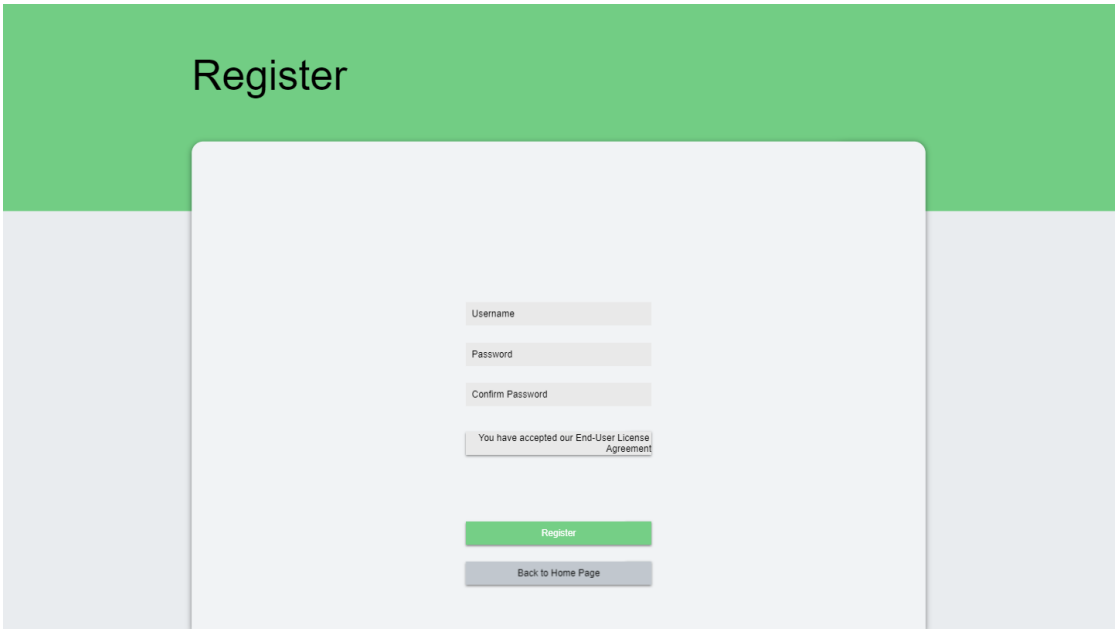


Figure 4. 6 Register page of prototype

The register page will ask users for username and password. Additionally, the same password will need to be re-typed in the “Confirm Password” textbox. A tick box for agreement of End-User License Agreement (“EULA”) between the system and users need to be ticked before user can register for an account. Finally, users can click on the “Register” box to finish registering an account. At this stage, the system will re-direct users to the Home Page in order for users to login with the account that just registered.

4.4.2 Login as user

Figure 4.7 below visualises the Login page. The login page will ask users for their username and password. The user will need to click on the “Login” box in order to log in.

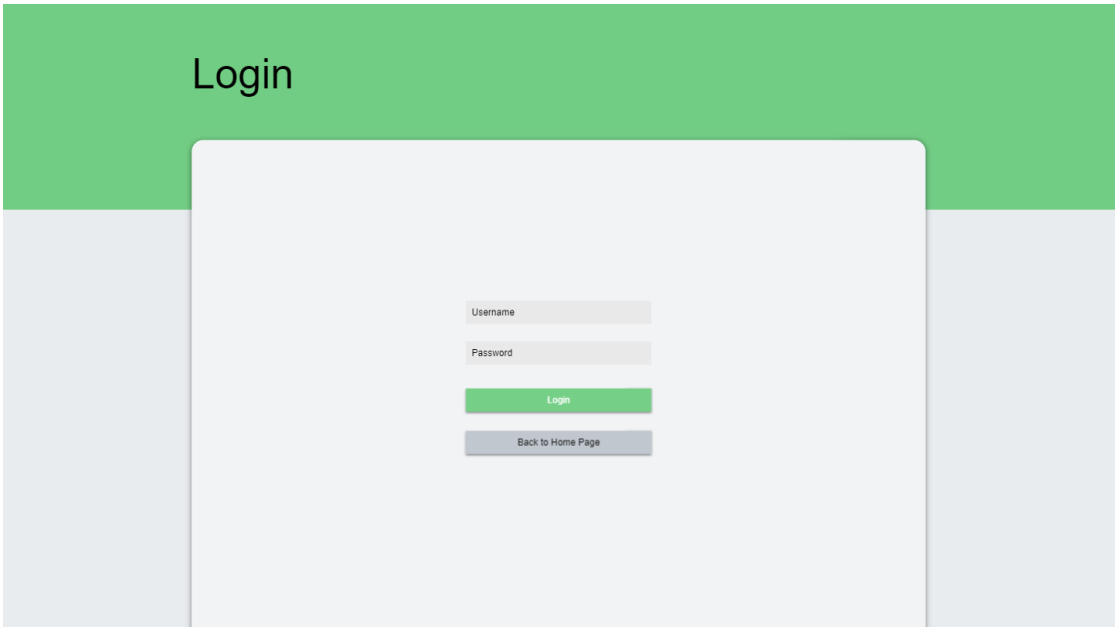


Figure 4. 7 Login page of prototype

After successfully logging in, users will be directed to their Mailbox page, where they can access their chats there. **Figure 4.8** visualises the Mailbox page. The mailbox page contains user's saved contact list in the left side, and an instruction on the right side where it will ask users to select any contacts to start messaging.

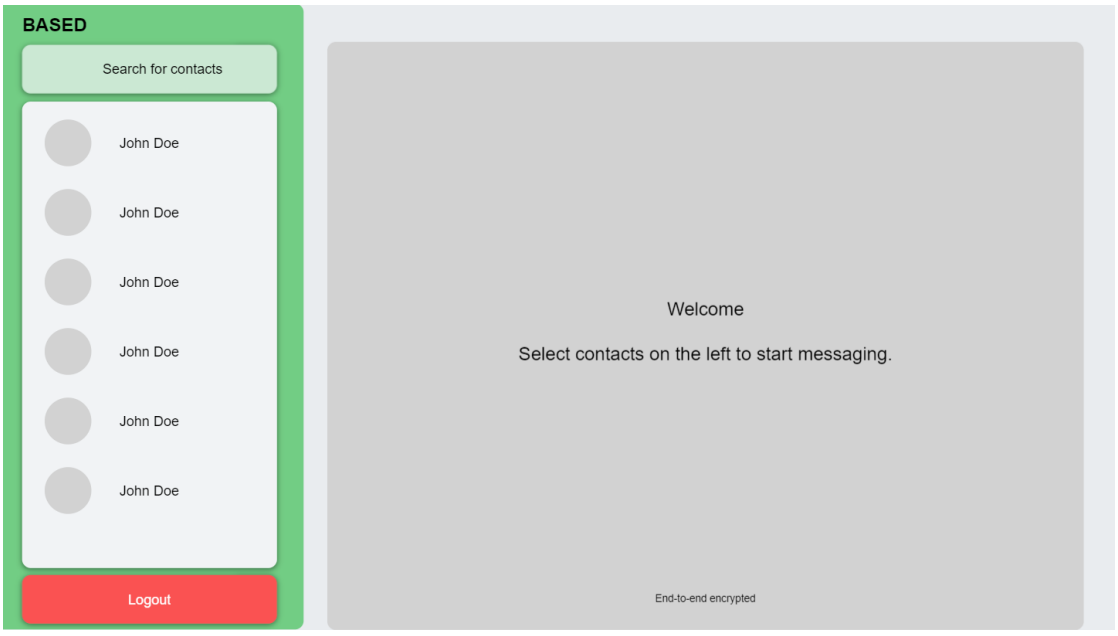


Figure 4. 8 Mailbox page of prototype

4.4.3 Search user

For users to search another user in the application, first the user must click the “Search for contacts” box on the top left side in the Mailbox page. Then the system will prompt user to enter the username that the user wants to exchange messages with. **Figure 4.9** visualises this process.

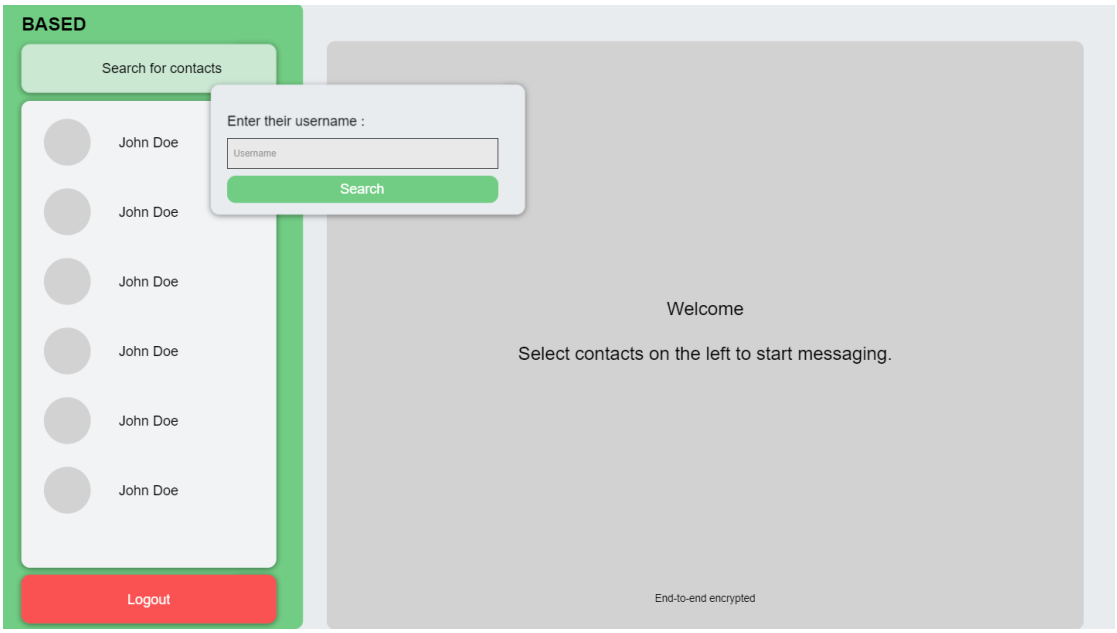


Figure 4. 9 Searching for contacts in prototype

If the user searched is found, then the system will ask user to verify that is the public key of that user belongs to the person using the key’s fingerprint. **Figure 4.10** visualises this process.

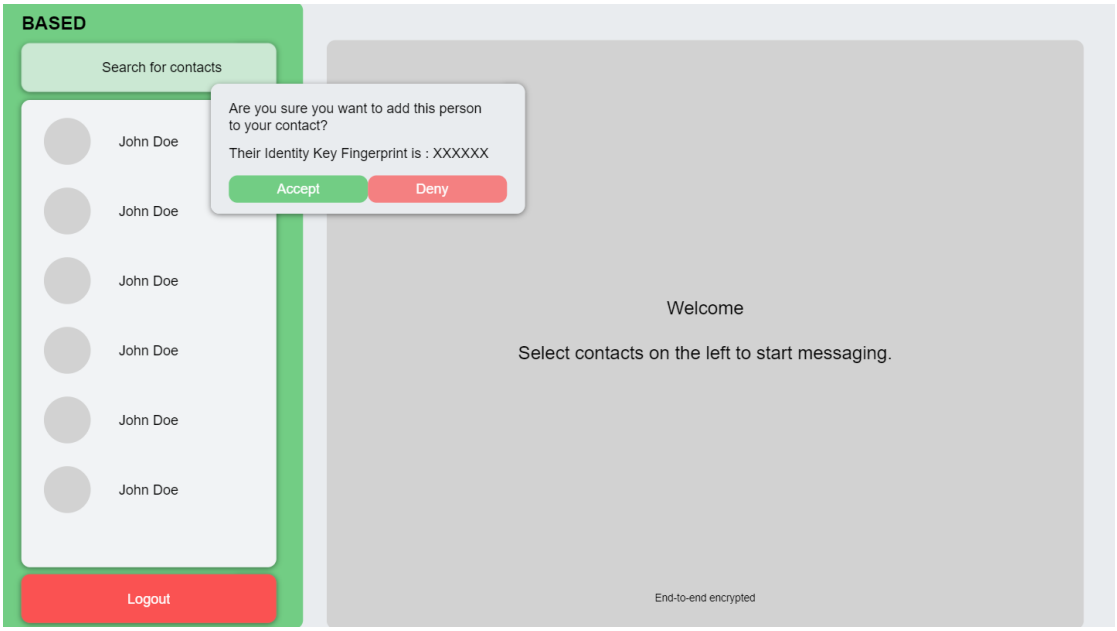


Figure 4. 10 Verification of new contact in prototype

If the user confirms that the key belongs to the person, they can add that person by clicking the “Accept” button; else they can reject the process by clicking the “Deny” button.

The newly added contact will be put in the contact list on the left side of the Mailbox page.

4.4.4 Send and receive message

To send message to another user, first the user will need to click on the user that they wish to message in their contact list. The right side of the Mailbox page will transform to provide space for typing messages, as well as showing conversation history. **Figure 4.11** visualises this process.

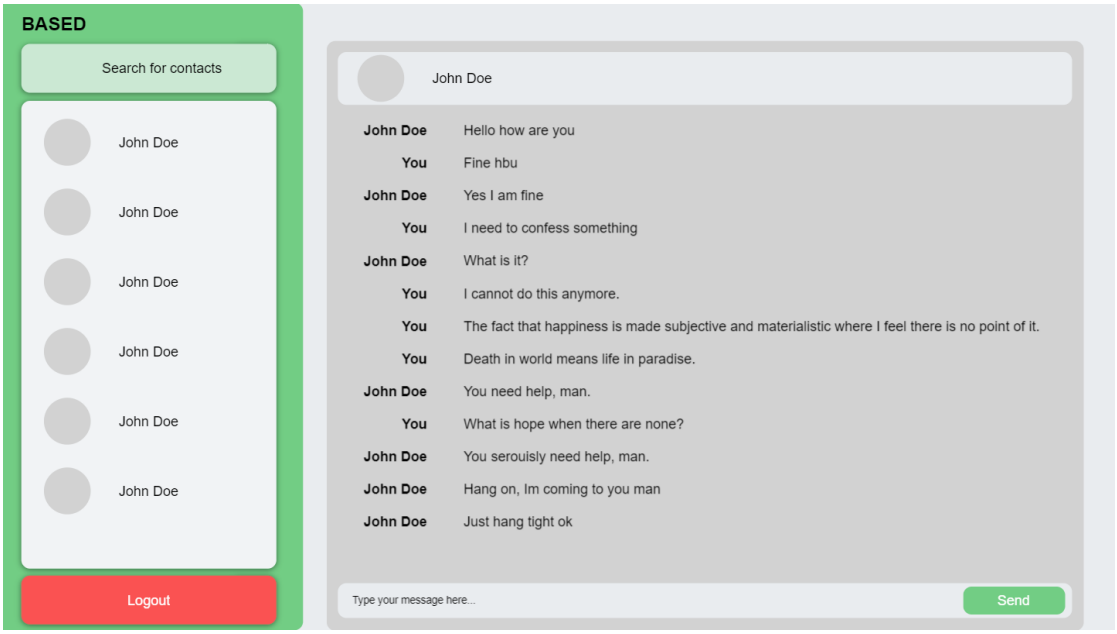


Figure 4. 11 Messaging space in prototype

The top section will display the recipient of the message i.e. another user in the chat session. The middle section will display the conversation history between the two users, and the bottom section will have a text box for user to type their message. Additionally, there is a send button next to the text box for user to send their message.

New messages from recipient to the original sender will appear along the conversation history at the middle section.

4.4.5 Verify user and messages

To verify user that they are exchanging messages with, first user will need to click the information icon to the right side of the top section of the messaging section. A popup box will appear and details of user verification will be displayed. Additionally, messages will be verified in the background and if all messages received can be verified, it would show the verification there as well. Moreover, users can also verify the integrity of the chat session by comparing the shared security code that are also displayed there. **Figure 4.12** visualises this process.

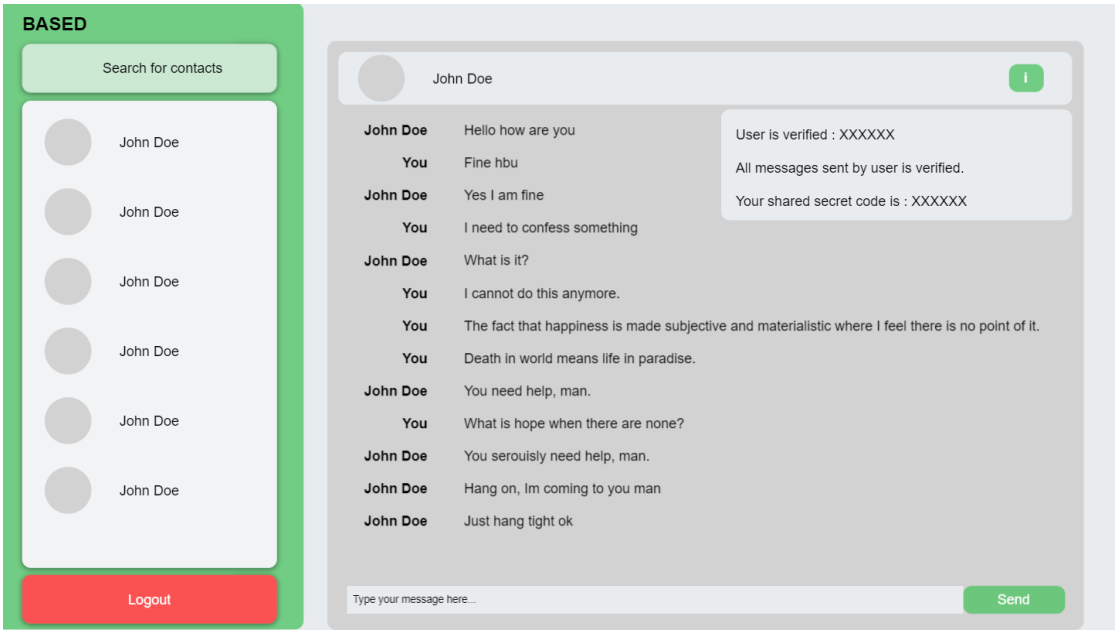


Figure 4. 12 User and session verification in prototype

At this stage, both users can continue to exchange messages securely, until the users decide to end the conversation. Users can either close the application or click the “Logout” button. This will disconnect the client from the server and effectively ends the application.

To conclude on this chapter, system architecture and design need to be planned beforehand to produce a working prototype during development based on designs made in this chapter; while implementing solutions in Chapter 3.

CHAPTER 5

IMPLEMENTATION

This chapter illustrates the flows and implementation of the proof-of-concept with the cryptographic suite selection fully integrated.

5.1 Libraries used during development

After considerations and testing of multiple libraries stated in **Chapter 3.6**, these are the libraries that was used in the development of the project:-

- **Python** as main programming language [117]
- **Tkinter** as main Python GUI library [130]
- **Socket** for handling all communication through the network [131]
- **Sqlite3** for storing all user's accounts and keypairs [119]
- Symmetric Encryption (both CBC and AEAD scheme) : **Pycryptodome** v 3.20.0 – AES [121]
- Asymmetric PKI and KEP : **Pycryptodome** v 3.20.0 - ECC, ECDH [121]

- Hash funct and Key Deriv. funct : **Pycryptodome** v 3.20.0 - SHA512, SHAKE128 [121]

- Password hashing and salting : **hashlib** and **secrets** [132], [133]

- Conventional Digital Signature : **Pycryptodome** v 3.20.0 – EdDSA [121]

- Post-Quantum KEM : **kyber-py** v 0.2.2 - Kyber1024 [124]

- Post-Quantum Digital Signature : **pypqc** v 0.0.6.2 - dilithium5 [134]

Note that dilithium-py was substituted with pypqc library. This is because during development, serialization and deserialization of the public and private keys from the key generation of dilithium-py encountered some fatal errors and cannot be reliably sent over the network to be stored in system's database.

5.2 System's Entity Relationship Diagram

There are a few changes from the proposed system's Entity Relationship Diagram in **Chapter 4.3 Figure 4.4**. The $publicKeys$ and $privateKeys$ table are merged into one table named $keys$; with addition of few fields. Other than that, the developed system's Entity Relationship Diagram remain unchanged. The latest developed system's Entity Relationship Diagram are visualised below.

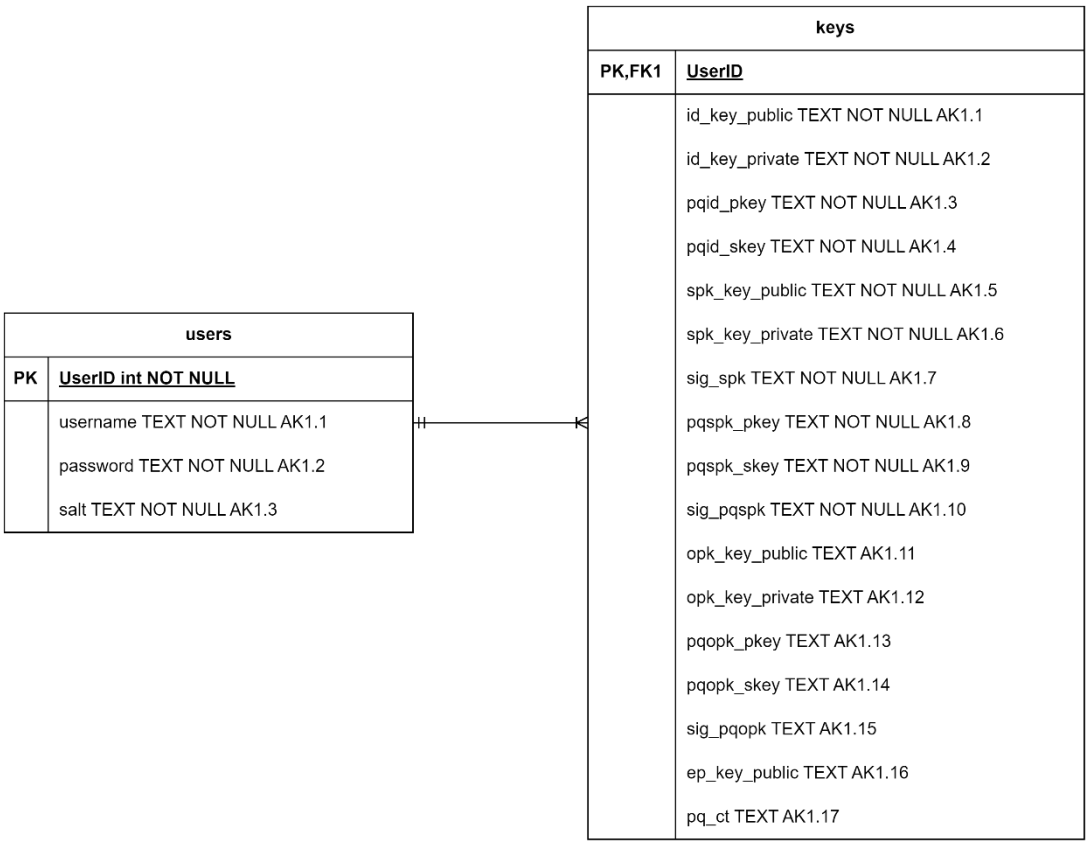


Figure 5. 1 Developed System's Entity Relationship Diagram

5.3 System's flowchart

Below is a high-level overview of the cryptographic implementation of the system. Comparing to system's flowchart in **Chapter 4.2, Figure 4.3**, some major tweaks were made in order to enhance the security of the application ; mainly on initial contact between clients and server.

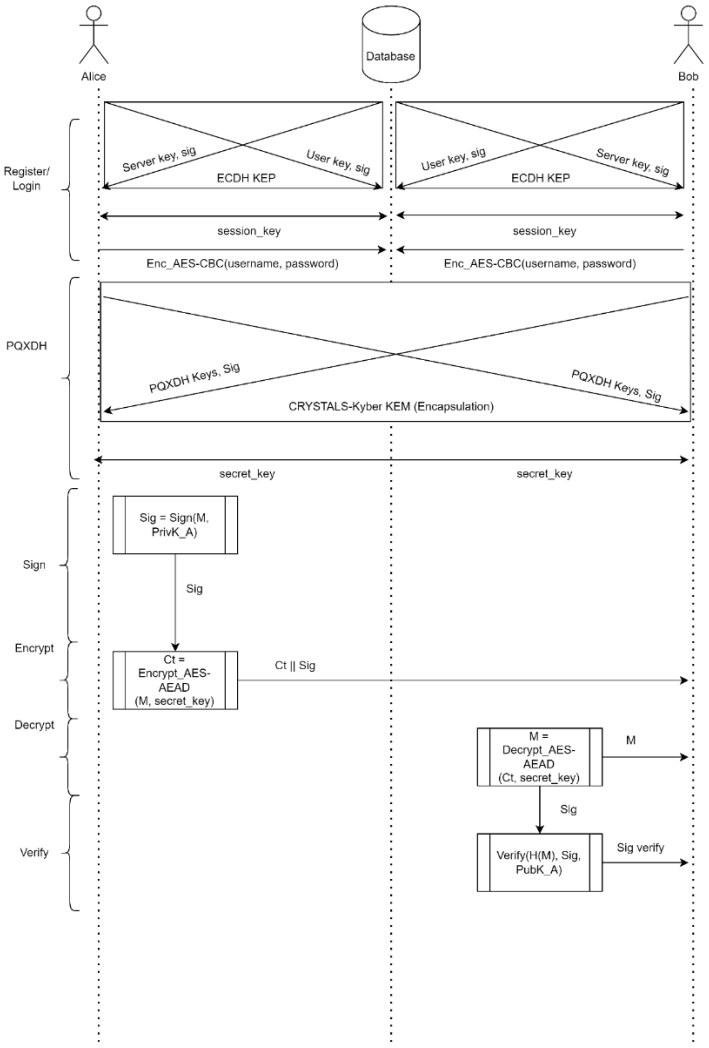


Figure 5. 2 High-level overview of developed system

In **Figure 5.2**, the latest flowchart of the developed system is shown. In "Register/Login" section, instead of using ECC Asymmetric Encryption, ECDH KEP is performed between each client and server. In result, for each client and server will share a session key for each session.

Username and password of the client will be sent to the server over the network with end-to-end encrypted. The credentials are encrypted on client before sending and will be decrypted on server after receiving. The encryption used is AES in CBC operation mode, with the session key generated from the ECDH KEP operation earlier as the key.

Another change was made in this flowchart was encrypting messages. The encryption used is AES in EAX mode, which is conforming to AEAD scheme, with secret key generated from PQXDH operation as key.

5.3.1 System's flowchart on register

Figure 5.3 below shows the system's flowchart on user registration.

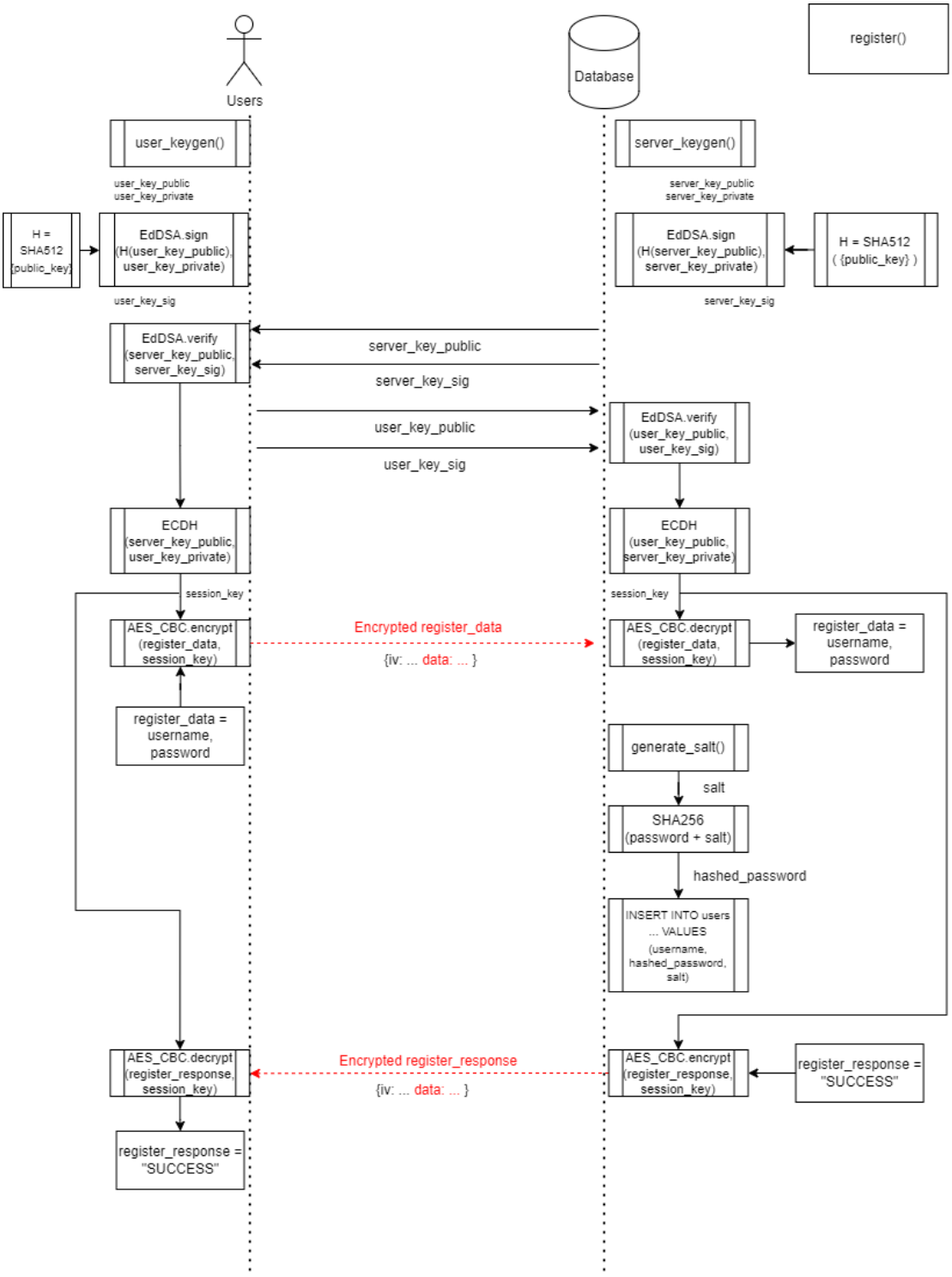


Figure 5. 3 Developed System's flowchart (register)

When the client and server starts, one keypair and signature is generated. When the client connects, both client and server exchanges their public keys and signature. Both

client and server verifies the signature against the public keys. If the verification is successful, key exchange protocol is performed and in result, a session key is generated.

Client will then encrypt the username and password before sending to server using AES CBC. The data will be packed in a JavaScript Object Notation ("JSON") format [135] which comprises of the IV that is generated randomly during cipher's initialization, and the encrypted data itself. Server will then receive the encrypted packet and decrypt the packet with the session key generated from the server-side.

Once the server decrypts the user's credentials, it will then proceed to generate a random 16-bytes salt to be hashed together with the password using SHA512. Finally, an entry is inserted into the database; into $users$ table, which are comprised of username, the hashed password together with salt, and the salt itself.

With adding salt in password hashing, in case of a threat actor manage to obtain the data in the table, the password data is protected against dictionary attack.

Finally, the server will send a response to the client to notify if the registration process is successful. This response is also encrypted and decrypted with the same method and key ; however with a different IV.

During the registration process, the initialization of PQXDH operation is done, where the client will generate all of the keypairs and signatures required for the operation. The initialization of PQXDH during register is visualised below.

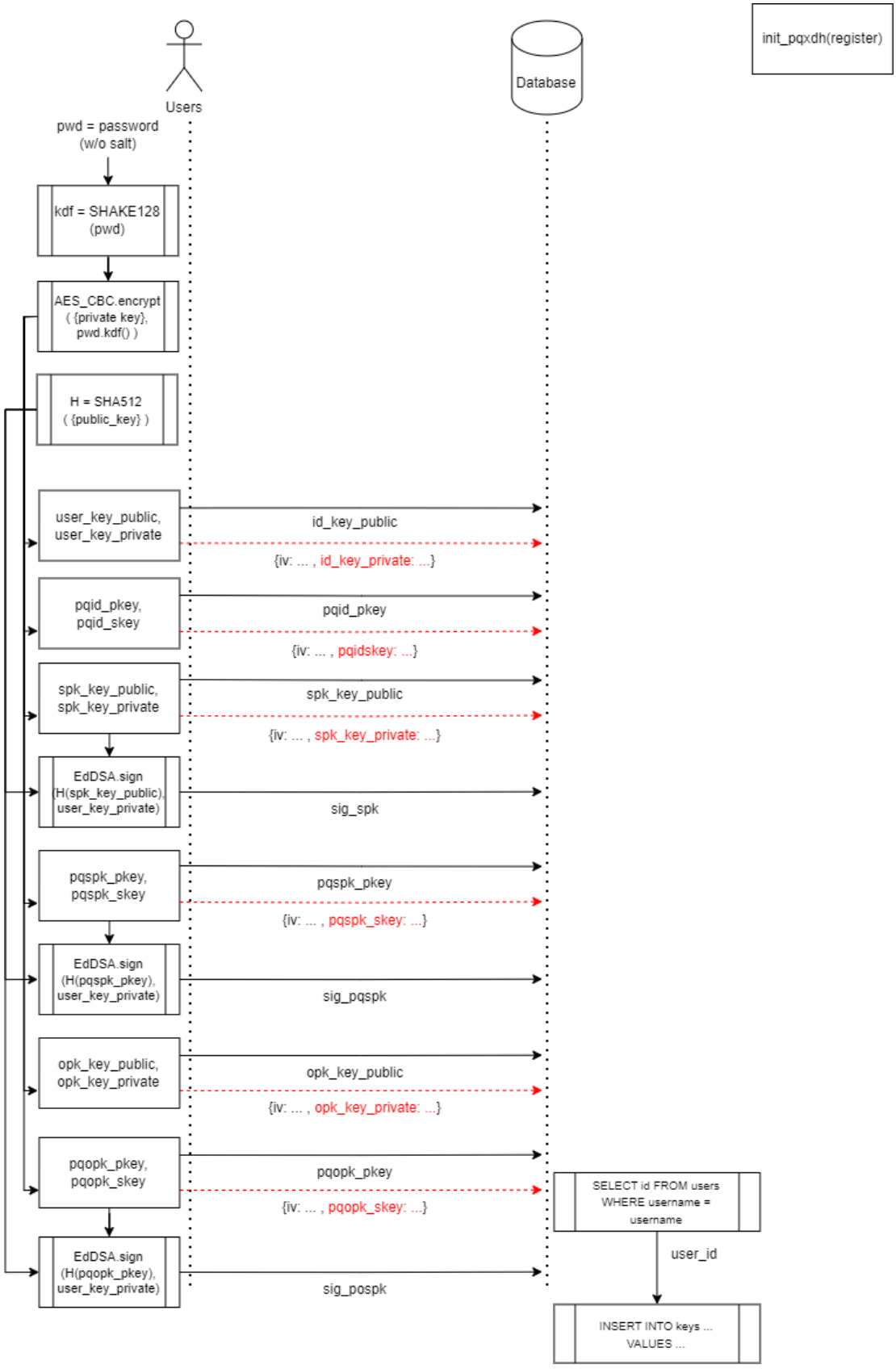


Figure 5. 4 Developed system's flowchart (PQXDH init. Register)

In the system's implementation of PQXDH initialization, all of the public keys and signatures are sent in plaintext; however with private keys, they are encrypted with AES CBC before sending them to server. The password from the user are then hashed using SHAKE128 as a Key Derivation Function ("KDF"). The digest from the KDF is used as the key to encrypt and decrypt the private keys. The generation of keys in the system follows as closely as possible to the PQXDH paper [84].

In the system's implementation of PQXDH, $ed25519$ EC curves is used instead of PQXDH's recommended curve which is $curve25519$. This is because there are no available libraries in Python that can reliably serialize and deserialize $curve25519$ keypairs to be sent over the network, at the time of development. This is also because $ed25519$ is birationally equivalent to $curve25519$, and it could theoretically be used to substitute $curve25519$ [136]. This also applies to key signing where in the system, EdDSA is used to sign the keys instead of using XEdDSA according to the PQXDH's paper.

It is important to note that even though the server receives the password in plaintext after the credential's decryption, it will never be used in any attempt to decrypt any of the encrypted private keys stored in the database. The value of the password in plaintext always hashed together with salt on server-side. Only client is able to perform the KDF using the password value without the salt in order to decrypt the private keys.

Finally, the server will receive all keypairs and signatures and insert a new entry into the database; into $keys$ table according to the user's ID.

5.3.2 System's flowchart on login

The process when a user log in to the system is visualised below.

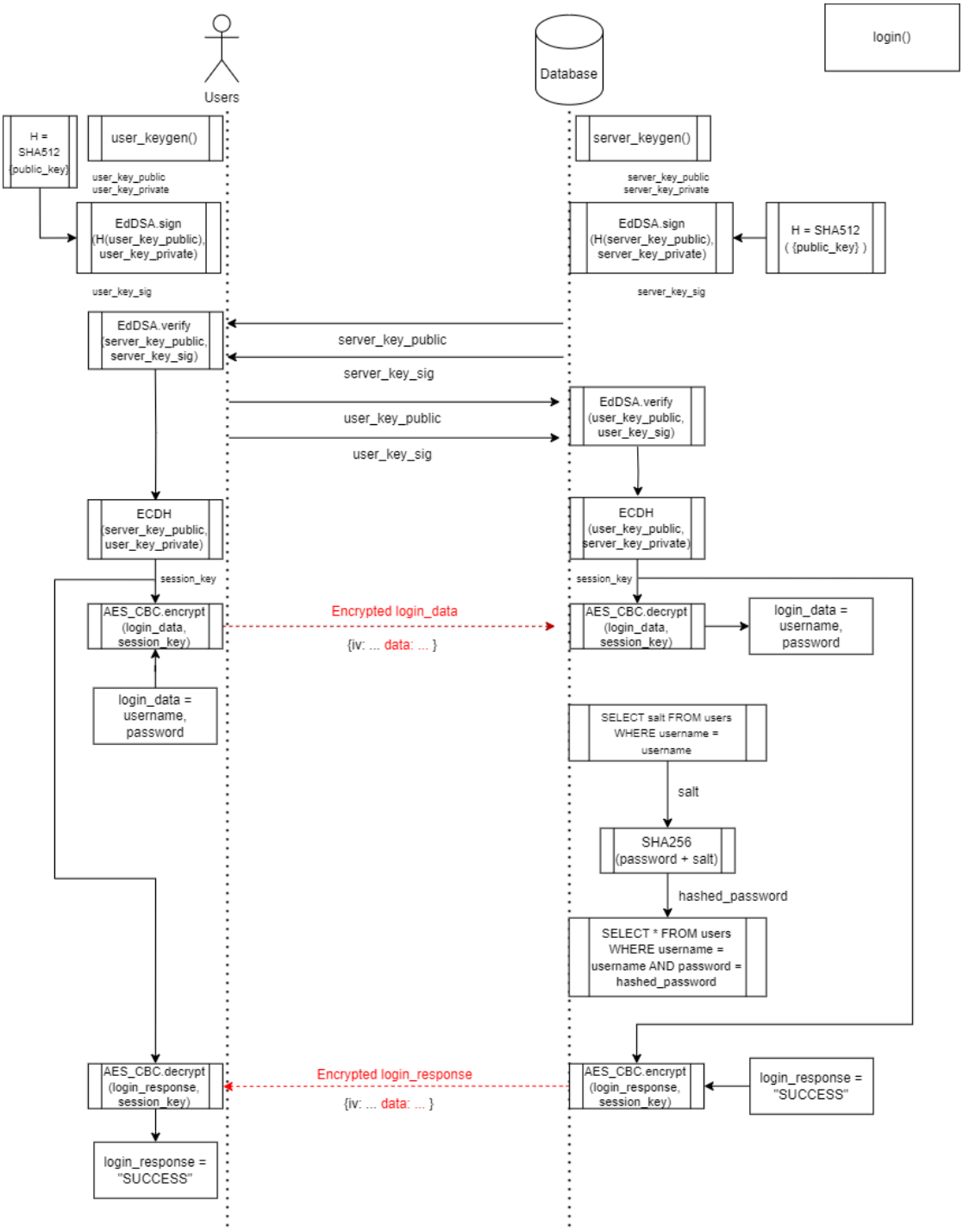


Figure 5. 5 Developed System's flowchart (login)

The process is almost the same as registration process; however the system will generate another session key after returning from registration. The server will query for the salt on the requested account, then hash the password together with the salt. Finally, another

query is performed to authenticate the credentials using the username and the hashed password.

Finally, the server will send a response to notify the client on successful login. This response is also encrypted using the session key set during the login process.

Now, the server will perform the initialization of PQXDH on login process, where the server will send all of the keypairs and signatures belonging to the user to be deserialized on client side. This process is illustrated below.

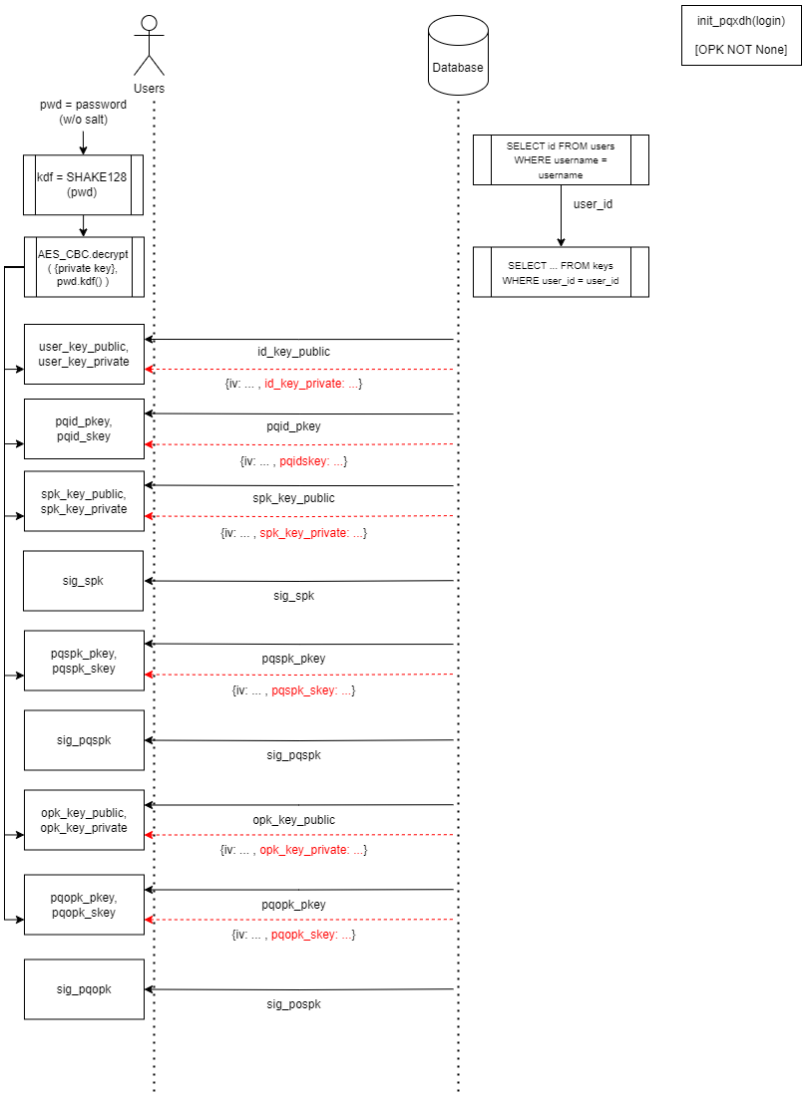


Figure 5. 6 Developed system's flowchart (PQXDH init. Login w/ OPK)

All of the keypairs and signatures belonging to the user are sent. The password from the client's login are inputted into the KDF function and the digest is then used to decrypt the received private keys.

However, there will be an instance when the user log in, the One-Time Prekeys ("OPK") are exhausted. This is because for once the PQXDH's secret key calculation is done, the OPKs are erased from the server; hence its name One-Time Keys. In that case, the server will send a string "NULL" to the client to note that their One-Time Keys are exhausted and need to be replenished. This process is visualised below.

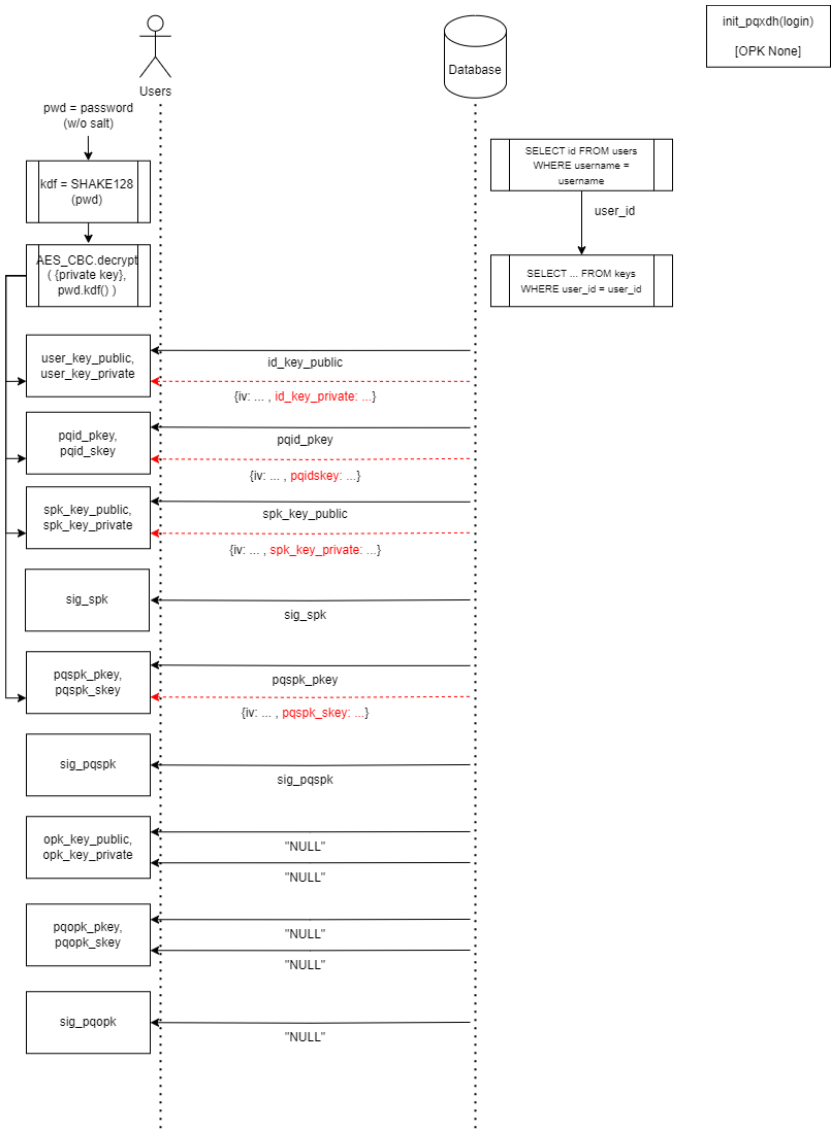


Figure 5. 7 Developed system's flowchart (PQXDH init. Login w/ SPK)

After successful login, the client will check if they have OPKs successfully deserialized to be used. If there are OPKs that are received during login, the client will send an update to the server to notify that the client is not requesting to update the OPK keypairs. This process is visualised below.

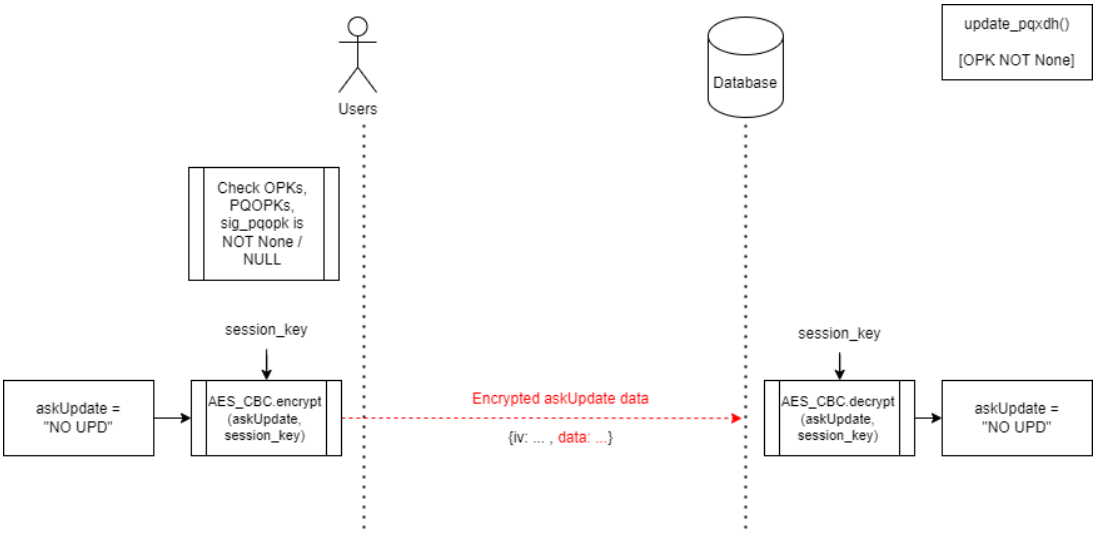


Figure 5. 8 Developed system's flowchart (PQXDH Update w/ OPK)

However, if the OPKs received are “NULL”, then the client will send a request to server to update the OPK keypairs; together with a newly-generated OPKs to the server. This process is visualised below.

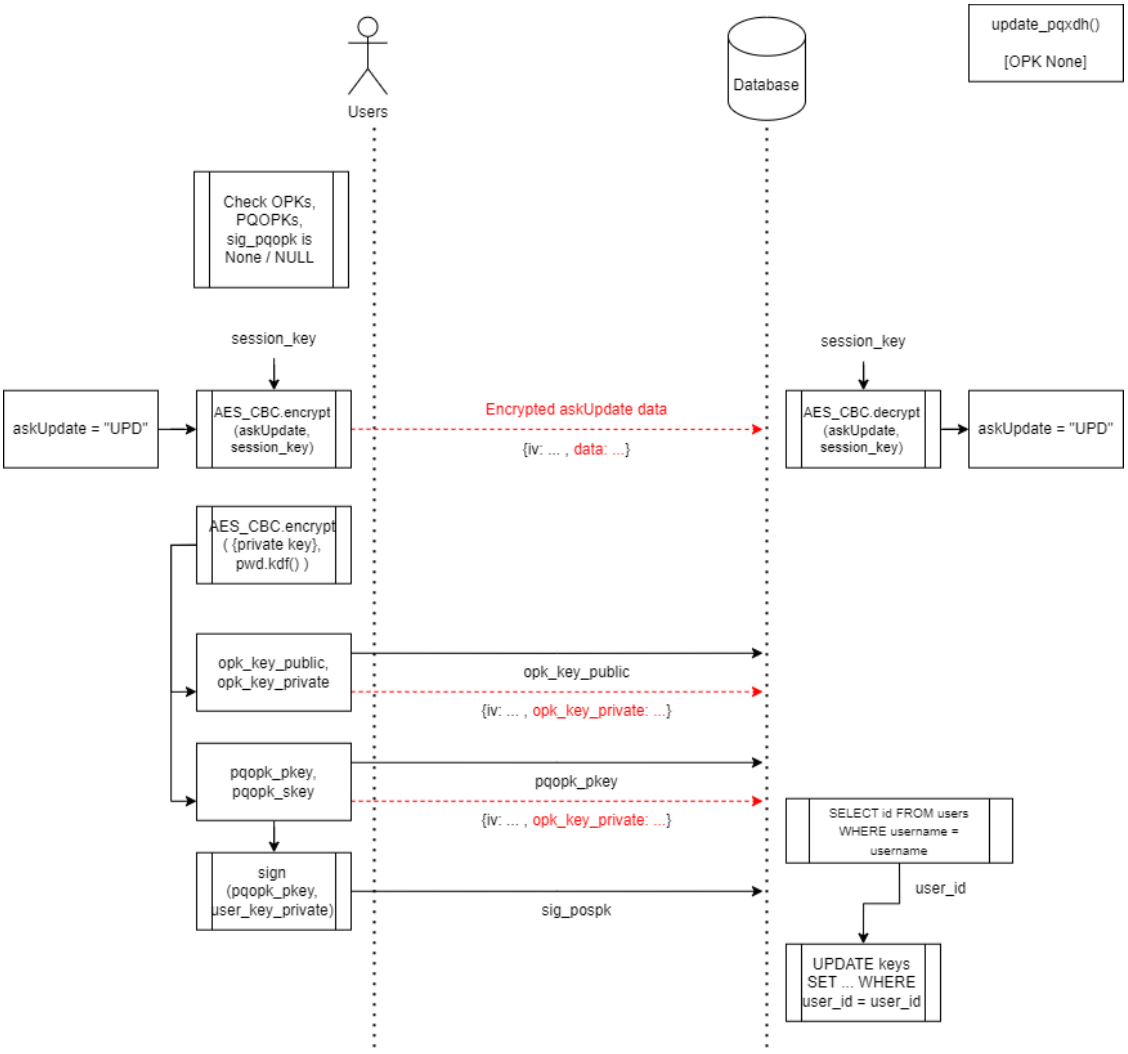


Figure 5. 9 Developed system's flowchart (PQXDH Update w/ SPK)

5.3.3 System's flowchart on PQXDH KEP

The process of the system's PQXDH KEP is visualised below. Note that one entity will be performing encapsulation and another entity will perform decapsulation. Below is a visualisation of the encapsulation process with OPK.

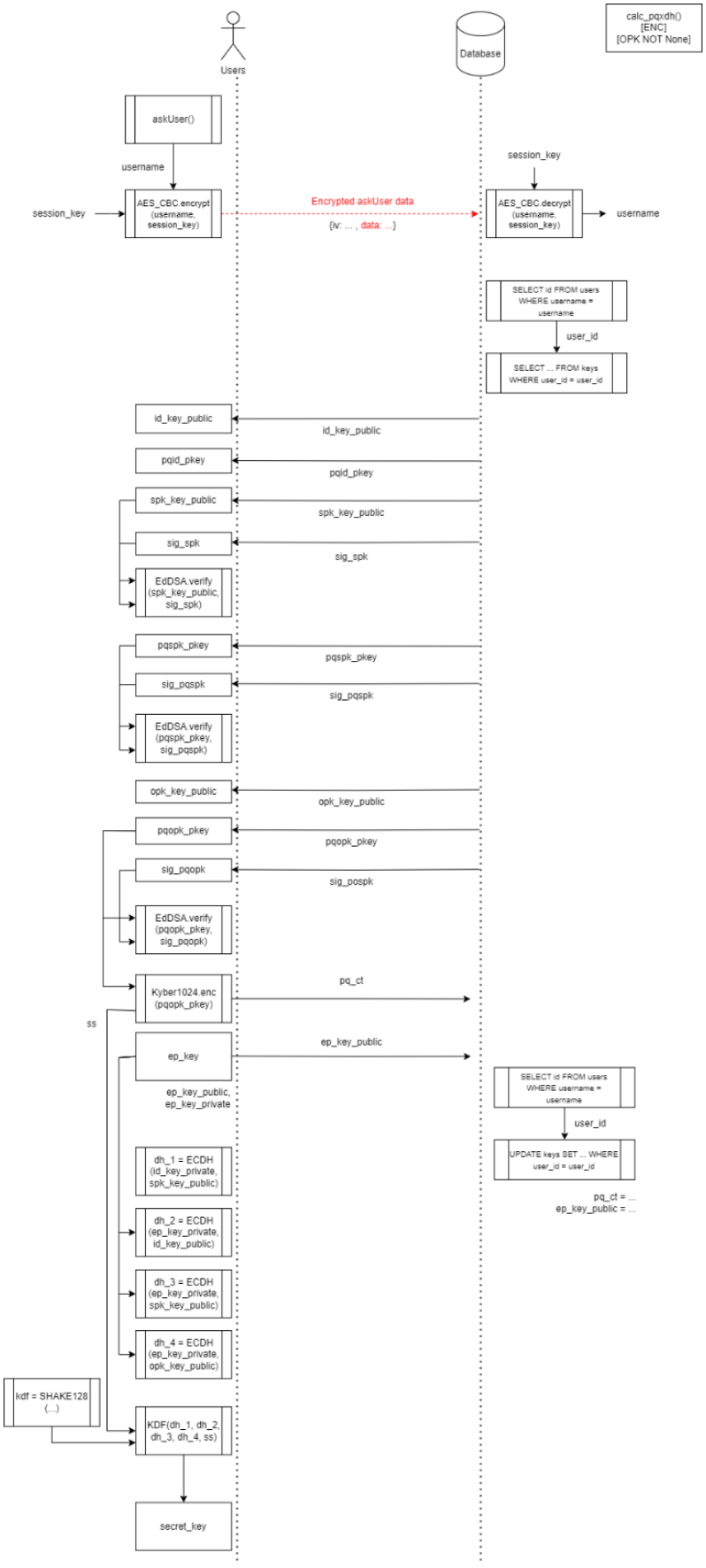


Figure 5. 10 Developed system's flowchart (PQXDH Encap. w/ OPK)

First, the system will ask user on who is their corresponding account username that will receive their messages. The correspondence's username is encrypted and sent to server. The server will decrypt the packet and proceed to search for the correspondence's account in the database.

If the correspondence's account is found, the server then proceeds to send every public keys and signatures belonging to the correspondence, to the client.

Once the client receives all its correspondence public keys and signatures, it first verifies the signatures on the public keys received.

If the verification successful, then the public keys are deemed authentic, and the client will perform Kyber's encapsulation with the received correspondence's PQOPK. The output of the encapsulation is a Ciphertext named pq_ct and a shared secret SS. Then, the client will also generate its own ephemeral key pair named ep_key_public, $ep\_key\_private$.

Four separate ECDH KEP will be performed. The first KEP with own private Identity Key ($id\_key\_private$) and the correspondence's public Signed Prekey (spk_key_public). The second KEP with own private Ephemeral key ($ep\_key\_private$) and correspondence's public ID Key (id_key_public). The third KEP with own private Ephemeral key ($ep\_key\_private$) and the correspondence's public Signed Prekey (spk_key_public). The last KEP with own private Ephemeral key ($ep\_key\_private$) and the correspondence's public One-Time Prekey (spk_key_public).

Finally, all of the values from the four KEP just now, together with the shared secret SS from the Kyber's encapsulation is inputted together into a KDF which is SHAKE128.

The output from the KDF is used as a secret key, to encrypt and decrypt messages with the correspondence.

The client will send the pq_ct and ep_key_public to the server, to be fetched by the correspondence when they perform decapsulation later on.

Here, decapsulation where OPK exists is covered. The same process is reversed from the encapsulation. The system will ask user for their correspondence account. The correspondence username is then encrypted and sent to the server. The server will then search for the correspondence account; and if the account is found, will proceed to send all of the public key and signatures of the correspondence account to the client.

Once all of the keys are received, it is verified. If verification is successful, the client then performs Kyber's decapsulation on the pq_ct from the encapsulation operation earlier, using their own private PQOPK. The output of this process will be a plaintext called pt.

The four KEP is performed but with their respective counterparts. The first KEP with own private Signed Pre-Key ($spk\_key\_private$) and correspondence's public Identity Key (id_key_public). The second KEP with own private Identity Key ($id\_key\_private$) and correspondence's public Ephemeral Key (ep_key_public). The third KEP with own private Signed Pre-Key ($spk\_key\_private$) and correspondence's public Ephemeral Key (ep_key_public). The last KEP with own private One-Time Pre-key ($opk\_key\_private$) and correspondence's public Ephemeral Key (ep_key_public).

Finally, all of the values from the four KEP just now, together with the plaintext pt from the Kyber's decapsulation is inputted together into a KDF which is SHAKE128.

The output from the KDF is used as a secret key and is now shared between the client and the correspondence.

The server will finally erase all of the outputs of the operation just now, which are pq_ct, ep_key_public; together with all of the OPKs since it has been used.

The process of the decapsulation is visualised below.

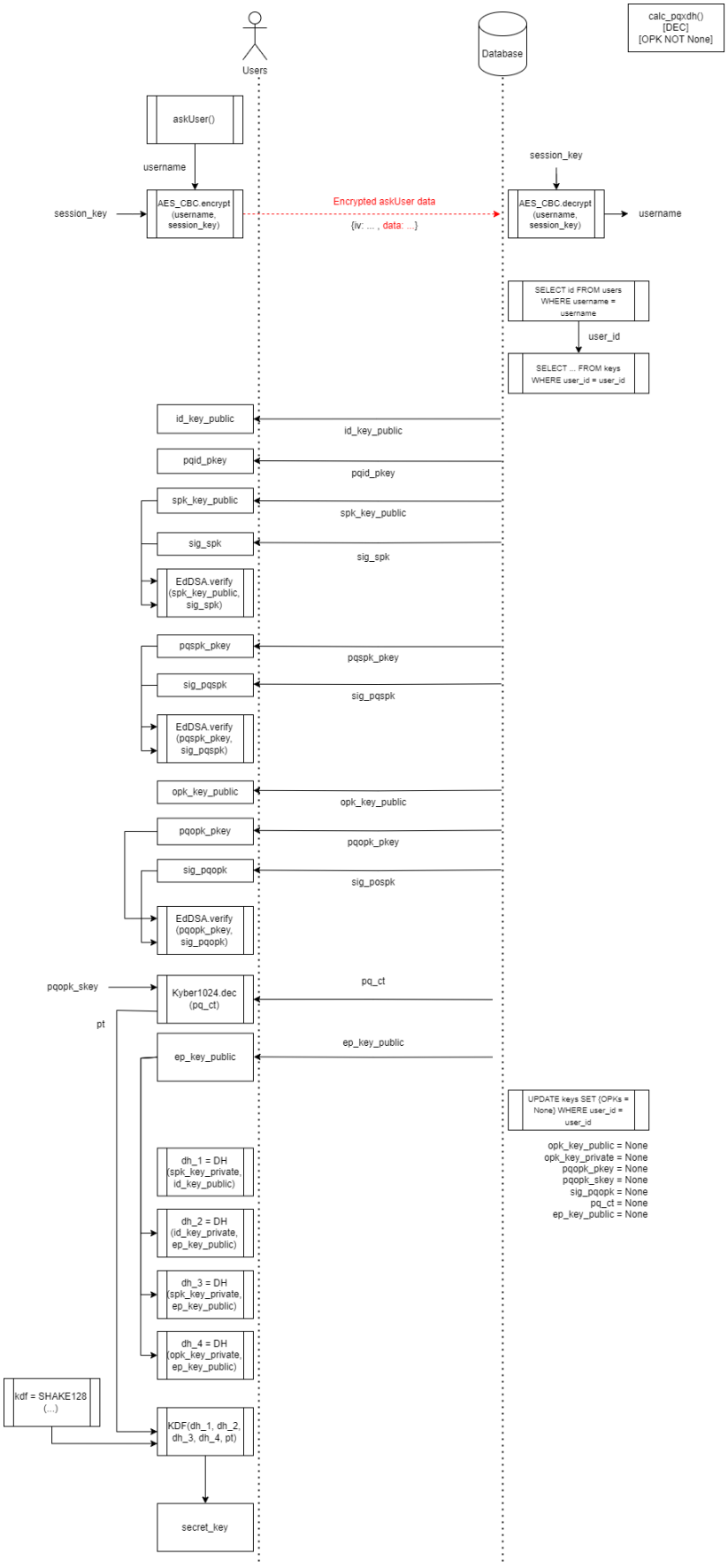


Figure 5. 11 Developed system's flowchart (PQXDH Decap. w/ OPK)

Here, encapsulation part in the case where there are no OPKs is covered. The process is almost the same, but the last KEP is not performed and encapsulation is performed with public PQSPK instead of PQOPK. The last KEP is also omitted from the KDF to derive the secret key. Otherwise, the process is all similar. This process is visualised below.

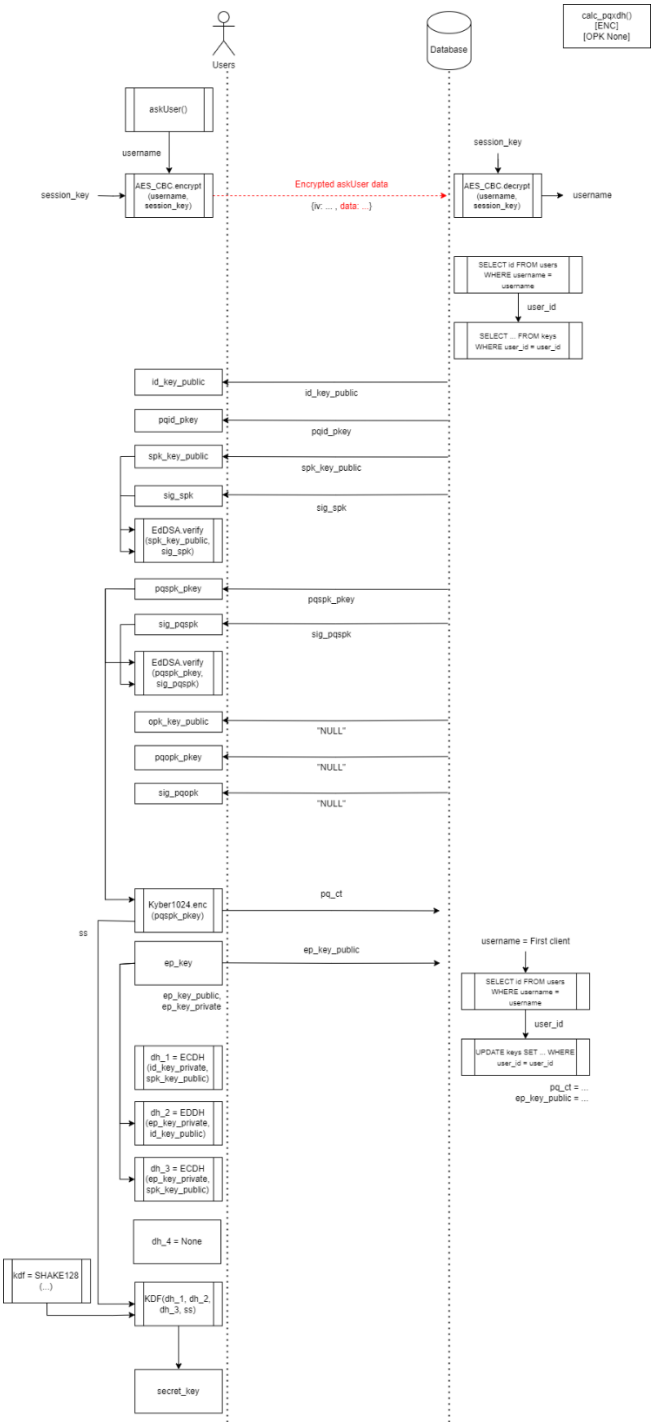


Figure 5. 12 Developed system's flowchart (PQXDH Encap. w/ SPK)

The decapsulation part without OPKs case is also similar, but the decapsulation operation is performed with own private PQSPK instead of using PQOPK. Also, the last KEP will be omitted from the KDF. This process is visualised below.

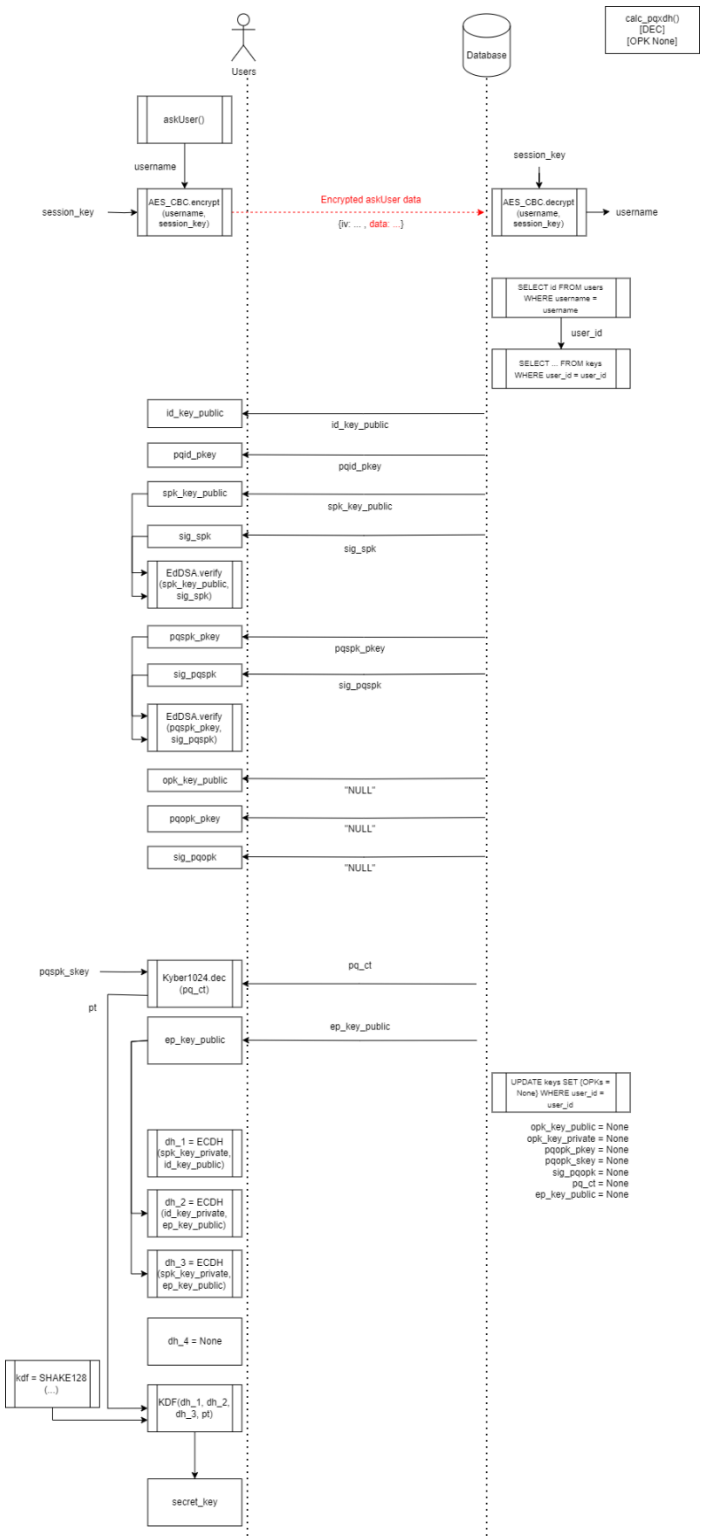


Figure 5. 13 Developed system's flowchart (PQXDH Decap. w/ SPK)

5.3.4 System's flowchart on sending/receiving message

The process of the system's message exchange is visualised below. Note that there are two entities named "test123" and "asdasd" exchanging messages in this scenario. "test123" correspondence is "asdasd" and "asdasd" correspondence is "test123".

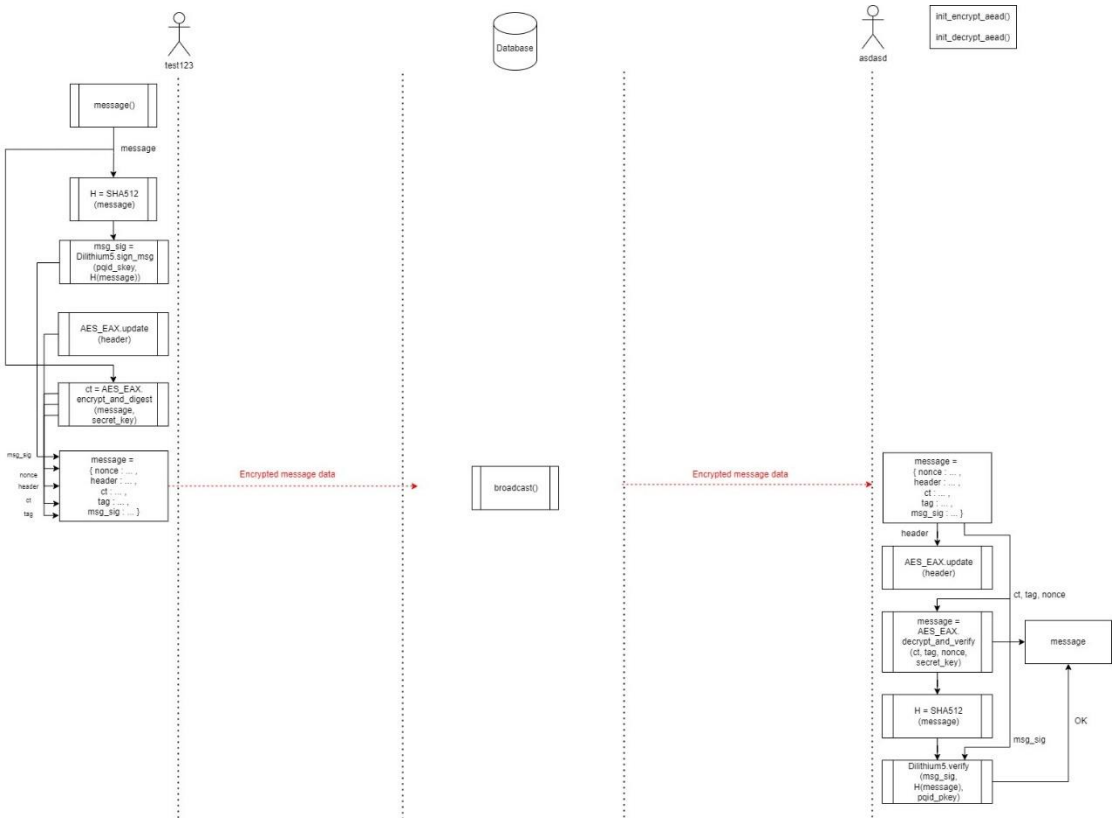


Figure 5. 14 Developed system's flowchart (Encrypt and Decrypt messages)

First, the client will take the message in plaintext after the user clicks send. The message is then hashed into SHA512. The hash digest of the message will then be signed by the client's own private PQ Identity Key ($pqid\_skey$) using Dilithium5; in this scenario "test123" $pqid\_skey$.

An AES with EAX mode cipher is initialized, and the cipher's header value is updated with client's own public Identity Key (id_key_public) converted to byte stream; in this

scenario “test123” id_key_public. The message is then encrypted and digested using the secret key generated from the PQXDH operation earlier. EAX mode is used to conform with AEAD scheme of PQXDH’s implementation.

The message is then packed into JSON format, comprising of nonce, header, the ciphertext itself named ct, tag, and the message signature named msg_sig. Finally, the message can be sent to server to be broadcasted to the correspondence; in this scenario to “asdasd”.

It is important to stress that all of the messages are end-to-end encrypted, and the server will just forward the messages to the correspondence; without any mechanism to decrypt the messages in-transit.

The correspondence, “asdasd” will receive the message, initiate the AES EAX cipher, and update the header from the message. The message’s ciphertext ct is then decrypted and verified using the nonce, tag and the secret key, to get the message’s plaintext.

The message is then hashed into SHA512, and the digest is verified against the message’s signature msg_sig with “asdasd” correspondence’s public PQ Identity Key ($pqid\_pkey$); in this scenario “test123” $pqid\_pkey$. If verification is successful, the message’s plaintext will be appended with “- OK” at the end; else the message will be appended with “- NOT OK” to denote that the message verification is unsuccessful.

5.4 Developed System's snapshots

The system starts when the server is started. Once it is up and running, the server will show a banner and notify that it is running and ready to accept connections. This process is visualised below.

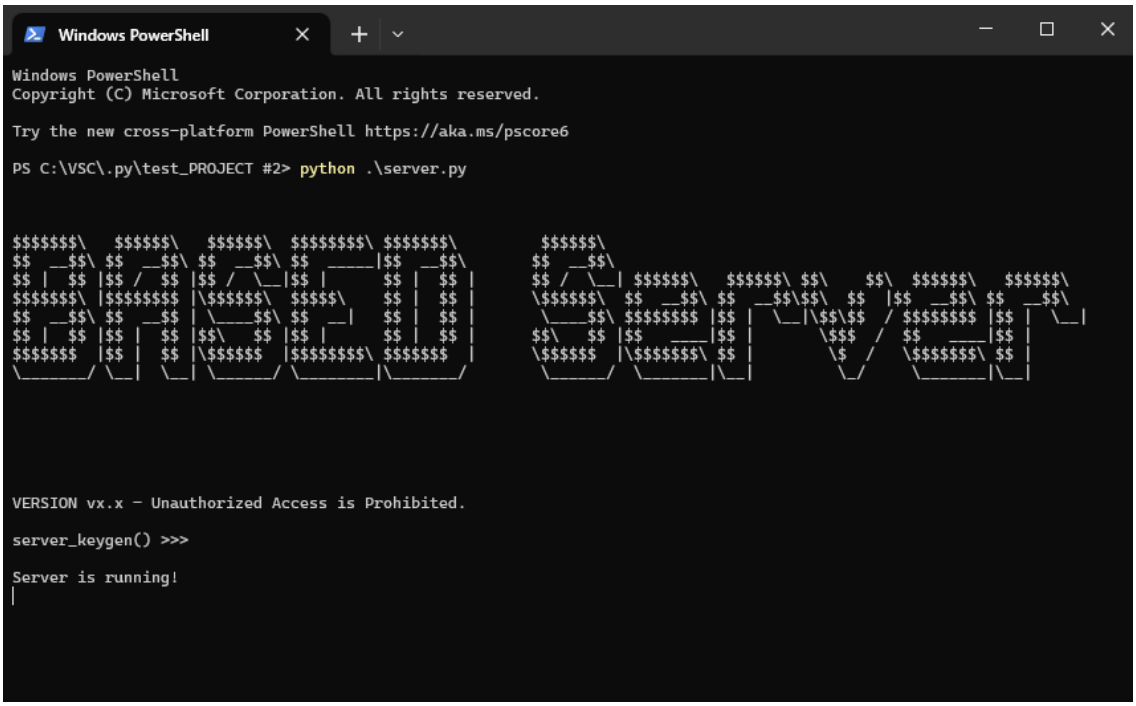


Figure 5. 15 Screenshot of server init.

Next, the client can be started. The user will be greeted with two options either to register an account or login. This process is visualised below.

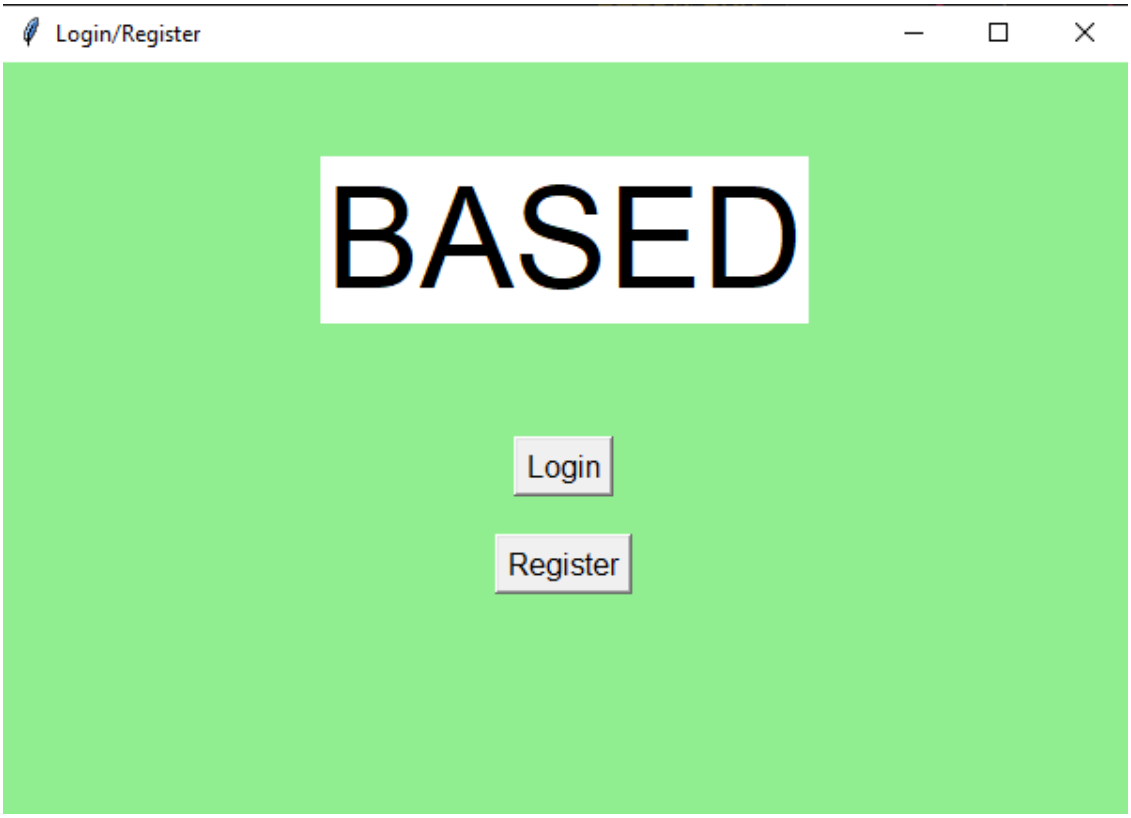


Figure 5. 16 Screenshot of client init.

5.4.1 System's snapshot on register

When the user clicks on register button on the client, the client will ask users for a username, password and confirm password. The password and confirm password first must match in order to register the account. This process is visualised below.

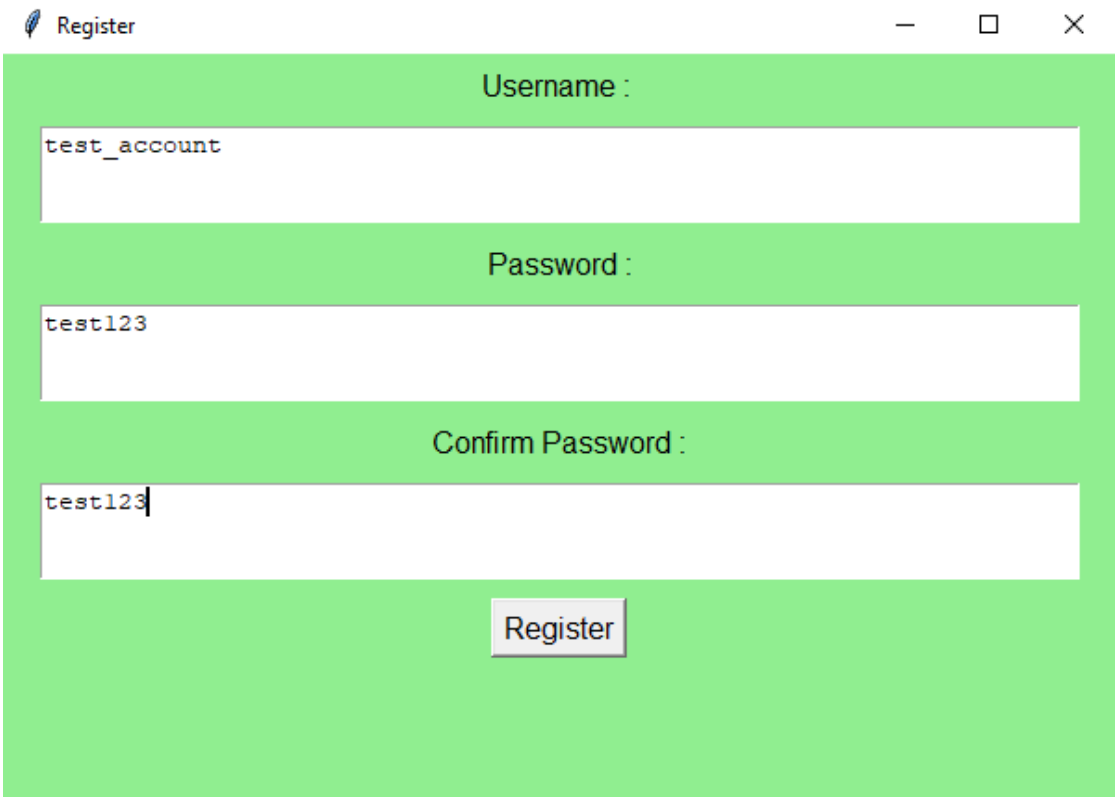


Figure 5. 17 Screenshot of client's register

The client will notify the user if the account registration is successful. This process is visualised below.

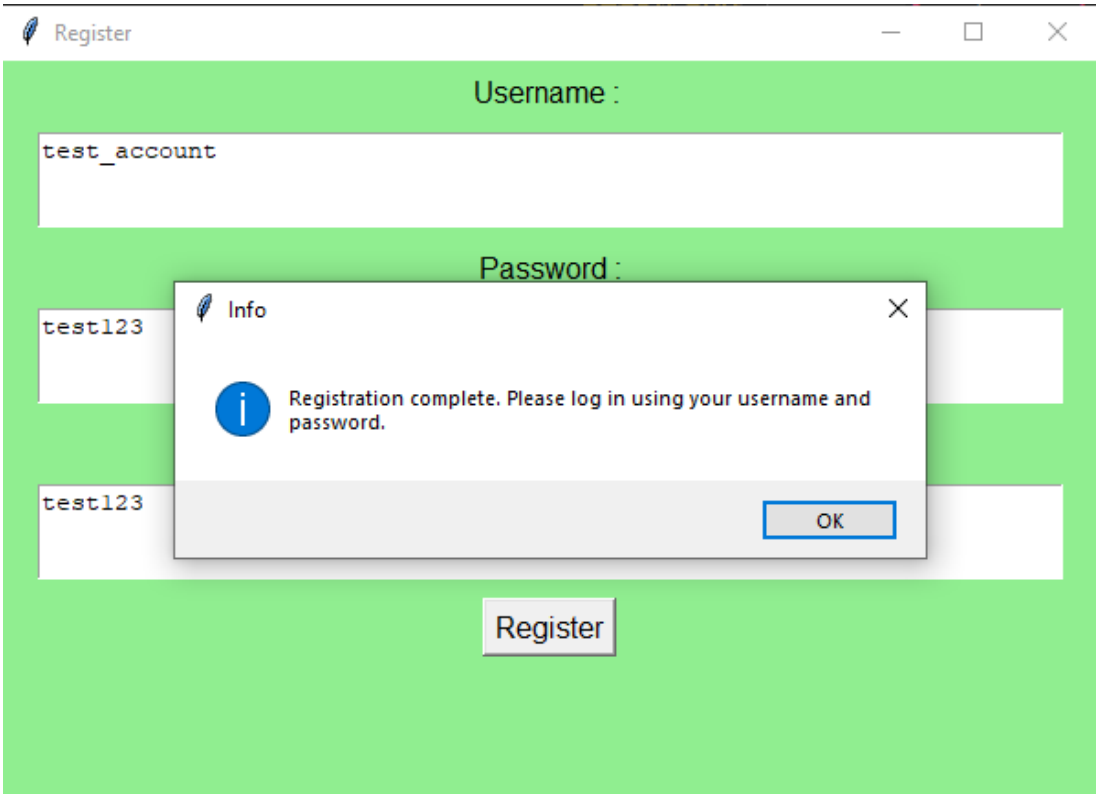


Figure 5. 18 Screenshot of client successful registration

5.4.2 System's snapshot on login

When the user clicks on the login button on the client, the client will ask users for their username and password to log into their account. This process is visualised below.

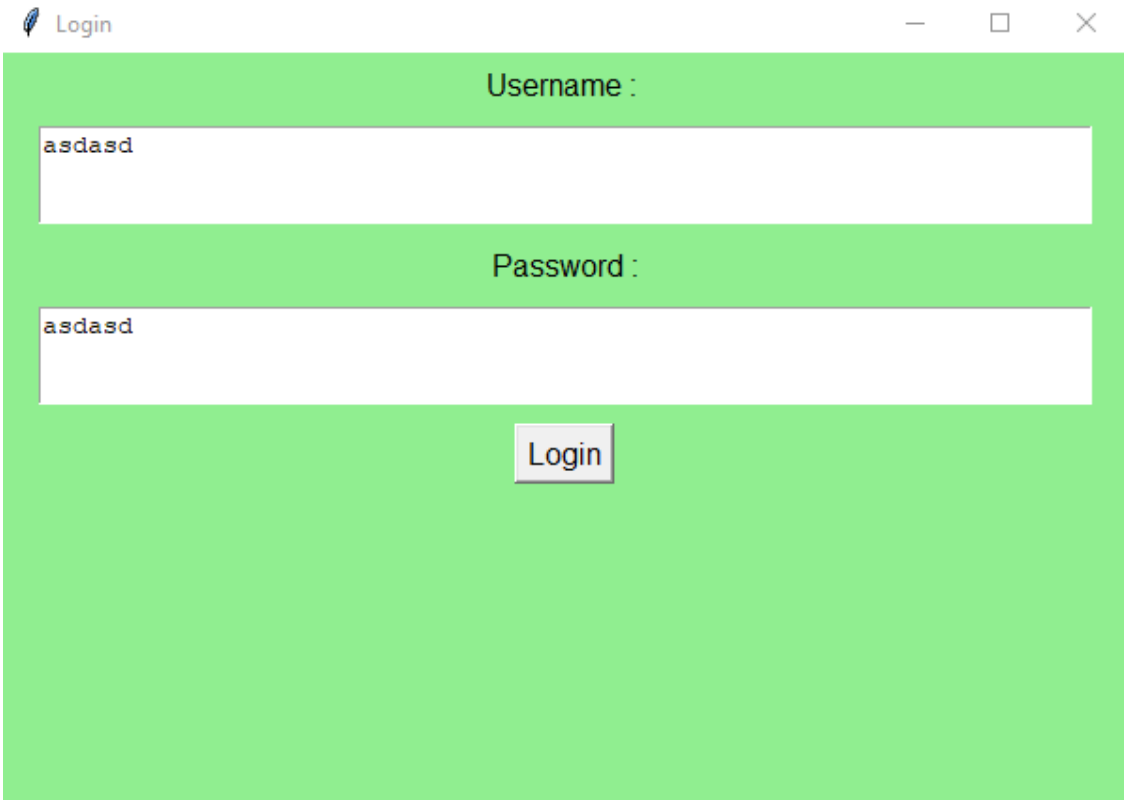


Figure 5. 19 Screenshot of client login

5.4.3 System's snapshot on set correspondence

When the user is successfully login, the client will then ask for their correspondence's username. This process is visualised below.

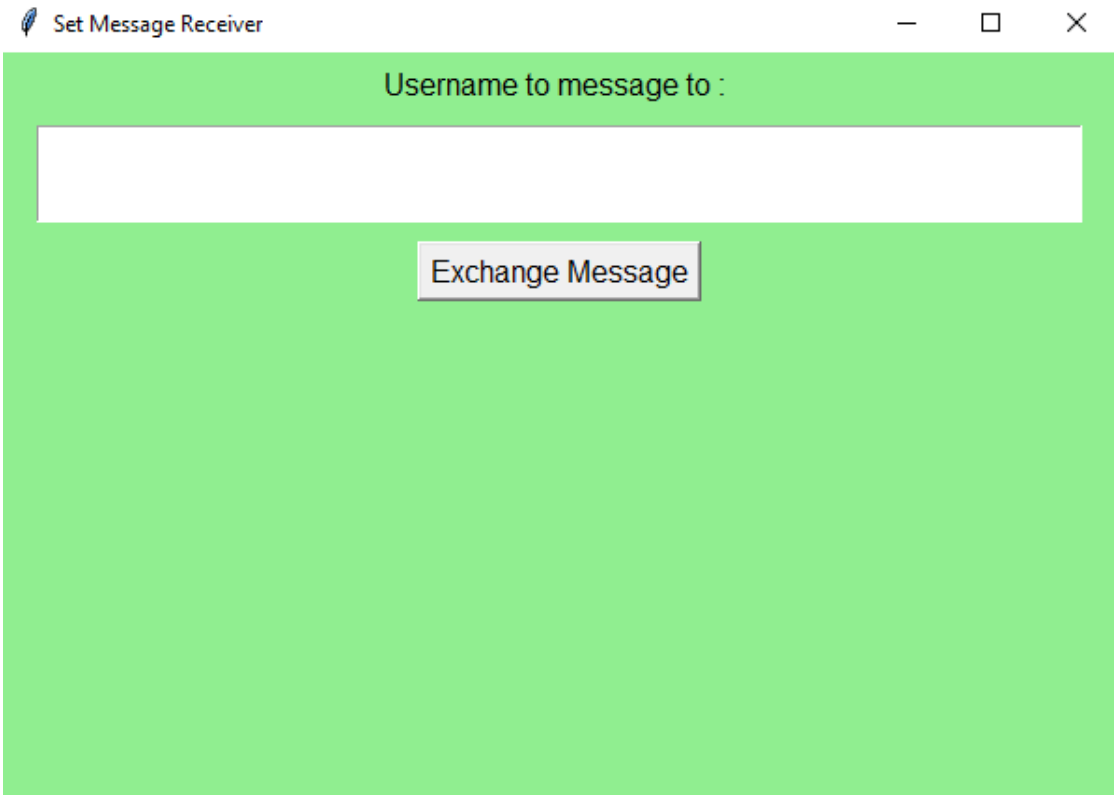


Figure 5. 20 Screenshot of client set correspondence

5.4.4 System's snapshot on message exchange

When the user's correspondence has been set, a message session will be started. The client is now ready to send and receive messages. This process is visualised below.

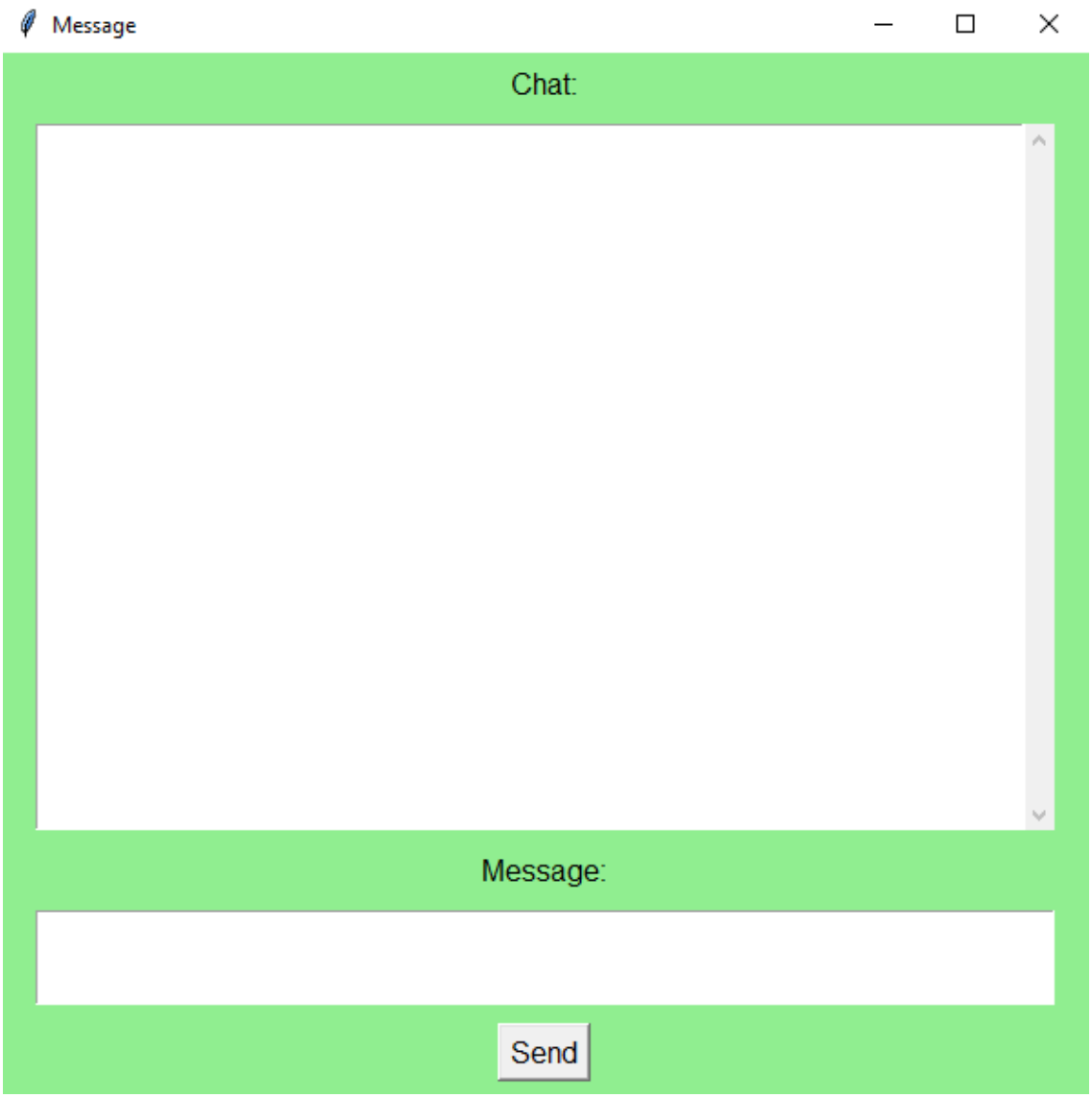


Figure 5. 21 Screenshot of message session init.

On server-side, the database is updated where the user has uploaded their pq_ct and ep_key_public to be fetched by their correspondence. This process is visualised below.

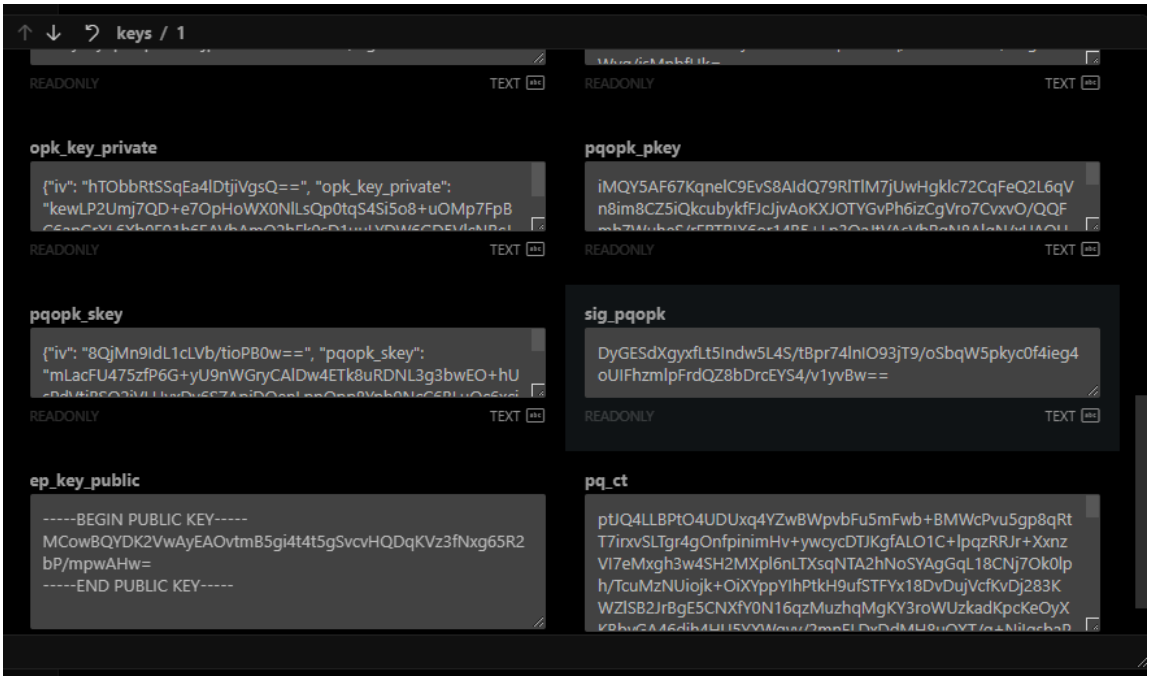


Figure 5. 22 Screenshot of server's updated keys table after PQXDH Enc.

Now assume there are two entities in a session, named “test123” and “asdasd”. When they both login to the system and start a session, the server will keep a log of who is currently log in to the system. This process visualised below.

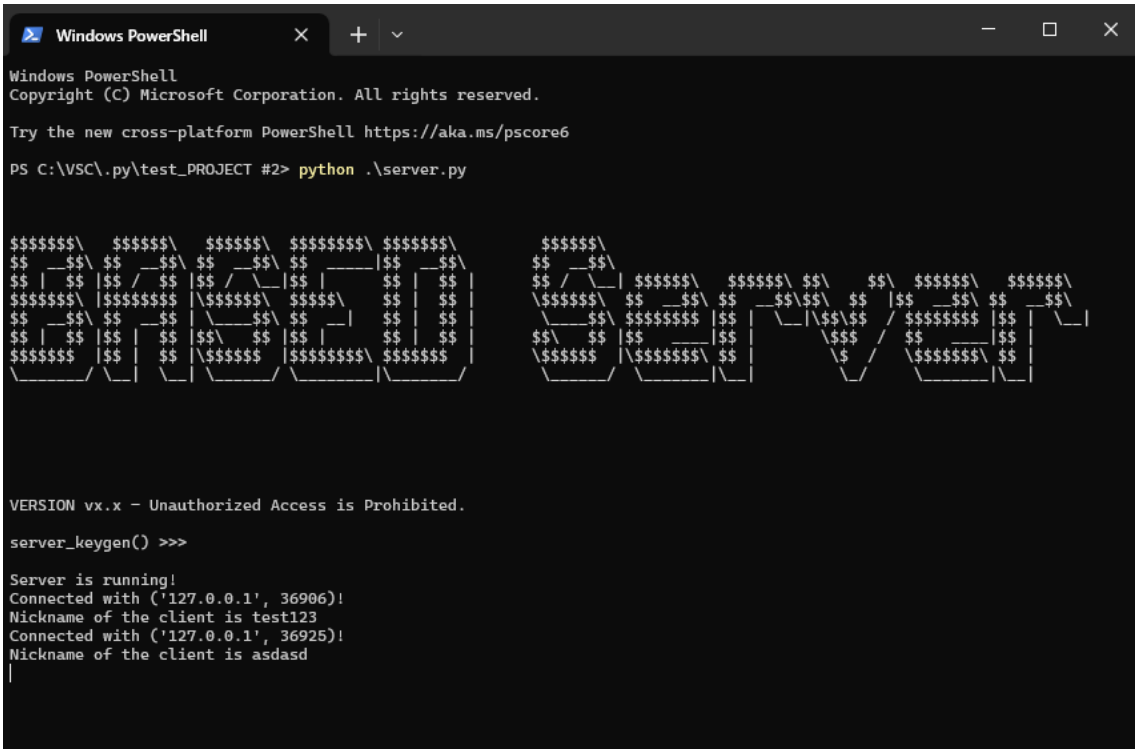


Figure 5. 23 Screenshot of server's client login log

Now, “test123” and “asdasd” can exchange messages securely since all messages are end-to-end encrypted and verified on receive. If a message is successfully verified, a string “- OK” will be appended to the end of message to note that the message is legitimately from the message sender. This process is visualised below. Note that the left chat is “test123” and the right chat is “asdasd”.

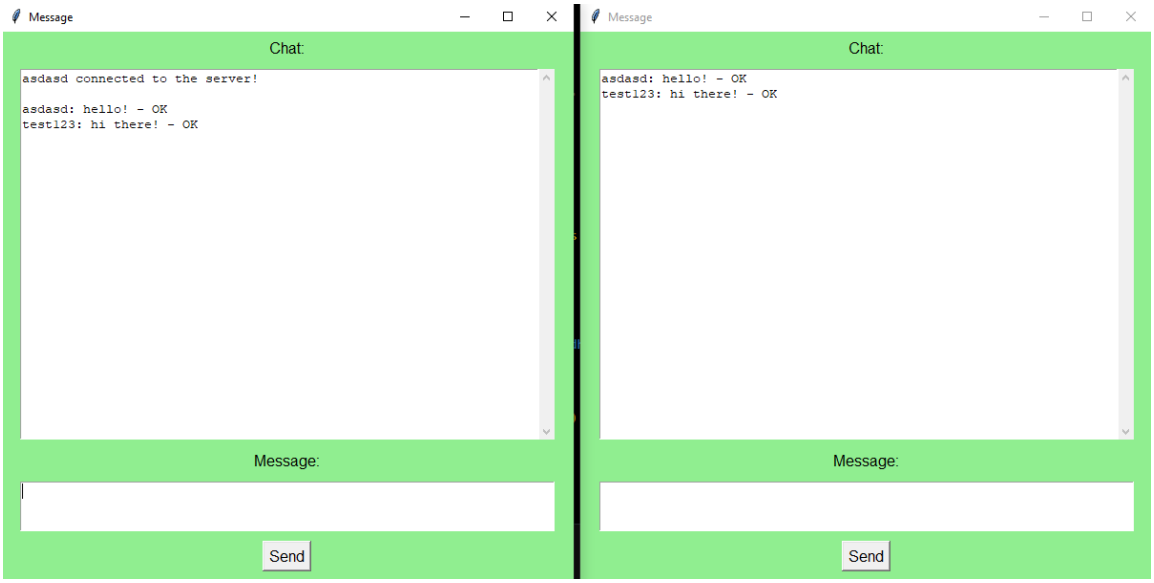


Figure 5. 24 Screenshot of clients' message exchange

On server-side, when the two entities has completed the PQXDH's Encapsulation and Decapsulation, the outputs from the PQXDH's Encapsulation and the OPKs of the user who initiated the Encapsulation will be erased to NULL.

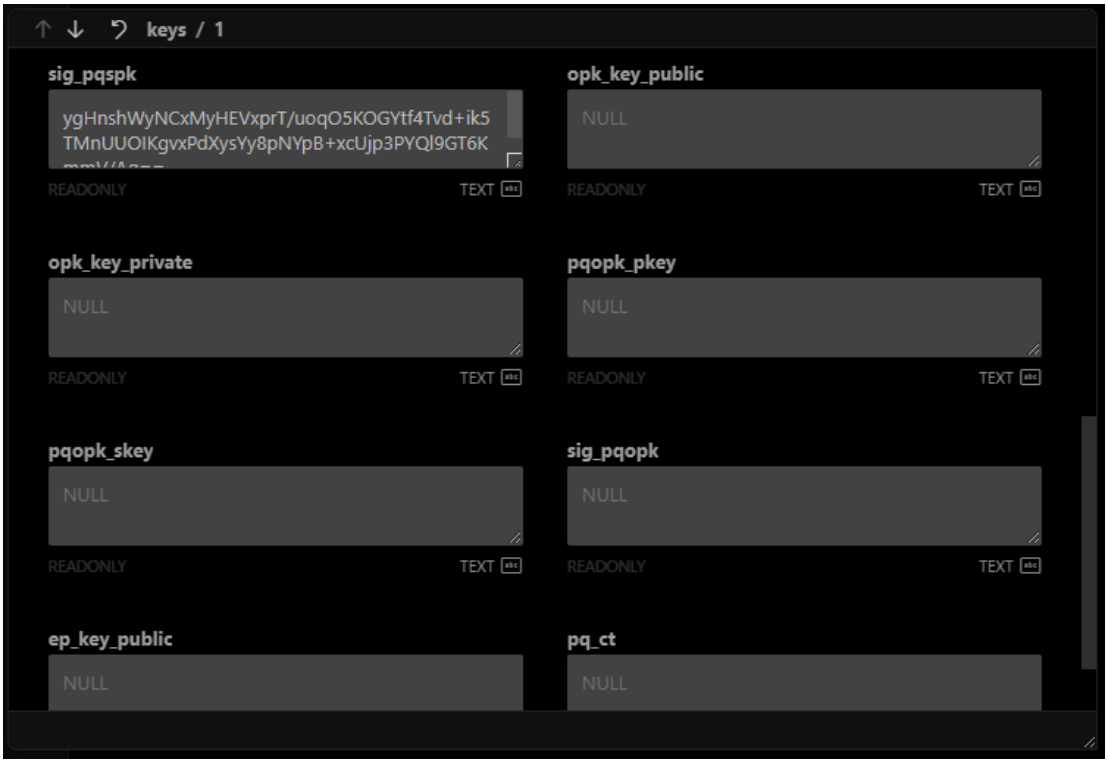


Figure 5. 25 Screenshot of server's updated keys table after PQXDH Dec.

Finally, when one of the client exits the application, it will automatically log out from the system, and notify the correspondence that the user has left the session. This process is visualised below. Note that “asdasd” client is closed.

Figure 5. 26 Screenshot of client’s exit session

On server-side, the server also updates the log of when the user exits the session. This process is visualised below.

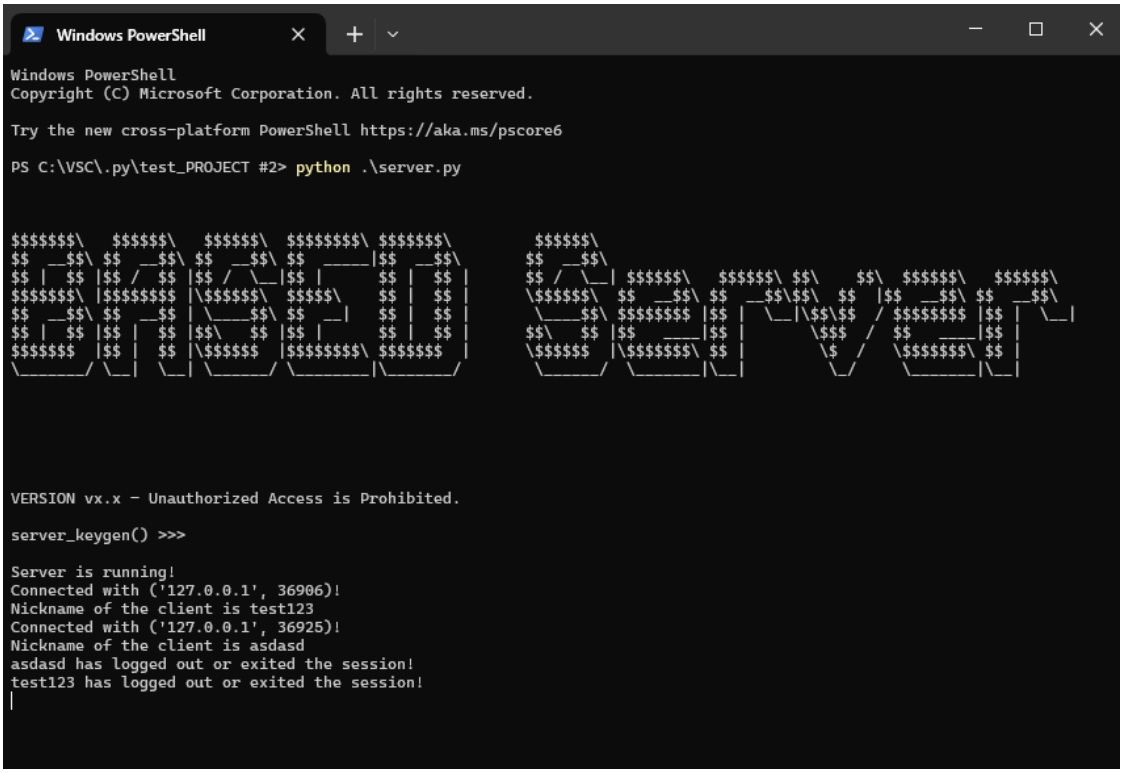


Figure 5. 27 Screenshot of server's log of client exit session

CHAPTER 6

CONCLUSION

6.1 Considerations on project

Although this project objectives and end-goal is to create a communication system implementing Hybrid Post-Quantum cryptosystem, it is far from ready to be widely used by the public. This is because in the Python's implementation of Kyber and Dilithium, the library's author disclosed that the library has not yet undergone a proper or formal cryptographic verification on the algorithm. The library's author also stressed a warning that it should not be used for any cryptographic applications.

Moreover, there are more unresolved issues and considerations in the system architecture and design that are not stated in this thesis; where proper methods of implementing cryptography such as initial connection with server, handling of keys over insecure network, key storage, etc. may need to be addressed during development of the system later on in the future.

Additionally, this proposed project has not yet been tested against possible practical attacks on the cryptography algorithms. For simplest example, the KEP process may theoretically be vulnerable to a Man-in-the-Middle ("MITM") attack where both users exchange keys with an intermediate threat actor in the middle. The threat actor then are able to read or modify then relay back the messages to the intended recipient [137].

Additional mitigations on these theoretical attacks need to be researched in order to make the proposed project more secure; unfortunately within time constraint of the project, the author cannot perform the mitigation research.

With the constraints of time also, some of the functionalities and features in this thesis may not be implemented in the system.

By all things considered, this project could prove theoretically that Hybrid Post-Quantum solution is feasible for current use on mitigation against Quantum computer threats. In the future, with the advancement of cryptography and increase of maturity in post-Quantum cryptography, the full potential of Quantum security can be harnessed by migrating towards full-Quantum cryptography, where the time has come to retire current conventional cryptography used. But, until the time comes, conventional Cryptography are still here to stay, as long as humanity needs it.

6.2 Challenges and Future work

During the development of this project, it takes a lot of time and effort to practically implement the PQXDH KEP from theory. This is because the complexity of the protocol that uses multiple keys and the keys have constraints on usage, such as the OPKs that are need to be deleted after one-time use.

Another challenge in this project is the complexity of managing the sockets and connection of the clients and server, and finding the right size and format of the packet's payload. Without time restrictions and JSON formatting of message payload, client and server cannot communicate reliably over the network.

Finally, the last challenge is to find the suitable libraries that can be practically used to integrate correctly without errors into the system.

For future work, the project could continue to finish off implementing fresh key generation for SPK as defined in PQXDH. With users' SPKs being refreshed after a set time, the message exchange can achieve true perfect forward secrecy.

Finally for future work, the latest updates on standardization of Post-Quantum Cryptography can be followed to make sure that the project stays up to date with NIST's recommendations for true Quantum Safe communication.

6.3 Postface by author

The author hopes that with this thesis published, the readers could understand the theoretic and practicality of basic to intermediate cryptography. This thesis has covered from the fundamentals and building blocks of cryptography, Symmetric and Asymmetric cryptography, Digital Signature Schemes, Key Exchange Protocols, and few proposed Post-Quantum Cryptography.

This thesis has also covered the introduction of conventional and Quantum computers; and what are their threats to current conventional cryptography. This thesis also have delved into the current and future outlook for Quantum computers to come.

This thesis has also looked at how the proposed project will be developed according to software engineering principles and what resources this project will need in order to be developed.

This thesis has also delved into the database and interface design of the proposed system; and how the user are going to experience the proposed system though the system flow.

The author hope that with the writing of this thesis, it could be beneficial to all academia and the society as a whole in the current time and in the future. With that, to conclude on this chapter, a quote by Niels Bohr to J. Robert Oppenheimer where he said [138]:

Algebra is like sheet music.
The important thing isn't "Can you read music?" It's "Can you hear it?"
Can you hear the music, Robert?

This quote shows that there are more than being able to do calculation or read knowledge. One must be able to visualise an insight of what these calculations or theories mean; and have inspiration within them. With this inspiration and firm grasp on fundamentals of our knowledge, a person can strive to be better.

REFERENCES

[1] P. W. Shor, “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,” *SIAM Review*, vol. 41, no. 2, pp. 303–332, Jan. 1999, doi: 10.1137/S0036144598347011.

[2] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC ’96*, New York, New York, USA: ACM Press, 1996, pp. 212–219. doi: 10.1145/237814.237866.

[3] J. Proos and Ch. Zalka, “Shor’s discrete logarithm quantum algorithm for elliptic curves,” *Quantum Inf Comput*, vol. 3, no. 4, pp. 317–344, Jul. 2003, doi: 10.26421/QIC3.4-3.

[4] C. Gidney and M. Ekerå, “How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits,” May 2019, doi: 10.22331/q-2021-04-15-433.

[5] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2007. doi: 10.1201/9781420010756.

[6] H. G. Liddel, R. Scott, and H. Drisler, *A Greek-English Lexicon*, 8th ed., no. 8. Harper & brothers, 1897. Accessed: Nov. 05, 2023. [Online]. Available: https://books.google.com/books?id=e04AQAAMAAJ

[7] P. Zimmerman, *An Introduction to Cryptography*. PGP Corporation, 2005. Accessed: Nov. 05, 2023. [Online]. Available: https://copeland.ece.gatech.edu/jac/6612/pgp/Intro%20To%20Cryptography.pdf

[8] B. Schneier, *Applied Cryptography, Second Edition*, 2nd ed. Wiley, 2015. doi: 10.1002/9781119183471.

[9] John. Viega, Matt. Messier, and Pravir. Chandra, *Network security with OpenSSL*. O’Reilly, 2002. Accessed: Nov. 06, 2023. [Online]. Available: https://www.oreilly.com/library/view/network-security-with/059600270X/

[10] “Difference between Active Attack and Passive Attack,” GeekForGeeks. Accessed: Nov. 05, 2023. [Online]. Available: https://www.geeksforgeeks.org/difference-between-active-attack-and-passive-attack/

[11] U. Maurer, “Information-Theoretic Cryptography,” vol. 1666, M. Wiener, Ed., Springer-Verlag, 1999, pp. 47–65. doi: 10.1007/3-540-48405-1_4.

[12] P. Guillot, “Auguste Kerckhoffs et la cryptographie militaire,” *BibNum*, vol. IX, pp. 5–38, May 2013, doi: 10.4000/bibnum.555.

[13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 2018. doi: 10.1201/9780429466335.

[14] C. E. Shannon, “Communication Theory of Secrecy Systems*,” *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct. 1949, doi: 10.1002/j.1538-7305.1949.tb00928.x.

[15] M. J. Dworkin, “Advanced Encryption Standard (AES),” 2023. doi: 10.6028/NIST.FIPS.197-upd1.

[16] M. E. Smid, “Development of the Advanced Encryption Standard,” *J Res Natl Inst Stand Technol*, vol. 126, p. 126024, Aug. 2021, doi: 10.6028/jres.126.024.

[17] D. Georgoudis, “Report on AES2, Day 1,” Cryptome. Accessed: Nov. 15, 2023. [Online]. Available: https://cryptome.org/jya/aes2-day1.htm

[18] D. Georgoudis, “Report on AES2, Day 2,” Cryptome. Accessed: Nov. 15, 2023. [Online]. Available: https://cryptome.org/jya/aes2-day2.htm

[19] J. Nechvatal *et al.*, “Report on the development of the Advanced Encryption Standard (AES),” *J Res Natl Inst Stand Technol*, vol. 106, no. 3, p. 511, May 2001, doi: 10.6028/jres.106.023.

[20] J. Daemen and V. Rijmen, *The design of Rijndael : AES--the Advanced Encryption Standard*. Springer-Verlag, 2002.

[21] D. Selent, “Advanced Encryption Standard,” *Rivier Academic Journal*, vol. 6, no. 2, p. 14, 2010.

[22] J. Daemen and V. Rijmen, *The Design of Rijndael*. in Information Security and Cryptography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. doi: 10.1007/978-3-662-04722-4.

[23] S. Roy, "What are the challenges faced in Symmetric Cryptography?," Encryption Consulting. Accessed: Nov. 16, 2023. [Online]. Available: https://www.encryptionconsulting.com/what-are-the-challenges-faced-in-symmetric-cryptography/

[24] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

[25] R. Rivest, A. Shamir, and L. Adleman, "Cryptographic communications system and method," 4,405,829, Sep. 20, 1983 Accessed: Nov. 18, 2023. [Online]. Available: https://patents.google.com/patent/US4405829

[26] C. Clifford, "Note on non-secret encryption (Declassified)," Nov. 1973. Accessed: Nov. 18, 2023. [Online]. Available: https://web.archive.org/web/20180928121748/https://www.gchq.gov.uk/sites/default/files/document_files/Cliff%20Cocks%20paper%2019731120.pdf

[27] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, May 2002, doi: 10.1109/MCOM.2002.1006971.

[28] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," Feb. 2003. doi: 10.17487/rfc3447.

[29] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings*, vol. 48, no. 177, Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 417–426. doi: 10.1007/3-540-39799-X_31.

[30] N. Koblitz, "Elliptic Curve Cryptosystems," *Math Comput*, vol. 48, no. 177, p. 203, Jan. 1987, doi: 10.2307/2007884.

[31] D. Jablon, “IEEE P1363: Standard Specifications for Public-Key Cryptography,” Nov. 2001. doi: 10.1109/IEEESTD.2000.92292.

[32] D. Moody, “Recommendations for Discrete Logarithm-based Cryptography:,” 2022. doi: 10.6028/NIST.SP.800-186.

[33] L. Chen, D. (orcid)0000-0002-4868-6684 Moody, A. (orcid)0000-0002-3930-527X Regenscheid, and A. Robinson, “Digital Signature Standard (DSS),” Feb. 2023. doi: 10.6028/NIST.FIPS.186-5.

[34] D. J. Bernstein, “A state-of-the-art Diffie-Hellman function.” Accessed: Nov. 20, 2023. [Online]. Available: https://cr.yp.to/ecdh.html

[35] D. J. Bernstein, “Curve25519: new Diffie-Hellman speed records,” Feb. 2006, Accessed: Nov. 20, 2023. [Online]. Available: https://cr.yp.to/ecdh/curve25519-20060209.pdf

[36] E. B. Barker and J. M. Kelsey, “Recommendation for random number generation using deterministic random bit generators,” Gaithersburg, MD, 2012. doi: 10.6028/NIST.SP.800-90a.

[37] A. Young and M. Yung, “The prevalence of kleptographic attacks on discrete-log based cryptosystems,” 1997, pp. 264–276. doi: 10.1007/BFb0052241.

[38] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, “Recommendation for the entropy sources used for random bit generation,” Gaithersburg, MD, Jan. 2018. doi: 10.6028/NIST.SP.800-90B.

[39] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[40] “Elliptic Curve Points | Desmos,” Desmos. Accessed: Dec. 05, 2023. [Online]. Available: https://www.desmos.com/calculator/ialhd71we3

[41] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” *Nature*, vol. 549, no. 7671, pp. 188–194, Sep. 2017, doi: 10.1038/nature23461.

[42] D. J. Bernstein and T. Lange, “PQ-CRYPTO.” Accessed: Dec. 02, 2023. [Online]. Available: https://pqcrypto.org/

[43] R. J. McEliece, “A Public-Key Cryptosystem Based On Algebraic Coding Theory,” Feb. 1978. Accessed: Nov. 27, 2023. [Online]. Available: https://ntrs.nasa.gov/api/citations/19780016269/downloads/19780016269.pdf#page=123

[44] D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds., *Post-Quantum Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-540-88702-7.

[45] E. Berlekamp, R. McEliece, and H. van Tilborg, “On the inherent intractability of certain coding problems (Corresp.),” *IEEE Trans Inf Theory*, vol. 24, no. 3, pp. 384–386, May 1978, doi: 10.1109/TIT.1978.1055873.

[46] D. J. Bernstein, T. Lange, and C. Peters, “Attacking and Defending the McEliece Cryptosystem,” 2008, pp. 31–46. doi: 10.1007/978-3-540-88403-3_3.

[47] M. R. Albrecht *et al.*, “Classic McEliece: conservative code-based cryptography: cryptosystem specification,” 2022. Accessed: Nov. 27, 2023. [Online]. Available: https://classic.mceliece.org/mceliece-spec-20221023.pdf

[48] T. Matsumoto and H. Imai, “Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption,” 1988, pp. 419–453. doi: 10.1007/3-540-45961-8_39.

[49] J. Patarin, “Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt’88,” 1995, pp. 248–261. doi: 10.1007/3-540-44750-4_20.

[50] J. Patarin, “Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms,” 1996, pp. 33–48. doi: 10.1007/3-540-68339-9_4.

[51] C. Wolf and B. Preneel, “Asymmetric Cryptography: Hidden Field Equations,” *Cryptology ePrint Archive*, no. 072, p. 22, 2004, Accessed: Nov. 30, 2023. [Online]. Available: https://eprint.iacr.org/2004/072.pdf

[52] J. Patarin, “Asymmetric Cryptography with a Hidden Monomial,” 1996, pp. 45–60. doi: 10.1007/3-540-68697-5_4.

[53] J. Hoffstein, J. Pipher, and J. H. Silverman, “NTRU: A ring-based public key cryptosystem,” 1998, pp. 267–288. doi: 10.1007/BFb0054868.

[54] J. Hoffstein, J. Pipher, and J. H. Silverman, “Public key cryptosystem method and apparatus,” 6081597A, Aug. 19, 1997 Accessed: Nov. 30, 2023. [Online]. Available: https://patents.google.com/patent/US6081597

[55] D. Coppersmith and A. Shamir, “Lattice Attacks on NTRU,” 1997, pp. 52–61. doi: 10.1007/3-540-69053-0_5.

[56] J. Hoffstein and J. Silverman, “Optimizations for NTRU,” in *Public-Key Cryptography and Computational Number Theory*, Berlin, New York: DE GRUYTER. doi: 10.1515/9783110881035.77.

[57] G. Alagic *et al.*, “Status report on the third round of the NIST Post-Quantum Cryptography Standardization process,” Jul. 2022. doi: 10.6028/NIST.IR.8413.

[58] Q. H. Dang, “Secure Hash Standard,” Gaithersburg, MD, Jul. 2015. doi: 10.6028/NIST.FIPS.180-4.

[59] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “The First Collision for Full SHA-1,” in *CWI Amsterdam, Google Research*, 2017, pp. 570–596. doi: 10.1007/978-3-319-63688-7_19.

[60] M. Stevens *et al.*, “Announcing the first SHA-1 collision,” Google Research. Accessed: Nov. 24, 2023. [Online]. Available: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

[61] kockmeyer, “File:SHA-2.svg - Wikipedia,” Wikimedia Commons. Accessed: Dec. 04, 2023. [Online]. Available: https://en.wikipedia.org/wiki/File:SHA-2.svg

[62] M. J. Dworkin, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” Gaithersburg, MD, Jul. 2015. doi: 10.6028/NIST.FIPS.202.

[63] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, “KECCAK Specification Summary,” TeamKeccak. Accessed: Nov. 24, 2023. [Online]. Available: https://keccak.team/keccak_specs_summary.html

[64] Armbrust, “File:SpongeConstruction.svg - Wikipedia,” Wikimedia Commons. Accessed: Dec. 04, 2023. [Online]. Available: https://commons.wikimedia.org/wiki/File:SpongeConstruction.svg

[65] D. Johnson, A. Menezes, and S. Vanstone, “The Elliptic Curve Digital Signature Algorithm (ECDSA),” *Int J Inf Secur*, vol. 1, no. 1, pp. 36–63, Aug. 2001, doi: 10.1007/s102070100002.

[66] S. Josefsson and I. Liusvaara, “Edwards-Curve Digital Signature Algorithm (EdDSA),” Jan. 2017. doi: 10.17487/RFC8032.

[67] D. Moody, “Module-Lattice-Based Digital Signature Standard,” 2023. doi: 10.6028/NIST.FIPS.204.ipd.

[68] S. Bai *et al.*, “CRYSTALS-Dilithium --- Algorithm Specifications and Supporting Documentation,” *Crystals (Basel)*, no. 3.1, p. 38, Feb. 2021, Accessed: Dec. 03, 2023. [Online]. Available: https://pq-crystals.org/dilithium/

[69] V. Lyubashevsky, “Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures,” 2009, pp. 598–616. doi: 10.1007/978-3-642-10366-7_35.

[70] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, “CRYSTALS-Dilithium,” in *Hardware Architectures for Post-Quantum Digital Signature Schemes*, Cham: Springer International Publishing, 2021, pp. 13–30. doi: 10.1007/978-3-030-57682-0_2.

[71] M. E. Hellman, B. W. Diffie, and R. Merkle, “Cryptographic apparatus and method,” 4,200,770, Sep. 06, 1977 Accessed: Nov. 25, 2023. [Online]. Available: https://patents.google.com/patent/US4200770A/

[72] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Trans Inf Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: 10.1109/TIT.1976.1055638.

[73] D. Gillmor, “Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS),” Aug. 2016. doi: 10.17487/RFC7919.

[74] S. Blake-Wilson, D. Johnson, and A. Menezes, “Key agreement protocols and their security analysis,” 1997, pp. 30–45. doi: 10.1007/BFb0024447.

[75] C. Kudla and K. G. Paterson, “Modular Security Proofs for Key Agreement Protocols,” 2005, pp. 549–565. doi: 10.1007/11593447_30.

[76] H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF),” May 2010. doi: 10.17487/rfc5869.

[77] M. Marlinspike and T. Perrin, “The X3DH Key Agreement Protocol,” *Signal Protocol Documentation*, no. 1, p. 11, Nov. 2016, Accessed: Dec. 04, 2023. [Online]. Available: https://www.signal.org/docs/specifications/x3dh/x3dh.pdf

[78] P. Rogaway, “Authenticated-encryption with associated-data,” in *Proceedings of the 9th ACM conference on Computer and communications security*, New York, NY, USA: ACM, Nov. 2002, pp. 98–107. doi: 10.1145/586110.586125.

[79] J. Bos *et al.*, “CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM,” in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, Apr. 2018, pp. 353–367. doi: 10.1109/EuroSP.2018.00032.

[80] D. Moody, “Module-Lattice-Based Key-Encapsulation Mechanism Standard,” 2023. doi: 10.6028/NIST.FIPS.203.ipd.

[81] R. Avanzi *et al.*, “CRYSTALS-Kyber -- Algorithm Specifications And Supporting Documentation,” *Crystals (Basel)*, no. 3.02, p. 43, Aug. 2021, Accessed: Dec. 05, 2023. [Online]. Available: https://pq-crystals.org/kyber

[82] E. Fujisaki and T. Okamoto, “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” 1999, pp. 537–554. doi: 10.1007/3-540-48405-1_34.

[83] S. Liu and A. Sakzad, “Lattice Codes for CRYSTALS-Kyber,” *IEEE*, Aug. 2023, doi: 10.36227/techrxiv.24031218.

[84] E. Kret and R. Schmidt, “The PQXDH Key Agreement Protocol,” *Signal Protocol Documentation*, no. 2, p. 19, May 2023, Accessed: Dec. 04, 2023. [Online]. Available: https://www.signal.org/docs/specifications/pqxdh/pqxdh.pdf

[85] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2012. doi: 10.1017/CBO9780511976667.

[86] C. Bernhardt, *Quantum Computing for Everyone*. The MIT Press, 2019. doi: 10.7551/mitpress/11860.001.0001.

[87] Smite-Meister, "File:Bloch sphere.svg - Wikipedia," Wikimedia Commons. Accessed: Dec. 06, 2023. [Online]. Available: https://en.wikipedia.org/wiki/File:Bloch_sphere.svg

[88] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation," *Fortschritte der Physik*, vol. 48, no. 9–11, pp. 771–783, Sep. 2000, doi: 10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E.

[89] World Economic Forum, "An Insight, An Idea with Sundar Pichai - Quantum Computing," YouTube. Accessed: Dec. 06, 2023. [Online]. Available: https://www.youtube.com/watch?v=eh78EUTas34

[90] K. Townsend, "Solving the Quantum Decryption 'Harvest Now, Decrypt Later' Problem," SecurityWeek. Accessed: Dec. 06, 2023. [Online]. Available: https://www.securityweek.com/solving-quantum-decryption-harvest-now-decrypt-later-problem/

[91] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds," in *Cryptology ePrint Archive*, no. 374, 2010, pp. 299–319. doi: 10.1007/978-3-642-13190-5_15.

[92] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique Cryptanalysis of the Full AES," 2011, pp. 344–371. doi: 10.1007/978-3-642-25385-0_19.

[93] D. Boneh and R. Venkatesan, "Breaking RSA may not be equivalent to factoring," 1998, pp. 59–71. doi: 10.1007/BFb0054117.

[94] T. Kleinjung *et al.*, "Factorization of a 768-Bit RSA Modulus," no. 1.4, 2010, pp. 333–350. doi: 10.1007/978-3-642-14623-7_18.

[95] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thome, and P. Zimmerman, "795-bit factoring and discrete logarithms," INRIA. Accessed: Nov. 25, 2023. [Online]. Available:

https://web.archive.org/web/20191202190004/https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2019-December/001139.html

[96] "Certicom ECC Challenge," *Certicom Research*, p. 47, Nov. 2009, Accessed: Nov. 25, 2023. [Online]. Available: https://www.certicom.com/content/dam/certicom/images/pdfs/challenge-2009.pdf

[97] A. Zieniewicz and J. L. Pons, "Kangaroo," GitHub. Accessed: Nov. 25, 2023. [Online]. Available: https://github.com/JeanLucPons/Kangaroo

[98] D. Khovratovich, C. Rechberger, and A. Savelieva, "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family," 2012, pp. 244–263. doi: 10.1007/978-3-642-34047-5_15.

[99] F. Mendel, T. Nad, and M. Schläffer, "Improving Local Collisions: New Attacks on Reduced SHA-256," 2013, pp. 262–278. doi: 10.1007/978-3-642-38348-9_16.

[100] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," *Quantum Inf Comput*, vol. 3, no. 2, pp. 175–185, Mar. 2003, doi: 10.26421/QIC3.2-8.

[101] L. K. Grover, "Quantum Mechanics Helps in Searching for a Needle in a Haystack," *Phys Rev Lett*, vol. 79, no. 2, pp. 325–328, Jul. 1997, doi: 10.1103/PhysRevLett.79.325.

[102] S. Imre and F. Balazs, "The Generalized Quantum Database Search Algorithm," *Computing*, vol. 73, no. 3, pp. 245–269, Oct. 2004, doi: 10.1007/s00607-004-0089-8.

[103] S. Boixo *et al.*, "Characterizing Quantum Supremacy in Near-Term Devices," Jul. 2016, doi: 10.1038/s41567-018-0124-x.

[104] F. Arute *et al.*, "Supplementary information for 'Quantum supremacy using a programmable superconducting processor,'" Oct. 2019, doi: 10.1038/s41586-019-1666-5.

[105] Y. Wang, Y. Li, Z. Yin, and B. Zeng, “16-qubit IBM universal quantum computer can be fully entangled,” *npj Quantum Inf*, vol. 4, no. 1, p. 46, Sep. 2018, doi: 10.1038/s41534-018-0095-x.

[106] D. Castelvecchi, “IBM releases first-ever 1,000-qubit quantum chip,” *Nature*, Dec. 2023, doi: 10.1038/d41586-023-03854-1.

[107] A. Blackstone, *Principles of Sociological Inquiry - Qualitative and Quantitative Methods*. Saylor Academy, 2018. Accessed: Jan. 05, 2024. [Online]. Available: https://saylordotorg.github.io/text_principles-of-sociological-inquiry-qualitative-and-quantitative-methods/

[108] S. Goundar, “Chapter 3 - Research Methodology and Research Method,” in *Cloud Computing*, ResearchGate, 2012, p. 43. Accessed: Jan. 05, 2024. [Online]. Available: https://www.researchgate.net/publication/333015026

[109] K. Isom, “Chapter 1: Introduction,” in *Practical Cryptography with Go*, LearnPub, 2015. Accessed: Jan. 06, 2024. [Online]. Available: https://leanpub.com/gocrypto/read

[110] K. Bhargavan, C. Jacomme, F. Kiefer, and R. Schmidt, “An Analysis of Signal’s PQXDH,” Cryspen Blog. Accessed: Jan. 12, 2024. [Online]. Available: https://cryspen.com/post/pqxdh/

[111] S. S, “A Study of Software Development Life Cycle Process Models,” *SSRN Electronic Journal*, 2017, doi: 10.2139/ssrn.2988291.

[112] I. Sommerville, *Software Engineering*, 10th ed. United Kingdom, Scotland: Pearson, 2016. Accessed: Dec. 30, 2023. [Online]. Available: https://testbankbell.com/product/software-engineering-10th-edition-sommerville-solutions-manual/

[113] K. Beck *et al.*, “Manifesto for Agile Software Development.” Accessed: Dec. 30, 2023. [Online]. Available: https://agilemanifesto.org/

[114] A. Mishra and D. Dubey, “A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios,” *International Journal of Advance Research in Computer Science and Management Studies*, vol. 1, no.

5, p. 6, Oct. 2013, Accessed: Dec. 30, 2023. [Online]. Available: https://ijarcsms.com/docs/paper/volume1/issue5/V1I5-0008.pdf

[115] “Visual Studio Code - Code Editing. Redefined,” Microsoft. Accessed: Jan. 07, 2024. [Online]. Available: https://code.visualstudio.com/

[116] “Welcome to Apache NetBeans,” Apache Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://netbeans.apache.org/front/main/

[117] “Welcome to Python.org,” Python Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://www.python.org/

[118] “Java | Oracle,” Oracle. Accessed: Jan. 07, 2024. [Online]. Available: https://www.java.com/en/

[119] “sqlite3 — DB-API 2.0 interface for SQLite databases,” Python Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://docs.python.org/3/library/sqlite3.html

[120] “MySQL,” Oracle. Accessed: Jan. 07, 2024. [Online]. Available: https://www.mysql.com/

[121] H. Eijs, “pycryptodome - PyPI,” Python Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://pypi.org/project/pycryptodome

[122] H. Schlawack, “argon2-cffi - PyPI,” Python Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://pypi.org/project/argon2-cffi/

[123] Badlock, “py2crypt - PyPI,” Python Software Foundation. Accessed: Jan. 07, 2024. [Online]. Available: https://pypi.org/project/py2crypt/

[124] GiacomoPope, “GitHub - GiacomoPope/kyber-py: A pure python implementation of CRYSTALS-Kyber,” GitHub. Accessed: Jan. 07, 2024. [Online]. Available: https://github.com/giacomopope/kyber-py

[125] GiacomoPope, “GitHub - GiacomoPope/dilithium-py: A pure python implementation of CRYSTALS-Dilithium,” GitHub. Accessed: Jan. 07, 2024. [Online]. Available: https://github.com/giacomopope/dilithium-py

[126] “Telegram Messenger,” Telegram FZ LLC. Accessed: Jan. 10, 2024. [Online]. Available: https://www.telegram.org/

[127] “WhatsApp | Secure and Reliable Free Private Messaging and Calling,” WhatsApp LLC. Accessed: Jan. 10, 2024. [Online]. Available: https://www.whatsapp.com/

[128] “A New Kind of Instant Messaging - Tox,” Tox. Accessed: Jan. 10, 2024. [Online]. Available: https://tox.chat/

[129] “ProtoPie.” Studio XID, 2023. Accessed: Jan. 10, 2024. [Online]. Available: https://www.protopie.io/

[130] “tkinter — Python interface to Tcl/Tk — Python 3.12.3 documentation,” Python Software Foundation. Accessed: May 30, 2024. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[131] “socket — Low-level networking interface — Python 3.12.3 documentation,” Python Software Foundation. Accessed: May 30, 2024. [Online]. Available: https://docs.python.org/3/library/socket.html

[132] “hashlib — Secure hashes and message digests — Python 3.9.19 documentation,” Python Software Foundation. Accessed: May 30, 2024. [Online]. Available: https://docs.python.org/3.9/library/hashlib.html

[133] “secrets — Generate secure random numbers for managing secrets — Python 3.12.3 documentation,” Python Software Foundation. Accessed: May 30, 2024. [Online]. Available: https://docs.python.org/3/library/secrets.html

[134] HarryR, “GitHub - HarryR/PyPQC: Python Wrappers for Post-Quantum Cryptography,” GitHub. Accessed: May 30, 2024. [Online]. Available: https://github.com/HarryR/PyPQC

[135] *ECMA-404 - The JSON Data Interchange Syntax*, 2nd ed. Ecma International, 2017. Accessed: May 30, 2024. [Online]. Available: https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf

[136] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, “High-speed high-security signatures,” *J Cryptogr Eng*, vol. 2, no. 2, pp. 77–89, Sep. 2012, doi: 10.1007/s13389-012-0027-1.

[137] theellipsis009, “Man in the Middle attack in Diffie-Hellman Key Exchange - GeekForGeeks,” GeekForGeeks. Accessed: Jan. 11, 2024. [Online]. Available: geeksforgeeks.org/man-in-the-middle-attack-in-diffie-hellman-key-exchange/

[138] “Oppenheimer (film) - Wikipedia,” Wikipedia. Accessed: Jan. 12, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Oppenheimer_(film)

[139] “draw.io - Flowchart and Diagram builder,” draw.io. Accessed: Jan. 12, 2024. [Online]. Available: https://app.diagrams.net/

All un-referenced diagrams are designed with own work using draw.io [139].

3 4

[3] Systematic Literature Review has not been done to this thesis.

[4] The Author of this work has exerted much efforts to keep the bibliography as accurate as possible; however there might be duplicates or inaccuracies in references. The use of DOI is extended wherever possible.

APPENDICES

APPENDIX A: Project Schedule – Gantt Chart

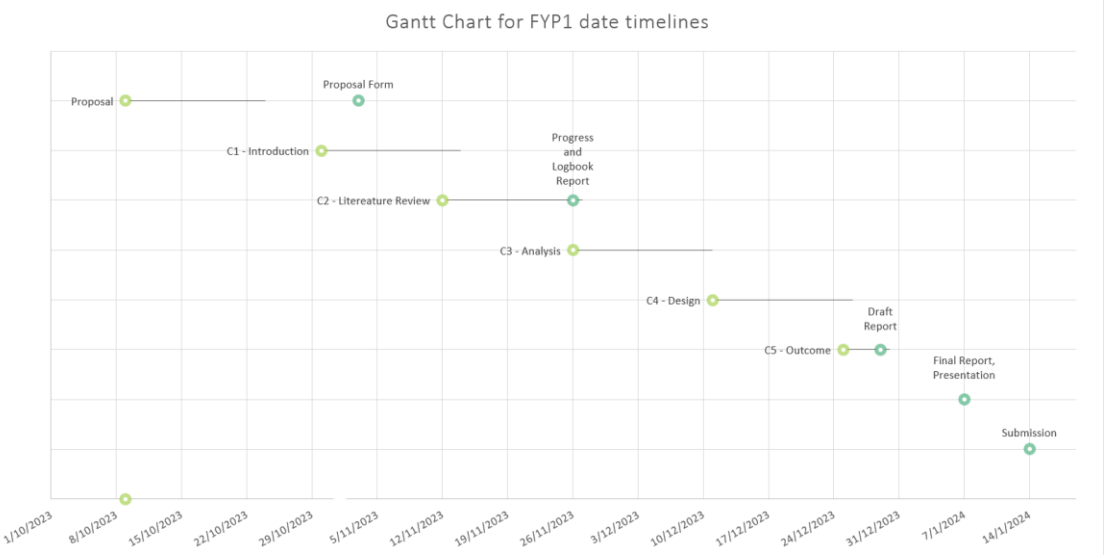


Figure A 1 Gantt Chart for FYP1 date timelines

[END PAGE]

retr0 et. al.,

2023–2024