Eötvös Loránd University
Faculty of Informatics

Post Quantum Secure Authentication methods suitable for Quantum Key Distribution

Dr. Zoltán Istenes
Professor at ELTE
Dr. Andreas Peter
Assistant Professor at UTwente
Dr. Benedek Kovács
Senior Specialist at Ericsson

Mohammad Rashid Zamani
Computer Science

UNIVERSITY OF TWENTE.

Budapest, 2018.

Statement

of thesis submission and originality

I hereby confirm the submission of the Master Thesis Work on the Computer Science MSc course with author and title:

Name of Student:	*Mohammad Rashid Zamani*
Code of Student:	*FANMN7*
Title of Thesis:	*Post Quantum Secure Authentication Methods Suitable for Quantum Key Distribution*
Supervisor:	*Dr. Zoltán Istenes*
	Computer Science

at Eötvös Loránd University, Faculty of Informatics.

In consciousness of my full legal and disciplinary responsibility I hereby claim that the submitted thesis work is my own original intellectual product, the use of referenced literature is done according to the general rules of copyright.
I understand that in the case of thesis works the following acts are considered plagiarism:

- literal quotation without quotation marks and reference;
- citation of content without reference;
- presenting others' published thoughts as own thoughts.

Budapest, July 4, 2018

student

Abstract

Quantum Key Distribution occurs over two communication channels: classical and quantum. While the quantum channel is somewhat secure by quantum principles, the classical channel need to be authenticated using cryptographic algorithms. Unfortunately, the authentication algorithms and the methods suggested are not taking into consideration the problems practical implementations are facing. Moreover, there are areas which has not been discussed by the literature such as the quantum application layer in communication over classical channel. This study identifies the hurdles practical implementations of QKD are facing and proposes a solution which take into consideration these needs. Moreover, it suggests new protocols and approaches suitable for QKD post processing authentication.

Dedication

To the butterfly and the bee.

Acknowledgement

"Always two there are,
no more,
no less.
A master and an apprentice."

— *Yoda*

Contents

List of Figures

List of Tables

Acronyms

AES	Advanced Encryption System. 4, 18, 19, 23, 24, 35, 43, 49–54, 77, 80
AH	Internet Protocol Security Authentication Header. vii, 65–67, 76, 83, 84
AL	Application Layer. 34, 62, 77
API	Application Programming Interface. 40, 57, 62, 77
APPP	Authenticated Post Processing Protocol. vi–ix, 12, 65, 67–76, 79–82, 85
BB84	QKD protocol, Bennett and Brassard proposed in 1984. v, 5–7, 9, 10, 13–17, 35, 36, 42, 44–47, 59, 61, 68, 69
CBC	Cipher Block Chaining. 20, 21, 24, 25
CMAC	Cipher-based Message Authentication Code. 20, 51
cpb	Cycles per Byte. 10, 27, 29, 31, 49, 51, 53, 54
CPU	Centeral Processing Unit. 19, 27, 28, 31, 48–52
CRC	Cyclic Redundancy Check. 44
CTR	Counter Mode. 22, 23, 31, 51, 52
DARPA	Defense Advanced Research Projects Agency. v, 34–37, 57, 59, 65, 80
DES	Data Encryption Standard. 2, 25, 50

ECB Electronic Codebook. 21

ESP Internet Protocol Security Encapsulating Security Payload. 35, 84

ETSI European Telecommunications Standards Institute. 57, 63, 80

FFT Fast Fourier Transform. 31

FIPS Federal Information Processing Standard. 28, 30

Gbps Giga bit per second. 10, 80, 81

GCM Galois/Counter Mode. vii, 22, 23, 51–54

GHz Giga Hertz. 49

GMAC Galois/Counter Mode Message Authentication Code. 22–24, 31, 52, 55

HAIFA HAsh Iterative FrAmework. 26, 30

HMAC Hash-Based Message Authentication Code. 29, 30, 35, 53, 55, 59

ICB Initial Counter Block. 22

ICV Integrity Checking Value. 66

IETF Internet Engineering Task Force. 50, 52

IKE Internet Key Encapsulation protocol. 35, 36, 57

IP Internet Protocol. 34, 38, 39, 42, 48, 49, 63–65, 67, 71, 73, 74, 76, 82–84

IPSec Internet Protocol Security. vii, 34–36, 41, 48, 49, 61, 62, 65–67, 76, 77, 81–84

ITS Information-Theoretic Secure. 2–4, 10, 18, 31, 34, 36–38, 43, 48, 50, 54–56, 66, 76, 80, 85

IV Initial Vector. 22

KDF Key Derivation Function. 30, 35, 55, 59, 65, 72

KEP Key Encapsulation Protocols. 34, 35, 57, 64

KMAC KECCAK-family Message Authentication Code. 29, 30, 53, 55, 76

KML Key Management Layer. vi, 34, 43, 62–64, 67, 76, 77, 80, 85

LDPC Low-Density Parity-Check code. 16, 44, 46, 47

MAC Message Authentication Code. v, 8, 9, 13, 14, 17–21, 23–25, 28–31, 49–51, 53, 54, 80, 85

MBps Mega Bytes per Second. 49, 80

Mbps Mega bits per Second. 80

MD Merkle-Damgård. 26, 30, 59

MHz Mega Hertz. 27

MITM Man-In-The-Middle. 8, 10, 15, 46

MTU Maximum Transfer Unit. 65, 83

NIC Network Interface Controller. 49

NIST National Institute of Standards and Technology. 28, 30, 50, 52, 54, 64

NQC Network-centric Quantum Communication. v, 43

NSA National Security Agency. 50

OSI Open Systems Interconnection. 34, 38, 41, 62, 64, 67

OTP One Time Pad. 2, 3, 36, 37, 41, 43, 57, 58, 77, 80, 85

PDU Payload Data Unit. 65

PKI Public Key Infrastructure. 2, 41

PMAC Parallelized Message Authentication Code. 21, 24

PRF	Psuedo Random Function. 18, 19, 23, 30
Q3P	Quantum Point-to-Point Protocol. vii, 37–40, 48, 49, 59, 64, 82–84
QAN	Quantum Access Node. 38
QBB	Quantum BackBone. vii, 36–39, 41, 58, 62
QBER	qubit Error Rate. 14–16, 46, 69, 72
QID	Quantum-device Identifier. 63, 71, 72
QKD	Quantum Key Distribution. ii, v–viii, 4, 5, 9–14, 17, 28, 32–43, 45–49, 56–59, 61–63, 65, 68–72, 76, 77, 79, 80, 82, 84, 85
QKDAL	Quantum Key Distribution Application Layer. 40, 83
QKDLL	Quantum Key Distribution Link Layer. 39, 83
QKDNL	Quantum Key Distribution Network Layer. 39, 83, 84
QKDTL	Quantum Key Distribution Transport Layer. 39, 83
QL	Quantum Layer. 34, 62–64, 76
QPPD	Quantum Post Processing Daemon. vi, 63–68, 71–74, 76, 77, 81–83, 85
RFC	Request For Comment. 49, 50, 65
SA	Security Associate. 35, 65, 67, 76, 77
SAD	Security Associate Database. 35, 49, 65, 67, 76, 77
SECQOC	SEcure COmmunication based on Quantum Cryptography. v, vii, 36–41, 56–59, 63, 64, 77, 81, 82, 84
SHA	Secure Hashing Algorithm. 26–28, 35, 53, 55, 59, 85
SPD	Security Policy Database. 49, 64, 65, 76, 77
SPI	Security Parameter Index. 65, 76

TCP Transmission Control Protocol. 34, 38, 39, 42, 49, 64–67, 69, 71, 73–76, 81–83

UMAC Universal Hash Message Authentication Code. 23, 31, 54

VMAC 64bit Universal Hash Message Authentication Code. 23, 31, 54, 55, 59, 80

VPN Virtual Private Network. 35, 36, 59

WDM Wave De-multiplexing Module. 34, 42, 81

XOF Extended Output Function. 28, 30, 59, 72

XOR Exclusive or. 2, 18, 20–25, 27, 30, 54

Chapter 1

Background

"Three can keep a secret,
if two of them are dead."

— Benjamin Franklin

Quest for secure communication dates back to ancient civilizations, near 2500 years ago, where simple mathematics and physical objects were used to create cryptosystem[1]. Naturally, attempts to break cryptosystems started. There are evidence indicating the challenge began in more than thousand years ago[2]. Although these early attempts might have had the sole *"evil"* intention of breaking the cryptosystems; nowadays cryptanalysis plays a major role on defining security of a cryptosystems. Throughout the course of history, as the science and technology advanced, cryptosystems enhanced too. Inevitably, these advancements improved cryptanalysis techniques to a degree where simple substitution and transposition cipher systems were not secure anymore. In order to gain a better chance against cryptanalysis, it was a common practice to add *security through obscurity* of the cyrptosystems.

Technology advancements enabled communication over long distances, and subsequently the need for secure communication over long distances arose – the biggest hurdle was to share a secret between the parties. Secrets could have been distributed either through face-to-face meetings, which might not have been possible and/or practical in many scenarios. Or they were shared via a trusted courier, that introduced a third party and increased the

[1] Cryptography existed for near 4000 years ago in Old Kingdom of Egypt but not for the purpose of secure communication. The first evident use of cryptography for the purpose of secure communication is Mlecchita Vikalpa, a substitution cipher listed as one of the arts in Kamasutra, for lovers to exchange private messages. Scytale transposition cipher, the first known physical object used as an authenticated encryption system by military in ancient Greek to authenticate and decrypt messages.

[2] Alkindi books on frequency analysis (800AD).

attack surfaces of the system such. Instinctively, a secret is safer when shared with less parties, as the opening quote to the chapter suggests. One reason justifying employment of security through obscurity could have been the high possibility of key leakage at that time – hiding the know-hows of the cryptosystem could be viewed as countermeasure – if an adversary get hold of the key, she would struggle to find out how to use the key.

Last two centuries, however, cryptography entered a new era so called *modern cryptography*. Alongside with rapid developments in science and technology during this period, in 19th century, it was suggested by Krechhoff that the security of cryptosystems shall solely rely on the secrecy of the key and he ruled out any need for obscurity in the mechanism of cryptosystem. This amplified the importance of the key which in return, magnified the key distribution problem. In fact, presently *"the strength of a cryptographic algorithm is directly linked to the difficulty of obtaining the secret key by the adversaries; thus, key distribution schemes can be identified as one of the most sensitive parts of the security systems in communication networks"*[DAGS08]. Another positive influence of Krechhoff principle was, as slowly as it got adopted[3], allowed cryptographers from around the world to exchange ideas and study cryptography more openly, similar to other sciences.

One of the most fundamental findings in modern cryptography, with no doubt, is *asymmetrical cryptography* — a novel technique in which a pair of keys is presented to each party in communication – one key for encryption, and one key for decryption. In these schemes, the encryption key (*public key*) is public knowledge and the decryption key (*secret key)* is kept secret. Messages are encrypted using one's public key and could only be decrypted by the one's corresponding secret key. Asymmetrical cryptography is the best solution to the key distribution problem and it is the backbone to *Public Key Infrastructure (PKI)*[4].

Another influential finding in 20th century was *information theory* and the concept of *information-theoretic* and *perfect* secrecy. An *Information-Theoretic Secure (ITS)* cryptosystem is considered cryptanalytically unbreakable even under the assumption that unlimited computing power is presented to the adversary – there is simply not enough *information* available to perform any cryptanalysis. Besides, if the cipher text created by the encryption algorithm of a cryptosystem does not reveal any information about the corresponding plain text, the cryptosystem in question has perfect secrecy. Claude Shannon, father of information theory, proved *One Time Pad (OTP)*[5] is ITS and has perfect secrecy.

[3]Data Encryption Standard also known as Data Encryption Standard (DES) algorithm was not known to the public even in the 1990s.

[4]PKI is the infrastructure used today over Internet and many other type of networks for key distribution based on asymmetrical cryptography. Asymmetrical cryptography is also known as public key cryptography.

[5]OTP is an encryption scheme in which the plain text is `XOR`ed with the key size of same length. Each key is only used once.

It was later proved for any system to have *perfect secrecy*, it is needed to have the same key size as the message and use each key only once, similar to OTP.

It seems like a very rational decision to utilize cryptographic algorithms with these idealistic degrees of security and put an end on the competition between *"Alice"*, *"Bob"* and *"Eve* – leaving *"brute-force"*[6] as the only attack possible. Most of the cryptosystems in-use today though, do not exercise these optimistic levels of security, and this is mainly due to the following:

- Most of the algorithms with these levels of security are not efficient – either computationally or in terms of key consumption.
- Unlimited computational power, clearly, does not exist. All our commercially available computational power is limited and the annual rate of its growth in future is predictable to some degree[7]. Thus, for many *"industrial settings"* scenarios *computational security* or *conditional security*[8] which is secure against current and near future computational power suffices.

Hence, many modern cryptographic algorithms gain their security legitimacy through *computational hardness assumption* i.e. to employ a trapdoor function[9] based on a problem which is assumed to be almost impossible to reverse with limited-existing computational power. Nevertheless, these assumptions may turned wrong – either through discovery of a new algorithm or the increase in computational power which would exceed a level that the problem in question would not be considered hard to solve anymore. Shor's quantum algorithm is an example of such. Deploying ITS cryptography would give us security in future, a term known as *forward secrecy*.

1.1 Quantum Information Theory and Cryptography

During the last decades and with the progress in the field of quantum physics a new theory emerged; *quantum information theory* which deals with *quantum information* i.e. information held in the state of a quantum system. *Quantum information processing*[10] opened new horizons both for cryptography and cryptanalysis, in another words, it breaks and creates cryptosystems.

[6] An attack in which the intruder tries all the possibilities.

[7] Moore's law predicts the computation power doubles about every two years.

[8] These are security notions under certain restricted computational power or other conditions.

[9] A one-way function which could be reversed efficiently by knowing a trapdoor or a secret value.

[10] Analogous notations of transmitting and processing information with algorithms and mathematics of computer science using *quantum computer* and *qubits* as basic element; instead of using digital computer and bits in classical information processing

Developed in 1994, Shor's quantum algorithm efficiently solves integer factorization problem, the discrete logarithm problem, and the elliptic-curve discrete logarithm problem in polynomial time. These problems are building blocks to almost all the key distribution algorithm presently in use and Shor's algorithm obsoletes security of many modern cryptography algorithms including RSA[11] and Diffie-Hellman[12]. Quantum computers are real threat to these cryptosystem. Presently, however, computation over small number of qubits with quantum computers are performed in research projects and development of actual quantum computers are still in progress[13]. Nevertheless, cryptographers are working on defining a class of cryptosystems resilient against quantum attacks called post-quantum secure i.e. there exist no efficient quantum or classic algorithm known to solve the problem these systems are based on – this does not imply unconditional security e.g. AES[14] is post-quantum secure but not ITS.

On a brighter side, there are principles in the nature of quantum mechanics and quantum field mechanics which allow performing cryptographic tasks with unconditional security. These principles – namely the aftermath of *Heisenberg*'s uncertainty principle and its result: the observer effect, *Quantum entanglement* concept, and *no-cloning theorem* – are the essential parts of *quantum cryptography*. These quantum physics principles make it impossible for an adversary to eavesdrop on a communication over a quantum channel without being noticed and even forbid obtaining a copying of the communication. Hence, *quantum cryptography* is ITS.

In 1984, Charles H. Bennett and Gilles Brassard illustrated one of the earliest applications of quantum cryptography. In their paper they demonstrated Quantum Key Distribution (QKD) to share a secret (key) between two parties using *"elementary quantum systems, such as polarized photons [...] to transmit digital information"*[BB84]. Nowadays, there exists various applications of quantum cryptography[15], yet recently QKD is receiving great attention. Progress in the technology has allowed practical developments of quantum communications within range of hundreds of kilometers. Some of these implementations are elaborated on in section 2.3. Given the fact that security of known key distribution algorithms are compromised by Shor's quantum algorithm; QKD seems to be a perfect alternative, given its unconditional security and proved practicality.

[11] Rivest–Shamir–Adleman asymmetrical encryption scheme based on factorization.

[12] Asymmetrical encryption based on discrete logarithm problem.

[13] The best progress as of 2018 is Bristlecone, Google's 72 qubits quantum processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`

[14] Advanced Encryption System is a block cipher explained in section 2.2.1.

[15] Quantum Authentication, Quantum Multi Party Communication, and Quantum Commitments are some of other applications developed in quantum cryptography.

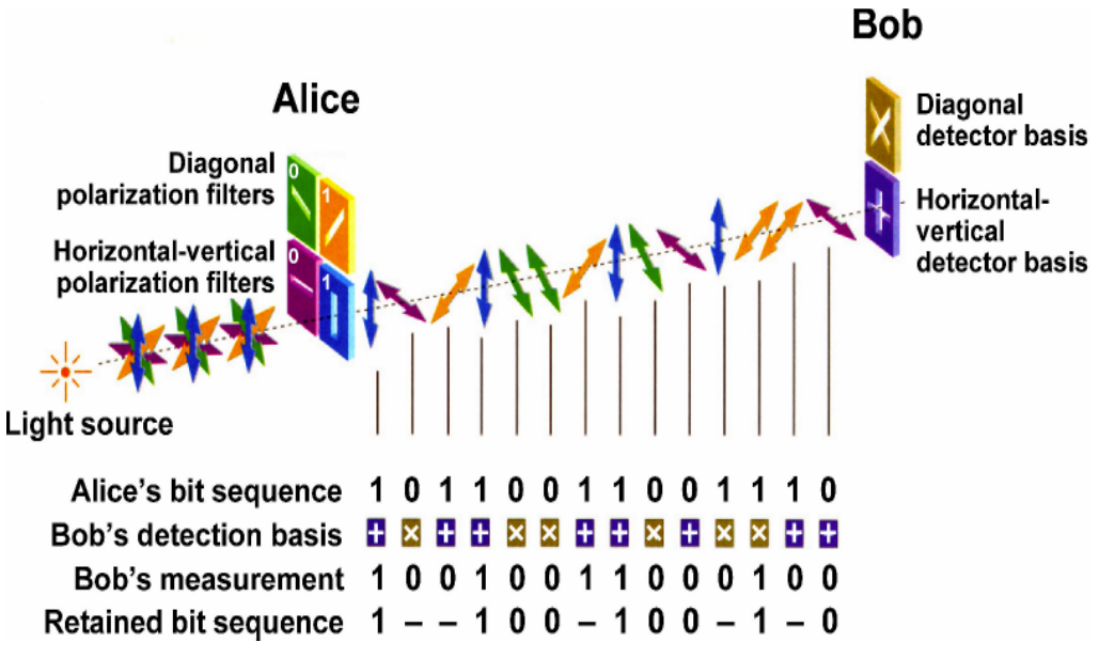

Figure 1.1: BB84 Quantum Key Distribution schematic protocol run.[Sha11]

1.2 Quantum Key Distribution

The method introduced by Bennett and Brassard in 1984 also known as BB84 protocol, inspired many others to develop different QKD protocols. Besides communication over quantum channels, most of these protocols including BB84 , require communication over an authenticated classical channel as well – means to provide this classic authenticated channel is the core of this study. Since BB84 is the foundation to many of QKD protocols, I have chose it as the QKD protocol of this research. The scope of this research is later detailed in section 1.4 and importance of authenticity of the communication over classical in QKD will be discussed in detailed in this dissertation.

1.2.1 BB84

Digital information in BB84 are encoded into elementary quantum system i.e. the polarization of a single photon. This is done by emitting single photons through different filters. Figure 1.1 depicts an illustration of BB84 semantic between famous *Alice* and *Bob*. As shown in the picture there are four filters with two different basis – one rectilinear (horizontal-vertical), and the other one orthogonal (diagonal). Therefore, there are two distinct polarizations for both bit 0 and bit 1 from these two basis. *Alice* encodes a random bit string (i.e. the key) using the described method. To encode each bit, she selects the filter's basis randomly. *Bob* also, chooses randomly between rectilinear and orthogonal

basis detectors and measures each photon. Once *Bob* receives all the photons the communication over quantum channel is finished – as discussed earlier this communication is prone to undetected eavesdropping.

Figure 1.2 demonstrates basic facts about polarized photon. Segment **(a)** of the picture shows when a photon is beamed into a polarizing filter, horizontal in this case, the result will always be as expected. If the measurements happen with the detector in the same basis as the one used to polarized the photon in the first place, the result of the measurement would surely be correct and as expected. This is shown in segment **(b)**, where measuring a single photon polarized horizontally using rectilinear basis detector yields correct result all the time. However, when the basis used in polarization differs from the one used in detection, the outcome would be probabilistic. Similar to section **(c)** in figure 1.2, where a detector with orthogonal basis is used to measure a photon polarized in rectilinear basis, the result would be measured as a photon angled 45° or 135° with the same probability.

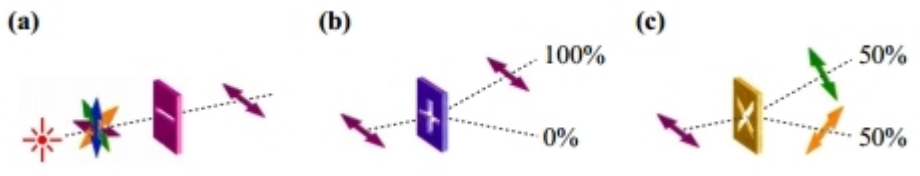

Figure 1.2: Basics of Photon Polarization [Sha11].

Since both *Alice* and *Bob* choose the basis randomly, mismatch in the basis are expected. Indeed, all the measurements *Bob* performs in different basis from the one *Alice* used to polarized the photon, are probabilistic and not reliable; thus, should be discard by both parties. Additionally, it is possible that that some photons are lost in the transmission or not detected correctly by *"Bob's imperfectly-efficient detectors"*[BB84]. This *Post Processing* procedure happens over a public authenticated classical channel, and once this procedure is finished both parties retained a bit sequence only known to themselves.

It is worthwhile to mention that *post processing* is the *Achilles heel* of BB84 – communication over the quantum channel is secured by quantum physic principles. If an intruder wishes to eavesdrop on quantum channel without being noticed, she needs to compromise the authenticity of communication over the classical channel, and then security of the whole protocol is jeopardized.

Post Processing

Once quantum communication over the quantum channel is finished, post processing initiates by public discussion between two parties. As stressed out before, this communication

should be authenticated i.e. recipient is certain about the identity of the sender to a very high degree and is convinced no alteration has happened in the content of the messages. However, content of the massages do not need to be encrypted – having knowledge of the information discussed over the public channel does not endangers the secrecy of the shared bit string over the quantum channel[BB84].

Figure 1.3, is the original protocol flow of BB84. The protocol is divided into quantum transmission, explained earlier, and public discussion. In the proposed method, *Bob* will start the public discussion once he has notified *Alice* that he received the photons. He start the procedure by disclosing the basis he used for detection, and *Alice* will confirm the correct ones. This step is also know as **key sifting**. Outcome of this step would be the presumably shared bit string between the parties.

The next step during post processing is the **confirmation**, where *Bob* reveals random bits of the presumably shared bit string and *Alice* confirms if they are correct. After this step they can calculate an estimated an error rate based on miss matches – high error rate could be a sign of eavesdropping. Since introduction of BB84, many variants and

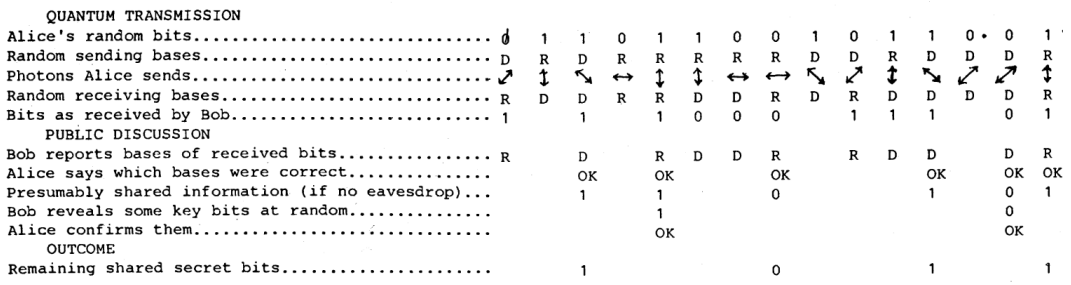

Figure 1.3: BB84 original protocol flow including quantum communication and post processing [BB84].

improvements on the protocol has been suggested – mostly for post processing – different algorithms and techniques has been suggested. In the original BB84 paper, confirmation is the last step of the protocol, and it is not explicitly mentioned how to deal with plausible error in the shared key due to transmission fault. Thus, in the succeeding papers it is very common for parties to perform **error correction** to eliminate possible errors. Another common step during post processing in QKD is **privacy amplification** in which parties use privacy enhancing techniques to boost the secrecy of the shred key – the idea is even if an eavesdropper have knowledge on some bits of the shared key, this step would diminished her knowledge drastically.

The sequence in which these steps shall be performed differs in some papers. Some once privacy amplification is done it is safe to assume the shared key is secret and error correction

is the last confirmation on the correctness of the key, while others argue believe the error correction might reveal some information about the key hence privacy amplification shall happen after that – this is described in more detail in next chapter.

1.3 Cryptographic Authentication

Cryptographic authentication is the mean to ascertain integrity and authenticity of messages. Ordinarily, authentication algorithms have two essential functions: one authenticates the message, and other verifies authenticated messages. These functions take a key alongside with other information as input to ensure authenticity and integrity of the message i.e. to be confident no one has modified the message or has impersonated the real sender – assuming the key is kept secret.

There are two types of cryptographic authentication algorithm in general: *Message Authentication Code (MAC)* and *Digital Signature*. MACs are symmetric algorithm – both authentication and verification functions use the same key. While, Digital Signature algorithms are asymmetric – there is a pair of key available. Secret key, only known to the sender, is used to *sign* messages, and public key is used for verification.

1.3.1 Authentication in QKD

If an adversary is able to inject message of her choosing into classical communication of post processing, then she can easily perform Man-In-The-Middle (MITM) attack. For this, the adversary needs to sit *"in-the-middle"* of the parties communicating link. Being in-the-middle of both classical and quantum communication, the adversary starts the quantum protocol with *sender* and impersonate herself as *receiver*, and start the protocol with *receiver* at the same time pretending she is *sender*. Then they start post processing, adversary performs post processing with both parties impersonating the other party for each end. If authentication on post processing communication is sound, this could be easily detected during post processing step confirmation (section) where parties reveal some portion of the exchanged key which both have used the same basis, also known as sifted key. Clearly, if most of these bits do not match, then the communication over quantum channel has been tampered with. Had the authentication scheme be subject to forgery, the intruder could manipulate messages in that step to her favoring and convince both parties they had shared a secret key with each other, where in reality it is the intruder who they have distributed keys with.

There are two main approaches to perform authentication over post processing messages: delayed and instance. As their name suggests, instant authentication, authenticates

messages in the communication as they are transfered, while delayed authentication happens at the last step when all the messages are sent. Most of the literature assume the shared secret used for authentication is present at the first run between the parties, and later on they consume from the keys generated through QKD. This is the reason this study focus is on MAC and not Digital Signature. Section 2.2 details post quantum secure MACs.

1.4 Scope

This dissertation is set to find *"Post Quantum Secure Authentication Methods Suitable for Quantum Key Distribution"*. In this endeavor, this thesis is obligated to find answer for the following:

- *What are the post quantum secure authentication algorithms?*
- *What does QKD uses the authentication for?*
- *What methods of authentication exist?*
- *What is suitable for QKD?*

In the original paper of BB84 and almost all other subsequent papers it is assumed that the parties have a pre-shared secret which they will use for authentication for the first ever run of the protocol. For the following runs of the protocol, both parties will use a portion of the shared secret generated in each run, for authentication of the next run. This implies the need of symmetrical authentication. Therefore, this study only looks into MAC algorithms. Algorithms that are discussed, are either submissions for standardization competition, or a modified version of them that has enhanced security and/or performance. This could also help to specify whether these algorithms are post quantum secure or not. Each submission is required to submit heavy cryptanalysis of their algorithm for the competition. Furthermore, standardization competitions, which are open to public, have many rounds over course of many years in which the submissions will go through heavy analysis by the community again. After the winner is selected, due to their adaptability being the standard algorithm or even finalist, cryptanalysis on them will continue. Hence, one could be more confident about the security of the algorithm as no attack has been discover during all these analysis.

There are some exception algorithm which has gained the community confidence base on the rate of deployment in industrial projects. These algorithms are mostly incremental innovation on known proved secure structures and advanced the security and performance with provable and explicit approach. These are mentioned later, and I try to back their security based on these facts. Nonetheless, post quantum security definition is rather loose

– after all the security is not proven and we base the security on the assumption that no known attack can breach the security of the cryptographic algorithm in question. The best generic attack known is brute-forcing using Grover's quantum algorithm [Gro96] which finds *"the"* input (e.g. key) in the space of n (e.g. key size) for a given function in complexity order of $2^{\sqrt{n}} = 2^{\frac{n}{2}}$. Therefore, if for any given algorithm with key size n, if it provides $2^{n/2}$ bits security, it is considered post quantum secure. Currently 2^128 bits is consider post quantum secure thus a key size of at least 256 bits. Algorithm in this study are also compared by their efficiency measured by cycles they require to process a byte and is measured in Cycles per Byte (cpb).

It is proven had the authentication scheme used is not ITS, the security of QKD is compromised [PAL+15]. MITM attack on the protocol was shown earlier which exploits forgery attacks on the authentication algorithm. In theory an almighty attacker can forge messages if the authentication algorithm is not ITS. Therefore security of the authentication algorithm plays a major role on the security of the whole system. However, in industrial *"real-life"* scenarios as discussed before, ITS is not necessary. In fact, practicality is much more important given unlimited power does not exist. It is true that unconditional computing power does not exist, though one could argue if the goal is to achieve post-quantum secure key distribution why use QKD in first place where there are already post-quantum secure key distribution protocols.

There are two reason to justify the security assumption. First QKD, apart from being ITS, has a great generation rate potential. Recent implementations demonstrate 1 Gbps speed encryption using QKD[EWL+10]. More importantly, the authentication security does not need to provide forward secrecy i.e. if the authentication could be forged later it would not matter. Therefore, if forging is hard enough for the time of post processing the secret shared is ITS. And that is why I studied post quantum secure algorithms and ITS. Therefore, the lower security bound of this study for authentication algorithm is being post quantum secure, while ITS remains as the higher security bound. Obviously achieving ITS authentication is more desirable.

Now that it is evident QKD *"uses"* authentication for post processing, it is useful to have a general overview of post processing. Understanding the nature of the messages and communication could help us to propose the suitable algorithm. Thus this study reviews post processing as it was suggested by original BB84 protocol and briefly mentions the variations. This is because of the fact that other protocols are derived from BB84 and post processing step and messages are similar. Knowledge of number of steps, type of communication, approximate message length could be decisive factors for selecting an authentication algorithm and an authentication method. An authentication method, employs an authen-

tication algorithm and it provides authentication total solution. A method of employing an algorithm is dependent to the *usecase* of the algorithm e.g. authentication using the same algorithms with two different methods one for network communication and the other with digital assets. Since the post processing is assumed to occur over classical channel, this study considers classical network authentication methods.

The most secure and most efficient algorithm and method is not necessary the best for QKD. It is important to understand what are the requirement of QKD to propose the *most suitable* solution. The best requirement analysis of QKD could be found in practical implementation of the protocol. Those study reveal the naked truth about the needs of the system and can better help us to define *what suitable for QKD is*. Hence, this study looks into well-known practical implementation of QKD as well. The outcome of analyzing these implementation would not only be useful to find the needs of QKD, but will also helps us to get an idea of the post processing steps and algorithms used in them, also the authentication methods and algorithms. I also use this as a ground to build on it, and also as a reference to compare my proposal to.

Eventually, based on my findings on the related literature, I propose a solution which provides authentication infrastructure complying with at least the lower bound security assumption specified for this study. The solution is software based and assumes it receives the *raw key* from the quantum device in software layer and shall perform the post processing. The main focus of this study is post quantum secure authentication method suitable for qkd, however, it is needed to cover other aspects of the system for the purpose of clarifying the whole picture or justifying design approaches. These *"out-of-scope"*s are discussed through out the text briefly. References are provided for interested reader to delve further on them.

1.5 Outline

This study is conducted in four chapters. Chapter 1 provided a background on the problem and introduce the topic of the research in abstract. In section 1.4 of this chapter research questions are detailed and methodology to find answers for them are explained. In chapter 2 related literature are analyzed and section 2.4 a comparative summary of the reviewed literature is provided which is the building block for the proposed solution presented in chapter 3. The solution could be considered as a design document which might be subject to minor changes once feedback from implementation and future research is available. This is discuss in more depth.

Proposed solution is a QKD network endpoint which could be used in any network

topology QKDs are working in and perform different QKD protocols. The architecture is presented in section 3.1 and it is well briefed over the scope of this study, namely authentication algorithm, authentication method, and QKD post processing. The other aspects of system related to the scope are detailed as well in section 3.2. To prove efficient post quantum secure authentication post processing, a simple version of APPP, the other main contribution of this study, is implemented in section 3.3.

And finally, chapter 4 discuss about the design rationale behind the proposed solution, and in section 4.2 it provides comprehensive comparison between the proposed solution and existing ones in different level, and also talks about where the solution stands from view point of recent literature. I conclude the study in section 4.3 where I argue my personal thoughts about QKD and the direction it is heading, and set backs which shall be addressed. I also present the possible outlook for future work on how to enrich and extend the proposed solution.

Chapter 2

Related Literature Analysis

"Secrecy,
once accepted,
becomes an addiction."

— Edward Teller

Thereafter publishment of BB84 protocol, efforts were put on to security analysis, enhancement, improvement, and also practical implementation of quantum cryptography. The level of secrecy provided by the quantum principles was so tempting that other cryptographic functions were also suggested based on these quantum principles. This level of acceptance allowed wide range of techniques and algorithms to be suggested for improvement on security of the BB84 to an extent that we have many Quantum Key Distribution protocols all derived from BB84; which itself has many varieties nowadays. Moreover, in the original paper communication on the quantum channel has been specified to a very clear extent. On the other hand, communication over the public channels are not described in detail – it seems out of scope of QKD and Bennett and Brassard contribution which was more focus on the quantum communication part. This resulted in various approaches and techniques to perform post processing.

Although the main focus of this research is analyzing authentication algorithms for QKD, yet it is necessary to clearly specify post processing and understand the type of messages traversed and the communication itself. These insights will be used to better identify what authentication algorithm and method is the most suitable for QKD. Thus, this chapter reviews the related works being done in the field of BB84 post processing with the intention of grasping an overall knowledge about post processing procedures and the communications required. Once the steps in post processing are discussed, post quantum secure MACs are presented. Subsequently practical implementations are reviewed, authen-

tication and communication requirements are extracted from these projects. Besides the issues the projects were facing which affects or is related to authentication are highlighted. At the end of the chapter, the comparative summary of all three, namely Post Processing, MAC, and practical implementations, is given.

2.1 QKD Post Processing

Post processing is the public discussion over an authenticated channel and happens right after quantum communication in which both parties obtained a bit string also known as ***raw key***. During post processing the basis mismatches in transmission and measurement are discovered and the corresponding bits are deleted from the *raw key* during sifting. The qubit Error Rate (QBER) is then calculated where both parties reveal some portion of the sifted key during confirmation step. Possibility of eavesdropping is measured in the same step – if QBER is above the threshold[1], the key is ignored and communication over quantum channel shall start again. Subsequently, plausible transmission and detection errors in the rest of the ***confirmed key*** are recognized and eventually last step will try to degrade possible knowledge of eavesdropper on the shared secret to minimum. Key life cycle during post processing is shown in figure 2.1 where each arrow represents one step of post processing.

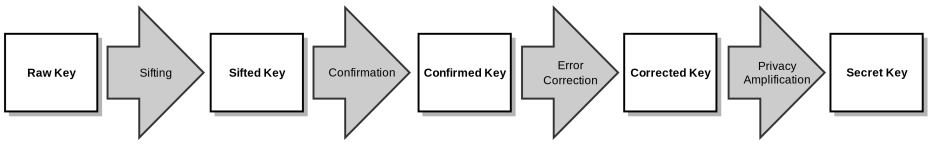

Figure 2.1: Key Life Cycle through Post Processing. Arrows resemble post processing steps.

Four main steps of post processing in the sequence of occurrence as suggested by original BB84 are explained here, known algorithms are described between *sender*, the one who initiates the protocol, and *receiver*. This section will give an overview of the messages, their approximate size, and their quantity during post processing. This Knowledge will be considered for both choosing between authentication algorithms and methods, and also in the design of the proposed solution.

[1]The threshold is the amount of accepted error in the sifted key based on packet loss, detection issues, and etc. It is directly dependent to the distance and hardware used in the project. Accepted QBER is not more than 20%.

2.1.1 Key Sifting

Key sifting is the first step in post processing – parties shall reveal used basis and winnow out the mismatched ones. Although there exist different methods of sifting, but at its essence, to sift, parties shall reveal all the basis used during quantum communication. To sift all the basis, a string as long as the key itself, shall be sent by one of the party where each bit represents the choice of basis. Same size bit string from the other party will confirm the correct basis i.e. matched ones. In original BB84, *receiver* initiates sifting.

The main different between key sifting approaches are due to different policies for termination of quantum communication. Although the original paper submitted time based policy in which at the end of quantum communication *receiver* notifies *sender* she has captured all the qubits, some have suggested iterative sifting and there is a different iterative method called non-iterative for the fact that it does not iterate the basis, and it rather uses a fixed based for communicating the key and use the other basis as decoy [WTC18]. These modifications are proposed to improve efficiency, practicality, and key rate.

In this study, however, I follow the time based termination policy original BB84 suggests – *receiver* is expecting a photon within a certain time slot, once the time for detection is over, *receiver* shall reveal the basis she chose during measurement. *Sender*, will then notify her which ones are correct. As figure 2.1 illustrates, the input in this step is the *raw key* and the output would be *sifted key*. There are two messages communicated in this step which could be as big as the *raw key*.itself.

2.1.2 Confirmation

During this step, some portion of the *sifted key* is revealed – in return parties can calculate the error rate over the quantum channel i.e. QBER. If the error rate is above a certain threshold, then parties abort post processing. As discuss earlier, there is an expected error due to losses during transmission and measurement. On top of that, quality of all most any connection drops as distance increases. The quality of the hardware used also affects the quality of the communication – higher QBER than expected is result of eavesdropping where the intruder tried to measure and disturbed the qubit or is trying to perform MITM.

Analogous to Sifting, there are different suggestion for confirmation step on which bits and what portion of *sifted key* shall be revealed, and who should initiates the step. However, this study follows BB84 original guidelines in which *sender* will ask for random indexes in the *sifted key* and *receiver* will reveal those bits. *Sender*, then, notifies her how many are correct. After this they can calculate the QBER and decided whether to continue or not. The original paper does not suggest how many bits shall be revealed, but follow up works

suggested up to half of the *sifted key*, and same size message is needed to confirm each bit. In this step, at least three messages are communicated. The revealed bits, obviously, would be removed from *sifted key* to form *confirmed key*.

2.1.3 Error Correction

In original BB84 paper, as shown in figure 1.3, confirmation is considered the last step of post processing. In practice however, at this stage, parties have a shared bit sequence which contains error as much as QBER with very high probability. Hence, it seems necessary for parties to re-conciliate these errors.

There are many different error correction algorithms which could be selected based on the implementation requirements and needs. There are two main approaches generally in these algorithms: (i) *Symmetric* where both parties participate in the process with the same load, and (ii) *Asymmetric* in which only one party plays the major role – symmetrical techniques are used in scenarios that computation for one party is expensive e.g. satellites or embedded systems.

Number of messages transmitted during this step depends on QBER, length of the shared secret and the algorithm used. It is out of the scope of this study to investigate error correction algorithms like BCH[2], LDPC, Cascade, and etc, deeply. Nevertheless, section 2.4 contains information of known error correction algorithms and scenarios they are fitted the most for.

2.1.4 Privacy Amplification

Since it is plausible to have error in the *confirmed key* due to mentioned causes, it is mandatory to use error correction. However, many error correction algorithms leak information about the data they are processing, in this case the *confirmed key*. Therefor, it is necessary to reduce the knowledge of adversaries after of this leakage or any other possibilities. This could be achieved by privacy amplification means – the most mentioned technique in the literature is using a family of hash functions. The output of privacy amplification function is set to a desired length for *secret key*.

Privacy amplification step has not been explicitly addressed in the original BB84 as well. Indeed it is out of the interest of this study as well – since if *sender* and *receiver* have already concurred on the privacy amplification technique, then the rest is a local computation for each one of the parties and no message is need to be communicated in this step; thus no need to authenticate anything. Since it is customary to use family of hash

[2]Bose–Chaudhuri–Hocquenghem codes[BR60].

functions for privacy amplification, it is safe to assume *sender* and *receiver* need to agree upon a function from the family for this step – this is the only communication needed for this step which, obviously, shall happen prior to the step itself.

2.2 QKD Authentication

Heretofore, we explained post processing in more detail for the purpose of clarifying each step and the messages communicated. After all, the intention of this study is to identify suitable means to satisfy the *"public channel"* requirements mentioned in BB84 for post processing – the whole QKD protocol is consider secure if and only if *"...public communication channel, assumed to be susceptible to eavesdropping but not to the injection or alteration of messages*[BB84] – the security of the protocol heavily relies on ***integrity*** and ***authenticity*** of the communication over the public channel. In general, there are two approaches for authenticating QKD post processing communication: instant and delayed. As their name suggest, instant authentication is when all the messages transmitted for each step are authenticated as they are sent and received – delayed authentication check the ***integrity*** and ***authenticity*** of the communication after post processing.

As mentioned in the previous chapter, MACs are cryptographic notions preserving ***integrity*** and ***authenticity*** of messages. In this section we present three different type of MAC algorithms which are mostly used in communication networks and are suggested by literatures to be deployed in QKD. MAC generates unique *"tag"* for a given message and a key. Tags could be later used to verify the ***integrity*** and ***authenticity*** of messages assuming the used key was only known to sender and receiver.

2.2.1. Definiton. (Message Authentication Code)

$$\begin{aligned} &\mathrm{G} : \mathcal{K} \times \mathcal{M} \mapsto \mathcal{T} \\ &\mathrm{V} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \mapsto \{0, 1\} \\ &\mathrm{V}_k(\mathrm{G}_k(m), m) = 1 \quad \forall\ k \in \mathcal{K}, m \in \mathcal{M} \end{aligned}$$

MACs are defined over arbitrary size message space $\mathcal{M}$, and finite size key space $\mathcal{K}$ and tag space $\mathcal{T}$. They also contain two keyed functions: tag generator function G and verification function V. Tag generator function G takes $m \in \mathcal{M}$ and $k \in \mathcal{K}$ and generates tag $t \in \mathcal{T}$ – verifying function V takes a tag and the corresponding message and a key, then verifies the tag: outputs either 1 if verification was successful meaning the tag was generated by the same message and key, or 0 otherwise. A MAC is consider secure if an adversary could not forge a verifiable tag for a message without knowledge of the key.

2.2.1 Block Cipher Based MACs

Block ciphers are symmetric encryption algorithm. They process the plain text in blocks and work in different modes of operations for different purposes.Advanced Encryption System winner algorithm, Rijndael also known as AES is a post-quantum secure block cipher which its security is based on assumption[3] and was introduce in 2001. It has been under heavy analysis since, and it is the most used symmetric encryption algorithm. By their essence, block ciphers are very efficient in hardware implementation[4]; however, concepts like substitution used in the algorithm are extremely slow in software. Due its high rate of adoption however, exclusive AES instructions[5] are implemented in high end CPUs and other methods of hardware acceleration are employed exclusively for AES to increase its performance in implementations.

The other finalists of Advanced Encryption System competition are the common alternatives to Rijndael, namely Twofish and Serpent. In the book *"Cryptography Engineering: Design Principles and Practical Applications"* by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno – the official designers and cryptanalyst of Twofish – they compare the three.

> *"Serpent [...] is built like a tank. Easily the most conservative of all the AES submissions, Serpent is in many ways the opposite of AES. Whereas AES puts emphasis on elegance and efficiency, Serpent is designed for security all the way. Twofish [...] can be seen as a compromise between AES and Serpent. It is nearly as fast as AES, but it has a larger security margin".*

However both have slow performance over software. Another related algorithm to Twofish called Threefish has been proposed which does not follow the substitution principle employed in block ciphers to avoid cache timing attacks, and achieves its nonlinearity dependency through Exclusive ors[FLS+10]. Substitutions are not very CPU friendly – thus removing them from the scheme make it very efficient in software implementation.

BEAR and LION are two algorithms which construct block ciphers by employing hash functions and stream ciphers – both very efficient in software implementation and can process big chunks of data. They are designed based on Luby and Rackoff proposal for constructing block ciphers from three PRFs. Both methods are proved to be as secure as used algorithms. BEAR adopts hash function H and stream cipher S. The algorithm splits input M into $[M_L|M_R]$; size of M_L is equal to the output size of M. The algorithm uses

[3] AES is not ITS, but there has not been an efficient attack found yet. Successful attacks exploit poor implementation rather than the algorithm structure itself.

[4] they do block by block

[5] `https://en.wikipedia.org/wiki/AES_instruction_set`

two keys K_1 and K_2 which both are bigger than the digest size of the hash algorithm used in size. Encryption happens as follows.

$$M'_L = M_L \oplus H_{K_1}(M_R)$$
$$C_R = M_R \oplus S(M'_L)$$
$$C_L = M'_L \oplus H_{K_2}(C_R)$$

Cipher text would be $[C_L|C_R]$ and it decrypts as follows.

$$M'_L = C_L \oplus H_{K_2}(C_R)$$
$$M_R = C_R \oplus S(M'_L)$$
$$M_L = M'_L \oplus H_{K_1}(M_R)$$

LION has a very similar construction as BEAR; instead it uses stream cipher twice and hash function once, which could be used with hash functions with weaker security assumption than the one used in BEAR i.e. the hash function does not need to be a PRF, it only needs to be collision-free [AB96]. It has been proved by the author that an attack on either of the algorithm would also break the hash function and the stream cipher used i.e. the algorithms are as secure as the stream cipher and hash functions used – it is the same case for proposed MAC algorithms i.e. the security of block cipher based MAC algorithms depend on the security of underlying block cipher used in any of these scheme presented here. We will see later AES with help of exclusive modification outperforms all the others on high end CPUs.

While from applicability perspective, it may not differ which variant of block cipher is chosen; it could be a great deal from performance point of view or security concerns[6]. Being the standard encryption scheme, Rijndael benefits from more attention in analysis and implementation.

Apart from the block cipher algorithm, its mode of operation could affect the overall performance – some modes are needed to be calculated sequentially while it is possible to perform other modes of operation in parallel to achieve better performance. Here we present modes which could be exploited to create authentication tags. At last we talk about another MAC algorithm which uses block cipher in a different fashion comparing to others to generate tag.

[6] All mentioned above are post-quantum secure and all their security is based on assumption, it is not proven mathematically which of them is more secure yet statements like Serpant is more conservative implies the better security of the algorithm, nevertheless it could not be measured.

Cipher Block Chaining MAC

Block cipher encryption algorithms operating in CBC mode could be used as MAC algorithm. The reason for this is the fact that in CBC mode there is feedback from the previous block which can assure the ***integrity*** of the message – if one block is changed or even be substituted with others, the tag would be different. Figure 2.2 shows both schematic of Cipher-based Message Authentication Code (CMAC) for scenario in which the message length is multiple of block length (in the right) and otherwise (in the left). As illustrated in the figure, the message is divided into block size and encrypted with the key K – except the first message block, the following message blocks would `XOR`ed with the previous cipher text, hence comes the name chaining.

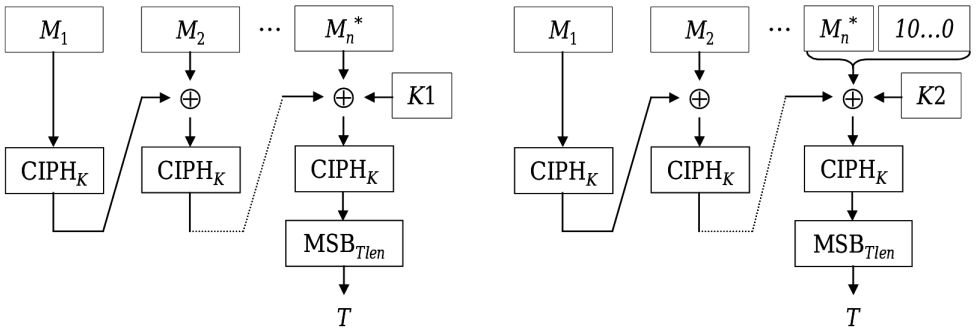

Figure 2.2: CMAC Schematic [Dwo05].

CMAC is an OMAC[7] that address security deficiencies found in CBC-MAC. If two pairs of tags and messages (m, t) and (m', t') are generated using the same key, the tag for the third message $m'' = m||[(m'_1 \oplus t)||m'_2||\cdots||m'_l]$ is also t' – while generating the tag for the m part of m'' generates tag t as expected, when the other part starts, t as the output of previous block would be `XOR`ed with the first block of second part and would be canceled out by the t in the first block of second part: $\texttt{CIPH}_k(m'_1 \oplus t \oplus t) = \texttt{CIPH}_k(m'_1)$. And this is exactly like computing the tag for m' which is t'. The issue could be solved by encrypting the last block with another key as ECBC-MAC does, however, the issue remains if the message size is not known in advance or size of the message is not multiple of block size. XCBC proposed a solution which requires three different keys. In contrast to XCBC and ECBC-MAC, CMAC algorithm derives K_1 and K_2 from the single secret K – hence comes the name One-key MAC – and apply either of them to the message depending on its length.

[7]One-key MAC are modified version of ECBC-MAC and XCBC which address CBC-MAC security issues but require two and three different keys respectively. CMAC could be seen as one-key XCBC

Parallelized Message Authentication Code

CBC style MACs are not very efficient due to the fact that their computation is linear and cannot be parallelized i.e. computation over a block could not be initiated until the computation over previous block is finished. Depicted in figure 2.3, PMAC is a block cipher based authentication tag generator which could be computed in parallel, thus it has superior performance compared to CBC variants.

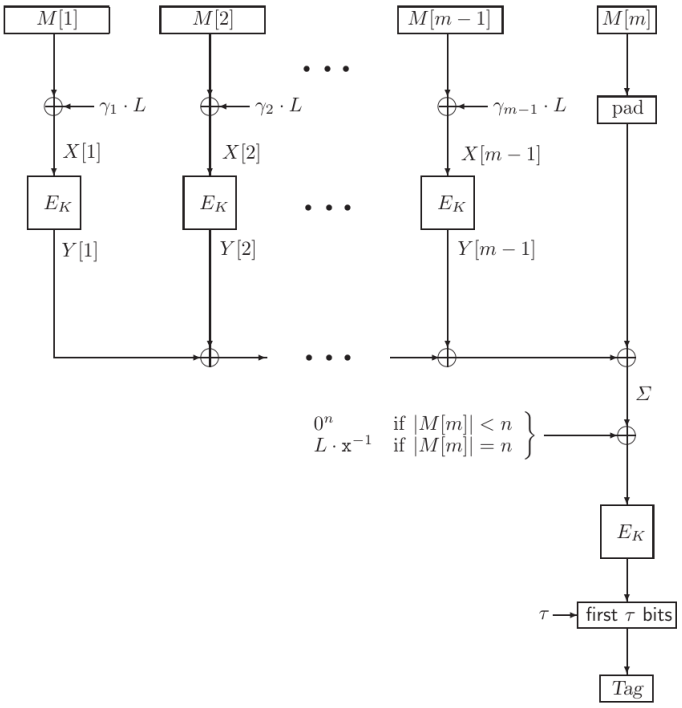

Figure 2.3: PMAC schematic [BR02]

.

As demonstrated in figure 2.3 PMAC resembles ECB mode of block cipher operation – messages are splitted into blocks of n bit size, then they are `XOR`ed with multiplication of constant γ_i derived from *Gray Codes*[8] and $L = \texttt{CIPH}_k(0^n)$, except the last block. The multiplication is defined in detail in [BR02] – it is a polynomial multiplication over $\mathbf{GF}(2^n)$. The results of `XOR` are then encrypted to create $Y[i]$s. As it can be observed the process on each block is happening independently of the others and could be performed in parallel. Subsequently, $Y[i]$s are `XOR`ed and eventually `XOR`ed with the last message block to construct

[8]Gray Codes are ordering $\gamma^l = \gamma_0^l\gamma_1^l \cdots \gamma_{2^l-1}^l$ of $\{0,1\}^l$ such that successive points differ (in the Hamming sense) by just one bit. For n a fixed number, PMAC makes use of the "canonical" Gray code $\gamma = \gamma^n$ constructed by $\gamma^1 = 01$ while for $l > 0$, $\gamma^{l+1} = 0\gamma_0^l\ 0\gamma_1^l\ \cdots\ 0\gamma_{2^l-1}^l\ 1\gamma_{2^l-1}^l \cdots\ 1\gamma_1^l\ 1\gamma_0^l$. It easy to compute successive points since for $1 \leq i \leq 2^n - 1$, $\gamma_i = \gamma_{i-1} \oplus (0^{n-1}1 << \texttt{ntz}(i))$. `ntz` is the number of trailing zeros; $\texttt{ntz}(7) = 0$ while $\texttt{ntz}(8) = 3$ [BR02].

Σ – ordering in `XOR` obviously does not matter. In case there were no padding added to the last message block, Σ will be `XOR`ed with $L.x^{-1}$ which means to reduce a degree from polynomial representation of L – there is an efficient algorithm for that in [BR02]. The final step is to encrypt the result and take the first τ bits as the authentication tag. It is obvious changes in any message block or their order will result in generating different tag.

Galois/Counter Mode

GCM is an authenticated encryption scheme based on block ciphers in CTR mode in which the encryption happens over a counter and the result is `XOR`ed with the message – figure 2.4 shows schematic of counter mode used in GCM which is called GCTR, ICB is the Initial Counter Block (ICB) or value which increments for each block. GCM calculation could be parallelized due to the nature of CTR mode.

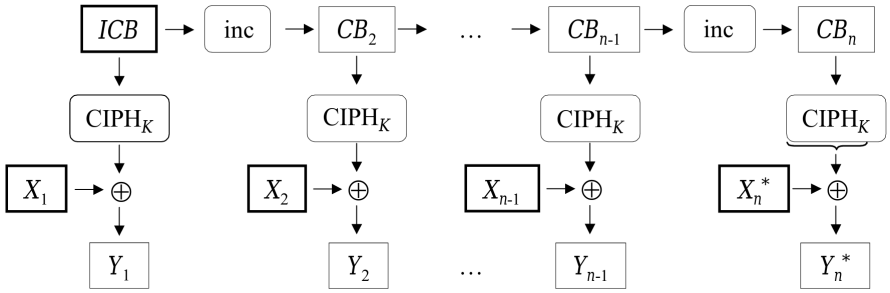

Figure 2.4: $\text{GCTR}_k(ICB, X_1||X_2||\cdots||X_n^*) = Y_1||Y_2||\cdots||Y_n^*$. Bold border lines on boxed denote they are input to the algorithm.

Figure 2.5 shows schematic of GCM authenticated encryption function. The algorithm takes confidential plain text P, and encrypts it using GCTR to construct cipher text C, and together with other inputs they form the input data the tag would be generated for – shown as input to GHASH_H in figure 2.5. Apart from the plain text and a key K, Initial Vector (IV) and additional non confidential data which user wants to authenticate denoted as A are inputs to the algorithm. *"An implementation may restrict the input to the non-confidential data, i.e. without any confidential data. The resulting variant of GCM is called GMAC. For GMAC, the authenticated encryption and decryption functions become the functions for generating and verifying an authentication tag on the non-confidential data"* [Dwo07]. Hence the input for GMAC could be only A and IV. The initial vector is used to define the block J_0[9] which is later fed to GCTRs as ICB.

GHASH, depicted in figure 2.6, is a universal keyed hashing function. It processes the

[9]If the size of IV is 96 bits, then $J_0 = IV||0^{31}||1$, otherwise IV is hashed using GHASH_H to create J_0 – the whole algorithm is detailed in [Dwo07].

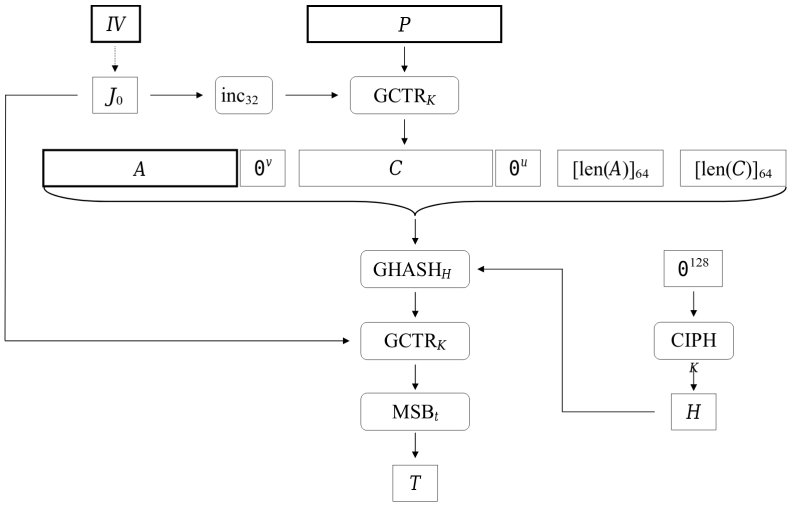

Figure 2.5: GCM schematic. Bold border lines on boxed denote they are input to the algorithm [Dwo07].

message in form of blocks. Each block is first `XOR`ed with the result from previous block and then it is multiplied by the secret H. The first block is `XOR`ed with block zeros before multiplication, and the multiplication itself is a block multiplication – an efficient algorithm to compute multiplication of two blocks is presented in [Dwo07]. GCM encrypts a block of 128bits zeros with the key K to arrange H, the secret key for GHASH which is used to hash the input data. The hashing output is then encrypted using GCTR and the t most significant bytes are chosen as the authentication tag – this is very similar to UMAC and VMAC, another MAC generator schemes explained in Universal Hash Based coming section. The reason GMAC was listed with other block cipher based scheme is GCM mode was suggested as block cipher operation mode and GMAC is a part GCM authenticated encryption algorithm, which is different from VMAC and and other solely authentication algorithms. More importantly, GMAC encrypts the results of universal hash while it is suggested by Carter-Wegman to `XOR` the result of hash with a psuedo random string. I think this would not make difference on the security of both algorithm as long as AES could be view as PRF, which this is acceptable in post-quantum era.

Verification of the tag follows the same procedure, the only difference is in the final step the computed tag would be compared to the original tag – if they are equal the tag is valid, otherwise it is not. In case GCM was used for authenticated encryption purposes, once the tag is verified, the cipher text C would be decrypted using GCTR decryption which is an identical procedure to GCTR encryption; the only difference is the input which is the cipher text C instead of plain text P.

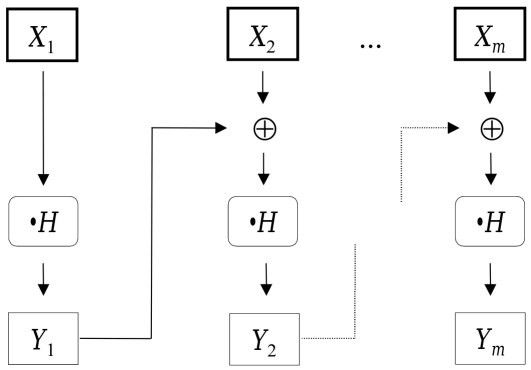

Figure 2.6: Ghash schematic [Dwo07].

Poly1305

Poly1305 is a MAC algorithm which hashes the message using an efficient **poly**nomial hashing in mod$2^{\mathbf{130}} - \mathbf{5}$ so called Poly1305, and then adds the hash output to encryption of a nonce. Poly1305 is different in nature comparing to other block cipher based MAC presented here as it does not rely on any modes of operation. Indeed, it might even be more similar to universal hashing MACs known as Carter-Wegman style. In the first publishment of the algorithm, it was suggested to use AES in order to encrypt the nonce (Poly1305-AES), however, later it was suggested to use more efficient encryption algorithms like chacha20. Chacha20 is a post-quantum secure stream cipher and the successor to Salsa20. It uses new round function which enhances the diffusion property of the scheme and also boosts the performance comparing to AES.

ChaCha follows the same principles as Salsa20 the eSTREAM certified portfolio1: software implementation, but has more diffusion per round, and this enables achieving the same security in less round, and obviously less round is more efficient. *"Salsa20/20 is more conservative design than AES, and the community seems to have rapidly gained confidence in the security of the cipher"*[Ber08].

The security of Poly1305 and justification for the design are expressed in [Ber05b]. Chacha20-Poly1305 has received a very good reception since its introduction and getting adopted into many applications substituting AES types of block cipher authentication specially in mobile device communication. Performance wise it is superior to CBC, PMAC and GMAC variations and implementation could be highly customized based on different architecture with low costs of implementation.

The algorithm implemented an efficient polynomial hashing function and hashes the message in blocks of 16 bytes (128 bits) and then `XOR`s the hash with the encryption of the nonce with the secret key k – classic Carte-Wegman where the result of the hash is

XOR with a random bit string here the cipher text. The random number r have certain restrictions, and values for some of its bits are fixed. In the below formula c_is are special polynomial representation of the message, and $q = \lceil \frac{\ell}{16} \rceil$ where ℓ is the message length.

$$(\underbrace{((c_1 r^q + c_1 r^{q-1} + \cdots + c_q r^1) \bmod 2^{130} - 5)}_{H(m)} + \texttt{CIPH}_k(n)) \bmod 2^{128}$$

Standard Poly1305 algorithm uses a 128 bits nonce n, 128 bits key k, 128 bits random number r, and creates authentication tag for arbitrary message length based on the above formula. However 128 bits is not post quantum secure, and recently versions with higher key size (256bits) are introduced.

2.2.2 Cryptographic Hash Based MAC

While using block cipher specially CBC-DES[10] was the most common approach to generate authentication tag, utilizing cryptographic hash functions to generate authentication tag emerged in mid 90s, mostly from sheer interest of Internet community *"where the development of security protocols has led to the need for simple, efficient, and widely available MAC mechanisms [...] the popular hash functions are faster than block ciphers in software implementation"*[**?**].

Cryptographic hash functions are not keyed primitives i.e. they do not take a secret as input, and this makes it harder to create secure cryptographic authentication tag generator from them. On top of this, cryptographic hash functions have their own notion of security – these security properties for a cryptographic hash function H and its digested output h, are listed below. Other attacks, except those listed below exist for hash functions, like extension attack, inner collision, or state collisions; yet these three remain the standard security properties of hash functions.

- *Pre-image resistance*: for $H(m) = h$ given h it must be impossible/hard to find m.
- *Second pre-image resistance*: for $H(m) = h$ it must be impossible/hard to find m' such $H(m') = h$.
- *Collision resistance*: it must be impossible/hard to find two messages $m \neq m'$ such that $H(m) = H(m')$.

Cryptographic hash functions are mapping arbitrary length domains into fixed length co-domains; hence it could not be claimed that there is no two pairs of input messages which

[10]Data Encryption Standard is a deprecated block cipher developed by IBM in 70s.

yield the same digest. Given the bigger size of the function domain comparing to its co-domain it is possible in theory for two inputs two have the same output. This fact directly affect the *Second pre-image resistance* and *Collision resistance* property, and that is why I have used "impossible/hard" to define these properties – the best solution is to use larger co-domains (e.g. 2^{256}) and design the cryptographic hash function as similar as possible to the notion of *random oracle*, i.e. distribute inputs to outputs as evenly as possible to make the effort expensive and then under some computational power assumptions, cryptographic hash function is considered secure.

For finding a pair which their digest collides, the attacker could choose any arbitrary pair of desired messages and check whether they output the same digest. To find second pre-image of a digest, however, the value of the digest is fixed. Finding collision is easier than finding a second pre-image, indeed a collision could be found in the order of $2^{n/2}$ while this number is 2^n to find a (second) pre-image where n is the size of hash function co-domains in bits. However, considering Grover's quantum algorithm, second preimages could be found in complexity of the same $2^{\sqrt{n}} = 2^{n/2}$ in post-quantum era[Ber10]. Assuming a function H is *pre-image resistant* for every element of the range of H is a weaker assumption than assuming it is either *collision resistant* or *second pre-image resistant*. Moreover, assuming a function is *second pre-image resistant* is a weaker assumption than assuming it is *collision resistant*[Sma16]. Hash functions are considered broken if collision could be found in order less than $2^{n/2}$ [BDPV07].

Merkle-Damgård (MD) is a well-know compression approach for creating collision resistance hash functions – algorithms such as MD5, SHA-1, and SHA-2 are benefiting from this structure. MD structure is a tree-based efficient method for calculation over software. As of 2018, however, finding collision on all variation of MD-5, SHA-1, and SHA-2 are practical [DEM15][EMS14], and except for two variation of SHA-2, namely SHA-512/224 and SHA-512/256, all of them are vulnerable to extension attack[11].

HAsh Iterative FrAmework (HAIFA) is another construction for creating secure cryptographic hash functions. It *"maintains the good properties of the MD construction[12] while adding to the security of the transformation, as well as to the scalability of the transformation"*; yet is simpler, more efficient, and faster. BLAKE, a post-quantum secure cryptographic hash function, was the third secure hashing algorithm finalist, which employs HAIFA as its domain extender[jCPB+12], and rely on same core permutation used

[11]Length extension attack on MD construction happens since the output is the internal state. If an attacker knows message m_1 and its length l_2, and the tag t which has been generated by key k, then she can forge tags for extension of m_1 – she simply has the inner state of the keyed hash function and forge the tag for $m = [m_1||m_2]$ under key k.

[12]A prefix-free MD construction whose padding rule is prefix-free.

in ChaCha stream cipher as compression function [ANWW13]. SHA-3 final reports credits BLAKE with having a *"very large security margin"*, and *"a great deal of depth"* in cryptanalysis performed on the algorithm.

In 2013 a enhanced version of BLAKE called BLAKE2 was introduced with astonishing performance specially on 64-bit CPUs. BLAKE2 operates on 256 bits (BLAKE2s) and 512 bits (BLAKE2b). BLAKE2b on 64bit-CPUs is 1.5 times faster than BLAKE2s [ANWW13] – on an Intel Core i5-6600 (Skylake micro-architecture, 3310MHz), BLAKE2b can process 1 Gibibyte per second, or a speed rate of 3.08 cpb. The performance could further be enhanced by using BLAKE2 in parallel mode. BLAKE2bp runs 4 instances of BLAKE2b in parallel, and BLAKE2sp runs 8 instances of BLAKE2s.

Sponge construction, yet another collision resistance method for creating secure hash functions introduced in 2008, is designed to *behave as random oracle*[13] – *"it takes a variable-length input and produces an infinite-length output"*[BDPV07]. Illustrated in figure 2.7, it uses a fixed length permutation f over $b = r + c$ bits also known as the *state* – r is the *bit rate* and c is called the *capacity*. The initial state is set to zero.

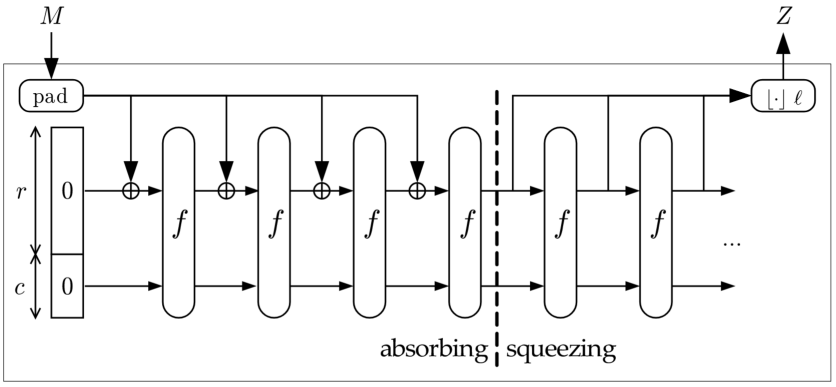

Figure 2.7: Sponge Construction [BDPV07].

Input to the construction is padded and splitted into r bits blocks, and during the absorbing phase all the blocks are `XOR`ed with the r bits of the state before being fed to the permutation f. Once all the blocks are processed, the sponge enters squeezing phase where r bits of state would be concatenated after each call to f to create the required length output. Sponge construction and its security proofs are detailed in [BDPV07][BDPV11]. An instance of sponge construction is called a sponge function. The winner of the third Secure Hashing Algorithm contest, also known as SHA-3, is a Sponge function called Kec-

[13] All the security complexity to find collisions and preimage are the same as random oracle, the scheme is immune to correlation attacks; length extension is not applicable to random oracle concept but sponge is secure against that as well[BDPV07].

CAK: *"the endpoint of a long learning process [...] at fixing PANAMA [26], resulting in RADIOGATÚN[4] [...] which become* KECCAK*"*[BDPV14]. In contrary to other hash algorithm, SHA-3 has satisfying performance in hardware implementation, but in software implementation it is slower than others[14][jCPB+12].

Analogous to encryption standard, SHA-3 benefits from the attention it receives being the latest standard; SHA-3 has been, and will be, under heavy analysis – no successful attack has been published yet. Moreover, there are tons of publication on secure and efficient software and hardware implementation beside suggestion on how to choose parameters to achieve required security and performance[15].

In 2016, two parallel hashing algorithm based on KECCAK-p were submitted for NIST fast hashing competition, namely ParallelHash and KANGAROOTEWELVE. The former achieves unlivable performance on Skylake X CPU platform and is more than 2 times faster than BLAKE2 parallel algorithms – the benchmark is presented in figure 2.8. KECCAK-p based algorithm's *"security assurance directly benefits from nearly ten years of public scrutiny, including all cryptanalysis during and after the SHA-3 competition"*[BDP+16]. Instead of 24 rounds of KECCAK permutation used in SHA-3, KANGAROOTEWELVE only does 12 rounds KECCAK-p permutation which is a safe margin considering the best attacks on KECCAK are happening over 6 rounds, and it utilizes SAKURA encoding for tree hashing to gain better performance on long messages, scenario more similar to reality usage of MAC in QKD. SAKURA, constructs tree of *hop*s instead of traditional nodes. A traditional node may contain message bits and chaining values of the tree simultaneously, while there are two different types of hops for that: *message hops* and *chaining hops*. SAKURA*-compatible tree hash modes are not required to generate all possible hop trees, but instead they can focus on the desired subset of them [...]. Such a hop tree determines the parallelism that can be exploited by processing multiple message hops or chaining hops in parallel"* [BDPV13]. There special hoping as well called *Kangaroo hop* which allows further parallelism. More detail about the algorithm could be found in [BDP+16][BDPV13]. SHAKE128, shown in figure 2.8 is Extended Output Function[16] version of SHA-3 256 which performs better than SHA-3 and provides arbitrary output size – it has been standardized on FIPS 202[17] and is presented in the figure as comparison reference.

In order to create authentication tags from these un-keyed hashing algorithms, a

[14]On Qualcomm's Krait micro architec-ture1 SHA-3-256 takes about 20% longer to hash a message than SHA-256 does, and on Intel's Ivy Bridge (3rd generation) micro architecture 2, SHA-3-512 takes about twice as long as SHA-512 does [ANWW13].

[15]Extensive list available at `https://keccak.team/papers.html`

[16]XOF is a function on bit strings in which the output can be extended to any desired length [KjCP16].

[17]`https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.202.pdf`

Function	SkylakeX	Skylake	Haswell
KangarooTwelve	0.55	1.22	1.44
KangarooTwelve ($\leq$ 8KiB)	2.35	2.89	3.68
ParallelHash128	0.96	2.31	2.73
Blake2bp	1.39	1.34	1.37
Blake2sp	1.22	1.29	1.39
SHAKE128	4.28	5.56	7.09
MD5	4.33	4.54	4.93
SHA-1	3.05	3.07	4.15
SHA-256	6.65	6.91	9.27
SHA-512	4.44	4.64	6.54
Blake2b	2.98	3.04	3.08
Blake2s	4.26	4.85	5.34
Blake-256	5.95	6.76	7.52
Blake-512	4.48	5.19	5.68
Grøstl-256	7.24	8.13	9.35
Grøstl-512	9.95	11.31	13.51
JH	13.04	15.14	15.09
Skein	4.48	5.18	5.34

Figure 2.8: Cryptographic hash cpb benchmarks.[BDP+16]

straightforward approach could be taking as input the concatenation of a secret key and the message, taking the output as the tag i.e. $H(m, k) = t$. This is know as keyed hashing – but this method is vulnerable to length extension attacks for some construction. Luckily, if a sponge function is used in this form and the inner state is kept secret from the adversary, the MAC is at least as secure as the sponge function used[sponge]. This is due to the fact that sponge construction are susceptible to extension attacks i.e. they do not out put the state of the hash. Indeed, Keccak family could be used in this efficient MAC generator algorithm called KMAC, constructed exclusively for this family of hash functions. It is guaranteed *"that the keyed sponge constructions can replace random oracles in any single-stage cryptographic system as long as the total complexity of the adversary is less than* $2^{(c+1)/2}$ *"* – for c being the capacity of the sate[BDPV11][BDPV08]. Therefore, the value for c is usually greater than 255 to have at least 2^{128} bits security. In order to resist extension attacks on other hash functions, HMAC principle could be followed.

HMAC

HMAC is one-keyed NMAC[18], both publish in 1996 together. It proposes an approach to make use of normal fixed IV (i.e. default initial state usually set to zero initially) hash

[18]Nested-MAC which uses hash functions to create MAC resistance to extension attacks – it requires two keys: $\texttt{NMAC}_{\{k_1,k_2\}}(m) = H(k_1 \mid\mid H(k_2||m)\ )$

functions to create secure MACs through the below nested formula. $\bar{k}$ is special key derived from k *"by adding 0's of k to a full b-bit block-size of the iterated hash function"*[BCK96]. `opad` and `ipad` are block-size length constants of `0x36` and `0x5c` values to perform outer a inner padding to the key, respectively.

$$\texttt{HMAC}_k(m) = H(\bar{k} \oplus \texttt{opad} \mid\mid H(\bar{k} \oplus \texttt{ipad} || m) \ )$$

HMAC does not necessarily share the same collision complexity as the hash function used in the scheme – it could be more resilient [BCK96][Bel15]. The concept is widely in used and analyzed extensively, more details and security proofs could be found in [Bel15]. BLAKE2 algorithm are designed to receive key as input – they pad the key with enough zeros to construct a block and append it to the beginning of the message. BLAKE2 variants only make one call to the compression function for hashing the first block with the zeros. This is because they use HAIFA a free prefix MD construction, hence becoming more efficient (at least for one less call). If HMAC used with BLAKE2, it will benefit from this enhancement and is called Prefix-MAC which slightly faster due to decreasing invokes of compression function [ANWW13].

KMAC

KMAC is designed to exploit sponge construction resistance against extension attack. The MAC is simply hash of the key and the message with a small tweak. During absorbing phase, first only the a padded version of the key would be `XOR`ed with initial state of zero, and then the message would follow. This model is called *outer-keyed sponge*. This allows the state to be unknown to the attacker before applying permutation f on the message block. The tag would simply be the output of the hash function. One could also place the key as the state itself, this method is called *inner-keyed sponge*[BDPV08].

Originally KMAC was proposed as outer-keyed sponge in two variant based on cSHAKE128[19] and cSHAKE256 [KjCP16]. KMAC could have arbitrary output length, thus it is recommended as PRF or Key Derivation Function[20] by NIST; the security precautions are discussed in [KjCP16].

2.2.3 Universal Hash Based MAC

Universal hash based MAC rely on a collection of hash functions instead of one and choose randomly amongst them in each run. For the collection to be considered $universal_n$ strong,

[19] cSHAKEs are defined in terms of the SHAKE and KECCAK[c] functions specified in FIPS 202. They either call SHAKE with XOF permutation or MACc[KjCP16].

[20] KDF generates session key from a secret seed.

it need to have $|H|/|B|^n$ functions to map elements a_i to b_i for $0 < i \leq n$ from domain A to co-domain B – $|H|$ is the number of hash functions in the family and $|B|$ is the size of co-domain or digested value. Domain on universal hash functions is mostly fixed and not arbitrary. Carter and Wegman introduced many classes of strong universal$_2$ and almost strong universal$_2$ family of hash functions and built MAC generator using universal$_2$. The scheme takes two key k_1, k_2, one to choose the hash function f_{k_1}, an the other one to pick an digest b_{k_2} from B and then tag $t = f_{k_1}(m) \oplus b_{k_2}$. Usually the message is splitted into n part and this procedure is repeated and the result are concatenated until all processed and then the tag is chosen from the final digest. It is important to not process the same message piece with the same keys twice [CW81].

It is proven using some provisions detailed in [CW81] the proposed scheme is ITS and unbreakable. UMAC and VMAC are two other implementation of Carter-Wegman style MAC which are implemented for 32-bit and 64-bit architectures, respectively. VMAC's VHASH algorithm uses integer multiplication rather than multiplication of polynomials, hence it is very efficient comparing to polynomial multiplications or block multiplications like GHASH. `crypto++` implementations of UMAC and VMAC perform at 0.22 and 0.40 cpb and GMAC operates 0.34 cpb with the help of exclusive instruction sets on Skylake (6th generation) Intel CPUs, otherwise the result for only hashing by GHASH and not encrypting using GCTR would be more than 10 times slower [BC14]. The performance comparison of the performances are detailed in section 2.4. VMAC generates tags adopting Carter-Wegman suggestion through below formula:

$$\texttt{VMAC}_{k_1,k_2}(m) = H_{k_1}(m) \oplus F_{k_2}(\texttt{nonce})$$

auth256, is another Carter-Wegman MAC which achieves astonishing performance faster additive Fast Fourier Transform (FFT) based algorithm to multiply polynomials – achieving Faster Binary-Field Multiplication, thus faster MAC generation [BC14]. auth256 is the only algorithm selected in this paper which has not been part of any competition or standard[21]. The justification for this selection is the fairly clear and straightforward claims and modifications done in auth256, namely modification of FFT which *"is well known in the FFT literature but we have never seen it applied to message authentication. It reduces the cost of FFT-based message authentication by a factor of nearly 1.5"* [BC14].

[21]The work was supported by the National Science Foundation and by the Netherlands Organization for Scientific Research.

2.3 QKD Network Implementations

Traditional QKD implementations contain two QKD endpoints connected to each other to form a QKD link which contains a quantum channel and classical channel – this type of connection is known to have the following flaws and restrictions:

- *"Traditional QKD is distance limited"*[Ell05].
- *"Key distribution [rate] exponentially decreases as a function of distance"*[PPA+09].
- Traditional QKD *"can only be used across a single physical channel (e.g. freespace or telecommunications fiber, but not both in series due to frequency propagation and modulation issues)*[Ell05].
- QKD link connections *"inherent point-to-point character of communication, which could be a significant obstacle in the majority of relevant application scenarios"*[PPA+09], *"and is vulnerable to disruptions such as fiber cuts because it relies on single points of failure"*[Ell05].

With advancement of technology, improvements are being done on restrictions caused by distance yet *"they are still dominating as of today"*[PPA+09]. In order to eliminate these hurdles, in late 90s, it was suggested *"to extend point-to-point connections to networks"[PPA+09]* – a natural progression analogous to evolution of classic networks – and the idea was studied theoretically and experimentally back then [PPA+09].

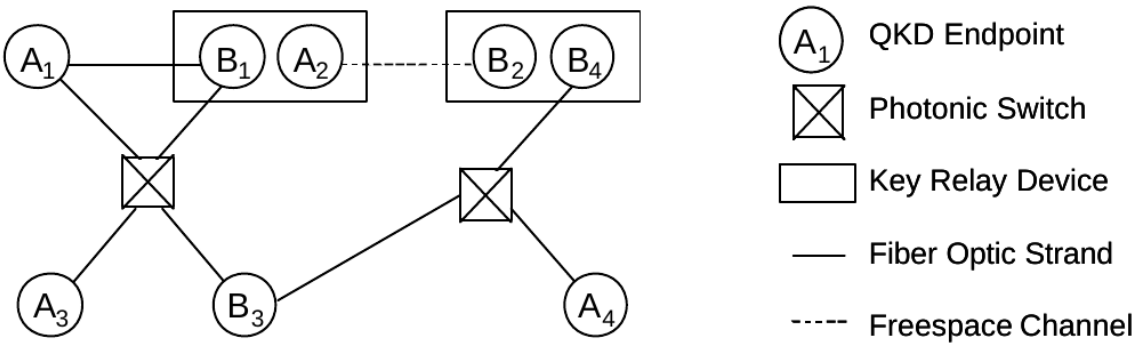

Figure 2.9: Abstract Schematic of QKD network [Ell05]

Figure 2.9 depicts a pictorial schematic of QKD networks. QKD networks consist of interconnected QKD links with various types of connections and protocols used amongst endpoints across the network – it lifts traditional point-to-point QKD links restrictions. For instance, QKD endpoint A_3 could share a secret with a distant QKD endpoint such

as B_4 through different QKD paths[22] – something which is not possible using traditional QKD.

Obviously, the knowledge gathered during designing and implementation of classical networks for almost half a century, was the backbone of designing QKD networks. Detailed description of QKD network, its components, and its security precautions regarding trusted and untrusted nodes are describe in the relevant literature [darapa] and are out of the scope of this section – in this section, I skim through some of the most well-known[23] implementations with the intention of understanding the general architecture and design principle used in them – this would help to better shape the requirements as well, as discussed before in section 1.4. Another beneficial outcome of the studying existing implementation which is the most relevant to this study is to inspect their approaches for post processing and authentication over the public classical channel.

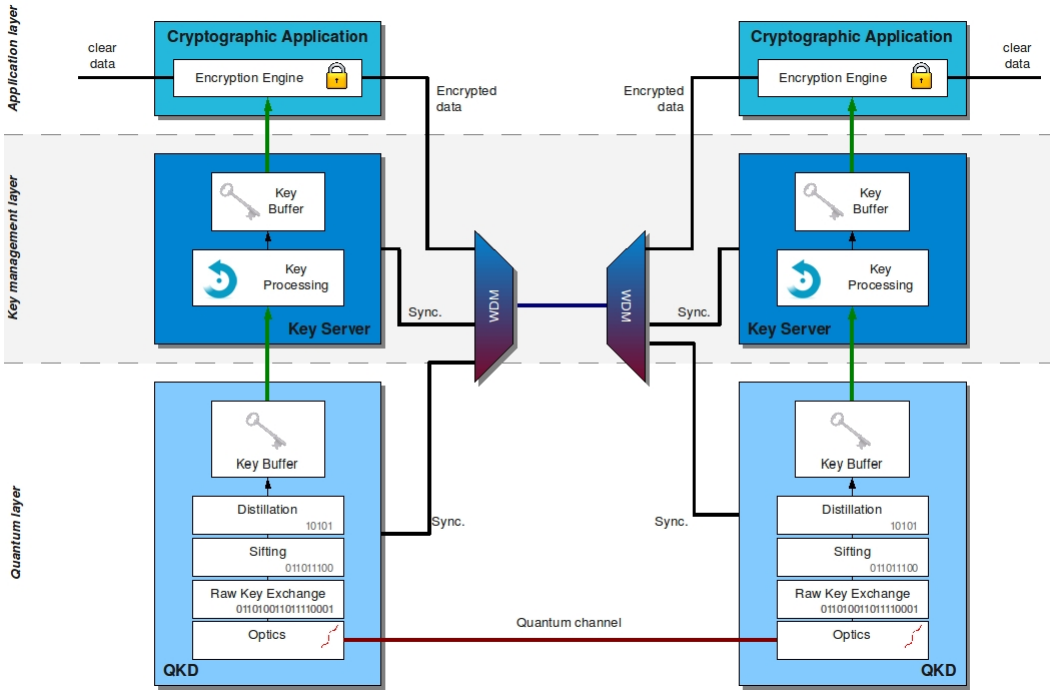

Figure 2.10: General architecture of different layers in QKD network [SLB+11]

All these project are utilizing a three-tier architecture similar to the one shown in 2.10 from SwissQuantum project. In that specific project, it was suggested to employ multiplexing over classical channel to reduce costs – this the reason the classic connections

[22] QKD path: a chain of QKD links and corresponding nodes[darpa]

[23] These implementation are those with rigorous amount of publication, mostly involvement of many different organizations and were functional for a long time (more than a year) and there is field study publication for them.

go through Wave De-multiplexing Module (WDM) before departing towards destination. This, however, is not common among all implementations. Nevertheless, the three below layers exist similarly in all the projects.

1. **Quantum Layer (QL)** is the layer responsible for generating secrets using QKD protocol and it contains all the necessary steps including post processing for the parties to obtain *secret key*. Once secrets are generated they will be pushed into the next upper layer to be stored.

2. **Key Management Layer (KML)** duty is to store and synchronize the keys generated by QKD and provide them to the application layer upon request or need. In some implementations there are other inputs apart from QKD into this layer such as keys generated by classical cryptographic algorithms like Deffie-Hellman [Ell05][SLB+11]. Moreover, this layer in many implementation is also responsible for transmitting keys into distant nodes over classical communication channel using Key Encapsulation Protocols[24]. KML also provides keys for authenticating messages during post processing.

3. **Application Layer (AL)** is the main consumer of the shared secrets which provides secure communication and other type of cryptographic applications for end users using secrets generate by QKD. Different projects have implemented different applications feeding ITS secure secrets into their *"crypto-engines"*.

The focus of this study is quantum layer, and specifically the post processing approach and authentication means used in these implementations. For communication over public channel all these projects are using classic network channels – some are exploiting existing packet structures and technologies used in Internet such as IP, TCP, and IPSec for their post processing protocols; while there are a few who have implemented new protocols from scratch, yet very similar to OSI model. It is important to note the architecture of network packets is layered design with each layer having definite purpose.

2.3.1 DARPA QKD Network

Defense Advanced Research Projects Agency (DARPA), sponsored Boston University and Harvard University to implement the first Quantum Key Distribution network in BBN laboratories in early 2003. The network consisted of six nodes back in 2004 – two free-space nodes and the other four nodes connected through fiber. The network architecture allows usage of different QKD protocols amongst nodes, and also connectivity of other

[24]KEPs encrypt secret keys and send across the network

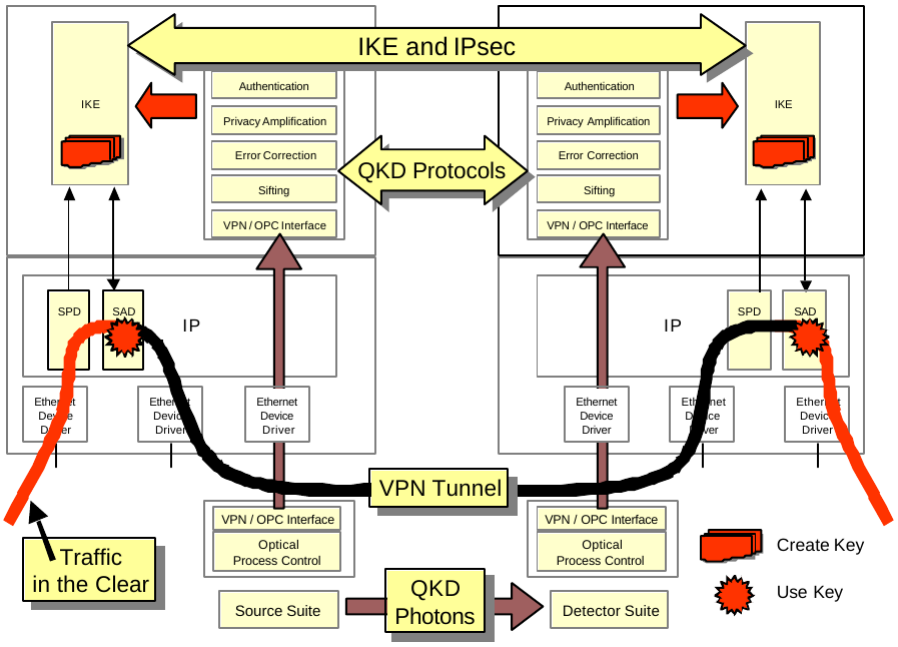

Figure 2.11: DARPA architecture for VPN [Ell05]

networks with different protocols[ref]. The QKD protocols used in this project are BB84 and BBN proprietary QKD protocol.

DARPA uses Internet packet structure for post processing. The network uses VPN tunnels between QKD endpoints to run post processing protocol, thus encrypting the discussion over the public channel – unlike original BB84. The VPN key agreement primitives are augmented or completely replaced by keys provided by quantum cryptography [ECP+05]. To my knowledge, there is no published details about the algorithms used in the VPN, or the nature of the augmentation done on traditional VPNs using IPSec ESP headers. Besides the post processing protocol are not discussed as well i.e. anything on top of Transport Layer (layer 4) is not elaborated on.

Figure 2.11 illustrates DARPA architecture. Keys generated from QKD and secrets shared using Internet Key Encapsulation protocol[25] will be used to generate new Security Associate[26] stored in Security Associate Database (SAD) for the VPN tunnel, or IKE. For encryption of the tunnel AES KDF (similar to IKEv2) is used with key rollover in every minute, and hashes are used for authentication (most probably SHA-2 HMAC given the project was implemented before in 2003 but it is not explicitly mentioned). Using IKE or a

[25]IKE protocol is a KEP that is the backbone to secret sharing over Internet. It uses symmetrical and asymmetrical cryptographic means to exchange session keys (shared secret) – current IKEv2 uses Deffie-hellman to generate shared secret keys between pairs and encrypt session key using shared secret.

[26]SAs contain the key for authentication.

key relay device, keys derived from QKD could be shared with other nodes in the network even if they are connected via quantum channel[27].

DARPA viewed QKD as a complementary technology to existing networks rather than something separate. The only application implemented for DARPA project is to feed ITS shared secrets into IKE which further would be used by IPsec to ensure secure communication using modified IPSec suite[28]. The system is implemented in C over NetBSD and runs in kernel space – modifications on the kernel has been done to allow QKD module interaction with IKE and IPSec daemon. The daemons themselves have been customized for the algorithms they use. At the time of publishment of the paper (2005), DARPA was considering about employing Carter Wegman tag for a final authentication – current status is unknown, but in 2014 it was announce DARPA is adding more wireless and satellite links into the project in news article.

The concept of three tiers illustrated in figure 2.10 was introduced later in SECQOC project, that is why in DARPA the distinctions are not very clear. In DARPA Quantum layer shares the medium for classical channel with key management layer and application. The same channel (VPN tunnel) is used for post processing and key management (key forwarding using IKE). The application layer is also defined within the key management layer – the application is using the keys shared by QKD in IKE which is used as key manager.

2.3.2 SECQOC QKD Network

SEcure COmmunication based on Quantum Cryptography (SECQOC) *"was a major research effort of 41 research and industrial organizations from the European Union, Switzerland and Russia, which was initiated in 2003 and carried out between April 2004 and October 2008. The SECQOC prototype in particular features six nodes connected by eight QKD links. The network was deployed in the internal glass fiber communication ring of Siemens (a SECQOC project partner) in Vienna, Austria"*[PPA+09]. It includes both free-space and fiber connections, and the project has implemented BB84 and SARG QKD protocols.

Unlike DARPA Quantum Network, SECQOC design philosophy recognized QKD as an entirely novel architecture of its own. Figure 2.14 illustrates an abstract schematic of SECQOC network architecture – it is consisted of interconnected Quantum BackBone (QBB) nodes by QBB links. Figure 2.12 demonstrates two QBB nodes in their most simple

[27] sent hop-by-hop through a trusted network in which each hop decrypts and then authenticates the key, and then encrypts the key with the secret it has shared with the next hop and send it to next hop until it reaches destination

[28] They claim they have plans to implement OTP over IPSec but do not discuss detail.

configuration having a single QBB link – QBB link is a point-to-point connection of many QKD device which perform the quantum communication to exchange ITS secret. This is the main difference between QKD links used in DARPA and QBB links introduces by SECQOC. More quantum channels provide higher rate of key generation and also higher reliability through redundancy. Each bundle of `n` QKD devices is accompanied with a classical point-to-point connection. The classical channel is responsible for all the necessary communication including QKD post processing, and secret key forwarding/routing. For all the communication over the classical channel SECQOC designed a protocol called Quantum Point-to-Point Protocol – its protocol stack is explained later in this section.

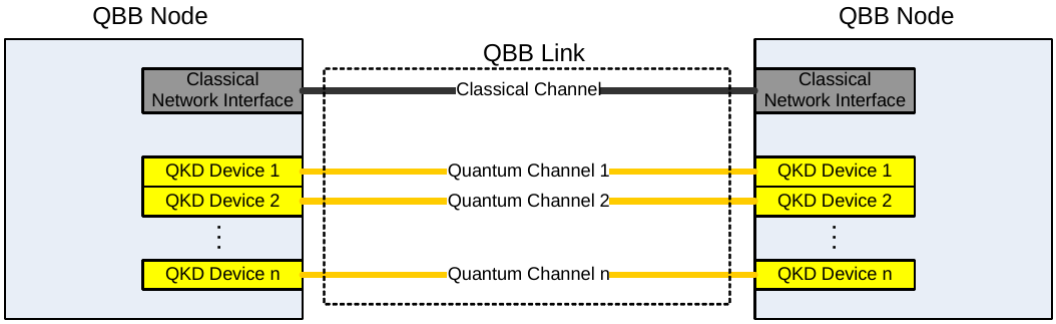

Figure 2.12: QBB link architecture [SBC+09]

Each QBB node could be connected to more than one QBB node. Figure 2.13 pictures a QBB node with arbitrary `n` QBB links to different QBB nodes in a network. Each QBB link is being handle by an individual instance of Q3P. Moreover, QBB nodes contain routing and forwarding module to pass the shared secret across the network. QBB nodes have other modules like quantum random number generators and post processing modules depicted as "Other modules" in the figure.

Clients located in different private networks get the same shared secrets from the QBB nodes located in their own private network, and then use those secrets to communicate over public network like Internet. Figure 2.14 shows this procedure. Clients in different private networks first need to register themselves with their QBB node. Then the QBB node will provide clients with the shared secret which the nodes had exchanged (shown in red dotted arrows). Subsequently, the clients use the secret to communicate over public network (shown in blue dotted arrow).

If the two clients are not connected through neighboring QBB nodes, the secret would be sent via intermediary QBB nodes in hop-by-hop fashion[29]. The secret is encrypted using

[29]Each hop authenticates, decrypts, find routes, encrypts using OTP, authenticates using Carter-Wegman

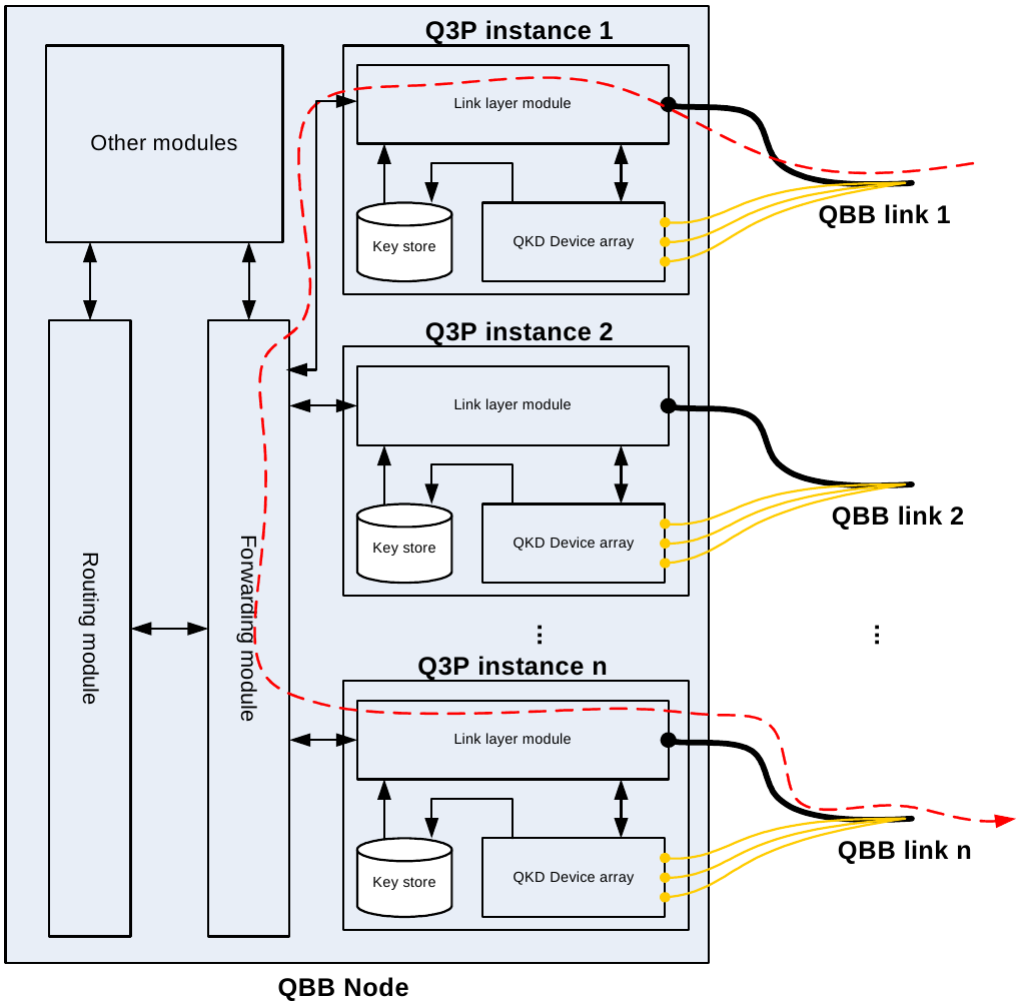

Figure 2.13: QBB node architecture [SBC+09]

the keys each hop has shared by their QKD mechanism until it reaches the destination. An illustration of secret forwarding could be seen in figure 2.13 in which the node is acting as a hop and a secret is being forwarded from a QBB node connected by `QBB link 1` to a QBB node connected through `QBB link n` – the path is shown by red dotted arrow.

Besides QBB, Quantum Access Node (QAN) could be used by clients to acquire ITS shared secrets. QAN nodes are connected to QBB nodes via secure links and they do not have quantum devices themselves yet take advantage of keys shared by other QBB nodes. The network topology detail and communication steps are described in [PPA+09][DAGS08]. For all the communication between QBB nodes including post processing and key forwarding Q3P protocol used.

SECQOC introduced a QKD protocol suit, illustrated in 2.15, which is a four layer protocol stack very similar to OSI model and TCP/IP stack, and lays on top of a classical

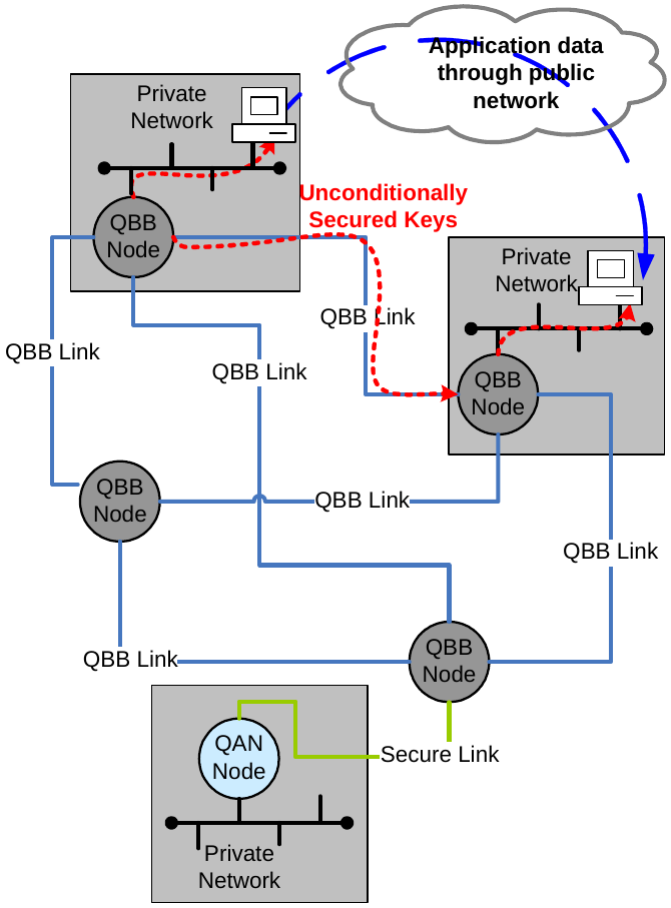

Figure 2.14: SECQOC network architecture [SBC+09]

TCP/IP socket or quantum channel interface. Quantum channel interfaces could be used when classical channel and quantum channel are multiplexed over the same fiber[SLB+11]. The first layer of QKD protocol suit is a link layer protocol, and links the connection of QBB nodes. In SECQOC, Q3P is used as Quantum Key Distribution Link Layer (QKDLL) and its header is demonstrated in figure 2.16. The two bit `EA` flag specify if the packet payload (the rest of the protocol stack) is encrypted and/or authenticated.

The next header in the protocol Q3P stack is the Quantum Key Distribution Network Layer (QKDNL) which has the exact same header as IPv4 [PPA+09][DAGS08] – on transport layer QKDTL shares the same header and mechanism as TCP to open a connection. Using similar technologies as classical network allows adopting similar routing protocols for forwarding the key in the QBB network – same as routing used in IP they are using OSPFv2[30]. There were attempt to address common issues current routing protocols are facing to decrease the packet congestion; no result of that has been published to my

[30]IP routing protocol.

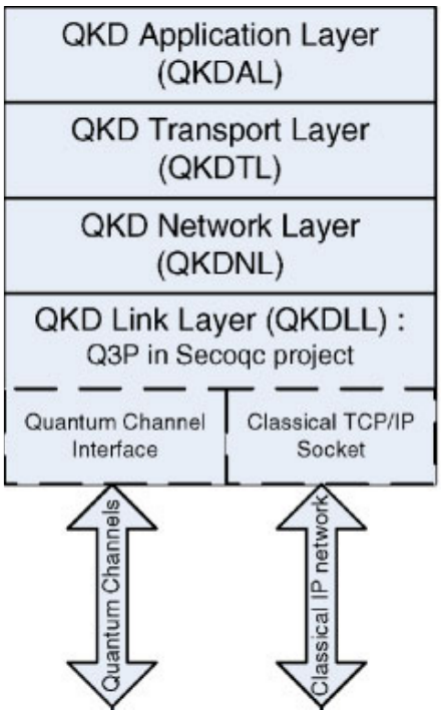

Figure 2.15: QKD protocol suit suggested by SECQOC[**?**].

knowledge though.

About application layer there is no specification; neither for post processing. Possible different use cases has been standardized [NFV13], yet no standardization is published for packet structures of Quantum Key Distribution Application Layer (QKDAL) – [NFV10] introduces API calls for different architectural levels to communicate with each other. It should be mentioned that communication authentication is not instantaneous as the Q3P approach is based on delayed authentication, taking place before a distilled key is declared 'secure' by the QKD link.

2.3.3 SwissQuantum

SwissQuantum network is the first international QKD network initiated in 2009 with 2 nodes in Switzerland and one node in France. Each node had two sub-nodes to maintain a point-to-point connection with the other two. The quantum layer of the project relied on trusted intermediate node and the used multiplexing of quantum and classical channel to reduce implementation cost. The classical channel is composed of post processing channel, key routing and forwarding channel, encryption applications channel, and/or monitoring channel.

The QKD protocols used in this project was SARG which is resilient against photon

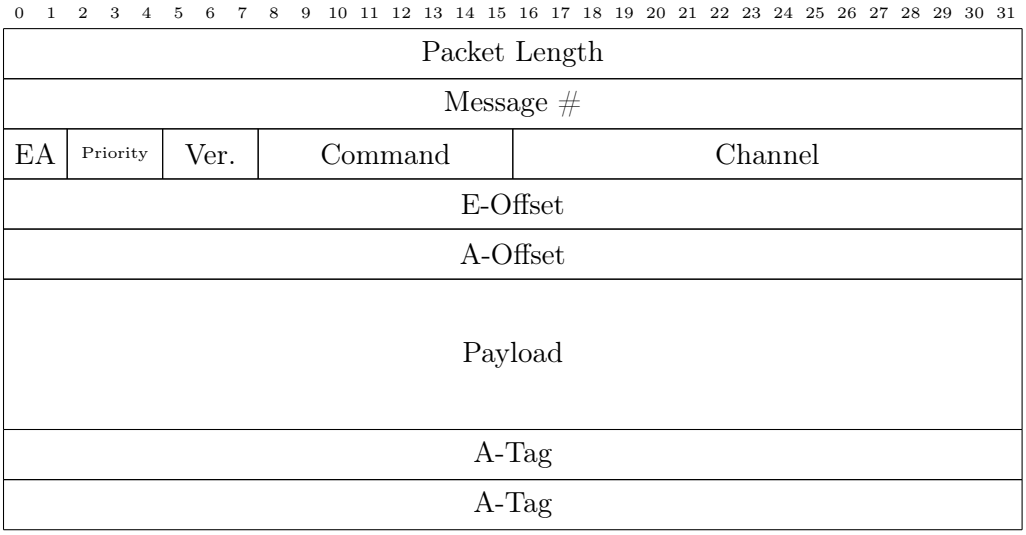

Figure 2.16: Q3P packet header.

number splitting attack and is more efficient in long distances [SLB+11]. CASCADE is used for error correction and Wegman-Carter scheme authentication is deployed to authenticate all classical communications. Raw key exchange over quantum channel took between 4-5 minutes, and 1 minutes for post processing[swiss00]. In order to increase performance, they introduced link aggregation on quantum channel – analogous concept to QBB link structure with aggregated quantum links.

The keys exchange by QKD will be combine with keys generated using RSA[31] PKI compatible with X.509 recommendations[32] to create the finale keys:

> *"Depending on the combination technique, this final key can be as secure as the more secure of the two initial keys [...] this combination is not used to increase the security of the resulting key, but for improving the reliability and availability of the applications in case of failure of the QKD layer".* [SLB+11]

The application of the project was to provide secure keys for encryption over Layer 2 (Ethernet or fiber encryptors) and Layer 3 (IPSec encryptors) in OSI model. Keys are encrypted by OTP schemes to traverse through the network. Detail about the protocol used for key forwarding and other communications such as post processing is not explicitly mentioned. However, given the fact that the project uses the same devices as SECQOC and exploits the same architecture, it seems QKD protocol suit illustrated in figure 2.15 is used.

[31] Rivest–Shamir–Adleman asymmetrical encryption.
[32] X.509 certificates authenticate public keys.

2.3.4 Wuhu QKD Network

In 2009, China also implemented a weak+vacuum decoy BB84 QKD network – it was the first hierarchal network consisting of four nodes in backbone and a subnet containing three more nodes, making it a 7 nodes network. As illustrated in figure 2.17, one of the backbone nodes works as a "Trust Relay" and extends the network.

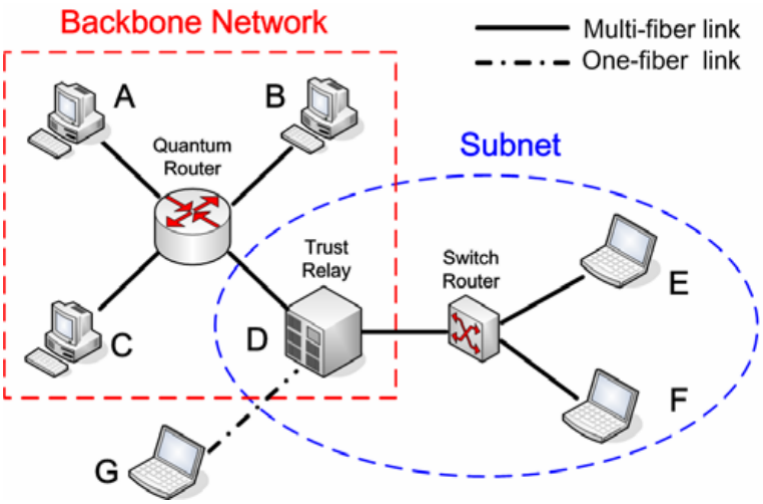

Figure 2.17: Wuhu QKD network diagram [XCW+09].

All the nodes located in the backbone – the important bureaus with high priority – could in theory play the role of a trust relay. Backbone Nodes are interconnected in a mesh topology through the Quantum Router with WDM which can establish $N(N-1)$ connection in a N port network at the same time.

The idea of subnet came from TCP/IP architecture – subnet enables extending the network in hierarchy fashion and create separate segments from the backbone to avoid information collision. In the subnets, delays are accepted to achieve better performance in overall. Trusted Relay acts as a hop between the subnet and the backbone and all the information transmitted between backbone and subnet is known to the trust relay.

Wuhu Quantum network is deployed in telecom stations and since there is Ethernet access point, they had to create their own LAN, but the communication module follows standard TCP/IP [XCW+09]. Cascade was used for error correction and universal hash function for privacy amplification – a quite similar choice between other implementations as well. Parameters from decoy state method are used to choose a the hash function from the family of hash functions for privacy amplification step.

There is no explicit information about the post processing protocol or authentication used over the classical channel. The final key generation rate was not high enough to

consider OTP. Hence, AES was used to *"encrypt the plaintext with the fast refreshed key sequence of 128 bits supplied by the QKD process"*[XCW+09].

2.3.5 Los Alamos National Laboratory NQC

In 2011, Los Alamos National Laboratory introduced a new concept called Network-centric Quantum Communication. NQC is a network in star topology as illustrate in figure 2.18 – it follows the same three tier type of architecture. In physical layer different clients distribute ITS secrets with a central node also known as "hub" and feed those secrets to Quantum Key Manager (KML) layer which enables secure communication between client. In star or "hub-and-spoke" type of topologies which are very common in wireless networks – the central node is a trusted entity which does the work of key forwarding and key sharing between clients in different level – this addresses the trust issues in the previous implementations where all the nodes had to trust each other. In "hub-and-spoke" topology, trust could follow a hierarchal architecture with the hub acting as the only trusted authority (TA) [HNM+13].

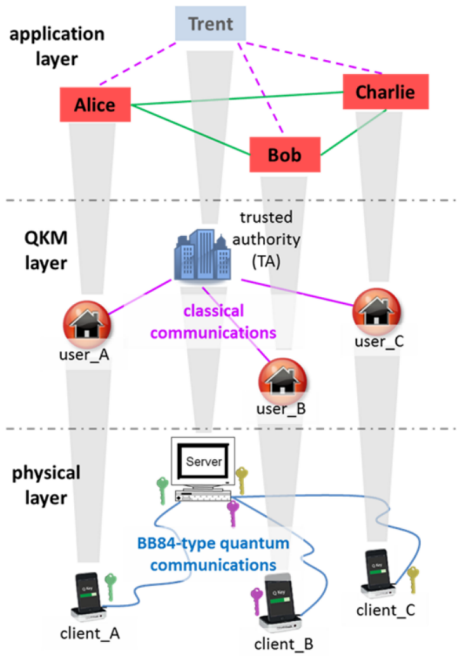

Figure 2.18: Los Almos NQC QKD architecture [HNM+13].

Los Alamos project is using a single multiplex quantum communication receiver in Trent and all the nodes are equipped with quantum communication transmitter – given the fact that the most expensive part in QKD implementation is the single photon detectors used in receivers [HNM+13] – the costs of implementation drops considerably. In order to further

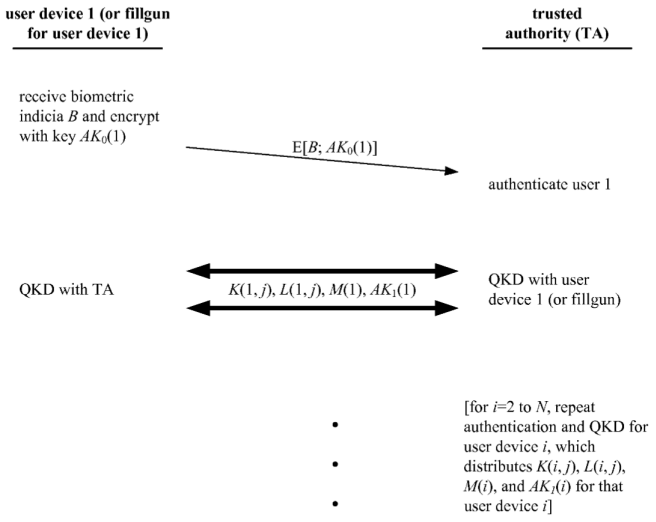

Figure 2.19: Los Alamos communication between client and trusted authority [WIPO WO 2013/048674].

reduce the costs, transmitters are using polarized qubits instead of phase-base qubits. This would also allow manufacturing of the transmitter device in a cheaper and smaller (more practical) scale.

The project implemented different type of BB84 protocol, and employed LDPC codes as error detection and correction technique. Toeplitz universal hashing was used for privacy amplification of the shared secret after reconciliation. In regard to authentication and post processing communication, no technical detail about the post processing protocol and implementation of the authentication method is given. It is mentioned cryptographic CRC hash function were used though – these are lightweight authentication algorithm. Authenticating the keys were generated by quantum identification protocol (QID) [HNM+13]. Moreover, QID could be used for client registration and revocation in the trusted authority.[33]

Figure 2.19 shows the registration procedure of client by the trusted authority. Upon first connection to the "hub", each user will use an identification like a biometric`B` and encrypts that using a pre-placed share key with the trusted authority. Once the message is authentication and required steps are taken to approve the identity, the trusted authority and the user will generate encryption keys `K` for the client to use with other j users in the network. The user will use these keys later on to communicate with other nodes in application layer. Besides, the trusted authority and user generate secrets `L` for key derivative function used between the user and other nodes. Authentication keys `M` are also generated

[33]Patented in WIPO WO 2013/048674.

and finally the trusted authority and the user refresh the authentication key between them to AK_1.

The trusted authority will go through the above procedure with all the nodes and later creates key pair $P(i,j)$ and authentication key $A(i,j)$ as shown below – the procedure takes about 8 minutes for each node. H is *"secure keyed cryptographic hash function like HMAC-SHA-256"*[PATENT].

$$P(i,j) = L(i,j) \oplus K(j,i) \quad \text{for } 1 \leq i,j \leq N \text{ and } i \neq j$$
$$A(i,j) = H[K(j,i); M(i)] \quad \text{for } 1 \leq i,j \leq N \text{ and } i \neq j$$

The pairs are later sent to users over public non-secure network – as a matter of the fact, once these pairs are published, the trusted authority could go off-line and the clients must be able to derive shared secret to communicate with each other based on these published pairs – user i calculates $K(j,i)$ to communicate with user j using its own key derivative corresponding to user j, namely $L(i,j)$ and the published $P(i,j)$, then with his authentication $M(i)$ and calculated $K(j,i)$, computes $A(i,j)$ and authenticates $K(j,i)$ – the other user j can derive and authenticate $K(i,j)$ following the same steps, and users have shared more than one secret which they could consume based on the application. Devices are constantly updating these values in the physical layer and the above formula is happening in the Quantum Key Management layer which provides these keys for secure communication in the application layer. On top of this, the patent of Quantum Key Management also include password based QID and some other methods for digital signature, multi-party computation and more.

2.4 Summary

Previously in this chapter, the post processing steps are explained and post quantum secure authentication algorithms which are mostly suggested for QKD are listed as well. Subsequently, QKD network implementation were discussed. In the final section of this chapter, I will try to give a comparative summary of each section in this chapter in which I try to compare all the explained techniques with the requirements explained in section 1.4. The arguments here are the building blocks of the proposed solution in next chapter, chapter 3.

2.4.1 Post Processing

As mentioned many times, BB84 definition of post processing itself is rather loose but it is explicitly mentioned in the paper post processing conversation need to be authenticated.

It feels like the paper is trying to lay out some insight and give suggestion on how to perform things. On the quantum communication, however, the paper is definitive; yet through further analysis was subject to modifications – a good example would be decoy based BB84 with slight change in communication over quantum channel to resist photon split attack.

It is evident that post processing is an inevitable part of QKD, even if the quantum communication becomes flawless and happens without any error. Certainly, parties need to discuss the choice of their detection basis in each measurement – random selection of detection basis is necessary to rule out MITM[34].

Post processing steps are subject to alteration and adjustment[35] – something that occurred with suggestion for error correction and further privacy amplification steps added to post processing. Even the sequences of performing can differ; especially for error detection and privacy amplification. I even encouter an implementation which had a final verification of the key after privacy amplification – a 200 bits sample [FDD+09].

For each step of the post processing, there are different approaches addressing different requirements which shine in specifc scenarios. This was more evident in sifting and error correction steps. For example if decoy based BB84 is used, instead of LDPC, Cascade algorithm is the first choice. Since it does not need an exact QBER and can work with a good estimate which parties could calculate from decoys, and this results in less communication and given Cascade algorithm performance yields faster operation, it also boost the perfromance of the system. However, if the estimate is not accurate, then the possibility of an error in the key after conciliation still exist. Considering this key would be fed into privacy amplification methods (figure 2.1) which surely have diffusion[36] property the final key each party will calculate could be significantly different. Hence, depending on the noise expectancy the suitable error correction could be used. It is also common to substitute the order of the steps depending on the needs of the project. For example if the project is deployed in complete private network or when Casacade is used which does not leak much information about the key, I have seen the privacy amplification step happening before error correction to re-conciliate errors in the privacy amplified or final key.

To make things even worst, there exist different BB84 variants; divergences are so extreme sometime that the derived protocol becomes a new protocol on its own. SARG is

[34]Otherwise, an intruder could easily perform MITM by transmitting the exact same qubit it detect from the sender to the receiver being located in the middle of communication

[35]In theory, once the quantum communication becomes error free, error detection could be removed, thus privacy amplification. Flawless communication in practice, however, seems far fetched.

[36]Diffusion means that if we change a single bit of the input of a function, then (statistically) half of the bits in the output should change.

an example of which – it is more efficient in long distances and resilient against photon number splitting attack. Instead of single photon, SARG operates on attenuated laser pulses. Clearly, detection of weak pulses requires more advanced hardware comparing to single photon, which increases implementation costs. Therefore, modified single photon protocols like BB84 are still the choice for short distances. This is the decisive fact in SwissQuantum and Los Alamos QKD network implementation choice of QKD protocol due to their different requirements. Short distance meant near 20 kilometers in 2000s, this should have improved given the technology advancements over the decade.

Moreover, distinctive related concepts in the domain of quantum physiques could reshape things in QKD e.g. quantum entanglement concept which lead to development of entanglement based protocols such as E91. These protocols gain their security based on the fact that if two particles are entangled in an object, measurement on any of them affects the overall system and can be detected[ref.E91] also known as *Quantum Qntanglment*, which is different quantum physique fact from the one used in BB84. Development of this concept led into rise of different QKD protocol families.

Therefore, it is easy to deduce that it is not the matter of which is the best protocol or the best approach; it is more about which one is suitable for your project requirements e.g. the projects prioritized quality attributes (e.g. security over performance, cost, efficiency, etc.), distance between the nodes, budget, type of the link (fiber, free-space), and etc. which were not available to me given the stage the research project I am involved in, currently is. Nonetheless, it was important to understand the essence of post processing and communications needed to propose suitable authentication method for it. These diversity shall be considered for the proposed solution and the infrastructure shall be able as generic as possible towords this layer to accomodate all needs. Furthermore, by studying post processing literature, one could see common choices of algorithms and approaches for each step of post processing in most of the literatures which could be view as *de-facto* standard of the industry. These are use of Cascade over LDPC for better performance, uses of universal hash function for privacy amplification with inputs from decoy states, choices of QKD protocol based on the distance and medium used.

2.4.2 Authentication

Authentication algorithms are the core of this study – the goal is to propose suitable authentication method for QKD post processing. Before arguing about authentication algorithms, however, it is constructive to recall the view point of the study over some definition. First and foremost, is the definition of security. As mentioned earlier in section 1.4 of this thesis, an algorithm is considered secure which could provide 128 bits security even at presence of

quantum computers. In order to comply with the security requirement set for this study, all the explained authentication algorithms are accepted as post-quantum secure, meaning no known quantum and classical attacks are known for them except brute-forcing the key space, at the time of writing this document. Hence, if they are used with key size equal or bigger than 256, they will provide 128 bits security at least. Throughout this study some algorithms which are in used today are labeled un-secure, simply because they cannot stand quantum attacks and there are attacks more efficient than 2^{128}. This includes even some ITS scheme due to their implementation and short key size (e.g. 128 bits key size which provides $2^{\frac{128}{2}} = 2^6 4$ bits security) which in presence of a quantum computer would not provide reasonable security[37].

Another view point of this study is algorithms performance efficiency and key consumption measured in bits. These are both extracted from the practical implementation of protocol and are the answers to *"what is suitable for QKD?"* question listed in the scope of this study in section 1.4. Both are stressed out in many literature and the efficiency is a necessity for the proposed solution in next chapter. The dilemma between performance and security is traditional in cryptography and even in the whole cyber security field, as one contradicts the other in most of the cases. However, having well defined the security requirements, I later argue that this may not be the case in this study, and it is possible to achieve the best performance and the best security within the established realm.

Algorithms perform differently over hardware and software due to different nature of architectures. No one could argue about the fact that an efficient hardware implementation will perform superior to software. However, logic circuit implementations are not flexible and for some implementations they might not be even reasonable to consider. Besides, hardware implementation could increase the cost of modification compared to software implementations. As argued before, QKD seems like a very dynamic field. Given the ever rising CPU power, parallelization, memory capacity and speed presented at software level nowadays, on top of flexibility and resources that exists there, I later assert proposed software implementation could achieve competitive performance to hardware implementations, if not better. The scope of the research assumes the implementation of post processing happens over software, however, I later suggests to even perform packet capture in software at user space level to reach higher performance.

Authentication tag generation and verification resides in a level in my proposed solution, where it is not rational to consider hardware efficient algorithm as well. Indeed, considering public channel as classic network using IP, authentication tags are either generated using standard schemes like IPSec or by project specific implementations like Q3P. In case of

[37]Grover's quantum algorithm can search a space of 2^n in complexity of $2^{n/2}$.

IPSec, the authentication tag is located after IP before TCP and in case of project specific implementation it is on top of TCP. With IPSec, some NICs implement authentication algorithms listed in IPSec RFC-6071. They get the information about the secret keys from kernel module which provide them with the key from SAD and the policies from SPD. These calls from hardware to software, and kernel stack calls are not very efficient as I reason later. Moreover, none of the implemented algorithms listed in IPSec RFC-6071 comply with this study security requirement except for AES and that is not the authentication algorithm choice of the proposed solution in next chapter. If a customize protocol is implemented for authentication purposes like Q3P, then authentication tags are presented on top of transport layer known as application layer, and by default the payload of link layer leaves kernel space and enters user space. Then to once again send it to hardware for MAC verification or generation does not seem rational.

Suggesting efficient algorithms without foreseeing the bigger picture would make no contribution. Hardware based proposal should be considered for projects where the outcome is developing QKD devices like IDQuantum – mass production of hardware based devices would result in cheaper overall price. However, I believe even in QKD devices software based implementation might seem to be a better choice given the diversity and probable modifications that could happen. Current development benefit from hybrid architectures, where both software and hardware perform their parts.

Mentioned in the scope section 1.4, the research assume the raw key is presented in software, most probably in RAM. To measure performance in software, algorithms would be classified based on CPU cycle per byte (cpb) i.e. how many cycles would take an algorithm to process a byte of the message. 1 `GHz` CPU does 1,000,000,000 cycle per second, and could process 1 Giga Bytes of data over a second for an algorithm with 1 cpb performance. Two different algorithm could operate at the same speed if the input data rate is low e.g. a 10 cpb algorithm on data input rate of 100 MBps on a 1 `GHz` perform the same as 1 cpb algorithm. Thus, fast performance is dependent to the load as well. There are implementation techniques to dedicate a CPU core to a process[38] – a normal Intel Core i5 Skylake (6th generataion) has four 3.2 `GHz` cores.

It should also be noted that the algorithms which were subject to analysis in this study are chosen from those who have been under heavy cryptanalysis and they are accepted in the community. Usually submitted algorithms to standardized competition have undergone

[38] `isolcpus= cpu_number` kernel parameter removes the specified CPUs, as defined by the `cpu_number` values, from the general kernel SMP balancing and scheduler algorithms. Processes could be load onto or off an *"isolated"* CPU through the CPU affinity syscalls. `cpu_number` begins at 0, and the maximum value is 1 less than the number of CPUs on the system.

a reasonable amount of analysis both before submission (as a requirement for submission), and during and after the competition. And I dub a cryptosystems accepted by the community when there is accepted RFCs and standard specification for them by either IETF, NIST, or other well-known organizations.

Block Ciphers

Block ciphers are one of the common choices for MAC generation, specially AES due to the its exclusive enhancements on high end CPUs and hardware implementation, as discussed earlier. Many big names in the industry like Google, CloudFlare, and many more are migrating from AES based to stream cipher based MAC generators like Chacha20-poly1305 though. This is due to poor performance of AES on mobile and low end CPUs, and steady high perfromance of **Chacha20** over all platforms.

Another reason commonly found to justify picking alternatives to AES is its security. There has been some attacks during the two decades the algorithm has been around. While most of these attacks were used to be on the implementation or other security consideration such as re-use of nonces with the same key rather than attacks on construction of Rijndael itself; recently there has been evidence of attacks on the construction of the algorithm. One example is AES long key scheduling time needed by the scheme which is not very efficient and makes the scheme vulnerable severely. In 2005, Daniel J. Bernstein, a well known cryptographer, inventor of Salsa and Chacha stream cipher families also auth256 ITS MAC generator, demonstrated *"successful extraction of a complete AES key from a network server on another computer"* using and he claimed *"the same technique can extract complete AES keys from the more complicated servers actually used to handle Internet data"* [Ber05a] – there are many more examples of *cache-collision timing attacks* against AES, recent implementations have take consideration against known ones.

Although skeptics think these new vectors of attack will propagate and the security of the algorithm will be eventually obsolete, if it is not now[39]. It may sound like a paranoiac view point since it is not based on any evidence; this could be rationalized better in military or very confidential settings though.

Threefish, designed in 2008, is a block cipher related to Advanced Encryption System finalist Twofish, and is secure against cache timing attack thanks to its different key scheduling. Threefish originally was part of SHA-3 finalist Skein; these facts imply the algorithm has undergone a reasonable cryptanalysis. Threefish out performs AES in software on CPUs with out AES instructions. Another option would be using BEAR and LION, two

[39]Some believe NSA or similar organization might have a back door for AES, a similar case actually happened for DES exported devices during 80s and 90s.

provably secure block cipher constructions building block ciphers from hash functions and stream cipher. Both BEAR and LION can process messages in very bigger block blocks than other i.e. 1KB to 1MB. It is not mentioned in the paper what to use as nonce for case of short messages or what other security precautions are needed – assuming you need a secret initial seed for that, both algorithms need 3 keys, but can process big chunk of data. however, they have 3 rounds of function which comparing to AES with exclusive instructions are slow.

Apply these to the scope of this study, AES performance is not a concern for this study's criteria though. AES instructions are implemented on many AMD and Intel CPUs which makes AES outperform all other algorithms. **ChaCha** is known to have the best performance among all mentioned algorithm across all platform, still AES with help of exclusive CPU instructions can even outperform fastes **ChaCha**, *ChaCha8*[40]. Table 2.1 compare the cpb of both algorithm on x86_64 CPU architecture[41]. **ChaCha** is a stream-cipher and it is more accurate to be compared to AES in CTR mode.

Message Size	**aes256ctr**	**chacha8**
8 bytes	19.25	19.25
64 bytes	2.41	**2.06**
576 bytes	**0.69**	0.92
1536 bytes	**0.56**	0.80
4096 bytes	**0.50**	0.78
Long messages	**0.48**	0.77

Table 2.1: x86_64 stream cipher architecture AES benchmark.

Thus I argue AES performance is not a concern given the fact we are using high-end CPUs. And regarding security, I personally believe AES is benefiting from all of the analysis. As soon as an attack is possible, the community responds rapidly to address the issues. Hence, I conclude AES is most suited block cipher algorithm for the purpose of this study.

The performance of block cipher MAC generators also depend on the mode of operation. CMAC is the least efficient amongst them due to its linear and sequential computation nature. Parallelized MAC, was patented initially which lowered its rate of adaption, comparing to GCM mode which was introduced years later. I could not find any benchmark on

[40]ChaCha8 has 8 rounds, and still is post-quantum secure. Best attack for Salsa family happens with 7 round. The most in used ChaCha variant is with 20 rounds.

[41]`https://bench.cr.yp.to/results-stream.html`

PMAC, and about others the benchmarks are not from validated sources. I found a benchmark of the algorithm implemented in cryptopp library and some personal benchmark[42] which all shows GCM mode's GMAC outperforming others. Results for *cryptopp* library implementation[43] are demonstrated in table 2.2. I have also included Threefish CTR in the table. Unfortunately, I could not find any benchmark for Threefish used in authentication modes, thus to have a better comparison I added AES CTR as well, both with 256 bits keys. Although Threefish is suppose to outperform AES, but benefiting from the exclusive CPU instruction, Threefish is almost 10 times slower than AES despite a slight better performance over the key setup. And that is the reason I did not study Threefish in detail.

Algorithm	**cpb**	**Cycles for setup**
GMAC(AES)	0.34	995
Poly1305(AES) (256-bit key)	2.47	471
CMAC(AES) (128-bit key)	2.41	310
Threefish-256(256)/CTR (256-bit key)	6.86	597
AES/CTR (256-bit key)	0.77	630

Table 2.2: Skylake generation microarchitecture benchmark of cryptographic algorithm.

GCM, NIST selected block cipher mode for 2007, has been under comprehensive analysis. There has been successful forgery attacks when the tag length is short, when the nonce is re-used the IV would be the same and key H for $GHASH$ could be retrieve[21] — also enabling attacker to choose the IV would result in counter IC colliding which leaks the key, and finally there are known many weak keys for the scheme as well. However, these attacks were considered *"not [to] contradict the claimed security bounds by the designers"*[IOM12] and *"are outside the security model"*[IOM12]. In 2012, it was shown that the security bounds are lower from the one designers stated in the original paper. The same study demonstrates when the nonce size is equal to 96 bits, the security bounds are higher[IOM12].

All of these issues are addressed in recent implementations. **AES-GCM-SIV** calculates initial counter *"pseudorandomly"* for every different nonce/message pair and *"even if the actual nonce repeats, the effective nonce used to mask the encryption is different for different messages"*[GLL17]. In 2016, first IETF draft for *"**AES-GCM-SIV**: Nonce Misuse-Resistant Authenticated Encryption"* was published. There are precautions in regards to choice of parameter selection listed in literatures which one must follow for the

[42] `https://github.com/randombit/botan/issues/969`

[43] `https://www.cryptopp.com/benchmarks.html`

scheme to be secure, latest parameter suggestion and analysis I found for this mode could be seen in [BHT18] proceeding for EUROCRYPT 2018. All these are perks of heavy crypto analysis happening on the standards – no attack is yet found to treat the construction of GCM itself.

It should be noted that CAESAR post quantum secure authenticated encryption finalists has been announced on May the 5th 2018. Most of the finalists are AES based schemes. The competition started in 2013, and the selection results are soon to be publish – the result could be used to choose even more efficient and more secure MAC generator, assuming it would be possible to use them in authentication only mode like GCM. It is common to find major security flaws during each round. Near 50 algorithms were submitted, and now 7 finalists are competing for 3 different use cases since *"experience with previous competitions suggests that a single-algorithm portfolio is unlikely to provide as much value as a multiple-algorithm portfolio"*[44] and this means having standard algorithms for different use cases which relaxes selection process based on project requirement.

Cryptographic Hashes

Unlike block cipher, comparison between hashes are easier as there are only two to be considered: BLAKE variants or Keccak family based. It used to be justified that if performance specially on software is the highest quality attribute one is looking for, BLAKE variants should be chosen, and if security or hardware performance is what you are after the most, choose SHA-3. BLAKE used to be the only SHA-3 finalist with efficient potential of parallelism, and although it was used on HMAC nested style scheme which is slower in abstract design level compared to KMAC that just hashes the key and message together, BLAKE on HMAC would still be a faster alternative [ANWW13]. The algorithm has a great rate of adaption in many projects[45]. In 2018, however, Keccak team published KangarooTewelve which is very competitive with BLAKE variants, out performing them in near three times less cpb in some cases, as shown in figure 2.8. Although the algorithm has been recently published, but the family it derives from and all the construction used has been under heavy analysis for ten years, and as long as there is no attack on SHA-3 for 12 rounds, the algorithm is as secure. As mentioned before, the best attack known happens over 6 rounds [BDP$^+$16].

Differences with the original SHA-3 SHAKE are less rounds and efficient parallelism which both boosts the performance drastically. The fact that it has been proved all keyed sponge functions could be considered as random oracle is very assuring – the security claims

[44] `https://competitions.cr.yp.to/faq.html`
[45] List of users could be found here:`https://blake2.net/#us`

in the paper states the algorithm *"shall offer the same security strength as a random oracle whenever that offers a strength below 128 bits and a strength of 128 bits in all other cases"*[BDP+16]. Using the key as initial state in inner-keyed sponge with one less round of permutation comparing to outer-keyed sponge, seems to be a good choice. Rationale behind the security claims are explained fully in the paper – the fixed parameters chosen in KangarooTewelve like state size [BDP+16] remove the burden of choosing safe parameters from users and makes user feel more relieved to use the algorithm compared to GCM variants. Another benefit compare to block cipher based is the setup time of the algorithm. AES key setup and scheduling is known to take some time, while Sponge construction, as mentioned earlier, does not have heavy setup – input needs to be splitted and padded properly which is not time consuming, and then you can start `XOR`ing them with the state and start computing permutation. The performance for the algorithm is around 1 cpb in worst case scenario (figure 2.8).

Universal Hashes

The only ITS MAC generation algorithm which is cited in many literatures is Carter-Wegman style. It has been introduce 30 years ago, yet it is not very common to use them. The reason for this could be the fact that universal hashes are biased in performance i.e. they other are efficient in hardware or software, and even in software implementation are very architecture dependent. Another hurdle for using these type of scheme is their inefficiency in terms of key usage.

Two known implementations are UMAC (suitable x86 architecture) and VMAC (UMAC variant for X86_64). Last modification on implementation of VMAC happen in April 2007[46] and there is an implementation on cryptopp library, but NIST has removed cryptopp from the certified list in 2016. All of these are not very good signs. And the worst is both implementations use 128 bits keys i.e. 2^{64} bits security in post quantum era which is not considered safe.

In 2014, Daniel J. Brenstien creator of Salsa, ChaCha, poly1305, and many more *high-speed* cryptographic algorithms[47]; co-authored a paper, demonstrating auth256, a super efficient ITS MAC generator in Carte-Wegman style [BC14]. The software implementation is in public domain. Same as other Carter-Wegman style MAC generator, the adoption rate of auth256 does not look good and there is no benchmark of the algorithm apart from performance claims by the authors. They have compared the algorithm to VMAC, and two other fast hashing algorithm, namely Poly1305, and efficient exclusive $GHASH$

[46] `http://www.fastcrypto.org/vmac/`

[47] For a complete list please refer to `https://cr.yp.to/papers.html`

implementation by Intel on Sandy Bridge microarchitecture (2nd generation) and Inter Core 2 microarchitecture. The comparisons have not been performed very systematically, as results for all platforms are not presented.

Besides they do not reveal the result for VMAC and dont mention the size of the key used in VMAC nor the encryption algorithm. They only mention VMAC is faster than Poly1305 without giving any result for VMAC. The comparison between VMAC and auth256 is more relevant as others are only hash algorithms. For instance, both Poly1305 and GHASH still need to perform one encryption step before generating the tag. This imply auth256 outperforms GMAC which is very promising. Benchmark of HMAC-SHA-1 which is currently widely in used over Internet is also presented for comparison by the authors. Table 2.3 summarizes these benchmarks, for those microarchitectures which the paper has not presented benchmark, the corresponding cell in table is filled with "–".

Algorithm	**Sandy Bridge**	**Core 2**
auth256	1.43	1.89
GHASH (without PCLMULQDQ)	10	–
GHASH (with PCLMULQDQ)	1.79	–
HMAC-SHA1	5.18	6.74
Poly1305	1.22	1.89

Table 2.3: Benchmark comparison of auth256 with other algorithms.

Suitable Authentication

From the mentioned scheme, block cipher based and cryptographic hash based are satisfying the lower security bound of this study i.e. they are post quantum secure, and Carter-Wegman styles are complying with the higher security bound i.e. being ITS. For the lower security bound, KangarooTwelve is the most promising one. Apart from the performance, and unmatched security[48], the algorithm pis useful for other cryptographic means like KDF and etc. – something that is in more detail in chapter 3. KangarooTwelve is easy to use compared to others in regard to choice of parameters as the algorithm has taken care of many of them. Given the proof for the keyed sponge acting as random oracle, I believe the scheme could also be seen as ITS if the key size and the size of the capacity of the sate are big enough and their values are kept secret. This has not been mentioned anywhere, but it is easy to deduce, since the random oracle is unconditionally secure. Therefore, I believe KangarooTwelve can also satisfy the higher security bound

[48]Keyed sponge and KMAC are prone to many attacks known for other construction as written before.

as well, which might be ground breaking given all ITS scheme required twice the key – this need further research. Therefore, I suggest auth256 for the higher security bound as well. Reason for this are discussed earlier.

There also two very different mindsets towards authentication of post processing messages, namely delayed, and instant. In delayed authentication there are two subcategories as well, one is to authenticated all the post processing communication at the end of post processing. The other is to authenticate the last message or exchange only the authentication of the secret key generated. These are very implementation related. I could not find any tangible reason to favor each above the other since each one has its own benefits. Delayed authentication definitely are more efficient in terms of performance and key consumption (in case only the key is authenticated). However, it takes a long time before you can trust the key, while the confidence in the communication is gained immediately in instant authentication with the first packet. This is why both should be considered in a generic solution.

2.4.3 QKD Networks

Literatures discussing QKD post processing and authentication could be classified in two main categories: **(i)** those which are discussing in algorithm level. Most of these papers are proposing a new or modified version of universal hash family to achieve better performance or exceed the security bounds on message length, use of nonce and etc. I call them *theoretical* ones. **(ii)** The other class, study the problem in implementation level, hence labeled *practical* ones. I could not find any paper focused at this level, except field tests or other related publishment of QKD Network implementations.

For this study which thrives to propose post quantum secure authenticated infrastructure for post processing, practical literature is favored. This is one of the main rationale behind analysis of QKD network implementations. Theoretical literature algorithms usually does not have an implementation, even if there is one it has *"academic"* standard, usually as a proof of concept, and not an efficient, well analyzed, secure, implementation. Moreover, to propose a suitable authenticated infrastructure for post processing, it was needed to have a better understanding of the use cases of the system. Identifying different requirements and realization of the network architecture was possible though this analysis. Indeed, there are perquisites and specifications forced from the architecture into the design of this authenticated infrastructure. In this section, I will try to discuss these point which led to shaping the proposed solution architecture in section 3.1.

QKD networks implementation could be viewed in two different categories, namely SECQOC based, and others. With no doubt, SECQOC project, has had a major contri-

bution towards standardization of QKD networks. The well researched project is the first to introduce the three layer architecture of the network. The idea is so neat, clear, and natural that I could even identify the same three layers in DARPA project which was implemented before SECQOC. All implementations are following the same hierarchal architecture. In this scenario, as already depicted in picture 2.10, post processing happens in the lowest layer: Quantum layer. Whatever the algorithms used for authenticating the key is presented to the algorithm in quantum layer from the above layer, Key Management layer. ETSI has already standardized these API calls between layers [NFV10]. The key management, in all implementation, is receiving secret keys from quantum layer, and later will provide them to application layer or quantum layer for consumption. Key management layers could also consume these secrets when trying to forward keys in the network. In case shared secrets are not available between two nodes e.g. run zero of QKD when no key has yet been distributed, key management layers use known KEP such as IKEv2. Some implementation are using the same protocols for key forwarding (e.g. DARPA) while others perform hop-by-hop OTP (SECQOC).

Amongst all implementation, Wuhu and Los Alamos approaches and choice of network topology allowed them to almost remove the need of key forwarding for some users in the net work, thus result in different key management layer amongst the nodes in the network. Illustrated in figure 2.17, Wuhu implementation benefits from a Quantum router. Thus, each node in the Backbone network could efficiently share secrets with all other nodes in the same network at the same time. When two clients in different subnets would like to communicate, they need to get the one of the shared secrets in the backbone from their trust relay (the node in their subnet which is connected to the backbone network as well). Clients nodes have a quantum connection to their trust relay and can run QKD with them. Those keys could be used to encrypt the session keys to use fr a communication with a client in other subnet. End clients do not need to forward key and their Key Management Layer is different from backbone nodes. This approach, which could be seen as a small hop-by-hop (one hop) is definitely more efficient than SECQOC hop-by-hop variants, as there is no need for a routing protocol to identify fast and secure route, and the session keys are encrypted less, which decreases total key consumption rate, comparing to encrypting using a key between each pair of hop in SECQOC.

Los Almos implementation, however, exploited the "hub-and-spoke" star shape network topology and proposes a new way of key distribution in which key pairs would be constructed for communication of each pair, as explained earlier in section 2.3.5. Instead of key management layer, nodes have a trusted authority or a "hub" which provides them with information required for calculation of the keys. Thus key management layer is different

from other projects and there is no need for key forwarding. Further more, Los Alamos has the lower data collision amongst its clients, as they are all establishing connection with the trusted authority, the center-point-of failure and weakest-link to the whole security of the system[49] of the network. The scheme also suffers from long key setup times. Nevertheless, Los Alamos implementation *"for N connected nodes to a hub you can achieve N^2 connection in application layer"* without any data collision between the clients in quantum level [HNM+13], compare this to mesh topology of SECQOC or others, there are many more connection needed between the clients to efficiently achieve N^2 connection. The amount of key consumption of SECQOC for hop-by-hop OTP encryption of the keys is very large – to calculate the exact amount, the exact network architecture and routing tables are required.

Both Los Alamos and Wuhu suffer from scalability. Consider connecting trusted nodes to allow clients from two networks to communicate, suddenly the load is unmanageable – a weakness known to the star topology, which is addressed in mesh type of networks. Although SECQOC is a mesh type network, but it faces some barriers for scalability due to its trusted "hop-by-hop" nature. It is certainly not feasible to have great number of trusted "hops". SECQOC is definitely designed for infrastructural level such as telecommunication backbone, or ATM connections as the use case samples listed in [NFV13] suggests – in these scenarios trusted "hop"s could be better justified. This also explain the network schematic in figure 2.14, where nodes are located in private network. One issue I could not find an answer for in any documents I read about the project[PPA+09][DAGS08][**?**][FDD+09], is exchanging the key between QBB nodes and the clients in their private network. It is not mentioned anywhere if the key is transfered in plain or encrypted – if encrypted how QBB nodes are sharing keys with the users in their private network. And that seems like the weakest link of the project, no matter how secure QBB nodes are exchanging the keys, an intruder could easily penetrate into local network and obtain the shared secret and the security of the whole project is jeopardized. In contrary to SECQOC use cases, projects like Los Almos has demonstrated implementation of cheap, small, and efficient quantum transmitters which could even be incorporated in a personal computer. These developments will affect the target audience and industrial level QKD could be deployed in; a transition from backbone layers towards end users.

SECQOC QBB Link principle with introducing redundancy over quantum channel both for higher key rate and more reliability was another main contribution of the project. QBB Node, however, were not highly adopted by other projects. From the illustration in figure 2.13 and real pictures of the device, they are not very cost effective. SwissQuantum

[49] If the trusted authority is corrupted or out-of-service the whole security of the network is under threat.

project demonstrated a modified version with two quantum channel connected. Another modification was reducing number of link layer modules to save cost. This was implicitly mentioned in the Tokyo QKD network as well. Tokyo QKD network, a SECQOC based network which was not listed in this document due to its similarity to the listed ones. The projects implemented BB84, BBM92, and SARG04 QKD protocols over a mesh type network of 6 nodes with distances from 1 to 90 kilometers and uses Q3P protocol suite used in SECQOC. They SwissQuantum project reported 1 minutes of post processing for each 5 minutes of quantum communications – I later demonstrate this could be further improved.

None of the implementations performs post-processing on public channel communication. The closest implementation is DARPA which uses a VPN tunnel. Actually, the secure tunnel is the application of the projects which is receiving its keys through QKD. Since the tunnel is established between the nodes, they are exploiting it as public channel. And in SECQOC variants, it happens through the point-to-point classic communication, shown in figure 2.12. This is an expensive implementation choice to have point-to-point classical channel and unnecessary as discussed earlier. SwissQuantum propose to use the same medium for quantum communication through multiplexing the data with quantum data to use the maximum bandwidth of the medium and reduce costs. Q3P protocol in this scenario sits on QuantumChannel interface instead, as depicted in figure 2.15. A comprehensive discussion on Q3P is addressed in section 4.2 where it is compared to the suggested protocol presented in this study.

Finally, authentication scheme used in the implementations are mostly Carter-Wegman style. Although it is not explicitly mentioned which algorithms they are using, it might be the case that they are using VMAC with 128 bit keys which are not secrue anymore. Implementations which suffer from low key rate, adopt commonly used model on their time e.g. DARPA used HMAC (with either MD-5 or SHA-1, due to the time of the project). Obviously, the key rate is real issue in QKD networks. Tokyo QKD network, added a feature to its key management layer which would receive feedback from key generation rate in Quantum layer and *"resize[s] the key materials for absorbing the difference in key generation rate and key length of each QKD link"*[SFI+11]. This implies when the key rate drops some keys would be fed into some XOF function or KDF (e.g. Shake or KangarooTwelve) to generate longer secrets. Another suggested approach for better key consumption is to shorten the length of messages, specially during post-processing to achieve less number of communication thus less tag generation and key consumption.

To conclude, QKD networks are suffering from high cost of implementation which makes them expensive key generators comparing to classic ones. There are many suggestion on

hardware level to reduce costs as discussed. Another issue is the considerably low key generation rates. This could also be improved by utilizing better hardware, yet consuming less key is suggested by all implementation since the consumption on post processing is rather high. Apart from employing less secure algorithm to reduce key size, shortening the length of messages communicated is also suggested. On top of these, a generic solution should update the security and can serve in different network topologies.

Chapter 3

Contribution

"If you steal from one author,
it's plagiarism;
if you steal from many,
it's research."

— *Wilson Mizner*

The state of the art exhibited in latest pieces of academic literatures and cutting edge knowledge employed in current industrial implementations have been epitomized in chapter 2 and in its last section – section 2.4 – a comprehensive analysis of them is expressed. This chapter proposes a solution which complies with the requirements set for the research explained in section 1.4, and overcomes the shortcomings mentioned for QKD. This proposal is based on sate-of-the-art suggestions on how to improve things and also proposes new approaches where there is no suggestion in literature.

In this chapter, I present an authentication scheme based on IPSec protocol suite using algorithms studied before in this document. I also propose an enhance architecture of quantum end point which deploys the authentication scheme – I label the enhanced quantum endpoint together with authentication scheme ***the solution***. Finally, a protocol is proposed for handling post processing steps of quantum key distribution. The protocol is designed based on BB84, however, the generic design principles of the protocol accommodates usage of other quantum key distributions protocols as well. Prior to these, the requirements of the solution are detailed and a general schematic of the solution's architecture is presented.

3.1 Solution Architecture

The architecture of my solution also complies with the three layer architecture common in all implementations. The highest level, application is a layer out of scope this study which

does not need specification. Application Layer (AL) could use the keys generated from QKD for any cryptographic application. Similar to other implementations, there would be API calls for Quantum Layer (QL) and AL to obtain keys from Key Management Layer (KML). In order to provide keys to other layers KML traditionally needs to: synchronize the keys, generate keys, and manage the key-in use. The idea of providing feedback about key rate to KML proposed in Tokyo implementation is also included in KML, and I take it further and illustrate how such an information could be used to obtain dynamic efficient authentication algorithm based on the rate of the key generated and quantity of shared secrets available. The focus of this study in KML is more on how KML provides secret to QL for post-processing, although other features would be discussed abstractly and recommendation on how to implement them would be suggested in related cases.

QL follows the QBB node design principle in quantum channel, namely redundancy in quantum links. However, the solution take into account that for cost saving purposes, there might be only one classical channel present for many quantum channels connected to different nodes. In QL, the target of this study is post processing, and most importantly authenticating the communication. Therefore, in coming sections, I detail an authenticated post processing communication between two nodes in the network. Adopting OSI model for communication, the solution is not restricted by the network topology of Quantum Node and could service any topology i.e. these two nodes could be two nodes from any of the mentioned networks. The adaptability of the solution on network topologies and some hypothetical use cases are discussed in next chapter.

Figure 3.1 depicts an abstract schematic of the architecture. Parts which would be detailed in this document are shown in green rectangles from both KML and QL. Both rectangles could be physically located in the same machine, something seen in most recent implementation like Tokyo QKD Network. As mentioned earlier, quantum layers of all implementations are also key consumers for post processing. Hence each quantum end point needs a key manager layer to manage the shared keys i.e. which to use where and how. In some implementations like Los Alamos, however, key management is solved in other fashions and client quantum end points could calculate the secrets and do not need synchronization between the shared keys. In those scenarios, The IPSec modules should be located on the same machine which performs post processing. This is the case for use cases where quantum endpoint are more like end user compared to infrastructural backbone nodes. IPSec provides authentication for communication and the portion of it related to the scope of this study is explained later.

Each endpoint can be connected to arbitrary number of endpoints in the network. At lowest level, arbitrary number of quantum devices are connected to other quantum nodes

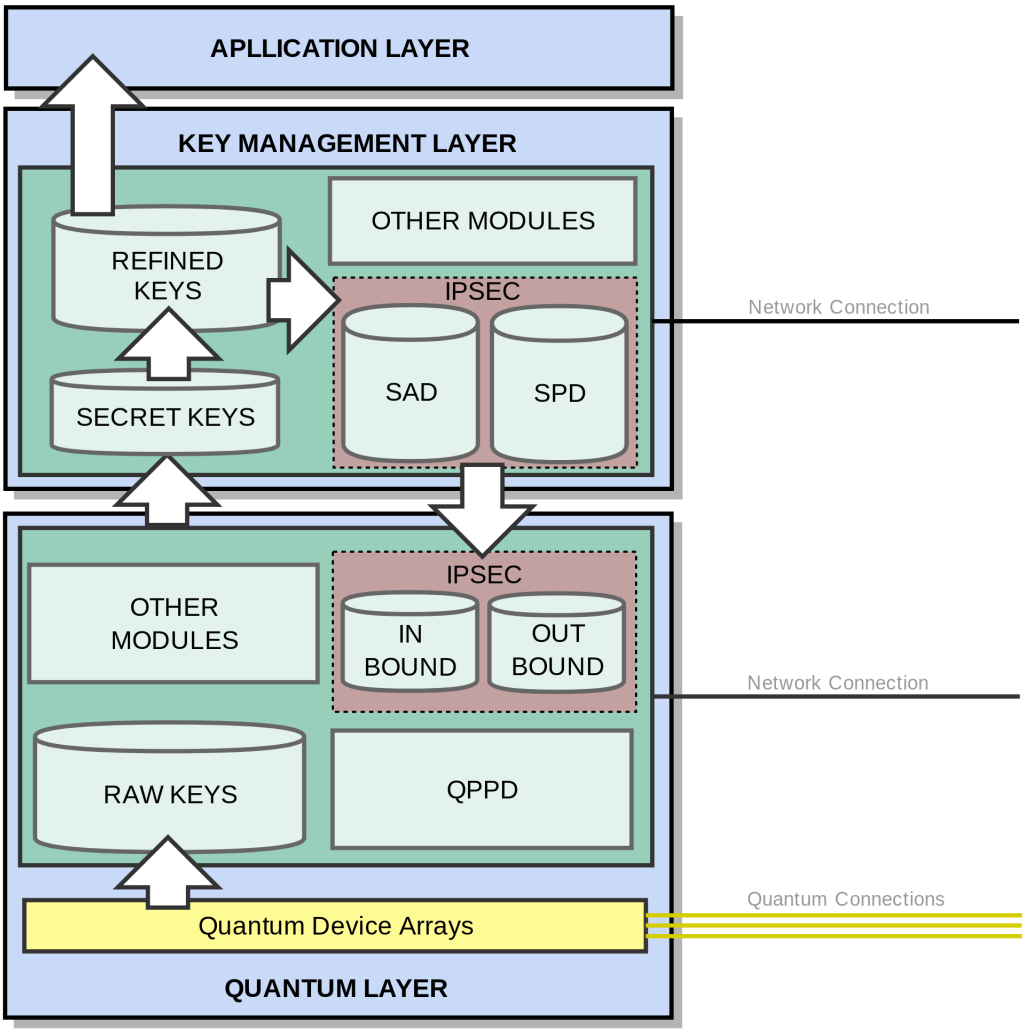

Figure 3.1: Proposed Quantum Endpoint Solution Architecture.

across the QKD network via quantum connection. It is not important if it is point-to-point connections, or connected to a quantum router like Wuhu. Each quantum connection has an ID – Quantum-device Identifier (QID) in ETSI standard derived from SECQOC project and is 4 bytes long similar to IPv4 addressing. I adopt the same QID. All quantum devices push the raw keys into RAW KEYS data base. Quantum Post Processing Daemon (QPPD), will use these raw keys and performs the post processing. Once the secret keys are derived, they would be inserted in SECRET KEY database in KML. There are two other databases in QL related to QKD. IN BOUND contains the keys for QPPD to verify authentication tags of incoming traffic. OUT BOUND includes the keys that QPPD will use to generate tag for packets it transfers. Other modules also exist in QL layer which

will not be explained in this study, to name some:

- Random number generator which feed random sequence to quantum devices both for key to transmit and choice of detection basis. Some implementations use quantum random number generators.

- A module to gather feedback from quantum communication. This module could compare the size of raw keys and the final secret key which is sent to KML to obtain quantitative feedback of quantum channel quality, and also the key generation rate. And keep track of consumption of secret key, and stretch different configuration to guarantee the system's availability.

Once keys are placed in SECRET KEYS database, KML manage the keys and insert them in REFINED KEYS database. For example secret keys could be used as initial seed to extending algorithm to stretch the key size. KML also generates keys as mentioned before. Those keys also would be placed in REFINED KEYS database. It is known which nodes in the network have the same keys in REFINED KEYS. Any application, would receive keys from REFINED KEYS database. Two other data bases exist as well; these are Security Associate Database (SAD) and SPD. They contain the secrets and policies needed by IN BOUND and OUT BOUND in QL. There also other modules in KML layer, the key one is the KEP. This is the module which creates key, forward them to other nodes, and etc. NIST competition for post quantum security with the first round of submission in 2017, already has some promising candidate[BP18]. Certainly result of the competition could help choosing the appropriate algorithm for this module. Both QL and KML need classical network connection. This is achievable in many forms, both layer could share the medium as well, or use the quantum channels. For the purpose of simplicity, they are shown separately. Quantum Connections could also be terrestrial or satellite communication.

3.2 Quantum Post Processing Daemon

Quantum Post Processing Daemon (QPPD) is the software performing the post processing. QPPD adopts TCP/IP protocol stack from OSI model for post processing communication, hence the method of authentication is TCP/IP based. We can see this adaption in other projects as well except SECQOC variants; they use Q3P protocol stack. Q3P protocol stack, apart from the first layer, is 100 percent identical to TCP/IP, the routing protocol to route Q3P packets is the one used in IP, namely OSPFv2. All other network functionalities are achieved in identical ways as well. Certainly there is no need to deviate from the standard. The comparison between our proposed protocol and Q3P is written in section 4.2.

TCP/IP protocol stack already takes care of network functioning and has taken into consideration security of the communication as well i.e. authenticity, integrity, and privacy. Furthermore, TCP enables reliable communication. Once a TCP connection is established, it makes sure the Payload Data Unit (PDU) would arrive at destination. The three way handshake to establish sequence numbers for both direction makes TCP the first choice for reliable bidirectional communication. It is common for PDUs to be fragmented if their size exceed a certain amount also known as Maximum Transfer Unit (MTU) for efficiency purposes — default IP packet MTU is 1500 bytes. The recipient TCP node would assemble the fragmentation to construct the same PDU, an identical scenario to all communication over Internet.

IP achieves security through IPSec suite. IPSec is the backbone to secure communication. RFC-4302 details Security options possible for IP using IPSec suite, and it includes Internet Protocol Security Authentication Header (AH). AH packet header is shown in figure 3.2, and it authenticates the whole IP packet, except for mutable fields – these field are subject to alteration during transition fro performance purposes. Security consideration are in place in AH and detail in the RFC e.g. sequence number to resist replay attacks. Security Parameter Index (SPI) refers to a Security Associate which contains the key and initially is stored in Security Associate Database. Later depending on the policies in Security Policy Database it is either placed in IN BOUND or OUT BOUND data base. RFC uses the secret in SAs to generate keys to use on the communication. The procedure is the same as standard IPSec. Usually the key in the SA is used as a seed for a KDF to stretch the key even further. Having the sequence number in the packets, parties can both use the appropriate number and derived key round and more importantly decided when to drop the key and use a new one. This procedure is known as re-keying. Usually implementations use time base fast re-keying, DARPA project and other suggest changing in every minute. QPPD uses KangarooTwelve for KDF. QPPD uses feedback from the key rate to stretch even further when necessary.

QPPD implements Authenticated Post Processing Protocol (APPP) to perform post process steps. APPP is a generic protocol which accommodate post processing for different QKD protocols. It is described extensively in next section. Unlike other solutions, QPPD benefits from Fast User Space Packet Processing, a technique which is been used in network industry to achieve higher packet processing performance unachievable in Kernel Space. The most well known choice is Intel Data Plane Development Kit(DPDK). This performance is necessary to be able to handle the load of post processing for all the quantum devices.

QPPD is listening on a certain TCP port for post processing requests and instantiates

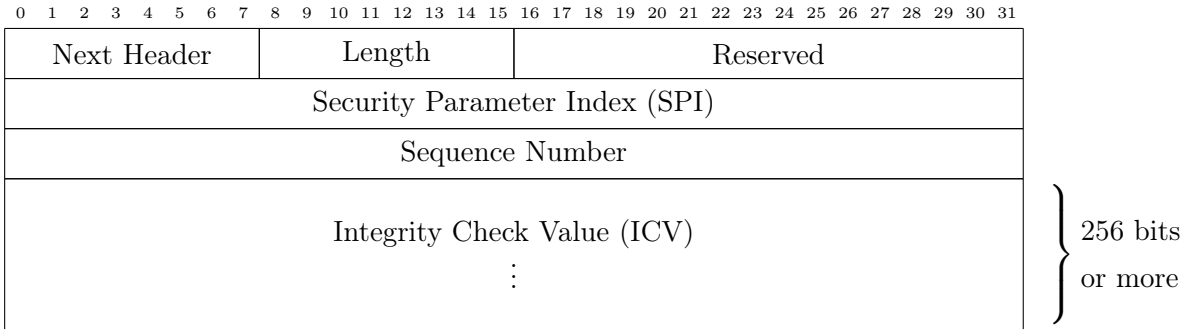

Figure 3.2: IPSec AH packet header.

a TCP connection on a different port to carry on with the actual post processing. This is explained later with the sample run in section 3.4. Protocol stack used by QPPD is shown in figure 3.3. QPPD generates tags and put them in ICV, or verify them for the incoming packets.

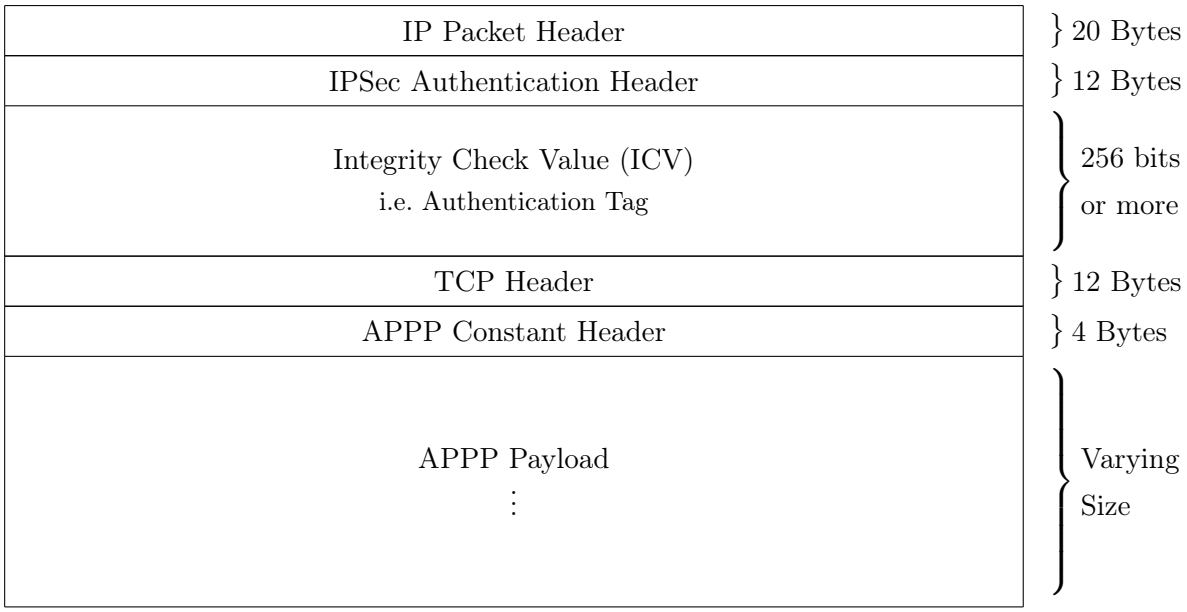

Figure 3.3: Proposed solution's protocol stack.

3.2.1 Authentication Algorithm

Lower bound security assumption of this study for authentication algorithm is that the algorithm should be post-quantum secure. Yet, the higher security bound, that is an ITS authentication algorithm, is more appealing. QPPD implements KangarooTwelve for lower bound security assumption. The algorithm could also satisfy the higher bound se-

curity assumption. The reason for this selection, on top the efficiency and sound security proofs, is the simplicity of the algorithm – it looks like a *"turn-key"*, *"plug-and-play"* algorithm. And for the higher bound assumption QPPD utilizes auth256. Security Associate, alongside with the secret key, includes choice of authentication algorithm. QPPD uses the information in the available SAs in INBOUND and OUTBOUND databases to select between the two. SAs are prepared in KML and based on the different factor decides which SAs from SAD should be available to QPPD.

3.3 Authenticated Post Processing Protocol

Authenticated Post Processing Protocol (APPP) is an application layer (layer 7) protocol in OSI model which sits on top of TCP as shown in figure 3.3. No other implementation discussed about this layer of communication. The TCP `NEXT HEADER` field specifies this payload which would be one of IANA experimentation number between `0xFD-0xFE`[1]. This way, all ordinary network devices could be used to route the packets.

QPPD implements APPP to perform post processing. A sample implementation of the protocol is given in section 3.4 of this chapter. As the name of the protocol suggest, it assume the connection is already authenticated. Indeed, following the layer concept of OSI, application layer should contain the application data. Packet authentication needs to happen in lower layer to authenticate both data and the channel and the network. This is why IPSec AH is placed at lower level and authenticates IP and TCP packets as well.

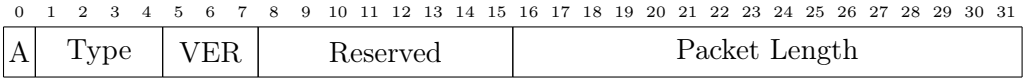

Figure 3.4: APPP constant header.

As argued before, message size is crucial in post processing communication. To achieve shortest possible size, APPP follows a dynamic size header design. Such designs could be seen in bandwidth restricted network protocols such as satellite protocols. APPP has a constant 4 bytes header illustrated in figure 3.4. Header fields are described in table 3.1. Based on the the type of packets necessary fields would be attach to the constant header. The field `A` is a flag for delayed authentication. If set the parties shall keep the messages traversed for a later authentication.

[1] RFC3692

Field	Name	Size in bits	Description
`A`	Authentication flag	1	If set, indicates delayed authentication happens on packet.
`Type`	Type Indicator	4	Specifies the type of APPP packet based on the values in table 3.2.
`VER`	Version	3	Indicate compatible implementation version of the protocol with the packet.
`Reserved`	Reserved for future use	8	This field could be exploited by the protocol developer for application specific purposes.
`Packet Length`	Length of the packet	16	Specifies packet length excluding constant header in bytes.

Table 3.1: APPP constant header fields.

3.4 Sample Run

In this section, I describe an abstract implementation version `0b000` of APPP. It should be noted that this solution could be considered as a design draft and could be subject to slight modifications based on the feedbacks from implementation. From APPP point of view it is not important what QKD protocol is used or in what sequence the post processing steps are being performed. Version `0b000`, explained here will follow original BB84 in the order discussed before. It should be noted that this implementation of APPP is detailed to be compatible with my proposed solution architecture, although the protocol could be implemented otherwise to accommodate other implementations.

Although post processing starts after quantum communication but APPP version `0b000` starts the communication before initiating quantum communication to negotiate on some parameters. This is suggested in some theoretical literature [WTC18]. After Initial Setup, quantum communication begins and parties i.e *sender* who instantiate the quantum communication and send the key qubits and the *receiver*, share the raw key. Once quantum communication is over, both parties' QPPDs collect the raw keys in RAW KEYS database, and split them into pieces called *chunks* and performs post processing on the chunks. Chunk by chunk process of raw key was proposed in SwissQuantum to perform Post Processing in parallel.

Value (in hexadecimal)	**Packet Type**
0x0	Initial Setup Packet
0x1	Sifting Step Packet
0x2	Confirmation Step Packet
0x3	Error Correction Step Packet
0x4	Final Confirmation Step Packet
0xE	Error Packet

Table 3.2: APPP version `0b000` Header Type values.

Table 3.2 demonstrates types values for APPP version `0b000`. These values would fill the type value field in the constant header of APPP. Other type numbers not listed on the table are reserved for future use. Depend on each type, additional data would be attached to the constant header. First, I explain the error packet which is sent if parties face an error. Then other steps as they would happen in a real run are demonstrated. This section does not explain Error Correction Step Packet type `0x3`. Error correction algorithm are out of scope of this study, but the same rationale used in other step happens there. Also Confirmation Step Packet type `0x2` and Final Confirmation Step Packet type `0x4` is not detailed. These steps are very to similar to Sifting Packet Type `0x1`. Indeed, in sifting one party reveals a bit sequence representing choices of detection basis, and the other confirms that. Same analogy is happening in confirmation step where part of the sifted key is revealed to calculate QBER and evaluate the possibility of intruder eavesdropping. Also final confirmation follows the same principles, but reveal and confirmation happens over the final secret key after privacy amplification step.

Figure 3.5 depicts sequence of messages sent from the first step of instantiating the request of QKD, illustrated as Pre Processing Phase, up to the end of post processing. The *sender* first sends a QKD instantiation request to *receiver*, and the *receiver* accepts the request. After quantum communication is over and raw key is obtained, both parties will split the raw key into chunks to be able to perform post processing in parallel. *Sender* will send Chunk post processing request to *receiver*. In this request *sender* will specify another TCP port she is waiting for the receiver to start the post processing. *Receiver*, will then open another TCP connection to *sender* and starts the post processing process in the new TCP connection by sending her choices of basis detection to *sender* and await confirmation on it. This is to follow original BB84 post processing as a sample. Depend on the QKD protocol used, these steps and and the content traversed could differ. For each of the messages, I have included a packet sample as well which are explained in corresponding

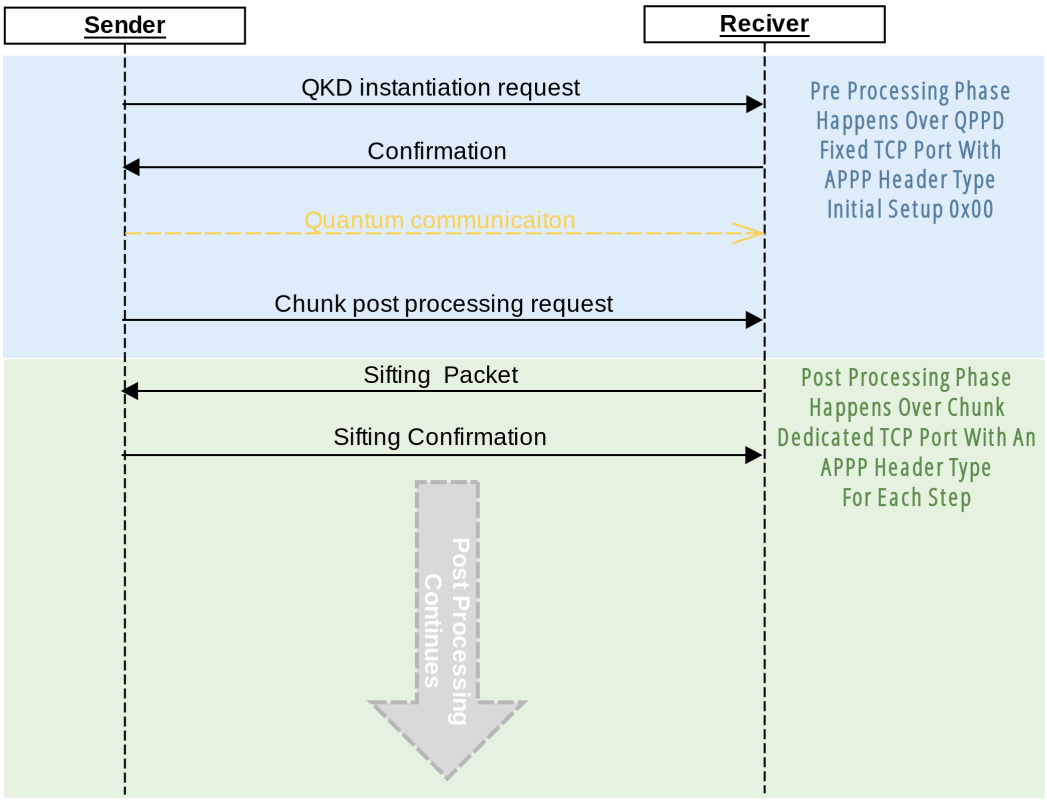

Figure 3.5: APPP version 0.0 message sequence from pre-processing to post-processing.

subsections. As mentioned before, from post processing sifting has been chosen for the purpose of demonstration. The rest of the post processing steps follow the same routine at network level.

3.4.1 Error `0xE`

Error packet header is shown in figure 3.6. In this type the reserved field in the constant header is used to show the error type. Depend on error type, additional information could be added, usually an identifier for a raw key chunk that is being processed.

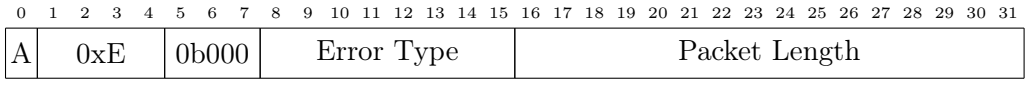

Figure 3.6: QKD Error header.

The packet could be sent payload less as well for example if an authentication tag is not working, or if a request to instantiate should be rejected. These could be sent only by

error type and without payload.

3.4.2 Initial Setup `0x0`

QPPD is listening on a constant specific TCP port for QKD instantiation request from other nodes in the network. A *sender* who wishes to start sharing key will construct a QKD instantiation request similar to the one shown in figure 3.7 and transmit that to the *receiver*. Instantiation request will have value `0xAA` in the reserve field of QKD constant header to be distinguished from other packets in this step. The other fields in the packet are self explanatory. After the constant header, we have the Raw Key Length the *sender* wishes to share. The length is given in bits representing amount of qubits she is going to transfer. After that, the sender QID which is an identification number similar to IPv4 for the quantum device. The QID shall have a point-to-point connection with the *receiver*. The Quantum-device Identifier of the device at the other end i.e. *receiver*'s QID will follows after that. Then the ID for this raw key is added. All these fields have 32 bit size. Logic for this is briefed in next chapter. Then there are four 1 byte fields to represent QID protocol *sender* wishes to use, same for the error correction algorithm, privacy amplification algorithm which would probably be a family of hash function, and number of chunks the *sender* wants to later split this raw key into and perform post processing on. Obviously there would be tables to associate these fields' values to corresponding algorithms and protocols. It needed to be studied what algorithms are going to be used and implemented before that. For those steps, this study did not perform any analysis, and this could be later defined. Therefore, the size of these last 4 fields could be modified later – maybe 1 byte is overkill.

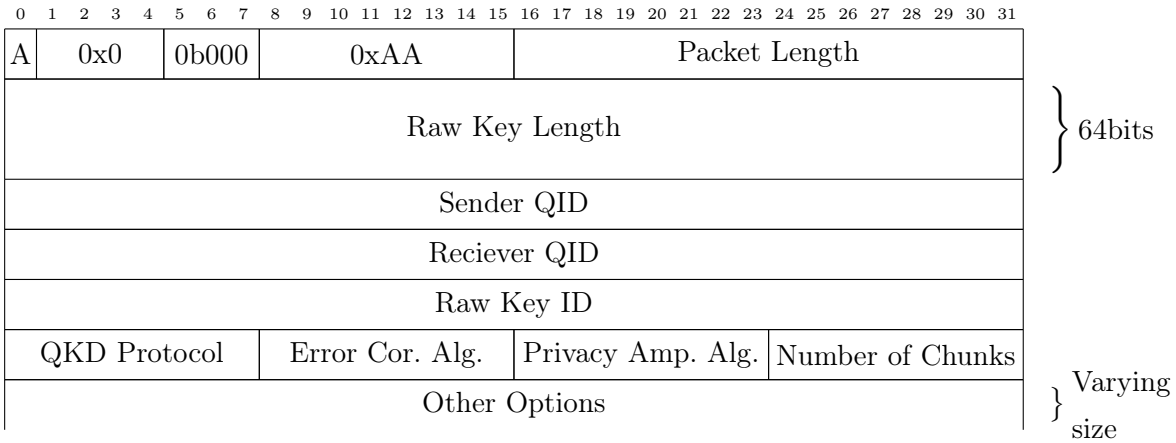

Figure 3.7: APPP Initial Setup header QKD instantiation request

Most likely more options would follow as well. To clearly define this, I need more

feedback from the project and maybe the quantum devices, to clearly specify what is needed in the initial setup. These options could include the expiry date of the raw key, accepted QBER threshold, qubit detection style (iterative or time based), nonces or tags that might be needed later e.g. if the key is going to be used for authentication later (this is to comply with projects like Los Alamos which generate specific purpose keys).

Upon receiving the QKD instantiation request, the receiver performs some check. Obviously first thing is the authentication tag. Authenticating the packets have been explained before. All the packets would follow the same instruction. Authentication in QPPD has happen before processing this packet. As stressed in APPP explanation, at this layer, protocol assumes the messages are authenticated, except the `A` flag is set[2]. Then some error checks would be perform for example to *receiver* would check for correction of the quantum connection with the QIDs given, if its QID is connected to the requested QID. Other checks could also be performed such as system load and if it can handle new protocol run i.e. if it is not overloaded. Future study and feedbacks from implementation could spot light on this. Based on the output of the check the receiver either accept or rejects the request.

I have not included the APPP confirmation packet because I have no knowledge about how to instantiate the connection on quantum channel. Maybe a constant header payloadless with its Reserve field set to another value e.g. `0x88` would suffice, or maybe there is need to send more data. Even it might be possible to send a pulse over quantum channel to confirm, in order to save consuming the key by less communication over public channel. In case the request is rejected a payload less error packet could be sent. Depending one the reserve byte value sender could realize the reason. In case the request is accepted by receiver, both QPPDs on both end will notify the quantum devices to start the quantum communication. Means of communication between QPPD and quantum device is out of interested of this study and out of scope this research. What I assume is QPPDs on both end will receive the RAW KEYS after the end of the quantum channel which they could correlate with the exact request.

While the communication on quantum end is happening, both QPPDs will start calculating Chunk IDs. The procedure is with the help of some XOF or KDF. My suggestion is construct the string I call ***"RAW KEY TAG"*** as follows: *[Raw Key Length in bits // Sender QID // Receiver QID // Raw key ID]* which is 128bits and put it as the capacity of the sate of KangarooTwelve and take the first 32 bits as Chunk ID and the next 32 bits of the same output block as Chunk hash ID. KANGAROOTWELVE output is 1600

[2]If the flag is set, then there is need for more fields in the first QKD instantiation request to specify what type of delayed authentication shall happen and what algorithm to use and etc., QPPD then will keep a copy of the packets to perform authentication on later.

bits in each round and we are taking the first 64 bits of each round. Chunks are sequential and both QPPDs should calculate the same Chunk ID and Chunk Hash ID. Obviously security is not the issue here, the reason for using this algorithm is the collision resistancy property. Reasons for this suggestion are argued in discussion chapter.

Once the communication is finished over the quantum channel, and QPPDs are presented with the raw keys, they can start splitting them into chunk and correlate them with appropriate chunk ID. At this point *sender* constructs the Chunk Processing packet and send it to *receiver* to start the post processing on the chunk of raw key. The packet header is illustrated in figure 3.8. The reserve field in the constant header is set to `0xBB` to be able to differ the packet.

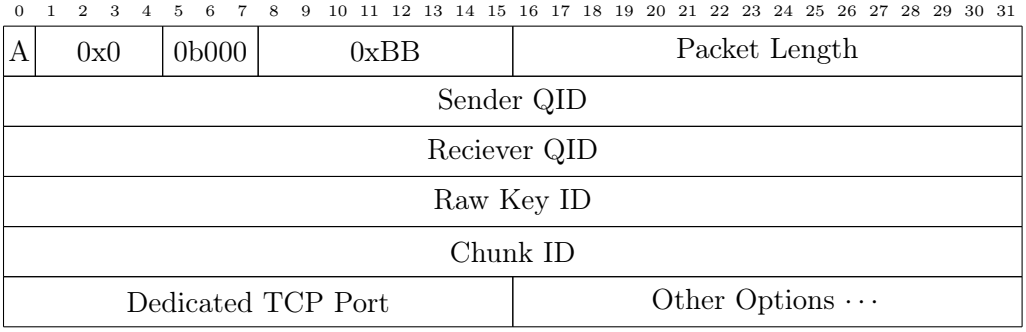

Figure 3.8: APPP Initial Setup header chunk post processing request.

Sender in this packet select the chunk she wish to perform post processing on and fill the packet header's according IDs. After that *sender* specifies the TCP port number she is expecting the receiver to start the post processing this chunk on. This is similar to the approach of many network application where dedicated port is listening for request and data transmission happens over other ports. This way the main port is always available to many nodes as the communications are short for requests unlike data transmission.

The constant header of APPP during post processing will always be 8 bytes shown in figure 3.9. Reserved field in most of the post processing act as an counter. This could come handy in error correction specially where number of packets could be more than other steps and gives QPPD a better chance organizing packets. Also in case of delayed authentication, packet sequence could be constructed simpler without tracing IP packets, and just by following packet type field and counter in the APPP header.

The Chunk Hash ID is the identifier which is going to identify this chunk in the communication over the dedicated port for this post processing. Each TCP connection could be viewed as a virtual tunnel between parties. More than one chunk post processing could

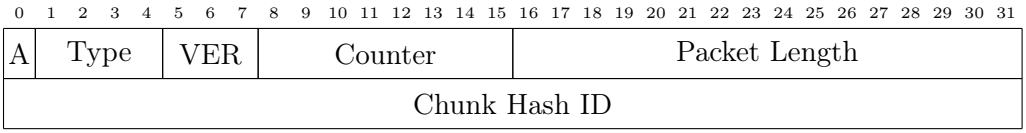

Figure 3.9: APPP Post Processing constant header.

happen in each TCP connection or tunnel. Within each tunnel the chunks are known by their Hash ID. Since there are less chunks in each tunnel comparing to what exist in RAW KEY database, the probability of the collision over 4 bytes is reasonable. The real identifier for raw chunks in the domain of QPPD is much longer than 4 bytes, however. This allow us to reduce the size of the messages transfered e.g. ID reduces from at least 256 bits to 32 bits. Outside the tunnel and in general chunks could be distinguished by the string I name ***SECRET KEY TAG***, at least 256bits (in case of IPv4) long and constructed as follows: *[Sender IP || Receiver IP || Sender TCP Port || Receiver TCP Port || RAW KEY TAG || Chunk ID || Chunk Hash ID]*. Obviously these could be fields in RAW KEYS database which would be filled by QPPDs upon receiving the raw keys from the quantum devices. Another thread could look into the database and pick the chunks its QPPD was receiver for and construct the chunk post processing request. One module could keep track of open TCP connection and perform a very simple load balancing to make sure post processing on chunks are performed efficiently.

3.4.3 Sifting `0x1`

Once *sender* sends the Chunk Process request, it starts listening on the mentioned TCP port. Upon *receiver* receives chunk request packet, it perform some check, like if the chunk is available, if QPPD is not overloaded, and etc. then decides to accept or reject the request. If she decides to reject she will send an error packet similar to what explained before. Otherwise, *receiver* constructs the packet called Sifting showed in figure 3.10.

After the Post Processing constant header, *receiver* will add number of detection she missed in Sifting Packet, these are lost qubits. This could be used by *sender* to decide if she want to continue with the post processing or if the number of errors are high. One could argue that the *receiver* which is constructing the packet could decide as well, based on parameters exchange in initial setup, and we could save sending one packet. But I still have this and give this decision to the *sender* who instantiated the process to decide terminate it. Beside this size could be useful for both parties at implementation level.

Following the number of missed detection, the *receiver* will put her choice of basis

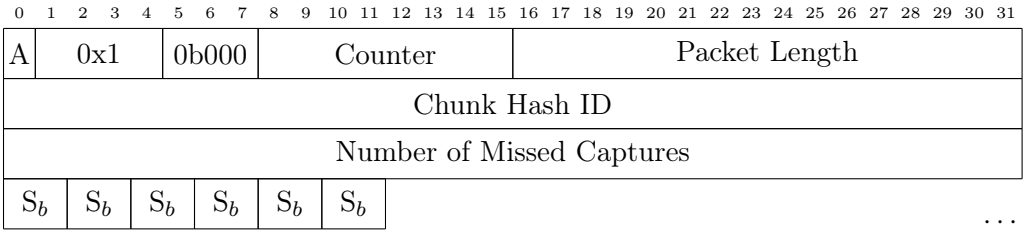

Figure 3.10: APPP Post Processing Sifting packet.

Value (in bits)	**Representation**
0b00	Reserved/Not Used
0b01	Not Capture
0b10	Rectilinear
0b11	Diagonal

Table 3.3: S_b values.

detection. Here I propose a very simple way by fast encoding the basis detection string to S_b values and that is to perpend each bit in the sifting bit string with either a zero or one, in case of missed match and detected respectively. The values for S_b are shown in table 3.3. This method is not efficient in size though as it doubles the size of detection bit string. There are compression algorithm and other size reducing techniques that could be applied. The focus of this study is, however, on efficiency of the header of the packets rather than payload. Once the packet is constructed, and she opens a TCP connection, if there is already no open connection with *sender* on the same port, and sends the packet to *sender*.

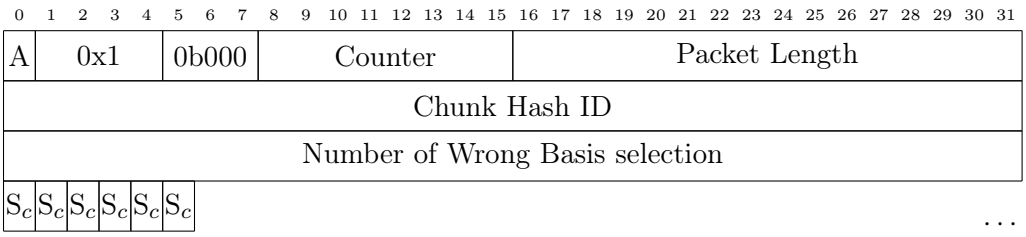

Figure 3.11: APPP Post Processing Sifting Confirmation.

Sender will process the S_bs to confirm the correct ones and construct Sifting Confirmation packet shown in figure 3.11 where after APPP post processing constant header,

sender will put the number of wrong detection, and then values of S_c to confirm which basis detection was correct and which one was wrong, so both parties could deduce the sifted key.

It should be easy at this point to have a rough idea on how the rest of the communication on post processing would look like and how the packets are constructed using APPP. After post processing, chunks of raw key are transformed to secret keys which would be provided to KML.

3.5 Key Management Layer

Once the post processing is done, secret keys are provided to KML. The purpose of whole system could be seen as filling the REFINED KEYS database. These are the keys which are either provided to application layer, or used by QPPD for classic communication authentication, or even used by KML itself. On top of the secret keys from QL, KML could receive some feedbacks about the quality of the connection on quantum level and rate of key generation, and use these information to efficiently use the secret keys. Secret key size are fixed, known and configurable because all of them had gone through privacy amplification step and the step's function output is set to a desirable length, usually 256 devisable. Secret keys in all the development go through a refinement process in which the length of the secret keys are stretched ti have more keys. Based on the feedback, KML could extend the derived key lengths accordingly to maintain secure communication and provide availability. This could also be used for filling SAD and SPD databases of IPSec. As mentioned earlier, SA is an entity which contains a secret key and other information associated with the key, like the algorithm this key should be used with, and SPI of the SA that should be expected on the AH packet header.

KML consumes refined keys to create SA and then places them in SAD. Based on the feedback it can decide on the policies that these SA should be used as well. SPD indicates which SA with which policies shall be applied on what traffic e.g. whether to be used for INBOUND or OUTBOUND, or to be used for which packets based on TCP/IP address and port. These policies could also be set based on the feedback. Consider the key generation rate is dropping, the KML then creates more keys with one refined key with help of KANGAROOTWELVE and fills SAD with keys for less consuming algorithms like KMAC. When the key rate is satisfactory, it can roll back and use Carter-Wegman ITS style auth256 which consume twice the key to provide better security. The same analogy could happen for the application layer. Recently a QKD implementation demonstrate the

same concept to reach 1 Gbps[3] encryption on Application Layer using AES with one minute re-keying policy[EWL+10].

Since quantum connections are known in quantum layer, it is also known to which nodes in the network QPPD would open a connection. Therefore, it is easy for KML to accommodate secrets shared by those nodes to SAD and later locate SAs in OUTBOUND and INBOUND databases through setting policies in SPD. These needs to be synchronized with other node's KMLs as well. They need to put the same same secret in to INBOUND and OUTBOUND used by their QPPD accordingly for the bidirectional authentication to happen.

Figure 3.12 depicts life cycle of keys within the system in abstract. It starts from the quantum layer where quantum device perform Quantum Key Distribution process to generate raw keys. Then either QPPD or the device will put them in the RAW KEYS database[4]. Raw keys are tagged as specified before, and would be splitted into chunks and tagged as explained before. Chunks of raw keys would go through post processing so secret keys could be derived from them. The processes the key goes through post processing are described before and shown in figure 2.1. These secret keys, then, would be placed at SECRET KEYS database in KML. This is the last step QPPD is involved with the keys. Secret key tags, as they are constructed now, could also be exploited by KML as meta data for routing, and generating SAs for IPSec.

Secret Key Tags include the address for the parties who has shared the key. KML then picks these secret keys and creates refined keys to put in REFINED KEYS database. Refinement process for the secret keys is out of the scope of this study. A suggestion was briefly discussed earlier. The purpose of these process is to create 256bit keys which would later be provided to consumers. It should be restated that refined keys are not only generated from QKD secret keys. KML can generate keys and share with other nodes on its own as well. For those communication KML uses refined keys to encrypt a packet and encapsulate the key and send to other nodes, like run zero of QKD when the nodes have not started the QKD yet. Another example is synchronization needed for IPSec secrets, or other management and synchronization communication needed in this layer for expiring keys or deleting keys and etc. SECQOC and it variants are using OTP to encrypt these packets. This is not very efficient, while in the most of the cases just encrypting the key with OTP suffices and reduces key consumption to only 256 bits instead of more than 100 bytes per packet. Network application layer implementation for KML could be used as

[3]Giga bit per second.

[4]Communication between Quantum Device and QPPD, as discussed before, could be detailed once I get my hands on one or a technical document of their implemented API

well, where encryptions happen over some parts of layer 7 e.g. the key being forwarded.

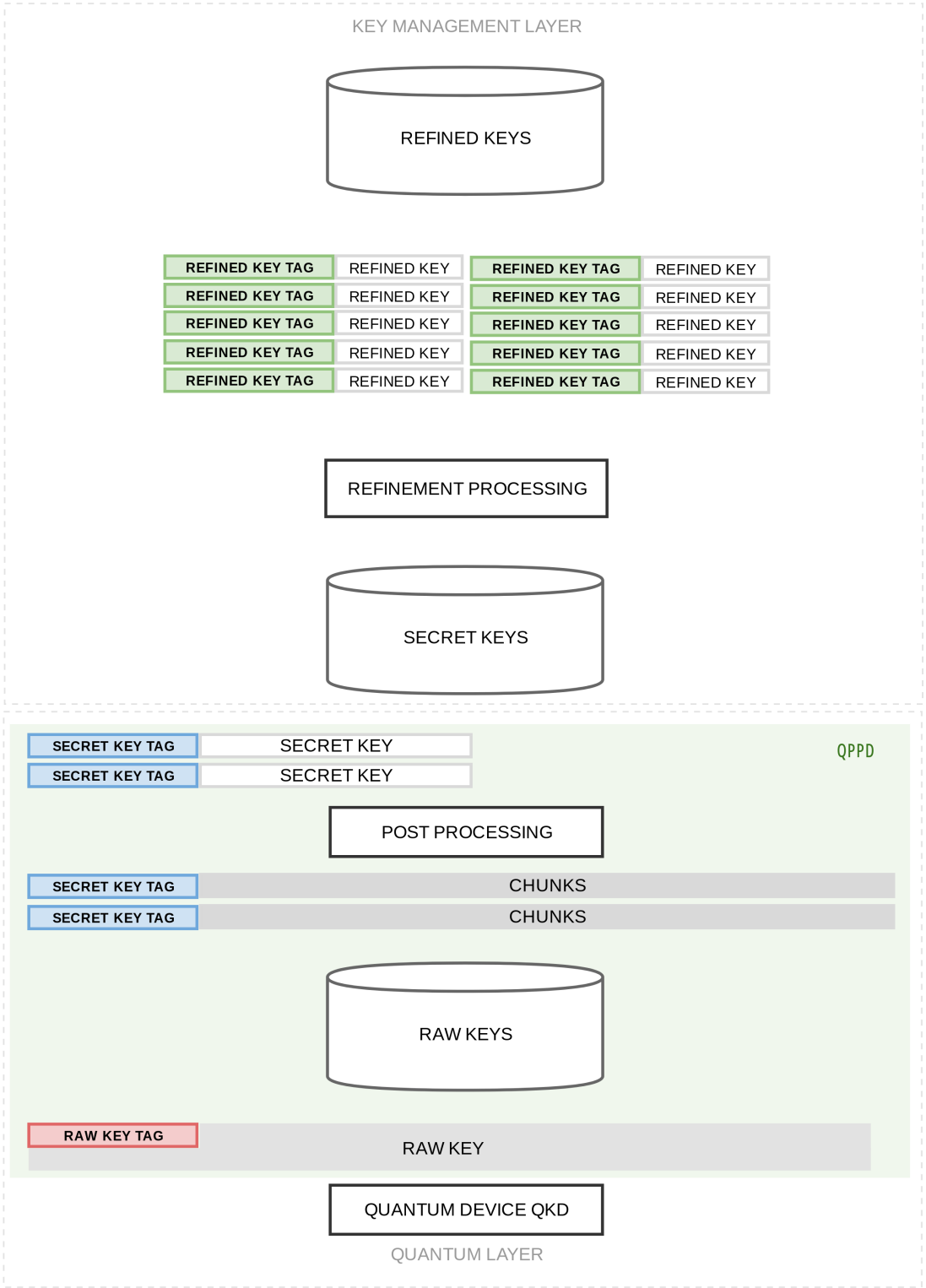

Figure 3.12: Keys transformation though the system.

Chapter 4

Discussion

"The aim of argument,
or of discussion,
should not be victory,
but progress."

— *Joseph Joubert*

On the final chapter of this dissertation, I provide the rationale behind the proposed solution. The main theme of reasoning behind most of the decision making through out this project was to follow the standards and try to build on top of what has already been built. The purpose of the design was to stay as generic as possible to be able to service different needs, and also update the gap of over 10 years in many sections like implementation technology and security. The design philosophy would be discussed over three aspects of the solution namely: architecture, proposed authentication scheme, and the APPP protocol. The design philosophy is more than reasons behind selecting of an approach or an algorithm – it is about the idea which resulted in having certain criteria of selection which ended up choosing an algorithm. Subsequently, I provide a fair comparison of the solution with existing ones. And at the end, I deduce the conclusion of this study.

4.1 Solution Design Philosophy

One of the main aspect that shaped my solution was following the technology trend and try to apply it to QKD where applicable. In the literature I found there was at least a 7 years gap. The last QKD network implementation paper I found was published in 2011 for Tokyo QKD network. There has been lots of progress recently in QKD domain though, with field test for 307km fiber distance QKD [KLH+17] or 1200km over satellite[YCL+17]. These studies and many others are focused on the quantum communication, apparatus,

and techniques in order to perform more efficient communication over quantum channel. Most of these papers do not even consider post processing. It feels like post processing has been neglected. Maybe one reason for this is that due to still poor performance on the quantum channel comparing to the price of the appliances, the need for enhancement in those area is greater.

Another thing about outdated implementations was the security concerns. Most of the QKD network implementations were not explicit about choice of algorithm used for authentication. Those who mentioned their choice did not comply with this study's security bounds, like SHA-1 in DARPA. This is very important to note, VMAC with AES 128 bits is a wide in-use ITS Carter-Wegman MAC generator but it is not post quantum secure. Because the secret key (128 bits) would not provide 128bits security but instead 64 bits of security in post quantum era. A quick fix to this could be enlarging the size of the key, yet I included examples that there are some attacks on the construction of algorithms found recently. Thus, I decided to take a step back and look for new findings that might have been introduced during these years. Moreover, there were some parts of the whole QKD which was not well defined in the literature like the application layer of QKD classical communication (APPP).

One of the key thing in designing a system is to understand the load the system will face. Unfortunately, I did not have any statistics on key generation rate to grasp the load of the system. Therefore, I performed educated guessing to define system load. The device that was considered for this study claimed generating 3kbps secret keys. The highest key generation rate I found in the literature was 26.2 Mbps at the highest [ILC+17]. Assuming half of the key is revealed in post processing confirmation step this brings this size to 52.4 Mbps. And half of the raw key most probably has been sifted and there is packet loss as well so it is safe to assume the raw key generation rate was near 110 Mbps or 13.75 MBps. If we consider each packet sent in network to be an average of 1000 bytes this mean only for one party to reveal her choice of detection base which is as long as the key size, 13750 packets per second are needed. These results are obviously the worst case scenario. As mentioned earlier it is suggested to use compression algorithm on post processing data to shorten message length.

The other fact to consider is number of keys generated in each node. In KML layer ETSI suggests space of 512 bits to identify all the keys in the network. In the quantum layer, however, there is no suggestion. A recent study claims that in order to achieve 1 Gbps encryption at encryption level with 1 minute re-keying policy for AES not OTP, 8.6 Gbps key generation rate is required[EWL+10]. This implies both long length raw keys and huge number of them for just one encryption link. Clearly, quantum endpoint should

be connected to more than one node to construct a network. And this enlarges the number of keys generated in each end point, thus a long ID size is needed.

4.1.1 Architecture

I based my designed on SECQOC, and applied the suggested enhancement for cost reduction, namely aggregated quantum links, WDM, lowering the number of link layer module to preferably one personal computer (the one that hosts QPPD in my solution). For authentication I had to add the IPSec into the three layer design as my protocol stack was different from the one suggested in SECQOC. Suggestions like getting feedback from lower quantum level to upper key management level is suggested in related literature for extracting more/less keys based on the quality of the quantum channel. I tried to have it in my design for the choosing more efficient authentication algorithm as well. Key processing and key tagging is derived from network packet processing application where metadata about the connection is passed to higher level for further use. This reduce inter communication between layers which is both a good design practice and an efficient solution. Suggestion for use of Fast User Space implementation is also from network programming domain which is the industry standard nowadays.

Recommending one QPPD for all the quantum devices is solely a cost effective decision and could be wrong based on the load. This is very dependent to the implementation and benchmark that could be achieve during tests. Another thing is the load which was discussed earlier. The main reason I suggested software based development was the great performance of software base packet processing project like Intel DPDK which are enable 10Gbps packet processing for network security purposes which often are time consuming, like a behavioral detection intrusion detection system or next generation firewall. This also need to be put in test. Huge number of keys generated each second also was the reason to use long key tags to avoid any collision.

However, long key tag meant longer packets. In order to reduce the size of the packets, I decide to add Chunk Hash ID concept and together with the TCP connection the post processing is being done in, I was able to reduce the key ID size which further reduces the packet size with out risking collision on the short key size ID. As I calculate 13750 packets are *only* sent by one party in a second for just one step of post processing. Reducing only one byte from the packet size results in 110,000 bit shorter messages per second only in one step of post processing performed by one party. Suggested solution at least has reduce the size of packets by 28 bytes during post processing of a chunk (32 bits instead of 256 bits). Key tagging is not discussed in other literature.

QPPD name has been selected modestly, like APPP. In fact, both of them are capable

of performing more than post processing and this was considered in their design. Through out the previous chapter, I hinted out some of possibilities, but for this study I stick with the name. QPPD is more like a QKD daemon. Its concept of functionality is very similar to other networking application, but it was not discussed in related QKD literature.

4.1.2 Authentication

Given the communication for post procesing happens over classical channel, avoiding TCP/IP would not be reasonable. All the implementations are using same TCP/IP[1]. The best authentication method known for TCP/IP is IPSec. This made selection of authentication method the easiest as all others were using IPSec as well. SECQOC Q3P is also using very similar techniques. Although they are calling their protocol Q3P, but the protocol stack is very similar to tunneling mode of IPSec. IPSec has kernel daemon but none of the implemented algorithm by the standard complies with our security requirement. This is why I proposed two different algorithms. Section 2.4 discusses the reasons for selection of the algorithms.

4.1.3 APPP

APPP was designed to perform post processing of generic QKD protocol. I tried to use only mandatory field to reduce the size as much as possible. The protocol header and the fields has been introduce in this document for the first time. I definitely could not follow any principle from related QKD research. This is why I had to improvise and adopt techniques from similar communications where bandwidth is expensive.

4.2 Solution Comparison

At architectural level, the solution is definitely an improvement over the already implemented networks based on the information published about those project. This is easy to justify. I have picked the enhancement and modification suggested by literature and further extend it. An example is introducing dynamic authentication algorithms based on feedback from quantum channel quality. While the idea was suggested to use the feedback to extend the secret key more, I suggest change of algorithm to a more efficient one. Other example of this small modification are stated through out the text in previous chapter.

Comparing authentication proposed in solution with others would not be justifiable though. At algorithm security level, all their implementation were consider some how safe

[1] Q3P might have different name for networking and application data layers but the packet structure, and functionalities are identical.

during their time of development, while now the same algorithms are not considered secure. About the approach for authenticating the packets, all the development are almost using the same technology i.e. IPSec. And about the performance, my solution is very fast theoretically. It needs to be implemented and tested properly against other to conclude. Other implementation does not reveal any benchmark from key rate generation or computational costs.

It is not possible to perform comparison for APPP either, since none of the implementation and any other literature has detailed this level. The efficiency of the design has been already discussed. The whole protocol stack used by QPPD, however, could be compared with SECQOC Q3P as both protocol as both protocol stacks are link layer (layer 2) payload. Q3P protocol stack is shown in figure 2.15. QKDNL and QKDTL are identical to TCP/IP, and header for QKDLL is shown in figure 2.16. There is no information available about QKDAL. My proposed IPSec based solution protocol stack could be seen in figure 3.3. The comparison is size based. Since information about the application layer is note provided for Q3P, I do not compare that layer, also the size of authentication tag is not considered as it is a security concern and both protocol allow varying size of authentication tag.

Comparing both protocol running on conventional Ethernet based network which runs TCP/IP is not fair because my proposed solution will easily outperform Q3P. Shown in figure 2.15, Q3P sits on top of TCP. A quick math will show us that from the MTU 36 bytes are used for TCP/IP[2] in both protocols and later again the similar 36 bytes are used by Q3P with QKDNL and QKDTL. Authentication header in my proposal is 12 bytes while Q3P which acts as the same authentication header in case used only for authentication is 16 bytes long. Not considering the authentication tag for each protocol stack, my proposal has reached application layer after 48 bytes, while in Q3P after 52 bytes, then again there 32 bytes of TCP/IP like packets for routing which makes almost doubles the size compare to my solution; thus 84 bytes for the same functionality.

Q3P, in my opinion, has been design to be used only over the Quantum Channel Interface i.e. when data is demultiplex-ed with quantum communication over quantum channel. The protocol as mentioned before has three modes of operation based of `EA` flags in the header to authenticate and/or encrypt the whole data. I am going to compare my solution to Q3P in all three modes over this type of medium where packets are not carried by Ethernet protocol. Since QKDNL and QKDTL are identical to TCP/IP and both are used once in both solution I take them out of comparison as well. Thus we are left with Q3P and IPSec AH. Q3P claims to be a link layer protocol, which is true but not accurate.

[2]IP header is 20 bytes and TCP is 16.

It is located in a place that a Link Layer protocol should be placed, but the packet does not provide functionality of a link layer protocol namely linking the connection. This is why on classic Ethernet which many devices are interconnected and can communicated with each other, Q3P cannot establish a link and substitute Ethernet – the packet header does not have any address field. In SECQOC however, there is no need for Link Layer. In fact if the data is demultiplex-ed in quantum channel the link is already there, quantum channel communications are point-to-point connection links and link layer protocol are not needed in SECQOC. Hence I conclude Q3P is just an Authentication/Encryption header comparable to IPSec. The fact that in SECQOC literature and figure also it is mentioned Q3P is the link layer only for SECQOC, and the name of the protocol itself makes feel more confident about my claim. Indeed, the protocol stack is very similar to tunneling mode of IPSec which I argue later.

In case the packets are need to be authenticated only, like in post processing which is the most of the communication, my proposed solution benefiting from IPSec AH uses 4 bytes less in header. For encryption only packets are not needed in my research, I did not discuss them. These could be seen in Key Management Layer communication which is out of the scope of my thesis. My solution for encrypting packets would benefit from IPSec ESP header designed for encryption. ESP header size is half of Q3P, 8 bytes. And in case both authentication and encryption are used both solutions have 20 byte header, thus identical.

My proposed used of IPSec is in transparent mode as it can be seen in figure 3.3. IPSec ESP in transparent mode does not encrypt IP header. While Q3P also encrypts Quantum Network Layer (analogous to IP). The only reason to encrypt the network layer (IP or QKDNL) is to conceal routing information. In that case IPSec could be used in tunneling mode where the security headers (AH and ESP) would be place before IP. In conventional network this would be the payload for another IP packet, and that is why this mode is called tunnel mode. Similar to Q3P, it can sit on Quantum Channel Interface without any IP before it, providing the same identical functionality. It is clear to see the dynamic proposed protocol stack out performs static Q3P.

4.3 Conclusion

This research proposed the most *suitable* authentication infrastructure for QKD post processing compared to all the existing alternative. It is suitable because it is designed to address the real needs of QKD. Based on my findings, practical implementations were facing the below obstacles.

- High costs of implementation almost makes it irrational to consider QKD compared to classical alternatives.
- Reliability and availability of the whole system in regards to providing secret keys to application layer is a concern and could be further degraded by high rate of key consumptions of the system over classical channel.
- Security notations were not justifiable with algorithms like SHA-1. Claims like employing Carte-Wegman style does not mean anything and could even be the case that the ITS algorithm used in the projects are not post quantum secure.
- There different topologies, and since solutions are not very generic in many layers, they are not adaptable on many projects. The need for a generic total solution is felt.

These I view as the first feedback the community received from the practical QKD network implementations Which happened over the course of 2000s. In the last decade, there is no practical implementation of networks. There practical implementation of two end point connecting to each other as a proof of concept for long distance quantum communication. Certainly, there are lots of areas to enhance. Different literatures have suggested many ways to improve certain aspects of QKD, yet there is no study considering everything in a total solution.

In this research, I tried to take in all these recommendations which could enhance the system and are related to post processing and authentication. Examples are the suggestion of using feedback and how it is extended in this study, or cost reduction hints on hardware level which are considered in the architecture of the solution. For the areas where there were no suggestion, yet I felt I could contribute and suggest an efficient algorithm, I proposed new techniques e.g. APPP, QPPD, key tagging and chunk hash ID, and etc.

Without any doubt, implementation of the solution and practical test could result in slight modification of solutions. Moreover, it could reveal new sets of requirements. More information about the hardware and statistics about the protocol such key generation rate, traffic load over classic communication channel, and etc., could also be useful. There are other areas which could be investigated, some are more generic compared to others. A generic future study could studying security proofs for KangarooTwelve and suggesting ITS MAC generation algorithm. One direction that could be taken to extend this research is definitely the KML, specifically the key forwarding, and OTP encrypting the forwarding keys which were discussed. Further development of APPP and describing other tasks for QPPD which were highlighted in the text are other directions to expand this research.

Bibliography

[AB96] Rose Anderson and Eli Biham. *Two Practical and Provably Secure Block Ciphers: BEAR and LION.* Springer, England, 3rd international workshop on fast software encryption edition, 1996.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. *BLAKE2: simpler, smaller, fast as MD5.* Springer, USA, acns, volume 7954 of lecture notes in computer science, pages 119–135. edition, 2013.

[BB84] C. H. Bennett and G. Brassard. *Quantum cryptography: Public key distribution and coin tossing.* IEEE International Conference on Computers, Systems and Signal Processing, New York, volume 175, page 8 edition, 1984.

[BC14] Daniel J. Bernstein and Tung Chou. *Faster binary-field multiplication and faster binary-field macs.* Springer, USA, international workshop on selected areas in cryptography p:92-111 edition, 2014.

[BCK96] Mihir Bellare, Ran Canettiy, and Hugo Krawczykz. *Keying Hash Functions for Message Authentication.* Advances in Cryptology – Crypto 96, USA, lecture notes in computer science vol. 1109, n. koblitz ed. edition, 1996.

[BDP+16] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, Ronny Van Keer, and Benoit Viguier. *KangarooTwelve: fast hashing based on Keccak-p.* Cryptology ePrint Archive, 2016.

[BDPV07] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *Sponge Functions.* NIST, UK, ecrypt hash workshop 2007 edition, 2007.

[BDPV08] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *On the Indifferentiability of the Sponge Construction.* Springer, USA, annual international conference on the theory and applications of cryptographic techniques p:181-197 edition, 2008.

[BDPV11] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *Cryptographic sponge functions.* KECCAK Family, 2011.

[BDPV13] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *SAKURA: a flexible coding for tree hashing.* Cryptology ePrint Archive, report 2013/231 edition, 2013.

[BDPV14] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. *The making of KECCAK.* Cryptologia, 38(1):26-60 edition, 2014.

[Bel15] Mihir Bellare. *New proofs for NMAC and HMAC: security without collision resistance.* Springer, USA, ajournal of cryptology 4 volume 4 p:844-878 edition, 2015.

[Ber05a] Daniel J Bernstein. *Cache-timing attacks on AES.* Palms Princeton, Chicago, cd9faae9bd5308c440df50fc26a517b4 edition, 2005.

[Ber05b] Daniel J. Bernstein. *The Poly1305-AES message-authentication code.* NSF, USA, 0018d9551b5546d97c340e0dd8cb5750 edition, 2005.

[Ber08] Daniel J. Bernstein. *ChaCha, a variant of Salsa20.* NSF, USA, 4027b5256e17b9796842e6d0f68b0b5e edition, 2008.

[Ber10] Daniel J. Bernstein. *Quantum attacks against Blue Midnight Wish, ECHO, Fugue, Hamsi, JH, Keccak, Shabal, SHAvite-3, SIMD, and Skein.* NSF, USA, 0152ab005327cb177476138d8ca74674 edition, 2010.

[BHT18] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. *Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds.* Springer, annual international conference on the theory and applications of cryptographic techniques p:468-499 edition, 2018.

[BP18] Daniel J. Bernstein and Edoardo Persichetti. *Towards KEM Unification.* Cryptology ePrint Archive, USA, report 2018/526 ver. 20180604:211444 edition, 2018.

[BR60] R. C. Bose and D. K. Ray-Chaudhuri. *On A Class of Error Correcting Binary Group Codes.* Information and Control, North Carolina, 3 (1): 68–79, issn 0890-5401 edition, 1960.

[BR02] J. Black and P. Rogaway. *A Block-Cipher Mode of Operation for Parallelizable Message Authentication*. Springer-Verlag., USA, advances in cryptology – eurocrypt '02 edition, 2002.

[CW81] J. Lawrence Carter and Mark N. Wegman. *New hash functions and their use in authentication and set equality*. Journal of computer and system sciences, USA, 22.3 p: 265-279 edition, 1981.

[DAGS08] Mehrdad Dianati, Romain Alléaume, Maurice Gagnaire, and Xuemin (Sherman) ShenMilton. *Architecture and protocols of the future European quantum key distribution network*. John Wiley & Sons, Ltd., Paris, France, security comm. networks. 2008; 1:57–74 edition, 2008.

[DEM15] C. Dobraunig, M. Eichlseder, and F. Mendel. *Analysis of SHA-512/224 and SHA-512/256*. Springer, USA, advances in cryptology - asiacrypt lecture notes in computer science, vol. 9453, springer, 2015, pp. 612–630 edition, 2015.

[Dwo05] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. NIST, USA, nist special publication 800-38b edition, 2005.

[Dwo07] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST, USA, nist special publication 800-38d edition, 2007.

[ECP+05] Chip Elliott, Alexander Colvin, David Pearson, Oleksiy Pikalo, John Schlafer, and Henry Yeh. *Current status of the DARPA quantum network*. International Society for Optics and Photonics, USA, quantum information and computation iii volume 5815 p:138-150 edition, 2005.

[Ell05] Chip Elliott. *The DARPA quantum network*. CRC Press, USA, quantum communications and cryptography p:91-110 edition, 2005.

[EMS14] Maria Eichlseder, Florian Mendel, and Martin Schlaffer. *Branching Heuristics in Differential Collision Search with Applications to SHA-512*. IACR Cryptology ePrint Archive, USA, 2014:302 edition, 2014.

[EWL+10] P. Eraerds, N. Walenta, M. Legré, N. Gisin, and H. Zbinden. *Quantum key distribution and 1 Gbps data encryption over a single fibre*. New Journal of Physics, Switzerland, 12 (2010) 063027 (15pp) edition, 2010.

[FDD+09] Simon Fossier, Eleni Diamanti, Thierry Debuisschert, André Villing, Rosa Tualle-Brouri, and Philippe Grangier. *Field test of a continuous-variable quantum key distribution prototype.* IOP Publishing, USA, new journal of physics vol.11 edition, 2009.

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. *The Skein Hash Function Family.* NIST, USA, version 1.3 edition, 2010.

[GLL17] Shay Gueron, Adam Langley, and Yehuda Lindell. *AES-GCM-SIV: Specification and Analysis.* IACR Cryptology ePrint Archive, 2017.

[Gro96] Lov K. Grover. *A fast quantum mechanical algorithm for database search.* STOC, Philadelphia, pages 212-219 edition, 1996.

[HNM+13] Richard J Hughes, Jane E Nordholt, Kevin P McCabe, Raymond T Newell, Charles G Peterson, and Rolando D Somma. *Network-centric quantum communications with application to critical infrastructure protection.* arXiv, USA, preprint arxiv:1305.0305 edition, 2013.

[ILC+17] Nurul T Islam, Charles Ci Wen Lim, Clinton Cahall, et al. *Provably secure and practical quantum key distribution over 307 km of optical fibre.* American Association for the Advancement of Science, USA, science advances vol.3 edition, 2017.

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. *Breaking and repairing GCM security proofs.* Springer, advances in cryptology–crypto 2012 p:31-49 edition, 2012.

[jCPB+12] Shu jen Chang, Ray Perlner, William E. Burr, Meltem S. Turan, John M. Kelsey, Souradyuti Paul, and Lawrence E. Bassham. *Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition.* NIST, USA, nistir 7896 edition, 2012.

[KjCP16] John Kelsey, Shu jen Chang, and Ray Perlner. *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash.* NIST, USA, nist special publication 800-185 edition, 2016.

[KLH+17] Boris Korzh, Charles Ci Wen Lim, Raphael Houlmann, et al. *Provably secure and practical quantum key distribution over 307 km of optical fibre.* Nature Publishing Groupe, Australia, nature photonics vol.3 edition, 2017.

[NFV10] Network Functions Virtualisation NFV. *Quantum Key Distribution (QKD); Application Interface.* New Journal of Physics, France, etsi gs qkd 004 v1. 1.1 (2010-12) edition, 2010.

[NFV13] Network Functions Virtualisation NFV. *Quantum Key Distribution; Use Cases.* New Journal of Physics, France, etsi gs nfv 001 v1. 1.1 (2013-10) edition, 2013.

[PAL+15] C. Pacher, A Abidin, T. Lorunser, M. Peev, R. Ursin, A Zeilinger, and J. Larsson. *Attacks on quantum key distribution protocols that employ non-ITS authentication.* arXiv, Austria, 1209.0365v2 edition, 2015.

[PPA+09] Momtchil Peev, Christoph Pacher, Romain Alléaume, Claudio Barreiro, Jan Bouda, W Boxleitner, Thierry Debuisschert, M Dianati, Eleni Diamanti, JF Dynes, et al. *The SECOQC quantum key distribution network in Vienna.* New Journal of Physics, Austria, iop publishing volume 11 edition, 2009.

[SBC+09] Valerio Scarani, Helle Bechmann-Pasquinucci, Nicolas Cerf, Miloslav Dušek, Norbert Lütkenhaus, and Momtchil Peev. *The security of practical quantum key distribution.* APS, Austria, reviews of modern physics volume 81, 3 p.1301 edition, 2009.

[SFI+11] Masahide Sasaki, M Fujiwara, H Ishizuka, et al. *Field test of quantum key distribution in the Tokyo QKD Network.* Optical Society of America, USA, optics express vol.19 p:10387-10409 edition, 2011.

[Sha11] Mehrdad S. Sharbaf. *Quantum Cryptography: An Emerging Technology in Network Security.* IEEE, California, 978-1-4577-1376-7/11 edition, 2011.

[SLB+11] Damien Stucki, Matthieu Legre, F Buntschu, B Clausen, Nadine Felber, Nicolas Gisin, L Henzen, Pascal Junod, Eleni Litzistorf, Patrick Monbaron, et al. *Long-term performance of the SwissQuantum quantum key distribution network in a field environment.* New Journal of Physics, Switzerland, iop publishing volume 13 edition, 2011.

[Sma16] Nigel P. Smart. *Cryptography Made Simple.* Springer, UK, isbn 978-3-319-21935-6 edition, 2016.

[WTC18] Weilong Wang, Kiyoshi Tamaki, and Marcos Curty. *Finite-key security analysis for quantum key distribution with leaky sources.* arXiv, Japan, 1803.09508v1 edition, 2018.

[XCW+09] FangXing Xu, Wei Chen, Shuang Wang, ZhenQiang Yin, Yang Zhang, Yun Liu, Zheng Zhou, YiBo Zhao, HongWei Li, Dong Liu, et al. *Field experiment on a robust hierarchical metropolitan quantum cryptography network*. Springer, China, chinese science bulletin volume 54 p:2991-2997 edition, 2009.

[YCL+17] Juan Yin, Yuan Cao, Yu-Huai Li, et al. *Satellite-based entanglement distribution over 1200 kilometers*. American Association for the Advancement of Science, Australia, science vol.356 edition, 2017.