A SYSTEM-LEVEL THROUGHPUT MODEL FOR QUANTUM KEY DISTRIBUTION

THESIS

Robert C. Cernera, Civilian

AFIT-ENG-MS-15-S-069

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-15-S-069

A SYSTEM-LEVEL THROUGHPUT MODEL FOR QUANTUM KEY DISTRIBUTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Robert C. Cernera, BS

Civilian

September 2015

AFIT-ENG-MS-15-S-069

A SYSTEM-LEVEL THROUGHPUT MODEL FOR QUANTUM KEY DISTRIBUTION

Robert C. Cernera, BS

Civilian

Committee Membership:

Douglas D. Hodson, Ph.D.
Chair

Michael R. Grimaila, Ph.D., CISM, CISSP
Member

Gerald B. Baumgartner, Ph.D.
Member

AFIT-ENG-MS-15-S-069

Abstract

Quantum Key Distribution (QKD) is an innovative technology which exploits the laws of quantum mechanics to generate and distribute shared secret keying material. QKD systems generate and distribute key by progressing through a number of distinct phases, typically in a serial manner. The purpose of this research is to identify these phases, their relationships to each other, as well as their relationship to time, memory space, computational requirements, and hardware resources. A mathematical model is developed which enables the study of critical system parameters, identifies and demonstrates potential bottlenecks that affect the overall key generation rate of serial implementations, and facilitates the analysis of design trade-offs in terms of parameters associated with specific implementations. Existing models of throughput performance make use of secure key rate equations which do not account for detailed system parameters and performance characteristics, particularly in the post-processing phases. In this research we build a model that is abstract enough to be applied to a wide range of QKD system configurations. The results of the model form an accurate prediction of throughput. The analysis contained herein provides QKD practitioners guidance in system analysis and design.

This work is dedicated to my family, for their unwavering support.

Acknowledgements

I would first like to thank my advisor, Dr. Douglas Hodson for providing me the opportunity to be part of the QKD research team and his guidance during the completion of this thesis. I would also like to thank Dr. Michael Grimaila for his unbridled enthusiasm for this project and the welfare of our team. Your patience and insights were invaluable during the monumental task of learning QKD.

To my lab partners, Major Logan Mailloux and Captain Ryan Engle, I want to thank both of your for brightening each day we worked together. Without your advice and encouragement I fear the road to graduation would have been much longer. You are both scholars and gentlemen.

I would like to give a very special thank you to Dr. Gerald Baumgartner, without whom the QKD team would not exist. Without your help and expertise, none of this would be possible. And finally, I would like to thank Howard Poston for our weekly discussions on cryptography, quantum mechanics, and information theory.

Robert C. Cernera

Table of Contents

List of Figures

List of Tables

List of Equations

Dictionary

QKD = Quantum Key Distribution

OTP = One-Time Pad cryptographic algorithm

DES = Data Encryption Standard

3DES = Triple Data Encryption Standard

AES = Advanced Encryption Standard

XOR = Exclusive-or operations

FPGA = Field-Programmable Gate Array

CPU = Central Processing Unit

GPU = Graphical Processing Unit

RSA = Rivest, Shamir, Adleman, in reference to the public-key cryptosystem

MPN = Mean Photon Number

ER = Error Reconciliation

PA = Privacy Amplification

FKG = Final Key Generation

A SYSTEM-LEVEL THROUGHPUT MODEL FOR QUANTUM KEY DISTRIBUTION

I. Introduction

1.1 Problem Statement

Quantum Key Distribution (QKD) is a method that offers theoretical unconditional security in the transmission of secure cryptographic key between two parties. As a key distribution technique, it is unique in that an eavesdropper can be detected during transmission over the quantum channel. This is made possible by employing the laws of quantum mechanics. Observation of quantum information effectively changes its state in transit, which will increase the error rate between the communicating parties. When paired with an unconditionally secure cryptosystem, such as the One-Time Pad (OTP), it possesses the potential for provable information-theoretically secure communications, with the conservative attribution of all errors to an eavesdropping adversary [1].

The performance of QKD with respect to key generation rate is determined by many factors, such as loss and inefficiency associated with photon transmission, the need to correct errors, and mechanisms to assure an eavesdropper has gained little to no information about the key. As a result, in its current state QKD systems often generate key at too low of a rate to use OTP for transmission of large data sets. The rate at which systems can generate usable keying material is highly dependent on its design and implementation. Research in the field of QKD performance has primarily been focused on improving throughput (i.e. key generation rate) by increasing the performance of individual components, protocols, algorithms, hardware, etc., of a particular system bottleneck [11, 14, 20]. Novel solutions have historically been devised for implementation-specific problems, such as alterations to system components to achieve a secret key rate at a particular distance, of which system-wide implications are not well understood [26].

General throughput equations exist that claim to give predictive assessments on final key rates, but the scarcity of input parameters into these equations makes their accuracy coarse at best. No credence is given to the monumental task of classical information processing or the time it takes to accomplish relative to quantum transmission. The problem with using the current throughput equations is two-fold: they are not comprehensive enough to offer end-to-end insight when designing new QKD systems, and offer no avenue to study or search for optimizations in systems that have already been implemented. In order to find an ideal system configuration, a throughput model that captures implementation specifics is required. This can be accomplished mathematically with a thorough and in-depth understanding of the many interdependencies between input parameters to the QKD protocol phases.

1.2 Research Purpose

Most modern implementations of QKD increase key generation rate by selecting from optimal best practices and available technology at each phase of the system. For example, during Quantum Exchange the performance of optical detectors in Bob is determined by hardware available when the system was designed. Detectors are limited by detection rates and recovery time dictated by the hardware. Similarly, Alice's ability to generate ideal single photons is currently not possible given similar hardware limitations, forcing her to attenuate pulses down to some probability of single photon generation.

QKD cannot be accomplished without the use of both a quantum channel and a classical channel. Classical processing presents its own unique problems in that both algorithms and their implementations must be selected and communication between Alice and Bob can only occur at the speed limited by latency constraints. For example, regardless of the reconciliation algorithm used (e.g., Cascade, Winnow, LDPC) and its optimized implementation (e.g., FPGA, parallel GPU), there is an upper bound on the speed at which information reconciliation can occur, given the computational complexity and the number of messages that must be passed between Alice and Bob [20]. Little research has been published in the way of optimizing the system as a whole when designed, given there are certain limiting factors which govern the overall speed of QKD.

The phases of operation of a QKD system are often processed in serial fashion: encoded photons are exchanged for some arbitrary amount of time or Bob has received an arbitrary number of photons, quantum exchange ceases and sifting begins until completion, sifting ceases and error estimation begins on the sifted key, etc. [28]. For a serial process such as this, there exist dependencies throughout the system which QKD practitioners must consider. The purpose of this research is to develop a mathematical model which incorporates system-specific performance criteria that may be used in addition to the generally accepted throughput rate equations for the enhanced study of Quantum Key Distribution systems.

This thesis seeks to answers the following research questions:

RQ1: What are the parameters that define a Quantum Key Distribution system?

RQ2: What phases exist in all Quantum Key Distribution systems?

RQ3: What are the necessary input and output parameters to define a phase?

RQ4: Can the notion of system time and performance be measured at the phase level?

RQ5: Can a system-level throughput model be developed that incorporates time, system memory, computational power, and the speed of classical communication?

RQ6: If so, how can it be used to answer fundamental performance questions of QKD such as, "How many Quantum Exchange, Error Reconciliation, and Privacy Amplification routines are necessary to achieve a desired amount final key?"

RQ7: What are the implications of altering the amount of Alice's memory allocated for Quantum Exchange?

RQ8: What are the implications of altering computational power for Alice and Bob?

RQ9: How can this model be used to study Quantum Key Distribution systems?

1.3 Research Goals

The goal of this research is to model and study end-to-end QKD systems through the buildup of a series of "mini models" at the protocol phase level. Specifically, this overarching model will incorporate performance parameters associated with quantum transmission, classical transmission, algorithmic complexity and execution, various information theory techniques, and performance of specific hardware components present in the system. The intent is to provide a useful mechanism (i.e., tool or model) for QKD practitioners in the initial design or optimization of system configurations.

In this thesis, we illuminate the various relationships, interdependencies, consequences, and implications of the numerous design decisions required to implement QKD processes, with the goal of providing the cursory understanding necessary for the QKD practitioner to design their own system.

1.4 Thesis Structure

The remainder of this thesis is organized into six chapters. Chapter II is a literature review of pertinent background information to understand Quantum Key Distribution systems and the performance trade-offs in their design and implementation. Chapter III discusses the research methodology used to answer relevant questions from Chapter I. Chapter IV provides an in-depth discussion on how the model was developed, the mathematical equations that govern it, the sequence of classical communications, and a discussion of phase implementation in practical QKD systems. In Chapter V, we generate a baseline configuration used to gauge the effectiveness of the model, as well as model outputs. In Chapter VI, outstanding research questions are answered, conclusions from this research are discussed, and future work in this area is proposed.

II. Literature Review

2.1 Overview

The purpose of this chapter is to present the necessary background to understand the research proposed in Chapter III and IV. This discussion will present topics that are not covered explicitly in Chapter III and IV. In particular, the history of cryptography will be reviewed along with the use of cryptosystems for non-confidentiality purposes. Next, the need for secret key distribution is presented along with the current practice of public-key distribution. The advent of Quantum Key Distribution and the need for it will then be examined along with its most popular protocol.

2.2 Classical Cryptography

Cryptography is the study and implementation of techniques that ensure information remains confidential. The techniques from this field can also be used to authenticate communications, apply signatures, and offer non-repudiation. We will focus on the problem of confidentiality in the form of ciphers, or cryptographic algorithms.

The goal of a cryptographic algorithm is to transform a message, or plaintext, into an unreadable and seemingly useless set of data, known as the ciphertext, by means of encryption. The intended recipient, however, possesses the knowledge to revert the transformed message to its original form and gain the ability to read it. This knowledge comes in the form of a secret key. In modern cryptosystems, the algorithm is usually a known entity, taking both the plaintext message and secret key as inputs. According to Kerchoff's principle, the real security of a cryptosystem should rely on the security of the key, rather than the secrecy of the algorithm [33].

In a large network of communicating parties, it is not practical to change the algorithm used between every pair of individuals or nodes. The same algorithm is agreed upon by all communicating parties, and each pair will instead share a secret key. The key the sender uses to encrypt messages is the same as the key used by the recipient to decrypt messages. Encrypted communication with this attribute is known as symmetric-key cryptography, in which the same key is used for both encryption and decryption. Popular examples of these algorithms used today include the Data Encryption Standard (DES), the Triple Data Encryption Standard (3DES), and the Advanced Encryption Standard (AES) [36].

There exists a single known symmetric-key algorithm that is both information-theoretically secure and offers perfect secrecy, known as the One-Time Pad. Information-theoretic security is the ability to remain unbreakable in the presence of an adversary with unlimited computing power. To have perfect secrecy, knowledge of the ciphertext must not reveal any additional knowledge of the plaintext. The One-Time Pad is relatively simple compared so other symmetric ciphers. A secret key string of bits are randomly selected, equal to the bit length of the message to be sent. The random bits are then XOR-ed to the original message and transmitted to the recipient. As long as the message receiver has the random bit string, the XOR on the ciphertext can be undone to reveal the plaintext, as shown in Figure 1.

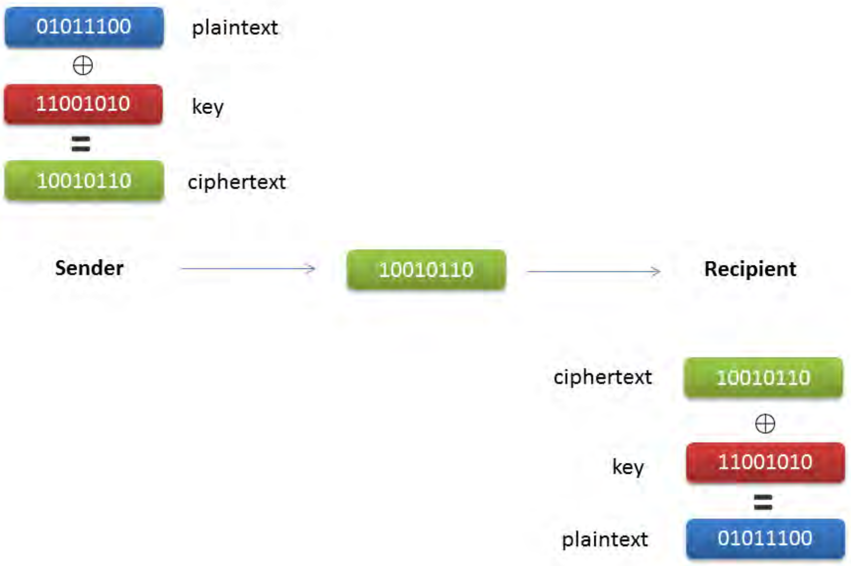


Figure 1. The One-Time Pad Algorithm

A secret key for the One-Time Pad carries several stipulations: it must be truly random, it must be as long as the message being sent, and must never be reused. This introduces two problems with its use: generating enough key to use the algorithm on large datasets and the ability to distribute the key from one party to another securely. In modern cryptosystems, a common solution for the latter issue is the use of public-key, or asymmetric algorithms, used to distribute the key.

2.3 Public-Key Distribution Methods

Public-key cryptography involves the use of two keys instead of one. The first key is known as the public key, which can encrypt messages. The second key, known as the private key, can decrypt messages. Asymmetric algorithms answer the question of whether or not it is possible to distribute a secret message (in this case the key) between two parties without ever physically meeting.

The first known instance of this technique is the Diffie-Hellman key exchange created in 1976, which is based on the calculation of discrete logarithms in a finite field [35]. Arguably the most popular asymmetric encryption scheme is the RSA algorithm, published a year later in 1977 by three postgraduates at MIT [34]. Much like Diffie-Hellman, RSA allows for the sharing of a secret key by two parties at a distance. Unlike Diffie-Hellman, however, it relies on the integer factorization problem for its security. That is, the difficulty of factoring a large number into two distinct prime factors. In computational complexity theory, both the discrete logarithm and integer factorization problems are considered difficult for a computer to solve [34]. The security of these algorithms is such that a modern supercomputer would take millions of years of brute-force searching to find a solution to the problem.

Supercomputing technology has made rapid advances in the last several decades. In addition, the advent of research into quantum computers has posed a threat to security by means of mathematical complexity. In fact, an algorithm has already been written for a quantum computer that would allow for finding a solution to the integer factorization and discrete logarithm problems in polynomial time, should a quantum computer come into existence [30]. In order to combat this threat, research in the field is being conducted to leverage the laws of quantum mechanics as a means for secure key distribution. If possible, not only would it break the reliance on the security of unsolved problems in mathematics and computer science, but it would allow for an efficient means of generating secret keying material. As a result, information-theoretically secure communications may be possible through the use of the One-Time Pad and Quantum Key Distribution.

2.4 Quantum Key Distribution

Quantum Key Distribution (QKD) is a technique that allows two communicating parties, normally referred to as Alice and Bob in the literature, to decide on a secret key between them. Once the secret key has been agreed upon, Alice and Bob can use it in any of the aforementioned symmetric algorithms as well as any others. It should be noted that QKD is not in itself a form of encryption, but merely a means of generating and distributing (i.e., growing) key. The goal of QKD is to guarantee and maintain the secrecy of the key.

Quantum Key Distribution relies on the laws of quantum mechanics. In particular, QKD relies on the Heisenberg Uncertainty Principle (or Principle of Indeterminacy), entanglement, Schrödinger's paradox, and the no cloning theorem [33]. The Principle of Indeterminacy states that no pair of the attributes positions, energy, or time can be measured simultaneously on an object. In quantum entanglement, the physical properties of particle pairs or groups of particles are correlated – the quantum state of each particle cannot be described independently. Schrödinger's paradox states that the observation of a quantum state collapses it, and the no cloning theorem forbids perfectly copying any unknown quantum states.

In classical communications, information is transmitted as bit values of 1 or 0. In quantum communications a quantum bit, or qubit, can be transmitted (or represented) as a 1, 0, or superpositions of a 1 or 0. Quantum Key Distribution represents these qubits as photons, or particles of light, which obey the rules of quantum mechanics previously discussed. They can be transmitted over terrestrial fiber optic cable or over an open-air free-space channel. These principles of quantum behavior allow for the detection of the presence of an eavesdropper attempting to learn information about the secret key, which is not feasible in classical cryptography.

2.4.1 QKD Protocols

The advent of Quantum Key Distribution began with Stephen Wiesner, who proposed the idea of encoding information on polarized photons using two conjugate bases in the late 1960s [2]. Almost two decades later in 1984, Charles Bennett and Gilles Brassard used this concept to develop the first QKD protocol, known as “BB84,” to generate shared secret keying material between two parties [3]. Current research in the field of QKD attempts to build or propose practical key distribution configurations and demonstrate their use in networked key sharing configurations.

In the BB84 polarization-based prepare and measure QKD system, the sender “Alice” prepares quantum bits (qubits) in one of four polarization states: $|\leftrightarrow\rangle$, $|\updownarrow\rangle$, $|\swarrow\!\!\!\nearrow\rangle$, or $|\nwarrow\!\!\!\searrow\rangle$. These qubits are encoded according to a randomly selected basis (“rectilinear” ⊕ or “diagonal” ⊗) and bit value (0 or 1) and sent over a “quantum channel” where they are measured by the receiver “Bob.” The quantum channel is used for photon transmission, and can be either a dedicated optical fiber or a free-space optical communication channel, as previously mentioned. A bit can be represented in one of two polarization bases: rectilinear and diagonal, with the bit value of 1 encoded as 135° and 90°, while 0 is encoded as 45° and 0°, for example [19].

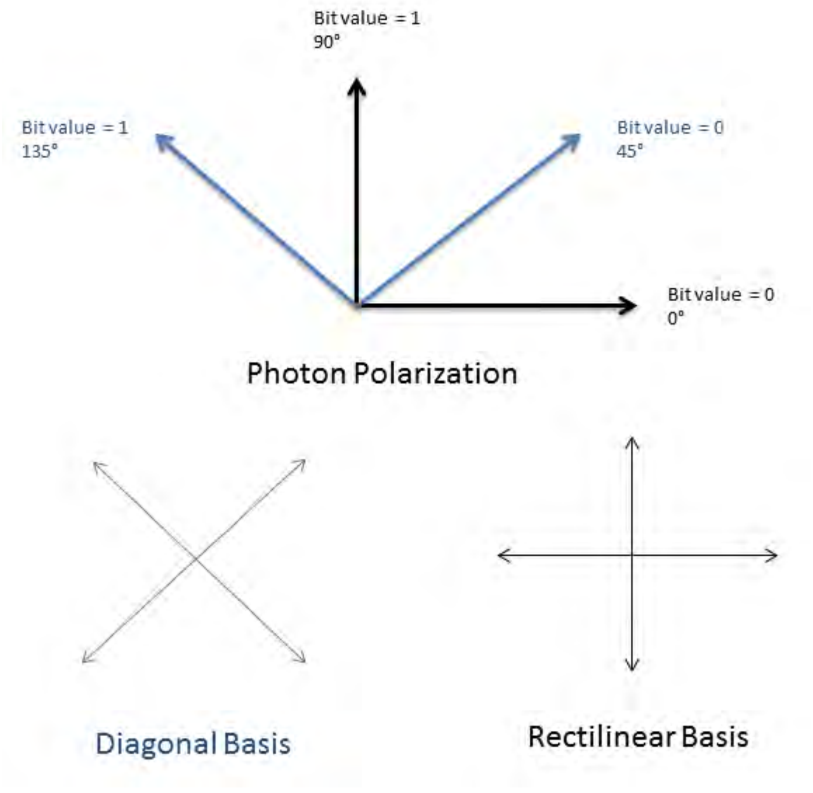


Figure 2. Photon Polarization [19]

Alice will randomly choose a bit and basis measurement. She will transmit one photon to Bob for each bit. Bob will then randomly choose a basis in which to measure the polarization of any photon he receives during quantum transmission. If Bob chooses the same basis as Alice, then both she and Bob should end up with corresponding bit values. If he chooses the wrong basis, however, the resultant bit value will be random. Measuring an encoded qubit disturbs its quantum state, a phenomenon unique to quantum communications [4]. This process of preparing, sending, and measuring qubits is known as “quantum exchange” and results in raw keys at both Alice and Bob.

After qubits are successfully exchanged between both parties, the system must employ techniques to generate a secure and error-free secret key. This process begins by first eliminating non-matching basis measurements. Bob announces his basis measurement information (not the measured bit values) for each detected photon over the classical communications channel. The result is a shared sifted key buffer between Alice and Bob, in which approximately half of the raw key bits are retained and half is lost due to Bob's random (i.e., incorrect) basis selection. Next, any errors that exist in the sifted key buffer will be corrected with a error reconciliation algorithm, used to minimize the amount of information an adversary can gain during the error correction process. Finally, an advanced information theory technique known as privacy amplification is used to ensure that any residual information an adversary may have obtained about the secret key is negligible. The shared secret key can then either be used to prime a symmetric encryption algorithm such as DES, 3DES, or AES or used in the unconditionally secure One-Time Pad (OTP) implementation.

After comparing preparation and measurements bases and eliminating the measured bits corresponding to mismatched bases, any remaining errors (bit mismatches) in the sifted key buffers may indicate the presence of an eavesdropper. Even if the errors were created by channel noise or any other kind of interference, all remaining errors are attributed to a listening adversary, conventionally known as Eve. An eavesdropper can introduce errors into the key because she is forced to measure the photons in transmission from Alice to Bob in order to determine their states, before resending random photons to Bob. The no cloning theorem protects the photon transmission because it is impossible to perfectly replicate a particle that has an unknown state [31]. At the time of measurement, Eve has no knowledge of the basis Alice used to encode the bit, so she must guess. If she measures incorrectly, the information encoded on the other bases is lost due the

Uncertainty Principle. When these photons reach Bob, his measurements will also be random, and will only guess the correct bit 50% of the time. Eve guesses the measurement basis correctly only 50% of the time as well, which means 25% of Bob's bits will differ from Alice's bits [32].

In addition to the originally designed protocol, there are practical security concerns that are addressed after key sifting has completed. Although all errors are attributed to Eve, some of them may in fact naturally occur due to noise in the channel. To address this issue, Alice and Bob will go through an error correction routine to eliminate remaining errors. To protect against the possibility that those errors might have been caused by Eve, and to maintain a usable key, the key buffer will go through a privacy amplification to obtain a shortened key of which Eve has little or no knowledge. The execution order of these processes largely remains the same from system to system, however their logical grouping tends to differ based on interpretation.

2.4.2 Phase Grouping

In the broadest terms, the BB84 protocol can be grouped into two major phases: quantum channel processing, and classical channel post-processing. The original protocol defines the post-processing to consist of the Sifting phase [3]. It was not until 1992 that the authors of the original protocol formalized the need for more post-processing in the form of error correction and privacy amplification, as well as authenticating the public channel [27]. The QKD post-processing aspect is often viewed differently from the protocol phases – those that exist up to and including key sifting.

This newfound understanding brought on differing opinions about the logical grouping of responsibilities per phase and how many phases should exist. The most popular groupings either consist of three or four relevant system phases. In a three-phase interpretation, there exists [37]:

1. *Quantum Exchange*
2. *Key Sifting*
3. *Classical Post-Processing*

Raw Key Exchange and Sifting are identical to those described above; however "Classical Post-Processing" encompasses Error Correction, Privacy Amplification, and Authentication.

The four-phase interpretation views Error Correction as its own category [12]:

1. *Quantum Exchange*
2. *Key Sifting*
3. *Error Correction*
4. *Privacy Amplification*

Regardless of how the phases are grouped, the processes and procedures that exist amongst them must be performed, in some capacity, within the Quantum Key Distribution System. They are: a quantum transmission, key sifting, error estimation and correction, privacy amplification, and authentication of a classical communications channel, described in the following sections:

We eight distinct processes required for the composition of a Quantum Key Distribution are defined as follows:

2.4.2.1 Authentication

Alice and Bob must authenticate their communication over the classical channel before transmission. It is assumed an adversary cannot alter any message exchanged over the classical channel during post-processing. The periodicity of re-authentication is at the discretion of the system designer [8].

2.4.2.2 Quantum Exchange

Alice prepares a sequence of qubits and transfers them to Bob over the quantum channel. For example, in polarization-based BB84, Alice randomly chooses a basis for each qubit: rectilinear or diagonal. Rectilinear basis can either be horizontal (0°) or vertical (90°), while diagonal can be ± 45°. Alice will then map a qubit state to each polarization. For instance, in the rectilinear basis, a 0 bit may be encoded as horizontal and 1 as vertical. Similarly, in the diagonal basis 0 may be encoded as 45° and 1 as 135° (or -45°) [3, 4, 19].

2.4.2.3 Raw Key Sifting

Bob must choose – at random – a basis with which to measure each qubit after receiving encoded photon pulses from Alice. He shares these bases with Alice and she confirms when their prepare and measure bases match. Any non-matching qubit measurements will be discarded. The remaining key should be identical for both parties, however channel noise or the presence of device non-idealities may inflict bit errors in the sequence [3, 4].

2.4.2.4 Error Estimation

Before these errors can be corrected, Alice and Bob must first determine if they have exceeded a predetermined error threshold, otherwise the unconditional security proofs will be violated [1]. Part of Bob's key material will be sampled at random and compared to estimate the overall average error rate. If it is too high, Alice and Bob may choose to abort this instance of key generation and start over [1].

2.4.2.5 Error Reconciliation

If the error rate is below the threshold, Alice and Bob perform an error correction scheme to ensure that they both possess identical keys. There are several reconciliation algorithms that can be used during this process, including Cascade, Winnow, and LDPC [20].

2.4.2.6 Entropy Estimation

Entropy loss estimation is an estimate of the amount of information Eve may know about the now reconciled key held by Alice and Bob. As we will discuss, the difficulty of estimating the total system entropy exists due to the multitude of design decisions that must be made with the potential to influence entropy. This is an inherently different problem than estimating the amount of information Eve can glean from classical cryptographic methods that rely on computational complexity [12].

2.4.2.7 Privacy Amplification

In an effort to further minimize the amount of information Eve has about the key, privacy amplification is used. During this process, more of the key is discarded in order to form a more secure final key. In general, the output of Entropy Estimation will determine the loss in final key size – the higher the entropy loss, the more key will be sacrificed to ensure Eve knows the least amount of information possible. This is accomplished by utilizing some variation of a Universal$_2$ hashing function [12, 15, 42].

2.4.2.8 Final Key Generation

The Privacy Amplified buffers held by Alice and Bob are checked one final time for consistency to ensure they are identical. This is accomplished through a final hash of both buffers and comparison of those hashes. If they do match, the Privacy Amplified key can be added to the final secure key buffer for use in an encryption scheme. It is unclear exactly how, when, and in what form this final hash should be accomplished. We will discuss a possible solution to this problem [15, 17, 42].

2.5 Secure Key Rate Estimates

In QKD literature, the performance of the system is usually described as the throughput of the secure key in bits per unit time [5, 18, 19]. For example, the equation in Zhou *et. al.* is described as [5]:

$$\boldsymbol{K_{secure} = q \cdot Q_{\mu}\{-H_2(E_{\mu}) + \Delta_1[1 - H_2(e_1)]\} \cdot f} \qquad (1)$$

K_{secure} = the secure key rate of a QKD system

q = protocol efficiency fraction

Q_{μ} = photon counting rate, or gain

E_{μ} = Quantum Bit Error Rate (QBER)

e_1 = QBER caused by single photons

$H_2(x) = -\sum_{i=0}^{n-1} p_i \cdot log_2(p_i)$, the Shannon Entropy

$\Delta_1 = \frac{Q_1}{Q_{\mu}}$, the ratio of single photon signals to all signals detected at Bob

f = the pulse rate at Alice

As we have discussed, QKD protocols describe the necessary processes up to and including sifting. The secure key rate equations make a deterministic prediction of the final key rate of the system by applying parameters that are specific to the protocol definition, particularly to the Quantum Exchange aspect of the system.

It is understood that the classical post-processing aspect of QKD is necessary to perform, but as such is not considered in this interpretation of gauging system performance. The variables listed above are specific only to the Quantum Exchange and Sifting aspects of the system, yet the equation attempts to determine key throughput for the system as a whole. The throughput equations in literature better serve as an upper bound on the performance of a system based on the pulse rate of Alice's laser. It is difficult to gauge actual performance, however, without specific knowledge of the implementation of computationally or communications intensive phases such as Error Reconciliation or Privacy Amplification.

III. Methodology

3.1 Overview

The purpose of this chapter is to describe the methodology used to conduct the analysis necessary for the development of a proof-of-concept empirical model. The rationale for answering the relevant research questions in Chapter I will be presented, in addition to a discussion on the experimental design and the methods that were used to validate the model.

3.2 Baseline Configuration

A thorough study was conducted on QKD system architectures. As described in Chapter II, and will be discussed more in-depth in Chapter IV, focus was given to systems implementing the BB84 protocol as well as the protocol itself. To estimate the performance of complete system designs, a system model needs to exist at a level of abstraction to include all the phases. The intent of this research is to establish the "least case" or the baseline at which a system can be called Quantum Key Distribution. There is consensus in the academic community about which procedures must be accomplished in order to be considered QKD. However, there is no universal agreement on which and how many of these procedures should be grouped into an individual phase and in which order each should occur.

In Chapter II, the generally recognized four phases of QKD were discussed. The model proposed here will break these into eight distinct phases. The four generally recognized phases were subdivided to provide a higher level of detail. For instance, Slutsky *et. al.* describe entropy loss estimation, privacy amplifying the key, and producing the final key as a single phase called Privacy Amplification [12], whereas our model assigns all three parts their own phase with unique inputs and outputs:

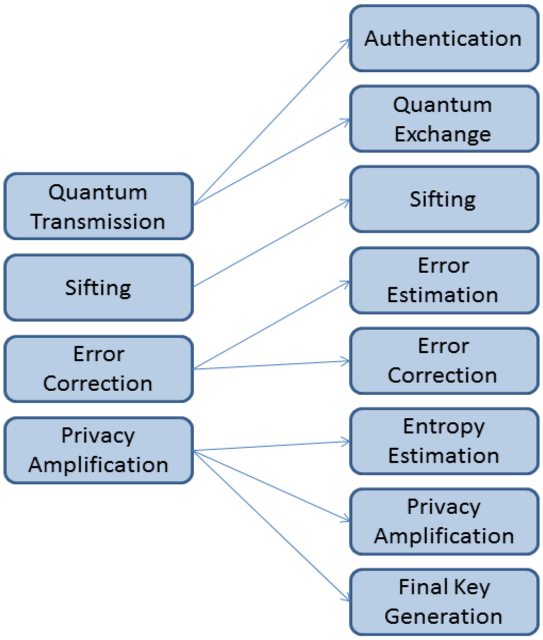


Figure 3. QKD Phase Relationships

3.3 Phase Model Development

It is worth reiterating that QKD is a technology used to distribute secret key, not an encryption scheme, and should only be considered as such. The effectiveness of a distribution system can be evaluated considering its performance in six areas [6]: confidentiality of keys, authentication routine requirements, the delivered key rate, systems robustness (behavior against denial of service, etc.), distance and location independence of communicating nodes, and resistance to traffic analysis. Contrasted with existing public-key distribution algorithms, there is much variability in QKD system designs. The inherent simplicity and lack of variability in public-key distribution allows for ease of implementation; they are algorithms which are selected for their individual qualities in the implementation of a complex cryptosystem or cryptographic suite. But such a method is not without its imperfections. One fundamental question for the QKD practitioner is: Does the cost of building such a system warrant the amount of key able to be generated, given that QKD is at least as secure of a distribution method as public-key distribution?

The lack of standardization in the design of QKD systems requires a level of understanding with high granularity. The system is not a singular entity, but a concatenation of eight distinct phases of operation. These phases define the operational aspects of a Quantum Key Distribution system. The need for their existence is dictated by the original protocols, such as BB84 [3]. The execution of each phase can be viewed independently, but given the serial nature of the system phases, there exist implications and/or consequences of design choices committed upstream (earlier phases) that affect subsequent execution and output of the system downstream (later phases). We view the QKD model in this way rather than as typical communications protocol stack model – the phases are, in fact, more akin to a pipeline stage than a protocol stack [5, 6].

The model developed for this research in Chapter IV represents the independent view of the system resource costs associated with each phase, as it relates to time, computational workload, and memory consumption. Each subsequent discussion highlights or enumerates the necessary considerations required in the design of a particular phase. The models depict: the input parameters into the phase, the major operations occurring during the phase and their resultant output as effects on the commodities of time and memory usage in Alice and Bob, as well inputs to the next phase.

The throughput of the QKD system is determined by the aggregate throughput of the phases, which is defined by a summation formula. Each discrete phase model's output provides, at minimum, the total time necessary to propagate through the phase, in addition to the amount of bits remaining in Alice and Bob's buffers at the conclusion of the required processing defined by the protocol. The inputs were determined, by a thorough analysis of the literature, as any tunable system parameters that could affect the outputs. Many of the input parameters possess dependencies that affect other parts of the system. The discrete phase equations were designed in such a way that when applied to the overall summation formula, dependency effects manifest themselves in final throughput. All of the dependencies that could be identified are discussed in detail in Chapter IV. All calculations are performed using number of bytes and amount of time in seconds, although the model is adaptable to other units of measure (e.g., kilobits, megabits, seconds, microseconds, etc.).

For any nondeterministic calculations (i.e., any computational processing that's determined at runtime by randomness, such as the Quantum Bit Error Rate), a variable for computational time is present. The value of this input variable represents the time it takes to propagate through the phase, or to complete a particular sub-block of the phase. For every phase aside from Quantum Exchange, communication across the public channel is required, and often halts the processing of the phase until Alice and Bob have communicated successfully. It is therefore natural to discuss computational time required in conjunction with communications overhead.

3.4 ICOM Models, Equations, and Sequences

Each phase is discussed in Chapter IV will be modeled in the same format. First, an ICOM (Input/Output, Controls, and Mechanisms) diagram is used to describe phase [39]. The ICOM diagram is paramount to the understanding of the flow of information and constraints governing execution. The following diagram shows an example of the ICOM format:

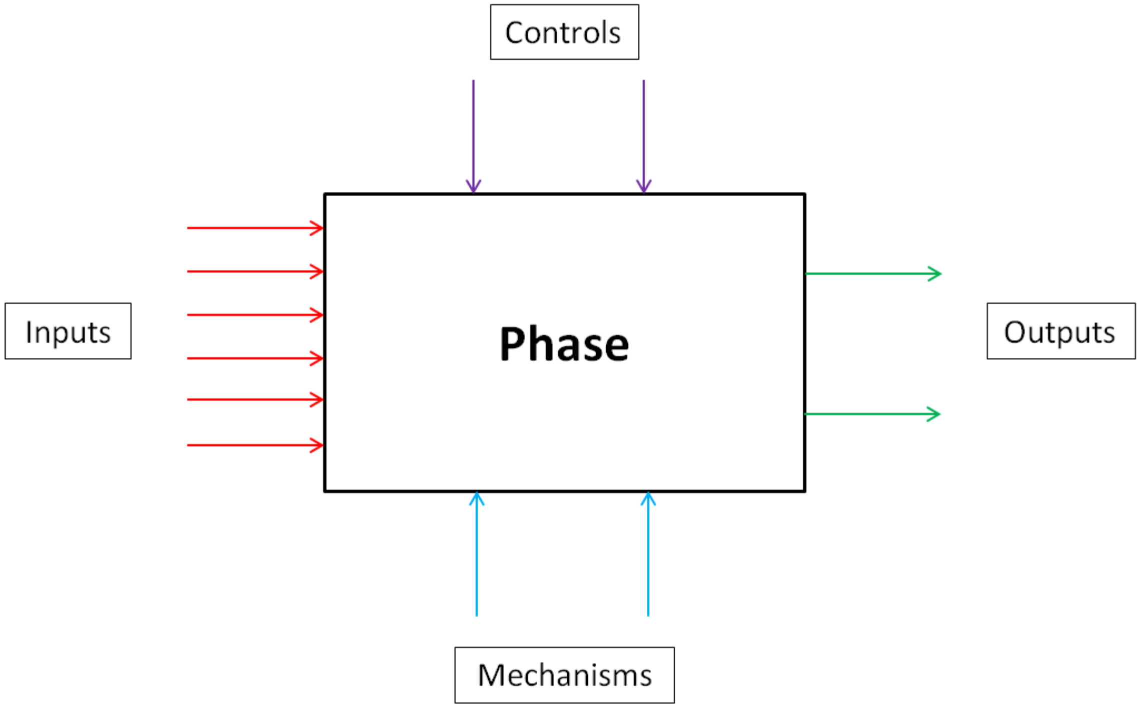


Figure 4. Example of ICOM model

Next, a series of abstract equations will be developed for each phase which describes the relevant behaviors of the phase (i.e., describing how inputs are converted to outputs). Finally, each phase will have a sequence diagram depicting the necessary classical communications required for execution. The following figure is an example of a sequence diagram in which the flow of communication occurs from left to right and vice versa, while the flow of time occurs from top to bottom:

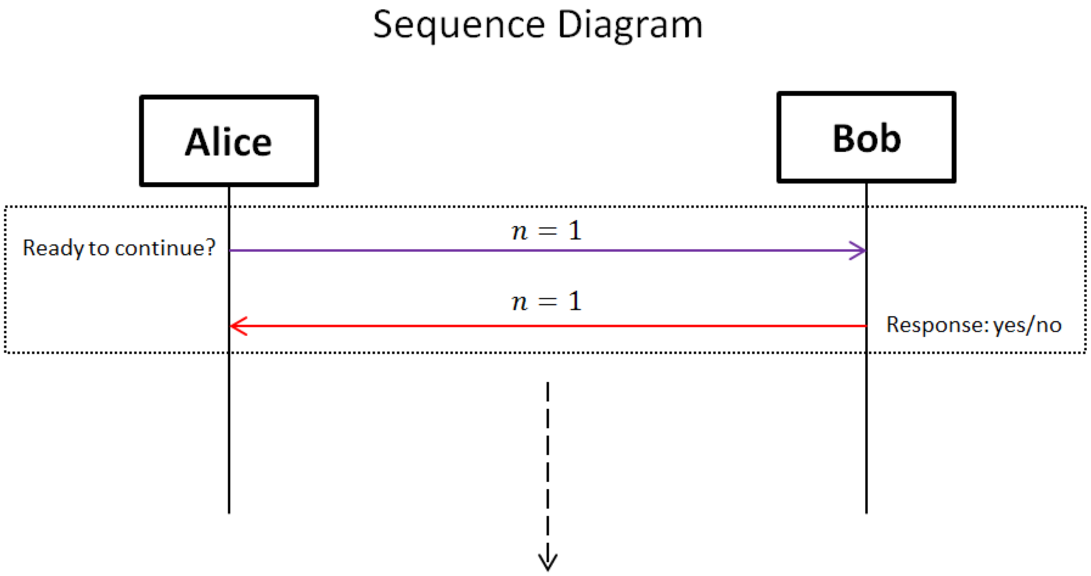


Figure 5. Example of Sequence Diagram

3.5 Baseline Configuration and Testing

After the model is defined, a configuration selected from QKD literature is used to establish a baseline model. The published findings' does not include all of the data in terms of input parameters needed for the model. Therefore, known values will be included as is, and reasonable assumptions about missing parameters are made. A series of exploration studies are conducted to answer relevant research questions. We will illustrate how design decision information and system behaviors can be understood, and how system optimizations can be found. An Excel model will be developed to encapsulate the phases into a serial sequence of phases.

IV. QKD System-Level Model

4.1 Introduction

In this chapter, the development of the system-level model is described in detail. First, an explanation of high-level system parameters will be discussed. Next, a model of each phase will be presented along with descriptions of phase-specific parameters, the sequence of classical communications during the phase, and a discussion of implementations in practical QKD systems.

4.2 System Characterization Model

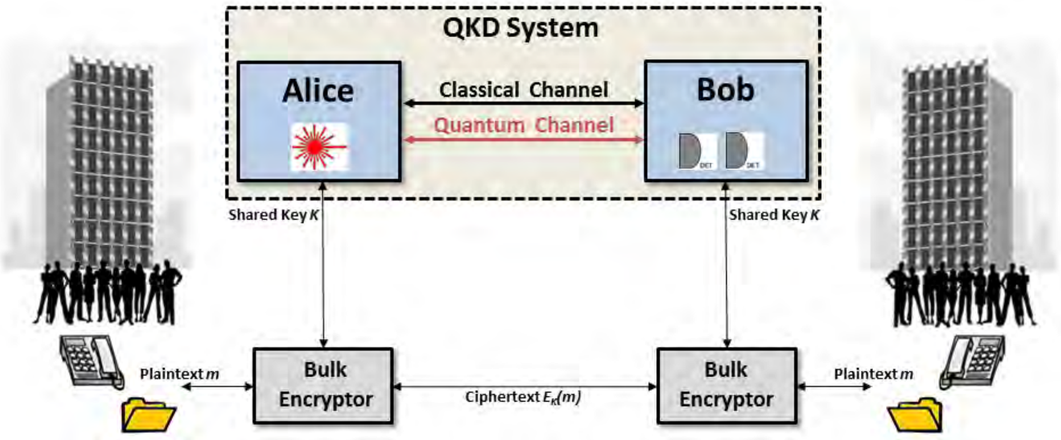


Figure 6. Overview of QKD System

At the system level, Quantum Key Distribution consists of the sender “Alice”, a receiver “Bob”, a quantum channel used to transfer photons, and a classical channel that supports conventional network communications. We first characterize each component by defining high-level system parameters, so that system designers can better understand, model and study the performance tradeoffs associated with particular design choices. The model outlined here uses several important metrics to both study system performance and describe the system behaviors of interest. We define each of these parameters in the following sections.

4.2.1 Alice and Bob

At a high level, we characterize the performance of a QKD system (consisting of an Alice, Bob, and associated networks) with consideration of computer processing power, network bandwidth and latency, and fundamental memory constraints. For example, a phase cannot execute unless a minimum amount of memory is available. Likewise, at the conclusion of each phase the memory usage associated with that phase will either increase or decrease.

4.2.1.1 Memory

We describe Alice and Bob's memory usage in terms of eight dedicated memory blocks of configurable size: one for each of the eight phases defined in Chapter 3. These memory blocks are pre-allocated and remain static for the duration of execution. While this assumption might not be as accurate as a model that considers dynamic memory allocation, it greatly simplifies modeling the entire system which provides insight into design tradeoffs.

The following memory map depicts an example snapshot how this allocation may be represented in the system:

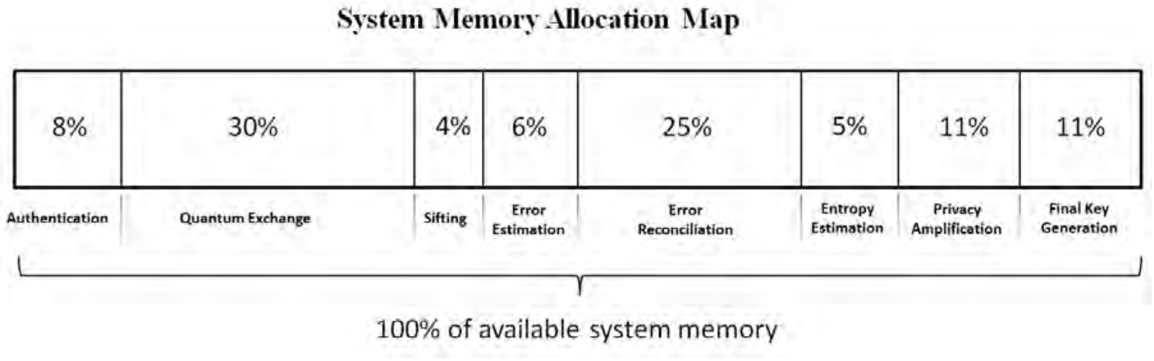


Figure 7. Example of System Memory Allocation Map

Each phase is allocated a percentage of available system memory. The total allocation cannot exceed 100% of the available memory. However it is possible to have a total allocation less than 100%, if desired. In general, the minimum amount of memory required for each phase is the size of the bit buffer passing through the phase and any additional memory overhead required for the processing of that buffer.

The additional memory overhead requirement is defined as any memory that is required in addition to the arbitrarily large input buffer to the phase, as well as any inter-phase memory requirements, such as the need to store information on pulses sent and received during Quantum Exchange. An example of additional memory requirement is any memory that is not immediately calculable during the phase, and must be defined from *a posteriori* knowledge of the system itself. For instance, in Error Reconciliation there exists memory overhead to complete computationally complex calculations for each block of key. The additional memory may therefore define the amount of computational memory required per block. This requirement must be defined per individual phase, and reduces the amount of available memory allocated for that phase.

4.2.1.2 Workloads

Each phase implicitly carries a workload representing the amount of work the processor must perform, captured by the "{Alice, Bob}$_{\text{cpu\_power}}$" parameter. Workload varies from phase to phase, but in this model it represents a standard unit of work that is treated equally to other units. We define one unit of work to mean a computational task that the processor must perform. The uniformity of how we treat work units lends itself to simplicity in a baseline configuration. The abstract workload parameters can be defined as detailed as needed to facilitate understanding tradeoffs.

4.2.1.3 Time

Aside from the Quantum Exchange phase of QKD, all communication occurs over the classical communications channel. The time necessary to complete this public discussion is relative to the amount of data that must be transferred between Alice and Bob. At minimum, each phase contains a "handshake" agreement between Alice and Bob deciding whether or not both parties are ready to continue execution. Depending on the phase and its implementation, any additional communications may be symmetrical (Alice and Bob send an equal number of messages of an average size), or asymmetrical (Alice or Bob sends many more messages than the other party, or the average size of the messages that one party sends is much greater than the other). In this model we consider that Alice and Bob each have a number of transactions that must be completed during each phase, each with a uniform or average message size.

In addition to the communications overhead, the total execution time of a phase also contains the time it takes for Alice and Bob's processors to complete (i.e., compute) the work associated with that phase. Each phase is associated with a static number work units, and Alice/Bob have a processor with a computational power defined as $\frac{workload}{time}$. In the definition of this model, we will assume processing power is constant across all phases. The total time it takes to propagate through the phases will be a summation of the times it takes for each phase to execute one or more times, dictated by system's target performance. Examples of performance targets include a desired final key rate, a desired number of final key bits, a desired key buffer size as an input to Privacy Amplification, or a desired key buffer size as an input to Error Reconciliation. The metrics used to define the system will also be used to define the system resource cost of reaching these performance targets.

4.2.1.4 Alice

The system-level input parameters for Alice are described in the following table:

Table 1. Alice System-Level Parameters

Parameter Name	Units	Definition	Detailed Description
pulse_rate	Mhz	The pulse rate of Alice's laser.	The pulse rate of Alice's represents the number of pulses she sends per second. Example: 4 Mhz = 4,000,000 pulses/sec
MPN	unitless	The Mean Photon Number (MPN)	The Mean Photon Number is an input to the Poisson distribution that describes the probability of n photon(s) appearing in a laser pulse with MPN = μ, described by: $P(n;\mu) = \frac{\mu^n \cdot e^{-\mu}}{n!}$
$signal_{percent}$	%	The percentage of signal states present in Alice's transmissions.	QKD practitioners may choose to implement a decoy-state protocol, which defines a percentage of the pulses sent are signal states, decoy states, and vacuum states. The final key rate is determined only be the number of signal-states that are successfully received at Bob.
$Alice_{total\_memory}$	bytes	The total amount of memory allocated in Alice for QKD.	Alice has a finite amount of total memory that is represented by the aggregate memory allocated by each phase.
$Alice_{cpu\_power}$	units/sec	The computational power of Alice's classical processor.	The power of Alice's processor is defined by the speed at which she can process units of work. A faster processor can compute more units of work in less time.

4.2.1.5 Bob

The system-level input parameters for Bob are described in the following table:

Table 2. Bob System-Level Parameters

Parameter Name	Units	Definition	Detailed Description
db_loss_Bob	dB	The loss as a result of propagation through Bob.	Pulses arriving at Bob have a calculable amount of loss (dB) due to the propagation through Bob's hardware. In this model we use the efficiency parameter for calculations, derived from the efficiency equation as a function of loss in dB: $Eff(dB) = 10^{\frac{-dB}{10}}$
$\eta_{detector}$	unitless	The efficiency of Bob's detectors.	A SPD suffers from poor detection efficiency due to recovery times, jitter times, quench times, dark counts, and afterpulses associated with the APD.
t_{dead}	sec	The dead time of Bob's detectors.	The dead time is a user defined period of idle time after a detection (i.e., an avalanche event, where a SPD is "turned off" to avoid afterpulsing). This is known as the dead time between detections, and limits Bob's overall detection speed.
Bob_{total_memory}	bytes	The total amount of memory allocated in Bob for QKD.	Bob has a finite amount of total memory that is represented by the aggregate memory allocated by each phase.
Bob_{cpu_power}	units/sec	The computational power of Bob's classical processor.	The power of Bob's processor is defined by the speed at which he can process units of work. A faster processor can compute more units of work in less time.

4.2.2 Classical Channel

A classical channel can exist in different forms, including free-space, over mixed use optical fiber, dedicated optical fiber, a public network, or even on the same fiber as the quantum channel. The system-level input parameters for the classical channel are described in the following table:

Table 3. Classical Channel System-Level Parameters

Parameter Name	Units	Definition	Detailed Description
dist_btwn_Alice_Bob	km	The distance between Alice and Bob.	Alice and Bob share a fixed distance between them. In this model we assume the distance of the classical channel and the distance of the quantum channel to be equal.
delay_per_unit_length	sec/km	The delay per kilometer incurred as a result of propagation.	The classical channel contains a delay associated with transmission that is bounded by the speed of light, *c*, in free-space and $\frac{2}{3} \cdot c$ through optical fiber [40]. Conventional networks may also have an added delay due to the communications protocol.
bandwidth	Mbits/s	Information capacity of the classical channel.	The classical channel has a limit on the amount of information that can be transferred in a given period of time, usually described in Mbits per second. Example: 100 Mbit/s bandwidth

All transmission times across the classical channel are calculated using the following formula:

$$\boldsymbol{transmission\_time(msg_{size}, bandwidth, num\_trans)} = \frac{\boldsymbol{msg_{size}(bits)}}{\boldsymbol{bandwidth\ (bits/sec)}} \cdot \boldsymbol{num\_trans} + \boldsymbol{t_{class\_prop\_delay}\ (sec)} \qquad (2)$$

Additionally, we will define the inputs related to the sending and receiving of messages that are shared across all phases:

Table 4. Shared Parameters of Classical Channel Communications

Parameter Name	Units	Definition	Detailed Description
AB_avg_msg_size	bytes	The average size of a message passed between Alice and Bob that is sent from Alice.	In phases that require communication over the classical channel, there is an amount of data that Alice must send to Bob. Although it is not required that each message carry the same amount of data, we represent the size of each message uniformly. For each phase where *AB_avg_msg_size* is defined, it is assumed that all messages passed sent by Alice to Bob are of that size.
BA_avg_msg_size	bytes	The average size of a message passed between Alice and Bob that is sent from Bob.	In phases that require communication over the classical channel, there is an amount of data that Bob must send to Alice. Although it is not required that each message carry the same amount of data, we represent the size of each message uniformly. For each phase where *BA_avg_msg_size* is defined, it is assumed that all messages passed sent by Bob to Alice are of that size.
AB_num_trans	unitless	The number of transactions between Alice and Bob that are initiated by Alice.	The number of transactions from Alice to Bob multiplied by the average message size from Alice to Bob calculates the approximate total amount of data that is passed in that direction. If the total number of transactions from A→B are added to the total number of transactions from B→A, we calculate the total number of transactions during the phase.
BA_num_trans	unitless	The number of transactions between Alice and Bob that are initiated by Bob.	The number of transactions from Bob to Alice multiplied by the average message size from Bob to Alice calculates the approximate total amount of data that is passed in that direction. If the total number of transactions from B→A are added to the total number of transactions from A→B, we calculate the total number of transactions during the phase.

In this model we assume the time it takes for Alice and Bob to communicate at the beginning of each phase (signified by a dashed outline in the sequence diagrams) to be insignificant.

4.2.3 Quantum Channel

Unlike the classical channel, the quantum channel is dedicated to quantum transmission – the transmission of photons from Alice to Bob. Its properties are similar to those of the classical channel with some slight differences, mainly to account for losses experienced during transmission. The system-level input parameters for the quantum channel are described in the following table:

Table 5. Quantum Channel System-Level Parameters

Parameter Name	Units	Definition	Detailed Description
$delay\_per\_unit\_length$	sec/km	The delay per kilometer incurred as a result of propagation.	The classical channel contains a delay associated with transmission that is bounded by the speed of light, c, in free-space and $\frac{2}{3} \cdot c$ through optical fiber [40].
$loss\_per\_km$	dB/km	The average amount of loss, in dB, experienced per kilometer.	Optical communication has losses associated with transmission. For example, optical fiber losses are approximately 0.2 dB/km. In this model we use the efficiency parameter for calculations, derived from the efficiency equation as a function of loss in dB: $Eff(dB) = 10^{\frac{-dB}{10}}$

4.3 Authentication

4.3.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Authentication phase:

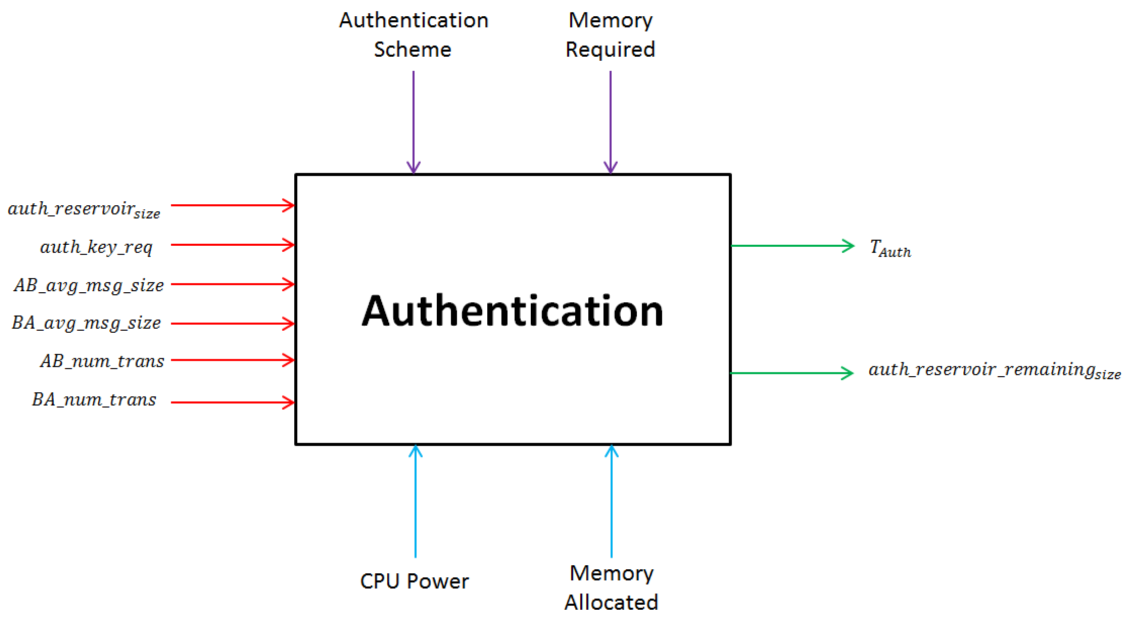


Figure 8. Authentication ICOM Model

The inputs and outputs local to Authentication are discussed in the following table:

Table 6. Inputs/Outputs Local to Authentication

Parameter Name	Units	Definition	Detailed Description
$auth\_reservoir_{size}$	bytes	The number of bytes in the authentication reservoir that can be used to authenticate a message.	The size of the authentication reservoir must be at least large enough at the beginning of each QKD round to authenticate the communication between Alice and Bob.
$auth\_key\_req$	bytes	The number of bytes required to authenticate a message.	For more information, see section 4.3.4.
T_{Auth}	sec	The time required to complete a single authentication.	If the QKD practitioner chooses to authenticate classical communications more than once per round, then the total time it takes to complete authentication will be a summation of time it takes to complete a single authentication multiplied by the chosen number of iterations.
$auth\_reservoir\_remaining_{size}$	bytes	The number of bytes remaining in the authentication reservoir.	The number of times Alice and Bob's communication can be re-authenticated during a QKD round is dictated by the amount of key remaining in the authentication reservoir.

4.3.2 Abstract Equations

The total amount of time required for a single authentication can be calculated with the following equation, which describes the time it takes to transmit data between Alice and Bob in addition to the time it takes to perform computations:

$$\boldsymbol{T_{Auth} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{Auth}}}{\{A,B\}_{cpu_{power}}}}$$

(3)

The amount of Authentication key remaining in the Authentication reservoir after the phase has completed execution can be calculated with the following equation:

$$\boldsymbol{auth\_reservoir\_remaining_{size} = (auth\_reservoir_{size} - auth\_key\_required)} \qquad \boldsymbol{(4)}$$

4.3.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

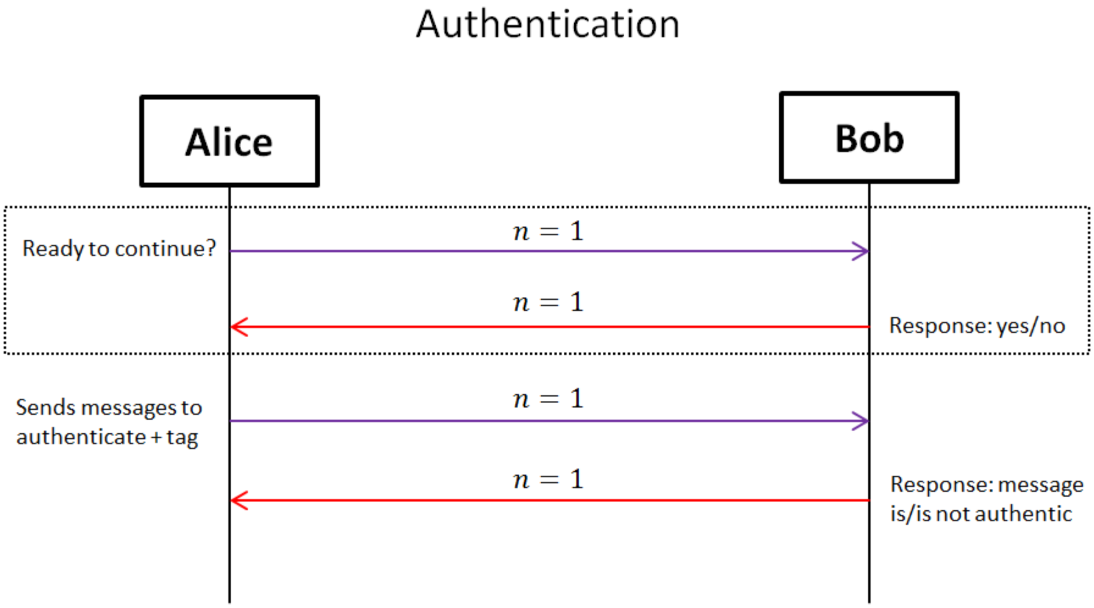


Figure 9. Sequence Diagram of Classical Communication During Authentication

4.3.4 Discussion of Practical Implementation

In applications outside of QKD, most authentications are done by utilizing computationally hard public-key methods. If this were the case in QKD, the security of the entire system would effectively be reduced to the computational security of the public-key algorithm [7]. Since QKD itself is provably information-theoretically secure, the public channel authentication must emulate the same theoretic security on the classical channel. The standard in QKD is the Wegman-Carter authentication scheme [8], which is based on ϵ-Almost Strongly-Universal$_2$ (ASU$_2$) hash functions, a subset of the strongly Universal$_2$ hash family discussed in depth in the Privacy Amplification section.

The required number of sacrificed secret key bits can be on the order of several bytes if necessary, making the Wegman-Carter routine very efficient [41]. This is an essential feature of QKD authentication because the system must produce more secret key than is lost during the Authentication phase. The sacrificed number of bits grows logarithmically with the message size. The authentication routine can be thought of in five steps [41]:

1) The sender (Alice) selects a message, m.
2) The sender (Alice) uses key bits, k, from previous round to create an authentication tag for the message, where $k \approx \log_2(|m|)$.
3) The sender (Alice) discards used key bits and sends the message to the receiver (Bob).
4) The receiver (Bob) reads the tag, computes his own based off of his own secret key, and compares them.
5) If both tags match, the receiver (Bob) sends a message back to the sender (Alice) confirming that the message is authentic and QKD can continue.

In order to achieve this type of authentication, Alice and Bob must acquire a prepositioned secret key in order to agree on a priming key for the initial authentication round. In every round thereafter, a portion of the final key from the previous iteration will be used to authenticate the new communication. It is not advisable to consider a public-key distribution of the priming key between both parties for the same reason an information-theoretically secure authentication method must be used – it effectively reduces the security of the system to a computationally hard problem [6].

Standard practice in electronic communications suggests the periodic re-authentication of the communicating parties. For QKD, the frequency of this practice is dictated by the system practitioner and has a direct impact on the final key rate, given that the secret key used to prime the Wegman-Carter routine must come from the identical secure key buffers shared by Alice and Bob. The nature of ϵ-ASU$_2$ hash functions for use in QKD does not allow the authentication of more than a pair of messages at a time [42], which means that new secret key must be sacrificed for every message pair. The evaluation of the hash functions is assumed to occur in constant time.

4.4 Quantum Exchange (QE)

4.4.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Quantum Exchange phase:

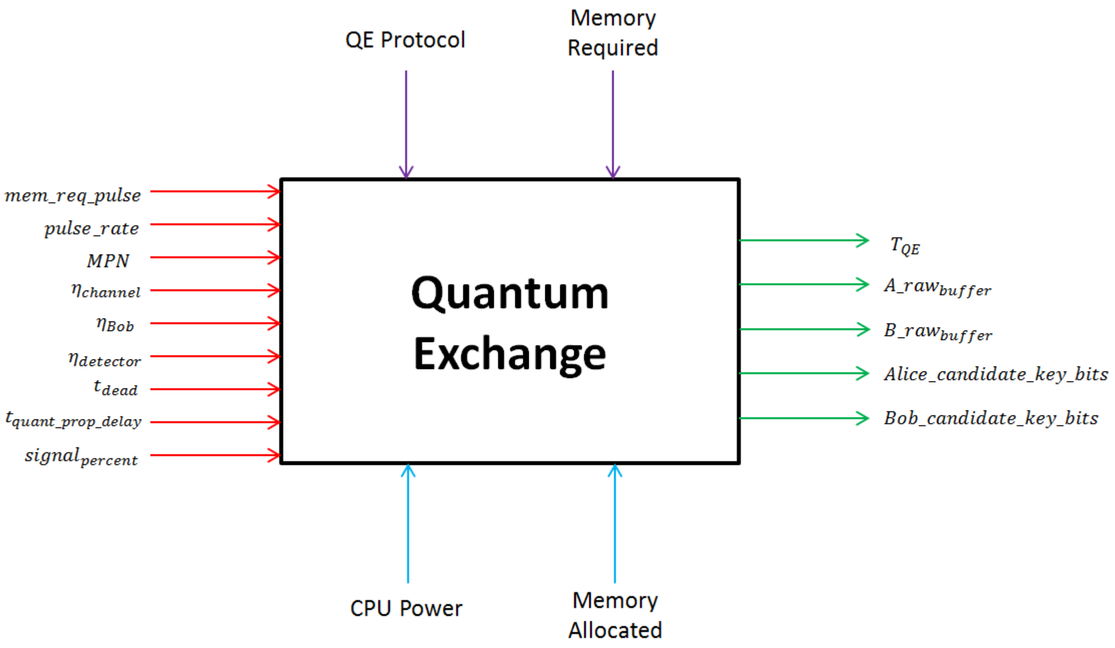


Figure 10. Quantum Exchange ICOM Model

The inputs and outputs local to Quantum Exchange are discussed in the following table:

Table 7. Inputs/Outputs Local to Quantum Exchange

Parameter Name	Units	Definition	Detailed Description
mem_req_pulse	bytes	The amount of memory required to store information on each pulse.	The memory required for each pulse is, at minimum, the space necessary to store the bit, basis, and timing (frame and slot) information. Alice stores information on every pulse she sends, while Bob store information on only the pulses he detects.
T_{QE}	sec	The time required to complete a single iteration of Quantum Exchange.	In this model we assume that Quantum Exchange will run until Alice's memory has been filled before moving to the next phase. To reach a desired number of final bits will most likely require multiple iterations QE during a single round of QKD. The total time of QE is calculated by the summation of time for the

			total number of QE iterations.
A_raw_{buffer}	bytes	The size of Alice's memory buffer after completion of QE.	If Alice runs QE until her memory buffer is full, then her raw buffer size will always be the maximum amount of memory allocated to her QE phase.
B_raw_{buffer}	bytes	The size of Bob's memory buffer after completion of QE.	Bob's memory buffer will be significantly smaller than Alices, due to his inability to store information on pulses he does not detect.
$Alice\_candidate\_key\_bits$	bits	The number of bits Alice possesses at the end of QE with the potential to be final key bits.	The number of bits Alice has will be equal to the number of pulses that she sends to Bob.
$Bob\_candidate\_key\_bits$	bits	The number of bits Bob possesses at the end of QE with the potential to be final key bits.	The number of bits Bob has will be equal to the number of detections he receives.

4.4.2 Abstract Equations

The time required to complete Quantum Exchange will either be the total time it takes for Alice to transmit all pulses required to fill her memory or the amount of time it takes to complete the classical processing necessary to process the information associated with each pulse, whichever takes longer. In this model the time to complete Quantum Exchange:

$$\boldsymbol{T_{QE} = max\left(\frac{num\_det\_at\_Bob}{actual\_det\_rate} + t_{quant\_prop\_delay}\,,\ \frac{\{A,B\}_{workload_{QE}}}{\{A,B\}_{cpu_{power}}}\right)} \qquad (5)$$

The Poisson distribution is used to calculate the probability that *n* photons will be present in a pulse. Thus, the cumulative Poisson probability describing the probability that one or more photons will be present is represent with the following equation:

$$\boldsymbol{Pois(X \geq 1) = 1 - e^{-\mu}} \qquad (6)$$

The total number of pulses that Alice sends is limited by the amount of memory she has allocated to Quantum Exchange. If the amount of memory required to store information (bit, basis, and timing) is known, the maximum number of pulses Alice can send is modeled by the following equation:

$$\boldsymbol{num\_pulses\_sent = \frac{A_{mem\ avail}}{mem\_req\_per\_pulse}} \qquad (7)$$

The number of detections that will actually be received at Bob is a product of efficiencies of the channel and Bob's hardware, the probability that a pulse contains a photon(s), the chosen signal percentage, and the total number of pulses sent by Alice. The approximate total number of detections at Bob can be modeled with the following equation:

$$\boldsymbol{num\_det\_at\_Bob = num\_pulses\_sent \cdot Pois(X \geq 1) \cdot sig_{percent} \cdot \eta_{channel} \cdot \eta_{Bob} \cdot \eta_{det}} \qquad (8)$$

The average time between the arrival of each pulse at Bob is represented by the inverse of the pulse rate after experiencing losses associated with photon probability, signal percentage, and the efficiencies of the channel and Bob's hardware in the following equation:

$$\boldsymbol{t_{time\_btwn\_pulse\_arrival} = \frac{1}{pulse\_rate \cdot Pois(X{\geq}1) \cdot sig_{percent} \cdot \eta_{channel} \cdot \eta_{Bob} \cdot \eta_{det}}} \qquad (9)$$

We assume the average pulse arrival interval, $t_{time\_btwn\_pulse\_arrival}$, at Bob is uniform. To account for the scenario where the dead time of the detectors exceeds the time between arriving pulses, we can calculate the average time between detections due to missed pulses with the following equation:

$$\boldsymbol{t_{avg\_time\_btwn\_detection} = ceil\left(\frac{t_{dead}}{t_{time\_btwn\_pulse\_arrival}}\right) \cdot t_{time\_btwn\_pulse\_arrival}} \qquad (10)$$

The actual detection rate (per second) can then be calculated by the inverse of the average time between detections. In the case where the dead time does not exceed the time between arriving pulses, $t_{time\_btwn\_pulse\_arrival}$, will be equal to

$t_{avg\_time\_btwn\_detection}$. In that case the pulse arrival rate would be equal to the actual detection rate, represented by the following equation:

$$\boldsymbol{actual\_det\_rate = \frac{1}{t_{avg\_time\_btwn\_detection}}} \qquad \textbf{(11)}$$

Given that Alice and Bob are required to store information about each pulse during Quantum Exchange, we can represent the size of their raw memory buffers as the amount of information required to store the number of pulses sent (Alice) and detections (Bob) with the following equations:

$$\boldsymbol{A_{raw_{buffer}} = num\_pulses\_sent \cdot mem\_req\_pulse} \qquad \textbf{(12)}$$

$$\boldsymbol{B_{raw_{buffer}} = num\_det\_at\_Bob \cdot mem\_req\_pulse} \qquad \textbf{(13)}$$

The number of candidate key bits (i.e., bits that have the potential to become final key bits) as a result of completing Quantum Exchange may be represented simply as being equal to the number of pulses sent (Alice) and detections (Bob) with the following equations:

$$\boldsymbol{A_{QE\_candidate\_key\_bits} = num\_pulses\_sent} \qquad \textbf{(14)}$$

$$\boldsymbol{B_{QE\_candidate\_key\_bits} = num\_det\_Bob} \qquad \textbf{(15)}$$

4.4.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the arrow pointing from Alice to Bob represents communication initiated by Alice, and the arrow pointing from Bob to Alice represents communication initiated by Bob:

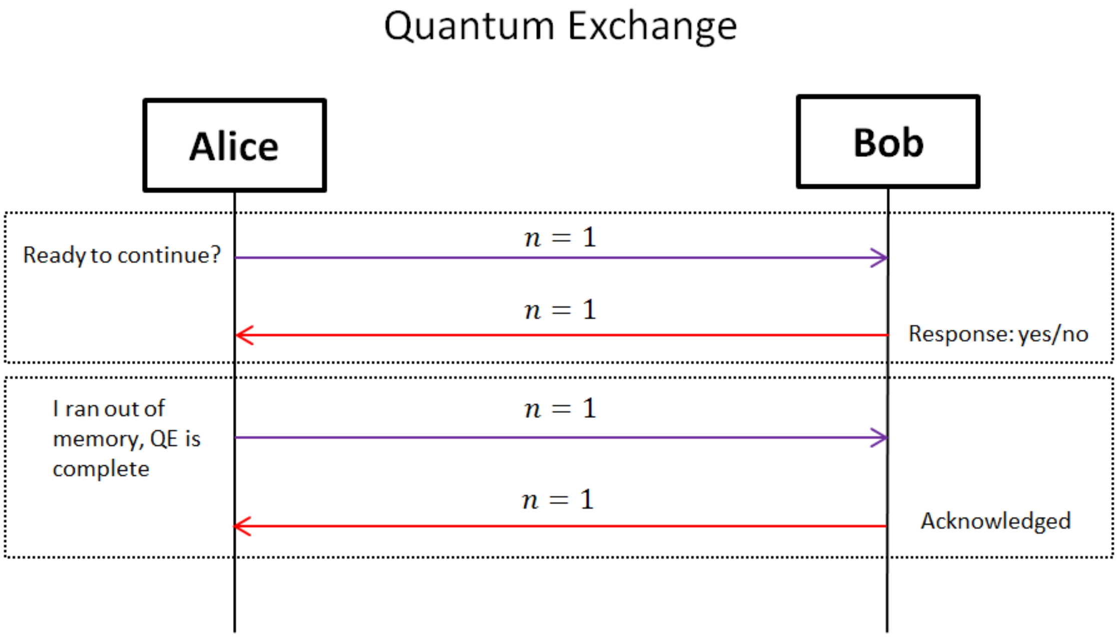


Figure 11. Sequence Diagram of Classical Communication During Quantum Exchange

4.4.4 Discussion of Practical Implementation

The objective of this phase is to maximize photon detection at Bob while overcoming potential obstacles in the system that limit Bob's detection efficiency. These realities primarily stem from the fact that neither perfect single photon generation nor detection is possible as of this writing. To address the lack of perfect single photon sources, photons are created at Alice with a defined probability – it is associated with attempting to maximize photon detection under these conditions, which by implication maximizes the final key generation rate. There are four major areas in which key generation rates can be significantly increased [21]: improvement in detection efficiency, increase in pulse rate at Alice, refinement of entropy estimates to reduce the amount of required privacy amplification, and increase in Mean Photon Number (MPN or μ, as is commonly denoted in literature). Note that QE is the only time during an iteration of the

QKD system where bits are being added to Bob's buffer. Successive phases seek to reduce the bit buffer, as sacrificial bits are required to complete their tasking.

The high amount of loss over long distances is generally overcome by utilizing Avalanche Photodiode (APD) detectors operating in the single-photon detection "Geiger-mode" in conjunction with a high pulse rate at Alice. The universal speed limit in these detectors, however, is dictated by the recovery or dead time between detections. This measurable recovery time exists due to the need to reset the device for a subsequent detection and prevent any charge carriers (e.g., electrons of holes) that were trapped in the device from being released during the next detector gating period, known as an after pulse, which may result in an arbitrary detection event that increases the number of errors. The recovery time of detectors must be long enough to limit or eliminate after pulsing but still generate sufficient key. It is worth mentioning that most detectors operate in a super-cooled state in order to reduce thermally generated contribution to the dark count rate [22], where a dark count is any arbitrary detection event that is not caused by a pulse from Alice.

The necessity for a recovery time of several microseconds forms one of the most important bottlenecks in overall system performance [21]. In addition, the efficiencies of APD detectors tend to be very low, typically around 10-20% [9]. In fact, the most fundamental trade-off in the operation of SPDs is that between dark count rate and photon detection efficiency [22]. A balance must exist then between Alice's pulse firing rate and Bob's ability to detect photons after incurring channel loss in the fiber, poor efficiency, and dead times. Since Alice must store information on every pulse she sends, if her pulse rate is too high she will quickly run out of available memory. Conversely, if her pulse rate is too low, the final key generation rate will be insufficiently low.

Another critical factor in determining the photon counting rate at Bob as well as final key rate is the selection of an optimal Mean Photon Number, or MPN. As stated, the lack of true single photon sources requires that Alice's pulses be attenuated so that they approximate single photon pulses. Bennett & Brassard, the original authors of the BB84 protocol, suggested the use of $\mu = 0.1$, however "contrary to frequent misconception, there is nothing special about a μ value of 0.1, even though it has been selected by most experimentalists" [29]. In other words, a system utilizing this probability will produce, on average, one pulse containing at least one photon for every ten pulses Alice generates according to the Poisson distribution. It is possible to increase the final key generation rate by simply increasing the MPN. In fact, the optimal probability is slightly over 1.0; however optimality varies between distinct systems [21]. The consequence of increased MPN is increased multi-photon entropy, which should be accounted for during the Entropy Estimation phase, and is the critical input parameter affecting how Privacy Amplification is performed. If the appropriate safeguards are built into the Entropy Estimation phase for a reasonable expectation of Eve's abilities, "one can in fact safely operate with a larger MPN … without any adverse effects on security" [21]. The major advantage of an increased MPN is an immediate increase in key generation without any detriment to system resources, namely memory space. A higher μ value increases the number of pulses that contain photons – pulses which have already been accounted for in memory by Alice. Thus, Alice's memory and MPN are independent, whereas Alice's memory usage and pulse rate are perfectly correlated.

The notion of dark counts in these detections, in terms of representation within the mathematical model, is such that if a dark count occurs, the result would be a skew in the number of detections at Bob, which is potentially more detrimental to the accurate representation of performance of a system than the actual ratio of dark counts to signal detections. If we assume Quantum Exchange will cease once Alice has produced an arbitrary number of pulses, yielding an equivalent bit count in her raw key buffer, the dark count rate will marginally skew the number of detections at Bob, potentially introducing errors. Dark counts and after pulsing therefore manifest their effects in the form of a higher error rate. As a result, if the calculated error rate exceeds its threshold during Entropy Estimation, no key will be generated in that particular iteration of the QKD system due to termination of execution. In general, the limit of secret key generation is the point which the probability of a dark count meets or exceeds the probability of detecting a signal bit [4], but dark count probabilities are so small that we did not include them in our general model. However, we note that when path loss is high (e.g., due to long propagation distances) and the photon detection probability becomes low, dark counts may become significant, and even dominate the detection rate. The desired speed of final key generation rate must be considered with the total distance of the system, which is true for all QKD implementations.

4.5 Sifting

4.5.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Sifting phase:

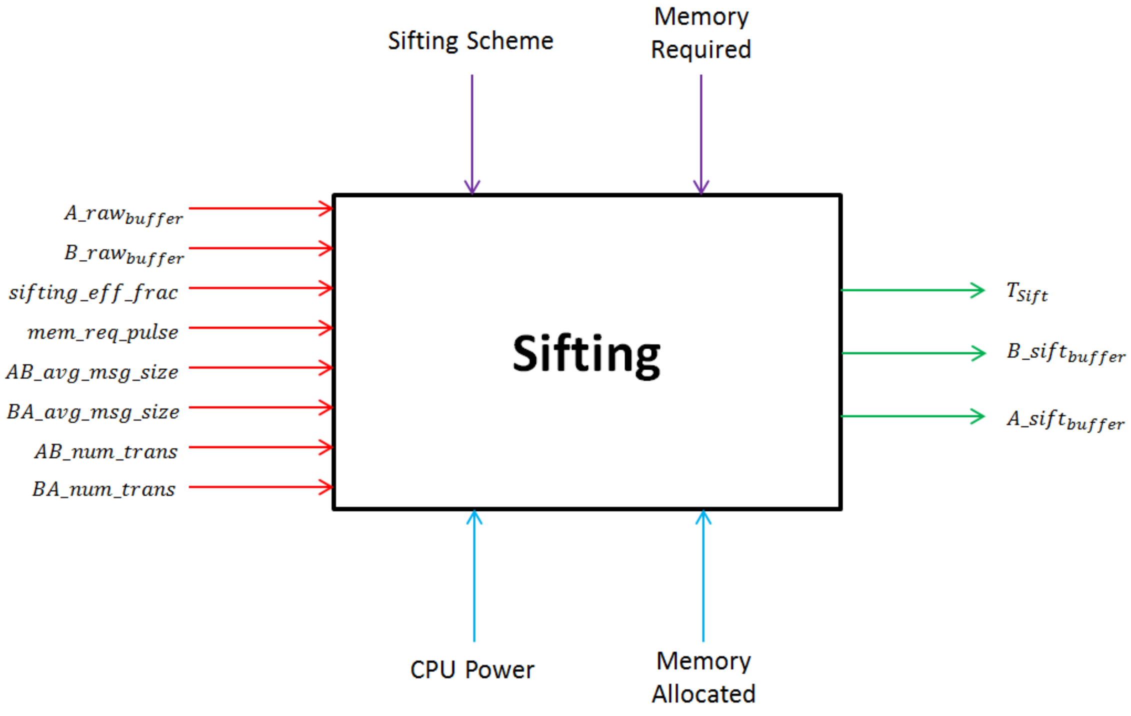


Figure 12. Sifting ICOM Model

The inputs and outputs local to Sifting are discussed in the following table:

Table 8. Input/Outputs Local to Sifting

Parameter Name	Units	Definition	Detailed Description
$sifting\_eff\_frac$	unitless	The approximate ratio of detections to correct basis measurements, as defined by the protocol.	The BB84 protocol, for instance, dictates that approximately half the time Bob will choose the correct basis measurement. The incorrect measurements will be discarded.
T_{Sift}	sec	The time required to complete the sifting process.	The time required to complete Sifting consists of the time it takes transfer all of Bob's basis measurements to Alice, for Alice to reply to Bob with the correct/incorrect measurements, and the processing time it takes to adjust the size of the bit buffers.
A_sift_{buffer}	bytes	The size of Alice's sifted key buffer.	After Alice eliminates pulses from her buffer with mismatched basis measurements, discards pulses that Bob did not detect and removes the extraneous information (basis, timing, etc.) stored on the remaining pulses, she is left with a sifted bit buffer.
B_sift_{buffer}	bytes	The size of Bob's sifted key buffer.	The size of Bob's sifted key buffer is the result of applying the sifting efficiency fraction and eliminating all unnecessary information that was required for Quantum Exchange (basis, timing, etc.)

4.5.2 Abstract Equations

The total time required to sift the raw key buffer is determined by the time it takes to transmit required data across the classical channel in addition to the computational time required, represented by the following equation:

$$\boldsymbol{T_{Sift} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{Sift}}}{\{A,B\}_{cpu_{power}}}} \qquad \mathbf{(16)}$$

After sifting has completed, Alice and Bob are no longer required to possess the basis and timing information for each bit. Therefore the size of Bob's sifted bit buffer is the result of removing both the unnecessary information about each pulse and mismatched basis measurements between himself and Alice. This process can be represented mathematically by the following equation:

$$\boldsymbol{B_{sift_{buffer}} = sift\_eff\_frac \cdot \frac{B_{raw_{buffer}}}{mem\_req\_pulse}} \qquad \mathbf{(17)}$$

The size of Alice's bit buffer should be equal to Bob's after removing all extraneous information related to erroneous basis measurements:

$$\boldsymbol{A_{sift_{buffer}} = B_{sift_{buffer}}} \qquad \mathbf{(18)}$$

4.5.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

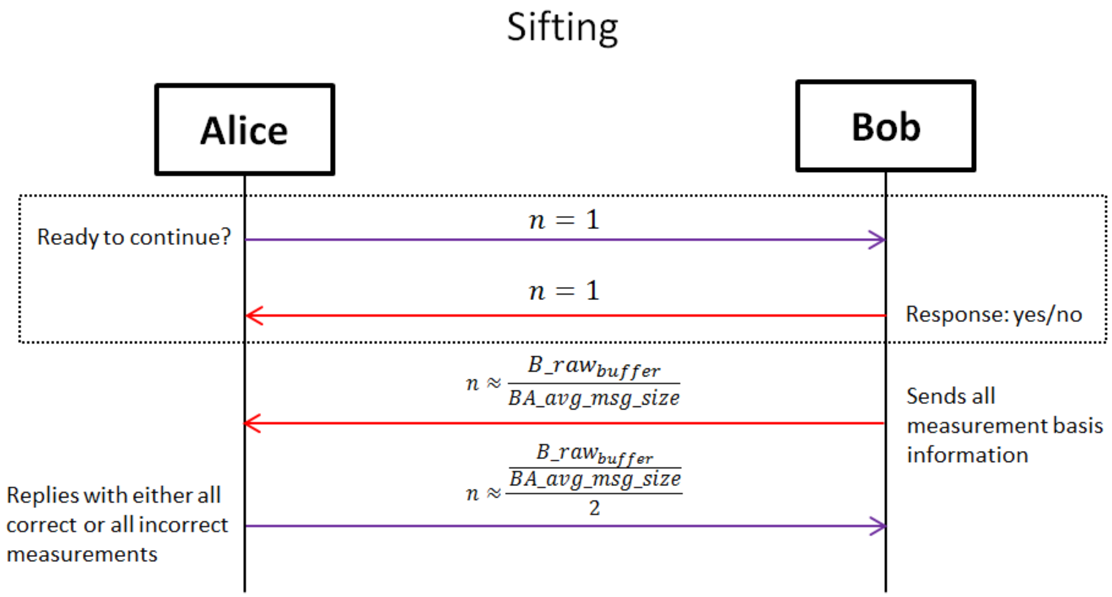

Sifting
Alice
Bob
Ready to continue?
$n = 1$
$n = 1$
Response: yes/no
$n \approx \frac{B\_raw_{buffer}}{BA\_avg\_msg\_size}$
Sends all measurement basis information
$n \approx \frac{\frac{B\_raw_{buffer}}{BA\_avg\_msg\_size}}{2}$
Replies with either all correct or all incorrect measurements

Figure 13. Sequence Diagram of Classical Communication During Sifting

4.5.4 Discussion of Practical Implementation

At minimum, the completion time of Sifting will require the time it takes to transmit all of the measurement basis information from Bob's raw key buffer to Alice in a series of messages across the classical channel, and a response from Alice for each message. Note that this is only one example of an implementation. In fact, Alice could theoretically send all of her basis preparation information to Bob and force him to complete the necessary processing to match and compare his measurements on received photons to Alice's entire buffer, then communicate that information back to her. The consequence of completing sifting in the latter fashion is the major increase in communications overhead that is required across the classical channel, which is directly correlated to the amount of loss on the quantum channel. Alice must store in memory, in some way, information about each pulse sent. At minimum, this includes the bit value, basis measurement, and timing information. Given the high amount of loss described in the previous Quantum Exchange section, if Alice communicates all of her pulse information to Bob after a Quantum Exchange that experienced 90% loss, there will be nine times as many messages required across the classical channel than if Bob initiated the Sifting phase by sending his basis measurements first.

There is a significant *increase* in available memory space after the Sifting phase has completed. In a signal-state-only BB84 implementation, for instance, the protocol efficiency fraction will be approximately one half [18]. In other words, 50% of Bob's raw key buffer will be discarded in addition to Alice discarding her entire buffer except those which matches Bob's buffer. Memory, however, is finite. Leveraging the memory space bottleneck is one of the most difficult challenges in the design of a QKD system. It limits the total number of photons that can be "in play" within the system at any given time, and is initially governed by the amount of loss in the channel. Until the Sifting phase has completed, Alice has to maintain information on every pulse that she sends, whereas Bob can only record what he detects. And as such, Alice suffers from a proportional increase in memory requirement to the amount of loss she must incur. The memory space being utilized in Alice is therefore disproportionately larger than in Bob before Sifting.

The nature of serialized phase execution dictates that before an attempt can be made to generate more key, the previous iteration of the system must have completed operation. It is therefore of interest to maximize the amount of key generated per system iteration while decreasing the time it takes to achieve the desired output of keying material. In general, the more signal bits detected at Bob during Quantum Exchange, the larger the output of secure key will be. The prospect of infinite detections, however, is hedged by the practical need for Alice to store information on all transmitted photons, the loss in both the channel and Bob, and the inefficiencies of Bob's detectors. Given practical limitations on the size of Alice's memory, a balance must be achieved between the amount of desired key in a given window of time, at a particular distance, for a single iteration of the system. It is crucial to affirm this concept as it relates to the Sifting phase because memory requirements, particularly for Alice, are at their peak at the conclusion of Quantum Exchange and the start of the Sifting.

4.6 Error Estimation

4.6.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Error Estimation phase:

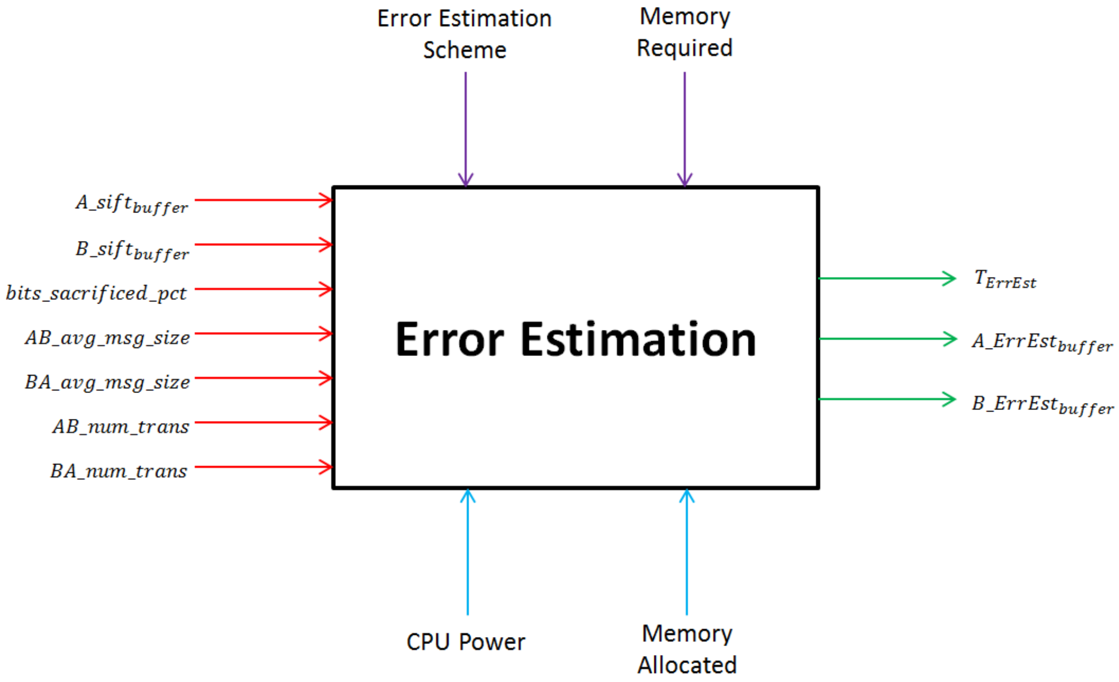


Figure 14. Error Estimation ICOM Model

The inputs and outputs local to Error Estimation are discussed in the following table:

Table 9. Input/Outputs Local to Error Estimation

Parameter Name	Units	Definition	Detailed Description
$bits\_sacrificed\_pct$	%	The percentage of the sifted key buffers sacrificed to provide an error estimate.	The percentage of eliminated sifted key for Error Estimation is entirely dependent on the needs of the practitioner. If a more accurate estimation is desired, the percentage can be increased at the detriment of final throughput.
T_{ErrEst}	sec	The time required to complete Error Estimation.	The time to complete sifting consists of communications time to transfer the sacrificed bits across the classical channel and processing time for error estimation calculations.
A_ErrEst_{buffer}	bytes	The size of Alice's Error Estimated key buffer.	The size of the Error Estimated buffer equals the size of the input sifted key buffer after eliminating the *bits sacrificed percentage* of the key.
B_ErrEst_{buffer}	bytes	The size of Bob's Error Estimated key buffer.	The size of the Error Estimated buffer equals the size of the input sifted key buffer after eliminating the *bits sacrificed percentage* of the key.

4.6.2 Abstract Equations

The total time required to estimate the error percentage of the sifted key buffer is determined by the time it takes to transmit required data across the classical channel in addition to the computational time required, represented by the following equation:

$$\boldsymbol{T_{ErrEst} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}workload_{ErrEst}}{\{A,B\}cpu_{power}}} \qquad \mathbf{(19)}$$

The size of the error estimated buffers for both Alice and Bob are determined the percentage of bits that are saved multiplied to the sized of the sifted key buffer, represented by the following equation:

$$\boldsymbol{\{A,B\}_{ErrEst_{buffer}} = (1 - bits\_sacrificed_{pct}) \cdot \{A,B\}_{Sift_{buffer}}} \qquad \mathbf{(20)}$$

4.6.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

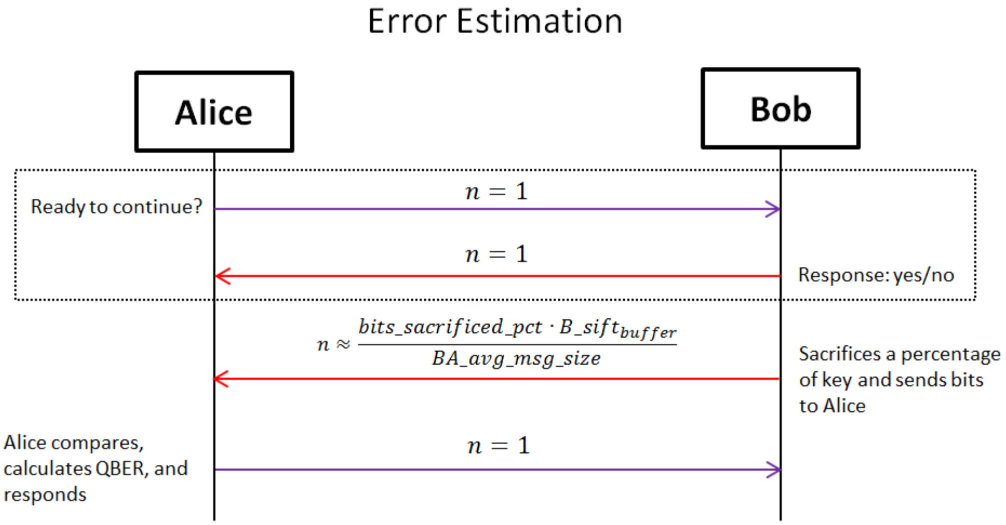

Error Estimation
Alice
Bob
Ready to continue?
$n = 1$
$n = 1$
Response: yes/no
$n \approx \frac{bits\_sacrificed\_pct \cdot B\_sift_{buffer}}{BA\_avg\_msg\_size}$
Sacrifices a percentage of key and sends bits to Alice
Alice compares, calculates QBER, and responds
$n = 1$

Figure 15. Sequence Diagram of Classical Communication During Error Estimation

4.6.4 Discussion of Practical Implementation

Time of completion in Error Estimation is primarily determined by how quickly a sampling of sacrificed bits can be transmitted between the parties. The Error Estimation phase presents the QKD practitioner with an opportunity to have a significant impact on the amount of final key generated by dictating the amount of key sacrificed to calculate the error rate. The trade-offs between sacrificing too much key or too little key can be significant. By sacrificing a high percentage of key it is also an immediate deduction in potential key at the conclusion of the final phase. But from this we gain a highly accurate estimate of the error rate, which may save vital resources required to propagate further through the system (assuming the error rate does not exceed the error threshold). A highly accurate error rate will also pay dividends during calculations for Error Reconciliation (e.g., performance optimization if the error rate is used to select the initial starting block size in the highly interactive Cascade algorithm). We must also consider that when leveraging phase execution time against amount of final key produced, there must be an intersection between how much final key is retained during Error Estimation and the subsequent gain in memory and final throughput per QKD iteration.

4.7 Error Reconciliation

4.7.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Error Reconciliation phase:

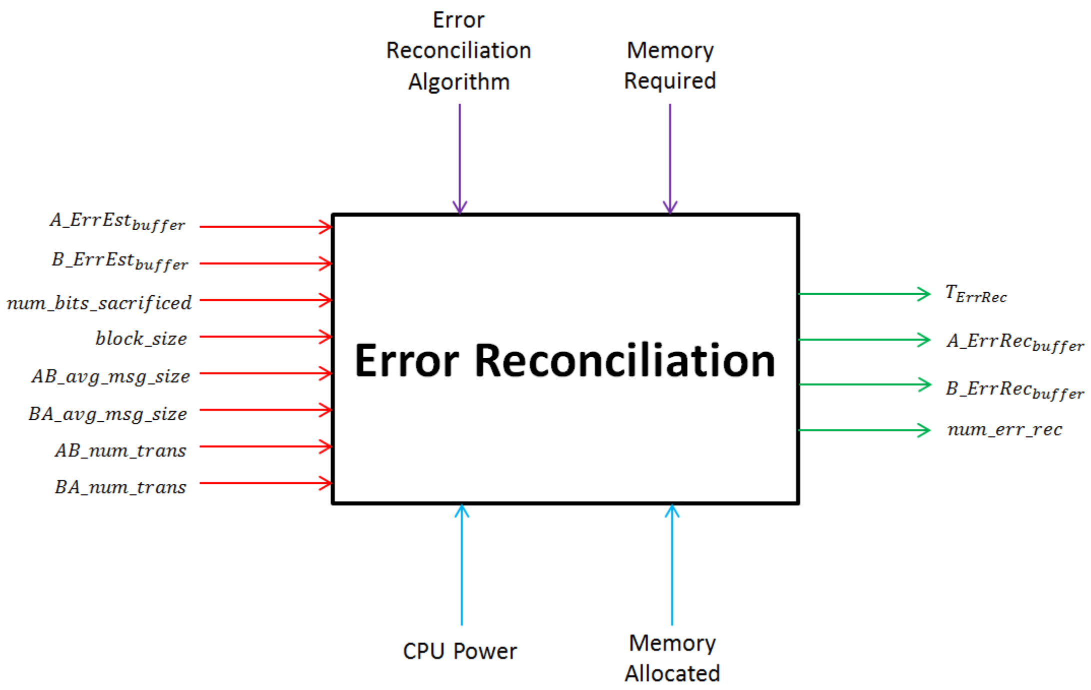


Figure 16. Error Reconciliation ICOM Model

The inputs and outputs local to Error Reconciliation are discussed in the following table:

Table 10. Input/Outputs Local to Error Reconciliation

Parameter Name	Units	Definition	Detailed Description
$num\_bits\_sacrificed$	unitless	The number of bits sacrificed for each Error Reconciliation routine.	Some QKD error correction algorithms sacrifice bits as a result of their processing. The most common algorithms, such as Cascade and LDPC, do not.
$block_{size}$	bits	The required input block size to perform an Error Reconciliation routine.	An error correction algorithm may have an optimal chosen block size that maximizes performance. In the case of LDPC, required a fixed input size based on the standard matrix selected for processing.
T_{ErrRec}	sec	The total time required to complete a round of Error Reconciliation.	The time to complete Error Reconciliation depends highly on the algorithm chosen, its interactivity on the classical channel, and computational complexity.
A_ErrRec_{buffer}	bytes	The size of the Error Reconciled buffer.	The size of the Error Reconciled buffer will be the same size as the Error Estimated buffer (assuming no bits are lost due to algorithm choice).
B_ErrRec_{buffer}	bytes	The size of the Error Reconciled buffer.	The size of the Error Reconciled buffer will be the same size as the Error Estimated buffer (assuming no bits are lost due to algorithm choice).
num_err_rec	unitless	The total number of Error Reconciliation routines required to process the input buffer.	The need to perform Error Reconciliation by block size means multiple rounds will be required if the input to Error Reconciliation exceeds the block size.

4.7.2 Abstract Equations

The total time required to complete Error Reconciliation, assuming that a single iteration of the algorithm occurs at a time, is equal to the time it takes to complete a single block of input key multiplied by the number of total blocks in the error estimated input buffer. We also assume that if the number of blocks does not divide the error estimated buffer evenly, the last block will be padded in order for the algorithm to run successfully, defined by the following equation:

$$\boldsymbol{T_{ErrRec}} =$$

$$\boldsymbol{ceil}\left(\frac{\{A,B\}_{ErrEst_{buffer}}}{block_{size}}\right) \cdot$$

$$\left(\boldsymbol{transmission\_time}(\boldsymbol{avg\_msg\_size}_{\{AB,BA\}}, \boldsymbol{bandwidth}, \boldsymbol{num\_trans}_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{ErrRec}}}{\{A,B\}_{cpu_{power}}}\right)$$

(4)

The size of the error reconciled buffer will be equal to the size of the error estimated buffer minus the number of bits sacrificed during all iterations of the algorithm (if any):

$$\{\boldsymbol{A}, \boldsymbol{B}\}_{ErrRec_{buffer}} = \{\boldsymbol{A}, \boldsymbol{B}\}_{ErrEst_{buffer}} - (\boldsymbol{num\_bits\_sacrificed} \cdot \boldsymbol{num\_err\_rec}) \qquad \textbf{(22)}$$

The total number of error reconciliation iterations depends on the number of blocks contained in the error estimated buffer, represented by the following equation:

$$\boldsymbol{num\_err\_rec} = \boldsymbol{ceil}\left(\frac{\{A,B\}_{ErrEst_{buffer}}}{block_{size}}\right) \qquad (5)$$

4.7.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

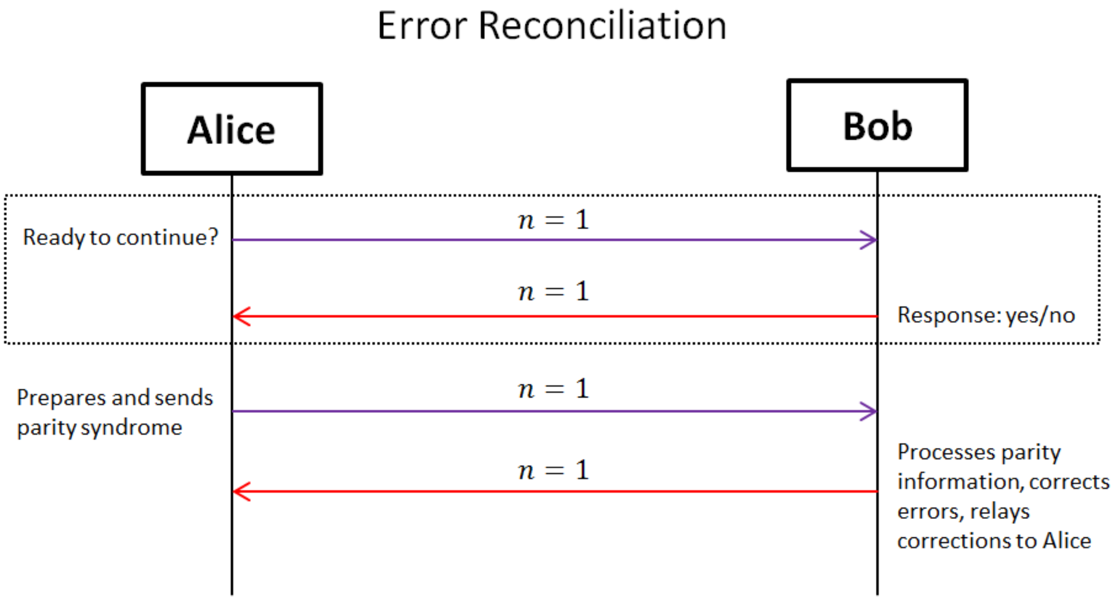

Error Reconciliation
Alice
Bob
Ready to continue?
$n = 1$
$n = 1$
Response: yes/no
Prepares and sends parity syndrome
$n = 1$
$n = 1$
Processes parity information, corrects errors, relays corrections to Alice

Figure 17. Sequence Diagram of Classical Communication During ER

4.7.4 Discussion of Practical Implementation

The timing equation for Error Reconciliation exists at a level of abstraction to represent the three major reconciliation algorithms, namely Cascade, Winnow, and LDPC (Low Density Parity Check). More importantly, it is robust enough to identify the inverse relationship between the number of messages sent and the computational time required for that dataset. For instance, Cascade requires high cost in communications overhead given a relatively large number of messages, which is variable depending on the error rate. It benefits, however, from low computational complexity and necessary memory space for the calculations that need to be performed, requiring 1 to 2 bytes per bit of data to be reconciled [20]. Latencies of a network are generally much higher than those of a CPU, which creates a bottleneck in processing in the case of Cascade. Winnow and LDPC represent the opposite scenario, requiring a very low number of communications and much higher computational time. In addition, necessary memory space for computation requires 20 to 30 bytes of memory per bit of data to be reconciled [20].

A strict sense of time is an important consideration for final throughput given that the algorithm chosen is viable. There are other considerations and trade-offs specific to each algorithm, however, that must not be ignored by the practitioner of a QKD system. For instance, Winnow suffers from the potential to induce errors during processing, and is therefore less effective than Cascade when discussing accuracy in the absence of time. Winnow also requires a *privacy maintenance* step in which a small amount of key is sacrificed during calculation – a step which is absent in both Cascade and LDPC [10]. Cascade also suffers from a higher variability in processing time, given that the number of messages that will be required per iteration is dictated by the error rate. Similarly, for LDPC codes the size of the error correction code and the computation time also depend upon the estimated error rate. The expected window of completion with respect to time would therefore need to be larger to account for that variability.

A study conducted at the National Institute of Standards and Technology (NIST) quantified the speed differential between FPGA implementations of Cascade and LDPC. It found that the throughput for Cascade dropped off from 5 Mbits/s to 3 Mbits/s between 0 and 100 km. LDPC, however, remained closer to a constant 2 Mbits/s [20]. This proved that the performance of Cascade is dictated mainly by the increased latency over the classical channel at greater distances. In short, the algorithm chosen is still highly implementation-specific regardless of its theoretical characteristics. The most intriguing part of the study, however, was that each throughput figure exists on a per-thread basis. That is, LDPC has the potential to run 2 Mbits/s per thread at 100 km. This presents the ability for linear scaling with parallel instantiations, with an upper bound on the thread count limited by the available memory, recalling that the required memory per bit reconciled is an order of magnitude larger for LDPC and Winnow than it is for Cascade.

4.8 Entropy Estimation

4.8.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Entropy Estimation phase:

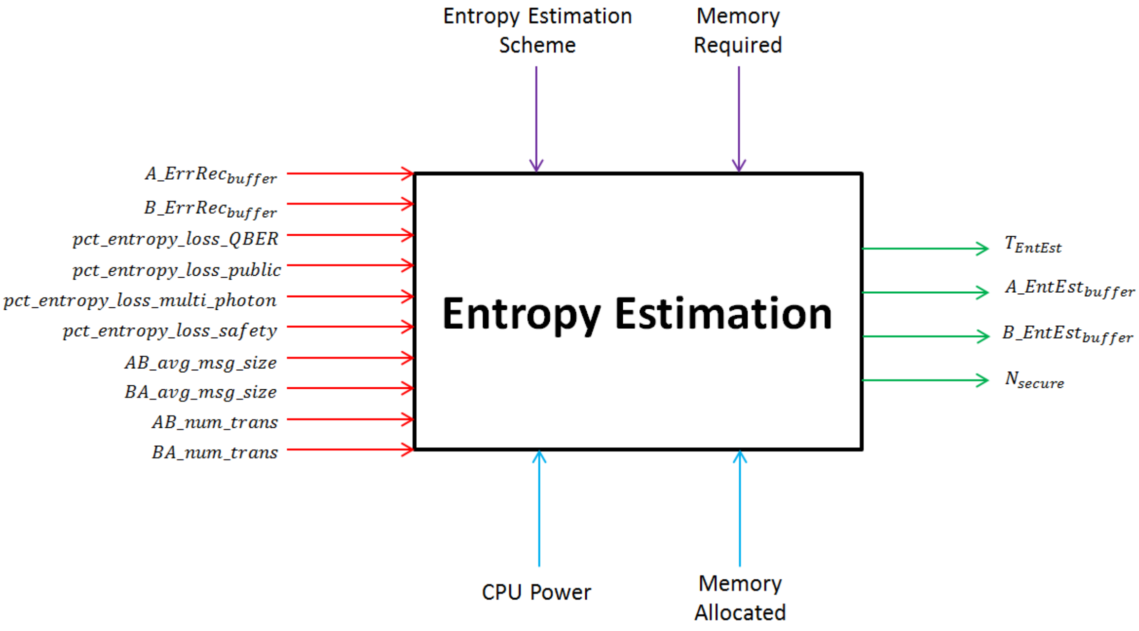


Figure 18. Entropy Estimation ICOM Model

The inputs and outputs local to Authentication are discussed in the following table:

Table 11. Input/Outputs Local to Entropy Estimation

Parameter Name	Units	Definition	Detailed Description
$pct\_entropy\_loss\_QBER$	%	The percentage of key lost due to entropy loss on the quantum channel.	Imperfect transmission on the quantum channel results in the possibilities of errors, all of which are attributed to Eve. Any leaked information as a result of this transmission must be subtracted away from the final key.
$pct\_entropy\_loss\_public$	%	The percentage of key lost due to entropy loss during Error Reconciliation.	Information is leaked during transmission of data across the classical channel during Error Reconciliation. Although these leaks occur in the form of parity information, an estimated amount of information about the key Eve might glean from this information must be subtracted away from the final key.
$pct\_entropy\_loss\_multi\_photon$	%	The percentage of key lost due to multi-photon pulses.	Information is potentially leaked with the presence of multi-photon pulses. Defense against the possibility of a Photon-Number Splitting attack may result in subtracting away the probability of all multi-photon pulses from the final key.
$pct\_entropy\_loss\_safety$	%	The percentage of key lost due to arbitrary safety margin.	An arbitrary amount of information may also be subtracted from the final key as a “safety net” to protect against any unaccounted for entropy loss.
T_{EntEst}	sec	The time required to complete Entropy Estimation.	The time required to complete Entropy Estimation is marginal, give that both Alice and Bob know the entropy estimation routine. Communication across the classical channel may not be required to complete Entropy Estimation.
A_EntEst_{buffer}	bytes	The size of the Entropy Estimated buffer.	The Entropy Estimated buffer will equal the size of the input Error Reconciled buffer. The bit buffers remain unaffected during Entropy Estimation.
B_EntEst_{buffer}	bytes	The size of the Entropy Estimated buffer.	The Entropy Estimated buffer will equal the size of the input Error Reconciled buffer. The bit buffers remain unaffected during Entropy Estimation.
N_{secure}	bytes	The number of bits that can be saved after Privacy Amplification.	The number of secure bits that can be saved post-Privacy Amplification is calculated by the summation of the entropy loss percentages applied to the Error Reconciled buffer.

4.8.2 Abstract Equations

The total time required to estimate the entropy loss is determined by the time it takes to transmit required data across the classical channel in addition to the computational time required, represented by the following equation:

$$\boldsymbol{T_{EntEst} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{EntEst}}}{\{A,B\}_{cpu_{power}}}}$$

$$\boldsymbol{(24)}$$

Considering no bits are lost during entropy estimation, the size of the entropy estimated buffer is identical to the size of the error reconciled buffer, represented by the following equation:

$$\boldsymbol{\{A,B\}_{EntEst_{buffer}} = \{A,B\}_{ErrRec_{buffer}}} \qquad \boldsymbol{(25)}$$

The number of bits that can be saved during the upcoming Privacy Amplification phase can be represented by the percentage of key remaining after losses multiplied to size of the error reconciled buffer, as shown in the following equation:

$$\boldsymbol{N_{secure} = (1 - total\_ent\_loss\_pct) \cdot \{A,B\}_{ErrRec_{buffer}}} \qquad (6)$$

4.8.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

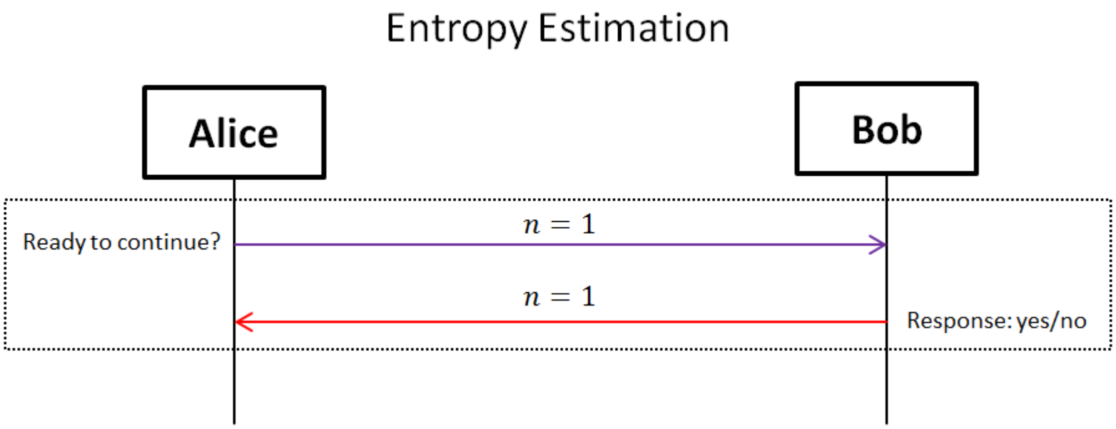


Figure 19. Sequence Diagram of Classical Communication During Entropy Estimation

4.8.4 Discussion of Practical Implementation

Although the execution requirements for this phase are trivial, the implications for the security of the overall system are significant. Time and space requirements for Entropy Estimation are minimal. The N_{secure} processing is at most an O(1) operation [3]. It is assumed that the inputs into the algorithmic calculation have been gleaned and retained from prior phases.

Earlier in this thesis we claimed that QKD as a distribution system is markedly different than public-key distribution methods. This notion becomes most apparent when discussing the representation of security in the form of a proof. Recent research has suggested that the catchall nature of proofs in conventional cryptography, in which Eve's actions are confined to the limits of mathematical processing, do not apply to QKD [11]. This stems from the fact that the exchange of quantum material is probabilistic, and as a result relies on the system's practitioner to estimate the entropy loss, or how much information Eve may have gleaned during Quantum Exchange. It is unclear if it is possible to know whether or not the amount of entropy accounted for is indeed provably secure, because unlike in conventional cryptography, the physical components (i.e. design choices of hardware) influence the theoretical security. It is therefore necessary to recognize the sources of entropy loss and its causes in order to generate a sufficient estimate of entropy loss.

Information leakage presents itself in three key areas [12, 23]: the presence of multi-photon pulses during Quantum Exchange, disclosure of block checksums or parity bits during Error Reconciliation (also known as the public discussion), and any eavesdropping that occurs on the quantum channel. Since it is impossible to distinguish between channel noise and eavesdropping, most security analyses make the conservative attribution of any detected errors to the presence of Eve. Of the few recognized entropy estimates based on information theory that exist, none take into account all three sources of entropy. The original BBBSS92 estimate from Bennett, et al. [27], Slutsky et al.'s Defense Frontier Analysis [12], and the Myers-Pearson estimate [23] are mostly concerned with information learned from measuring single-photon pulses, which is "intimately related to the observed error rate" [23]. Furthermore, within the singularly focused source of entropy, Eve is assumed to be conducting individual attacks in which she measures photons independently. The disregarded alternative, known as a coherent attack, occurs when Eve can measure several photons at the same time, which is infeasible without the advent of quantum memory.

The other two sources of information leakage, multi-photon pulses and public discussion, are not entirely ignored but rather account for their miscalculation in what Slutsky et al. refer to as an "arbitrary safety margin," which is subtracted from the final estimate for single-photon pulses to create an N_{secure} number of bits that can be used as final key. The focus on a perceived error rate as the primary source of entropy stems from the fact that a “combination of a low error rate and high information leakage is unlikely no matter what strategy the eavesdropper uses” [12]. The amount safety margin added is at the discretion of the practitioner to ensure a desired level of security. Higher estimates may be desired for added security. However, this will result in a smaller number of final bits, more Privacy Amplification, and a reduced final key rate. A balance must be achieved between security requirements and desired final key rate with the stipulation that the input QBER to Entropy Estimation be no greater than 11%, which is the point at which the Shannon entropy reaches zero [1]. In other words, an upper bound under which the QKD system remains provably secure.

The final number of N_{secure} bits can be derived with the following equation [13]:

$$\boldsymbol{N_{secure} = N_{corr}(1 - h_2(QBER)) - N_{leaked} - N_{multi} - N_{safety}} \qquad (7)$$

N_{secure} = final number of usable secure bits after Privacy Amplification

N_{corr} = number of bits after Error Reconciliation

$h_2(x) = -x \cdot \log(x) - (1 - x) \cdot \log\,(1 - x)$, the binary entropy function

QBER = actual Quantum Bit Error Rate

N_{leaked} = number of bits revealed during Error Reconciliation

N_{multi} = number of bits that must be discarded due to multi-photon pulses

N_{safety} = number of bits to be used as an arbitrary safety margin

In this model we chose not to neglect entropy loss as a result of public discussion. Although entropy loss cannot be directly estimated without running a real implementation of Error Reconciliation, we can instead use the Shannon bound for the minimum amount of information that will be revealed during an error correction routine. In fact, it is not possible to perform error correction without exposing information. The ratio between number of bits, N, needed to correct a sifted key of length η is given by [43]:

$$\boldsymbol{\frac{N}{\eta} = -e \cdot log_2(e) - (1-e) \cdot log_2(1-e)} \qquad (8)$$

where e is the observed error rate in the sifted key. This ratio is then multiplied to the number of bits in the sifted key, and reveals the amount of information that must be discarded. As long as we assume that an error correction routine is being used that is close to the Shannon bound, then use of a conservative safety margin should make the loss estimate a reasonably close approximation to the losses in a real system.

To estimate the entropy loss as a result of multi-photon pulses, we chose to implement a revised version of the conservative BBBSS92 estimate [27], which states that the percentage of discarded signal bits is equal to the probability of a multi-photon pulse. This probability can be described with the Poisson distribution [21]

$$\boldsymbol{m = 1 - \frac{\mu e^{-\mu}}{1-e^{-\mu}}} \qquad (9)$$

where μ is the Mean Photon Number. The irony of QKD is that although it is supposed to offer provable information-theoretic security in conjunction with a One-Time Pad, the existence of non-idealities in the system (i.e. lack of single photon source or detection, channel loss, etc.) make it evident that “the security that can be achieved is not absolute but probabilistic in nature” [12].

As discussed, this probabilistic security begins during Quantum Exchange, in which a Mean Photon Number is selected. We can increase system throughput by simply increasing the MPN, but as a consequence must alter our Entropy Estimate, and by implication the amount of Privacy Amplification incurred in order to maintain the same level of security. In fact, the optimal MPN value, which will provide the highest levels of both key rate and security, depends on the optical losses in the channel and on assumptions about Eve's technology [29]. By altering the arbitrary safety margin in the so-called entropy "defense function" to account for any new information Eve may have learned, it may be possible to safely operate with a much higher MPN. Specifically, we must account for an increase in the possibility for multi-photon pulses, which we recall are not directly considered in the standard entropy estimation functions. Pearson & Elliott have come to the conclusion that although there are numerous factors that affect key rate (most of which have been highlighted in this paper), that regardless of which initial entropy estimate is chosen (BBBSS92, Slutsky's Defense Frontier, Myers-Pearson estimate, etc.), the ideal μ value "will always be slightly over 1.0" – this translates into a ten-fold increase in system throughput compared to an MPN of 0.1 [21].

4.9 Privacy Amplification (PA)

4.9.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Privacy Amplification phase:

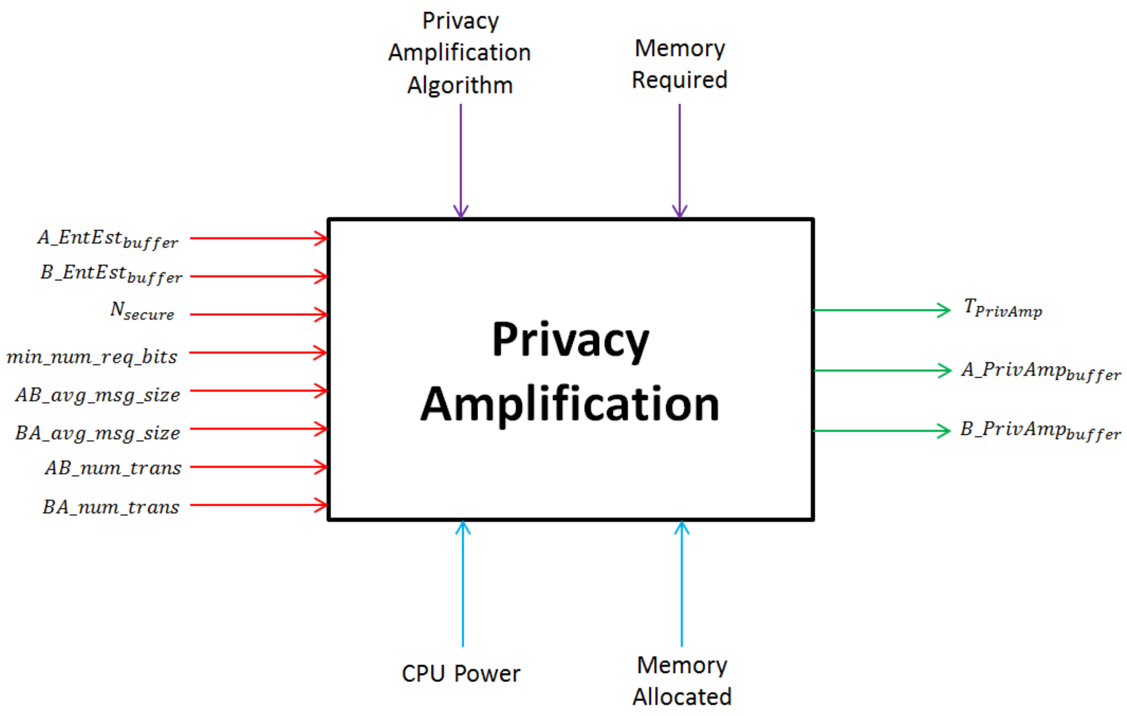


Figure 20. Privacy Amplification ICOM Model

The inputs and outputs local to Authentication are discussed in the following table:

Table 12. Input/Outputs Local to Privacy Amplification

Parameter Name	Units	Definition	Detailed Description
$min\_num\_req\_bits$	bits	The minimum number of required bits necessary to perform Privacy Amplification.	To reduce finite key size effects, it is recommended that the input to Privacy Amplification bet at least 1 Mbit [38].
N_{secure}	bytes	The number of bits that can be saved after Privacy Amplification.	The number of secure bits that can be saved post-Privacy Amplification is calculated by the summation of the entropy loss percentages applied to the Error Reconciled buffer.
$T_{PrivAmp}$	sec	The time required to complete Privacy Amplification.	The time required to complete Privacy Amplification is dictated mainly by computational processing power to do mathematical operations on large numbers.
$A\_PrivAmp_{buffer}$	bytes	The size of Alice's Privacy Amplified buffer.	The size of the Privacy Amplified buffer will be equal to the size of N_{secure}. More specifically, N_{secure} bits will be selected from the Privacy Amplified buffer after the computing the Universal-2 hash algorithm.
$B\_PrivAmp_{buffer}$	bytes	The size of Bob's Privacy Amplified buffer.	The size of the Privacy Amplified buffer will be equal to the size of N_{secure}. More specifically, N_{secure} bits will be selected from the Privacy Amplified buffer after the computing the Universal-2 hash algorithm.

4.9.2 Abstract Equations

The total time required to privacy amplify the entropy estimated key buffer is determined by the time it takes to transmit required data across the classical channel in addition to the computational time required, represented by the following equation:

$$\boldsymbol{T_{PrivAmp} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{PrivAmp}}}{\{A,B\}_{cpu_{power}}}} \qquad (10)$$

The size of the privacy amplified buffer will now be equal to the number of bits that can be saved as a result of entropy estimation, represented by the following equation:

$$\boldsymbol{\{A,B\}_{PrivAmp_{buffer}} = N_{secure}} \qquad (11)$$

4.9.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

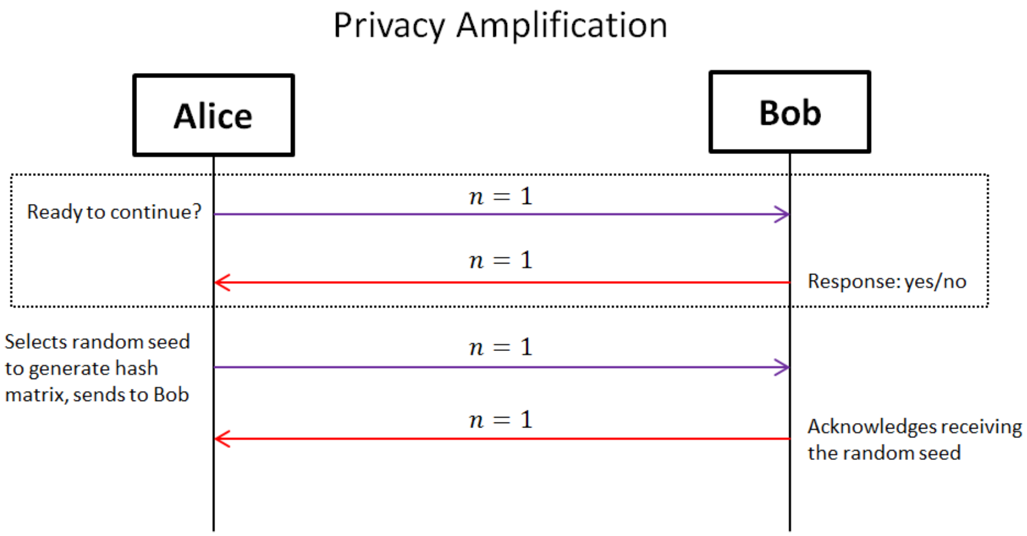

Privacy Amplification
Alice
Bob
Ready to continue?
$n = 1$
$n = 1$
Response: yes/no
Selects random seed to generate hash matrix, sends to Bob
$n = 1$
$n = 1$
Acknowledges receiving the random seed

Figure 21. Sequence Diagram of Classical Communication During PA

4.9.4 Discussion of Practical Implementation

The completion time of Privacy Amplification, much like Error Reconciliation, is highly dependent on its implementation and the specific algorithm chosen. The goal of Privacy Amplifying a key is to apply the leftover hash lemma to the error reconciled key. The lemma states that if there exists an n-bit secret key X of which Eve knows $t < n$ bits, we can produce a key of approximately $n - t$ bits of which Eve has no knowledge, if Alice and Bob were to randomly and secretly select a one-way hash function $h \in H$ family of hash functions [24].

By convention, in QKD a hash family is chosen that exhibits 2-universality. That is, a 2-universal hash function is one which the probability of collision between two distinct input keys is $\frac{1}{m}$ for every possible function $h \in H$, where m is the number of possible outputs of the hash function h [15]. More formally, a family of hash functions mapping A to B is said to be 2-universal if [14]:

$$\mathbf{Pr}\{\boldsymbol{h}: \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{y})\} \leq \frac{\mathbf{1}}{|\boldsymbol{B}|} \quad \text{for all } \boldsymbol{x} \neq \boldsymbol{y}, \quad (12)$$

where *h* is a randomly chosen hash function. There are several well-known classes of functions that exhibit such behavior suitable for use as a hash family, including multiplication of binary matrices (specifically Toeplitz matrices), modular affine transforms, multiplication in finite fields, and multiplication in binary fields [16]. The Privacy Amplification phase contains two parts: computation of the privacy amplified bit buffer, and the selection of final secure bits. The computation performs the one-way 2-universal hash function previously described, and the final secure bit selection uses the output of the Entropy Estimation phase to select the least-significant N_{secure} bits from the bit buffer.

As an example, we will select the first 2-universal hash function described by Wegman & Carter, in the form of a modular affine transform [15]:

$$\boldsymbol{K_{final}} = (\boldsymbol{m} \cdot \boldsymbol{K_{corrected}} + \boldsymbol{n})\ \boldsymbol{mod}\ \boldsymbol{p} \qquad (13)$$

K_{final} = key buffer from which final secure key will be selected

$K_{corrected}$ = error corrected key buffer

m,n = large composites < p, where $m \neq 0$

p = large prime ≥ $K_{corrected}$

The calculation of K_{final} technically has O(1) computational complexity, as does Entropy Estimation. However, the size of the $K_{corrected}$ buffer is on the order of hundreds of thousands (if not over a million) bits, which makes the seemingly straightforward calculation much more computationally intensive due to the sheer size of the numbers. Alice and Bob must also first agree, randomly and secretly, on the input parameters *m*, *n*, and *p* across the classical channel, which adds to the time through the phase. Although the class of functions can be publicly known, in this case affine transformations, the selection of a specific hash function from the hash family must be kept secret for it to remain universal [24].

At the output of Privacy Amplification there will indeed be a shorter, more secure key, but the hashing algorithm itself does not perform the reduction. The hash function will return a bit buffer, K_{final}, roughly the same size as the input buffer, $K_{corrected}$. It is not until we select the least significant N_{secure} bits from this buffer that we possess a privacy amplified key. Therefore the throughput implications come from the estimated loss of entropy in the previous phase and not from Privacy Amplification itself. Perhaps the only trade-off in system performance during this phase is the selection and implementation of the algorithm. Any hash family, so long as it is proven 2-universal, can be used. Some hash families, such as Toeplitz matrix hashing [14], will naturally have better computability than others, and may be more desirable for implementations in which the speed of post-processing is a critical factor.

4.10 Final Key Generation

4.10.1 ICOM Model

The following ICOM model depicts the inputs, outputs, controls, and mechanisms relevant to the Final Key Generation phase:

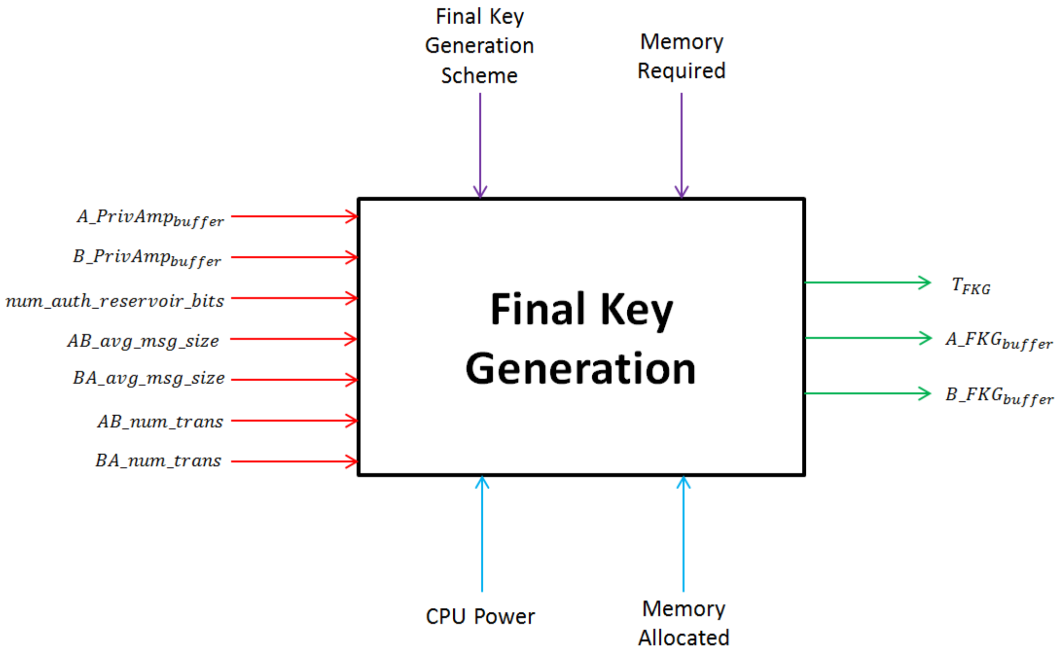


Figure 22. Final Key Generation ICOM Model

The inputs and outputs local to Authentication are discussed in the following table:

Table 13. Input/Outputs Local to Final Key Generation

Parameter Name	Units	Definition	Detailed Description
$num\_auth\_reservoir\_bits$	bits	The number of bits reserved for the Authentication reservoir.	A small number of final key bits must be reserved for Authentication of the next round of QKD.
T_{FKG}	sec	The time required to complete Final Key Generation.	The total time required to complete FKG involves the computational time to compute the hash of the final key by Alice and Bob, and the comparison of those hashed across the classical channel.
A_FKG_{buffer}	bytes	The size of Alice's Final Key Generation buffer.	The size of the final key buffer is the size of the Privacy Amplified buffer after the reduction of reserved bits for the Authentication reservoir.
B_FKG_{buffer}	bytes	The size of Bob's Final Key Generation buffer.	The size of the final key buffer is the size of the Privacy Amplified buffer after the reduction of reserved bits for the Authentication reservoir.

4.10.2 Abstract Equations

The total time required to privacy amplify the entropy estimated key buffer is determined by the time it takes to transmit required data across the classical channel in addition to the computational time required, represented by the following equation:

$$\boldsymbol{T_{FKG} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}})} + \frac{\boldsymbol{\{A,B\}_{workload_{FKG}}}}{\boldsymbol{\{A,B\}_{cpu_{power}}}} \quad (14)$$

The size of the final key buffer will be equal to the size of the privacy amplified buffer with a slight reduction due to the need to reserve bits for the authentication reservoir, as described in the following equation:

$$\boldsymbol{\{A,B\}_{FKG_{buffer}} = \{A,B\}_{PrivAmp_{buffer}} - num\_auth\_reservoir\_bits} \qquad (15)$$

4.10.3 Sequence Diagram

The following sequence diagram describes the classical communications necessary to execute the phase, where *n* signifies the number of transactions required, the purple arrow represents communication initiated by Alice, and the red arrow represents communication initiated by Bob:

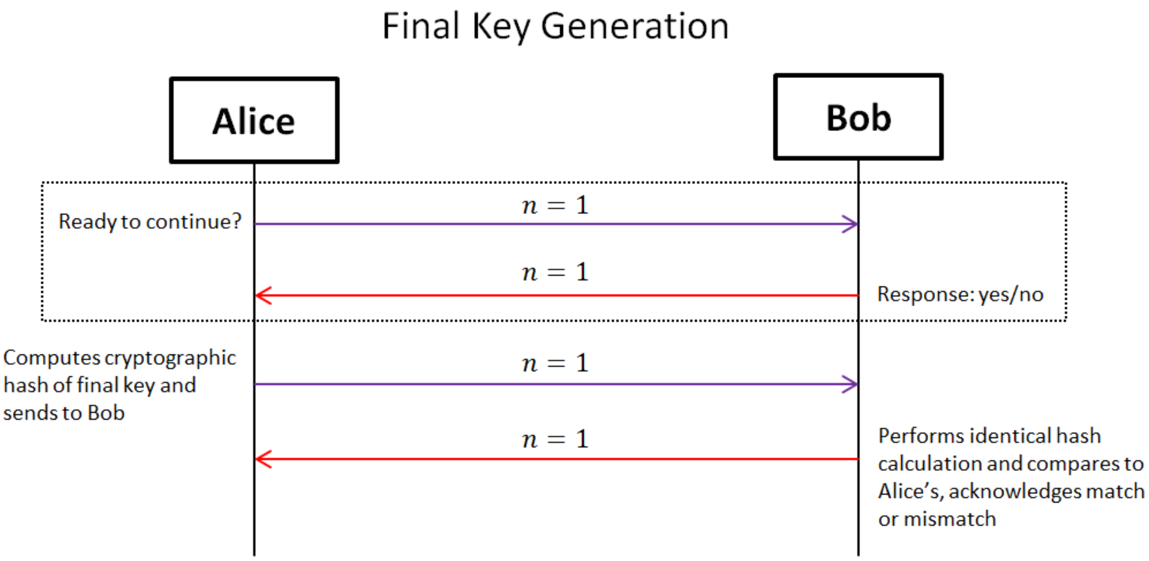

Final Key Generation
Alice
Bob
Ready to continue?
$n = 1$
$n = 1$
Response: yes/no
Computes cryptographic hash of final key and sends to Bob
$n = 1$
$n = 1$
Performs identical hash calculation and compares to Alice's, acknowledges match or mismatch

Figure 23. Sequence Diagram of Classical Communication During FKG

4.10.4 Discussion of Practical Implementation

Although Error Reconciliation's purpose is to provide matching key buffers between Alice and Bob with very high probability, there is still the possibility of masked errors existing in one or more key buffers. Even if only one bit is incorrect, the final key is useless. To protect against this scenario, standard practice is to hash the final key buffer after Privacy Amplification to make sure that keying material held by Alice and Bob does indeed match. Bennett & Brassard suggest conducting this step during Privacy Amplification, comparing the pre-privacy amplified key buffer, before the final reduction [17]. The practice of using a 2-universal hash function, however, is not cryptographic in nature, but sufficient for only randomizing the input [25].

The other problem with this approach is the use of the final key as input to the hash function. This effectively reduces the security of the QKD system to the security of the hashing algorithm itself. To protect against this security reduction, we suggest an alternative hashing routine which doesn't include the use of the final key directly. Since the goal of a final hash is to prove all bit errors were corrected during the Error Reconciliation phase, the errors are assumed to be corrected at the conclusion of that phase. In fact, the error corrected buffers are assumed to be identical; otherwise Privacy Amplification would be fruitless. It would then be beneficial to use the error corrected key buffer as the input to the final hash. If their comparison over the classical channel matches, it is assumed the final secure key will also match if Alice and Bob perform identical mathematical operations on the key thereafter. If the hashing is done in this way, even if the error corrected key buffer is recovered from a one-way, computationally-secure hash algorithm, an adversary would additionally require specific details of the algorithm chosen during Privacy Amplification as well an accurate knowledge of Alice and Bob's entropy loss estimation routine.

In addition, it may be beneficial to add additional safety margin to our entropy loss estimate, if desired. It is unclear if and how much entropy is lost during the conversion from a message to message digest in the application of a cryptographic hash function [25]. For this reason, an entropy loss adjustment can be viewed as adding additional arbitrary safety margin rather than introducing another variable in the calculation of N$_{secure}$ (refer to Figure 3):

$$\boldsymbol{N_{secure}} = \boldsymbol{N_{corr}}(\boldsymbol{1} - \boldsymbol{h_2}(\boldsymbol{QBER})) - \boldsymbol{N_{leaked}} - \boldsymbol{N_{multi}} - \boldsymbol{N_{hash}} - \boldsymbol{N_{safety}} \qquad (16)$$

We can conservatively assign N$_{hash}$ to be the number of output bits of the chosen cryptographic hash function. Again, the consequence of increasing the amount of information Eve might know results in a reduction of final keying material.

V. Baseline Configuration and Use Case

5.1 Introduction

In this chapter a baseline configuration is introduced that is used to study the relevant research questions presented in Chapter I. First, a list of modeling assumptions is made about implementing a practical QKD system in the model. Next, the parameter values that make up the baseline configuration are introduced and their resulting performance is discussed. Finally, the research questions pertinent to QKD performance are answered using the model and an analysis is provided of the results.

5.2 Modeling Assumptions of Practical QKD

In the representation of the configuration used in this model, we make the following assumptions about the implementation of a practical Quantum Key Distribution system:

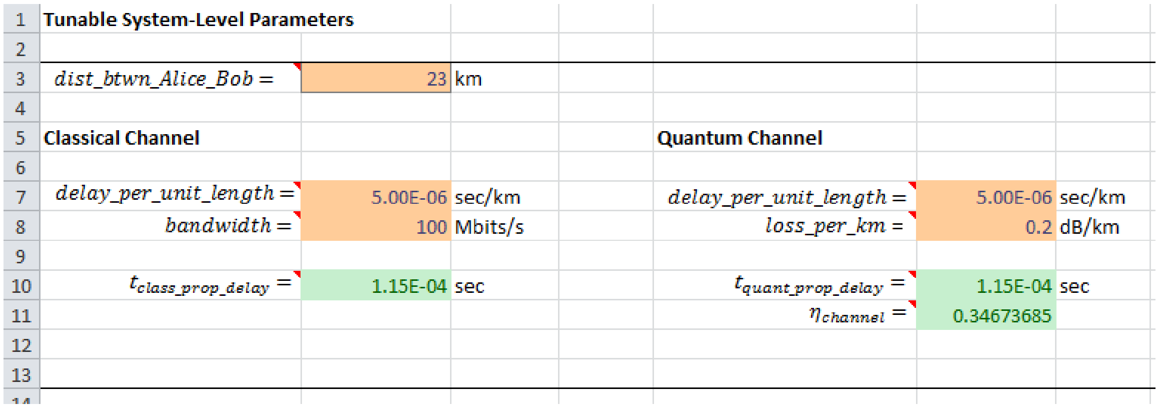
1) The quantum channel and classical channel are both optical fiber of the same length.
2) Communication over the classical channel is accomplished using the TCP/IP protocol with a maximum transmission unit (MTU) of 1500 bytes.
3) The propagation delay over both the quantum and classical channels is two-thirds the speed of light [40].
4) The bandwidth of the classical channel is 100 Mbits/s [45].
5) Alice and Bob share the identical processing power and total memory size.
6) The protocol efficiency fraction for BB84 is 0.5.
7) Authentication is done once per round of QKD.
8) Authentication is done using a Wegman-Carter ϵ-Almost Strongly Universal$_2$ hash function [8].
9) Alice ceases Quantum Exchange when her allocated QE memory is filled.
10) Weak coherent optical pulses arriving at Bob are uniformly spaced.
11) The information required for each pulse can be stored in 5 bytes for both Alice and Bob.
12) A maximum message size (MTU) is 1500 bytes.
13) The LDPC error correction algorithm is being used with a standard matrix that has a required block size of 54000 bits [20].

14) The memory overhead required to perform the LDPC algorithm is 30 bytes per bit [20].
15) LDPC can be performed with a number of bits less than the block size if it is padded to meet the block size requirement [20].
16) No communication is necessary to complete Entropy Estimation [20].
17) The minimum number of required bits as an input to Privacy Amplification is 1 Mbit [38].
18) If an insufficient amount of key is generated to conduct Privacy Amplification, Quantum Exchange is restarted.
19) Privacy Amplification implements a Strongly Universal$_2$ hash function using a Toeplitz matrix.
20) The matrix required to do Privacy Amplification hashing is approximately twice the size of the input to the Privacy Amplification phase [14].
21) A cryptographic hash function, SHA-512, is being used to perform Final Key Generation.
22) The minimum amount of memory necessary to perform small operations (e.g., the computation of a cryptographic hash) is 1024 bytes.

5.3 Baseline Configuration

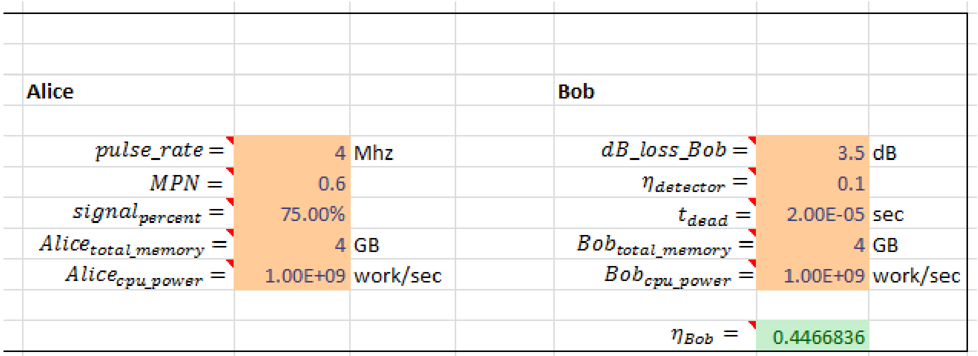
5.3.1 System-Level Parameters

In this model we chose to implement the so-called “Chen” as a baseline for conducting simulations and answering research questions [18]. In particular we attempted to model the Binhu-USTC link of the three-node network. The following figures show key performance parameters for our implementation:



1	**Tunable System-Level Parameters**					
2						
3	$dist\_btwn\_Alice\_Bob =$	23	km			
4						
5	**Classical Channel**			**Quantum Channel**		
6						
7	$delay\_per\_unit\_length =$	5.00E-06	sec/km	$delay\_per\_unit\_length =$	5.00E-06	sec/km
8	$bandwidth =$	100	Mbits/s	$loss\_per\_km =$	0.2	dB/km
9						
10	$t_{class\_prop\_delay} =$	1.15E-04	sec	$t_{quant\_prop\_delay} =$	1.15E-04	sec
11				$\eta_{channel} =$	0.34673685	
12						
13						

Figure 24. System-Level Parameters in Model, Part 1



Alice			**Bob**		
$pulse\_rate =$	4	Mhz	$dB\_loss\_Bob =$	3.5	dB
$MPN =$	0.6		$\eta_{detector} =$	0.1	
$signal_{percent} =$	75.00%		$t_{dead} =$	2.00E-05	sec
$Alice_{total\_memory} =$	4	GB	$Bob_{total\_memory} =$	4	GB
$Alice_{cpu\_power} =$	1.00E+09	work/sec	$Bob_{cpu\_power} =$	1.00E+09	work/sec
			$\eta_{Bob} =$	0.4466836	

Figure 25. System-Level Parameters in Model, Part 2

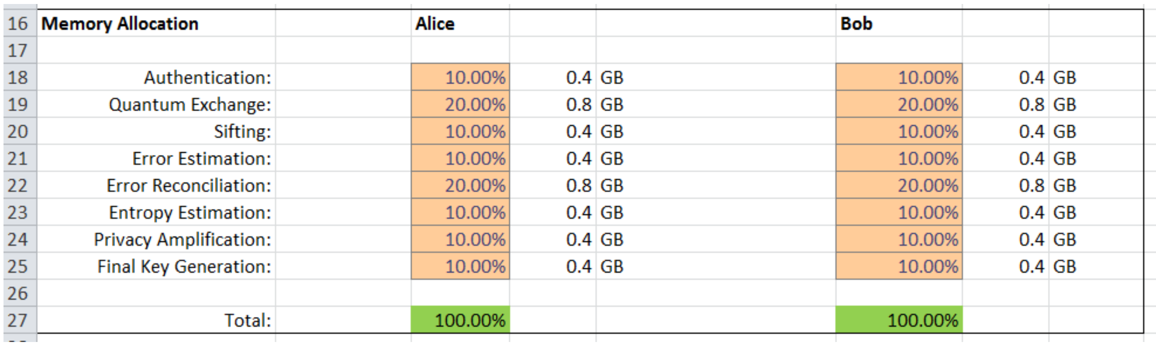
The following table describes the chosen parameter values for our implementation of the Chen configuration:

Table 14. System-Level Parameters of Chen configuration

Parameter Name	Units	Value	Justification
$dist\_btwn\_Alice\_Bob$	km	23	Although the Binhu-USTC link is technically 20 km, we chose a value of 23 km in order to match the efficiency of the quantum channel, which is measured in Chen as a "total attenuation" of 4.5 dB [18]. Chen specifically states the loss to be 0.2 dB/km, which equates to 4 dB of loss. To meet the total attenuation loss, we added 3 km to the total distance.
$delay\_per\_unit\_length$	sec/km	5.00E-06	Two-thirds the speed of light.
$bandwidth$	Mbits/s	100	The Fast Ethernet standard bandwidth.
$loss\_per\_km$	dB/km	0 2	Described by the Chen configuration [18].
$pulse\_rate$	Mhz	4	Described by the Chen configuration [18].
MPN	unitless	0.6	Described by the Chen configuration [18].
$signal_{percent}$	%	75	Described by the Chen configuration [18].
$Alice_{total\_memory}$	GB	4	Discussions with Subject-Matter Experts yielded values for Alice and Bob's total memory.

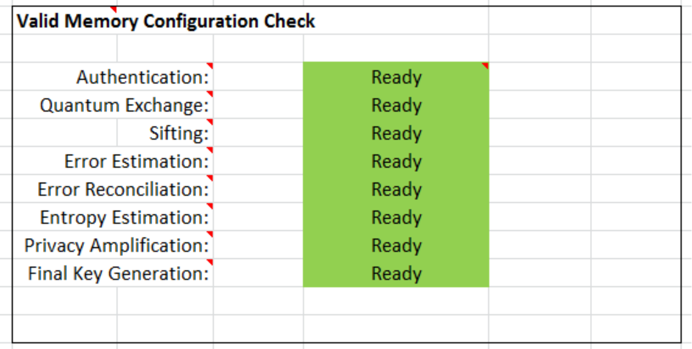
$Alice_{cpu\_power}$	work/sec	1.00E+09	Discussions with Subject-Matter Experts yielded values for Alice and Bob’s CPU power.
$dB\ loss\ Bob$	dB	3 5	Described by the Chen configuration [18].
$\eta_{detector}$	unitless	0 1	Described by the Chen configuration [18].
t_{dead}	sec	2.00E-05	Described by the Chen configuration [18].
Bob_{total_memory}	GB	4	Discussions with Subject-Matter Experts yielded values for Alice and Bob’s total memory.
Bob_{cpu_power}	work/sec	1.00E+09	Discussions with Subject-Matter Experts yielded values for Alice and Bob’s CPU power.

The following figures show the allocation of Alice and Bob’s memory to each of the eight QKD phases and the status of the validity of the configuration:



16	**Memory Allocation**	**Alice**			**Bob**		
17							
18	Authentication:	10.00%	0.4	GB	10.00%	0.4	GB
19	Quantum Exchange:	20.00%	0.8	GB	20.00%	0.8	GB
20	Sifting:	10.00%	0.4	GB	10.00%	0.4	GB
21	Error Estimation:	10.00%	0.4	GB	10.00%	0.4	GB
22	Error Reconciliation:	20.00%	0.8	GB	20.00%	0.8	GB
23	Entropy Estimation:	10.00%	0.4	GB	10.00%	0.4	GB
24	Privacy Amplification:	10.00%	0.4	GB	10.00%	0.4	GB
25	Final Key Generation:	10.00%	0.4	GB	10.00%	0.4	GB
26							
27	Total:	100.00%			100.00%		

Figure 26. Allocation of Alice/Bob's Memory Per Phase



Valid Memory Configuration Check	
Authentication:	Ready
Quantum Exchange:	Ready
Sifting:	Ready
Error Estimation:	Ready
Error Reconciliation:	Ready
Entropy Estimation:	Ready
Privacy Amplification:	Ready
Final Key Generation:	Ready

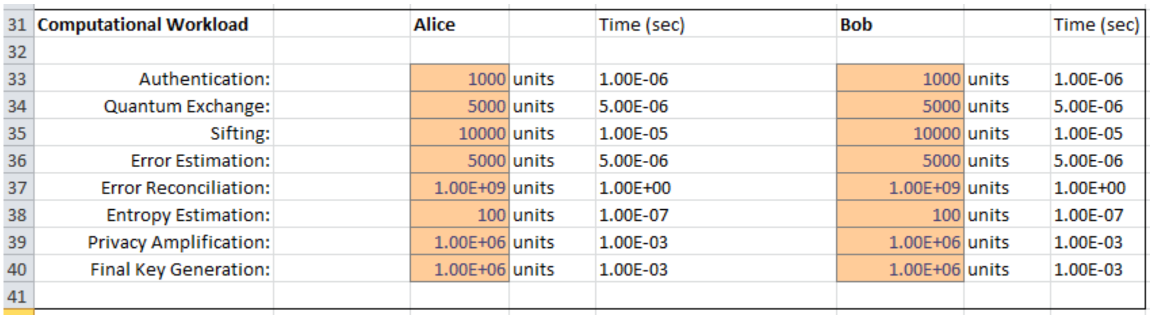
Figure 27. System Memory Configuration Check

We chose to allocate 10% of Alice and Bob’s total system memory to each of the phases, except for the more resource intensive Quantum Exchange and Error Reconciliation, which received 20%. The Valid Configuration Memory Check detects, via backend logic written in Visual Basic, if there is enough memory allocated to the phase for it to proceed based on all of the values that are currently implemented in the model. The following table describes the backend logic that is being evaluated in code:

Table 15. Valid Configuration Memory Check Logic

Phase	Evaluation Logic
Authentication	Authentication memory must be large enough to store the Auth key reservoir in addition to a very small amount of memory required for small operations, default 1024 bytes.
Quantum Exchange	Alice's memory must be at least large enough to hold the information required for a single pulse, otherwise QE can't run. Bob's memory must at least be larger than the ratio of sent pulses at Alice to detections at Bob, in relation to Alice's memory.
Sifting	Sifting memory must be at least large enough to hold the size of the sifted key buffers after the sifting efficiency fraction has been applied.
Error Estimation	Error Estimation memory requires only a small amount of memory necessary for small operations, default 1024 bytes.
Error Reconciliation	Error Reconciliation requires enough memory to receive the bit buffer from Error Estimation after key sacrifice, plus 30 bytes of computational memory overhead (NIST numbers for efficient FPGA implementation [20]) per bit of the block size.
Entropy Estimation	Error Estimation memory requires only a small amount of memory necessary for small operations, default 1024 bytes.
Privacy Amplification	Checks if the allocated memory for PA is at least twice the size of the input buffer. To reduce finite key size effects, the input to PA must be large - on the order of 1 Mbit or more. The matrices used in the hash calculation must be the same size as the input buffer, or very close to it, to obtain a privacy amplified key.
Final Key Generation	In this model we assume that a cryptographic hash is being used during final key generation, as opposed to a non-cryptographic Wegman-Carter style hash function that is more computationally intensive. For more information, see Chapter IV of thesis. The memory required must therefore be large enough to process a hash value using a standard hash function, such as SHA-512. Additionally it requires memory overhead for small computation, default 1024 bytes.

The following figure shows the computational workload assigned to each phase of the QKD system:



31	Computational Workload	Alice		Time (sec)	Bob		Time (sec)
32							
33	Authentication:	1000	units	1.00E-06	1000	units	1.00E-06
34	Quantum Exchange:	5000	units	5.00E-06	5000	units	5.00E-06
35	Sifting:	10000	units	1.00E-05	10000	units	1.00E-05
36	Error Estimation:	5000	units	5.00E-06	5000	units	5.00E-06
37	Error Reconciliation:	1.00E+09	units	1.00E+00	1.00E+09	units	1.00E+00
38	Entropy Estimation:	100	units	1.00E-07	100	units	1.00E-07
39	Privacy Amplification:	1.00E+06	units	1.00E-03	1.00E+06	units	1.00E-03
40	Final Key Generation:	1.00E+06	units	1.00E-03	1.00E+06	units	1.00E-03
41							

Figure 28. Computational Workload Assigned to Each Phase

Typically computational processing is measured in instructions per second, where a number of instructions are defined as the amount of work (an instruction is a single operation of the processor, which is defined by the processor's instruction set). In this model, we measure processing as units of work (rather than number of instructions) in order to reach a level of abstraction removed from any specific architecture. Ideally, we would run benchmark tests on the phases of operation in a real system to establish a baseline of work units required for each phase. Given that measurements for work attributed to a phase for nonspecific architectures are not present in QKD literature, we have discussed the appropriate values for computational workload with subject-matter experts and decided on appropriate values for each phase. The relative ratios between work assigned for each phase is more important than the numbers themselves (e.g., the work assigned to Authentication is much less than more resource intensive phases such as Error Reconciliation).

5.3.2 Phase Input Parameters

The following tables describe the input parameters chosen for each of the eight phases:

Authentication

Table 16. Input Parameters Local to Authentication

Parameter Name	Units	Value	Justification
$auth\_reservoir_{size}$	bytes	100	The Authentication reservoir must be large enough to authenticate an entire round of QKD. Since we are assuming that authentication is done once per round, 1250 bytes is more than sufficient.
$auth\_key\_req$	bytes	10	The amount of authentication key required from the reservoir to complete a single authentication. In practical QKD using Wegman-Carter style authentication, this is the amount of auth key required to authenticate a single message. Since authentication uses the ASU-2 version of Wegman-Carter universal hash families, the amount of key required for authentication can be on the order of only several bytes, if necessary. In general, the amount of key needed to authenticate a message is log$_2$(msg size).
$AB\_avg\_msg\_size$	bytes	1490	If the MTU of a message is approximately 1500 bytes and the amount of authentication key required is approximately log$_2$(msg_size), then 1490 should be sufficient size to carry the message to be authenticated with an appended authentication tag.
$BA\_avg\_msg\_size$	bytes	10	The log$_2$(1490) is approximately 10 bytes.
AB_num_trans	unitless	1	Wegman-Carter authentication only requires a single message to be passed from Alice to Bob: the message to be authenticated with the appended authentication tag.
BA_num_trans	unitless	1	Bob is only required to respond to Alice if the message she sent was authentic or not, based on his own calculation of the authentication tag.

Quantum Exchange

Table 17. Input Parameters Local to Quantum Exchange

Parameter Name	Units	Value	Justification
mem_req_pulse	bytes	5	Bit and basis can be stored as 2 separate bits. The timing information, assuming it consists of a frame and slot number, can perhaps be stored as an unsigned long, which is 4 bytes. Thus 5 total bytes should be sufficient.

Sifting

Table 18. Input Parameters Local to Sifting

Parameter Name	Units	Value	Justification
$sifting\_eff\_frac$	unitless	0.5	The BB84 protocol defines the efficiency fraction to be approximately one half.
$AB\_avg\_msg\_size$	bytes	1500	Assuming sifting is accomplished by Bob sending his basis measurements to Alice, she must respond with either which measurements are correct or incorrect. This is approximately half the amount of information that Bob sent to Alice. Representation of this scenario is possible in several different ways: Alice sends back the same number of messages that Bob sent, but half the size, or she sends back half the number of messages Bob sent but of the same size. Since a high volume of data (relative to the other phases) must be transmitted, we can set the message size to be a full MTU (Maximum Transmission Unit),

			which is 1500 bytes for the TCP/IP protocol.
$BA\_avg\_msg\_size$	bytes	1500	Assuming sifting is accomplished by Bob sending his basis measurements to Alice, she must respond with either which measurements are correct or incorrect. This is approximately half the amount of information that Bob sent to Alice. Representation of this scenario is possible in several different ways: Alice sends back the same number of messages that Bob sent, but half the size, or she sends back half the number of messages Bob sent but of the same size. Since a high volume of data (relative to the other phases) must be transmitted, we can set the message size to be a full MTU (Maximum Transmission Unit), which is 1500 bytes for the TCP/IP protocol.
AB_num_trans	unitless	1501	In the memory required per pulse, we claimed that bit and basis information can be captured in 2 bits. If Bob must send his basis information to Alice, he must send data approximately equal to the size of his pre-sifting bit buffer, to include timing information for Alice to compare against. We can therefore divide the raw bit buffer by the chosen average message size to get an approximate number of transactions required for Bob→Alice, and half of that number for transactions required for Alice→Bob.
BA_num_trans	unitless	3002	In the memory required per pulse, we claimed that bit and basis information can be captured in 2 bits. If Bob must send his basis information to Alice, he must send data approximately equal to the size of his pre-sifting bit buffer, to include timing information for Alice to compare against. We can therefore divide the raw bit buffer by the chosen average message size to get an approximate number of transactions required for Bob→Alice, and half of that number for transactions required for Alice→Bob.

Error Estimation

Table 19. Input parameters Local to Error Estimation

Parameter Name	**Units**	**Value**	**Justification**
$bits\_sacrificed\_pct$	%	25	The percentage of bits sacrificed must be good enough to get an accurate error rate, but small enough for sufficient throughput. We chose to sacrifice 25% of the key.
$AB\_avg\_msg\_size$	bytes	1500	Assuming Alice initiates Error Estimation, she must send data to Bob approximately equal to the total number of sacrificed bits. For most buffer sizes this should be enough to fill at least one MTU (Maximum Transmission Unit) worth of data, which is 1500 bytes under the TCP/IP protocol.
$BA\_avg\_msg\_size$	bytes	250	Bob is only required to send back to Alice the estimated error rate once he compares her sacrificed bits to his own. This action may be accomplished in 250 bytes or less. This action should only require a single message worth of data.
AB_num_trans	unitless	10	Assuming Alice initiates Error Estimation, she must send data to Bob approximately equal to the total number of sacrificed bits. The number of transactions required can be estimated by dividing the number of sacrificed bits by the average AB message size.
BA_num_trans	unitless	1	Bob is only required to send back to Alice the estimated error rate once he compares her sacrificed bits to his own. This action should only require a single message worth of data.

Error Reconciliation

Table 20. Input Parameters Local to Error Reconciliation

Parameter Name	Units	Value	Justification
$num\_bits\_sacrificed$	bytes	0	No bits are sacrificed while using the LDPC error correction algorithm.
$block\_size$	bits	54000	For LDPC this size is dictated by a pre-determined standard matrix. The ETSI DVB matrix, for instance, was selected for its reasonable 5/6 coding rate and requires a 54000 bit block [20].
$AB\_avg\_msg\_size$	bytes	1200	Alice is required to send Bob a parity syndrome of size S=N(1-R), where S is the size of the syndrome, N is the block size, and R is the coding rate of the LDPC matrix. For the ETSI DVB matrix, for example, the code rate is 5/6. In this instance, the size of the syndrome totals 9000 bits, which is approximately 1200 bytes of information [38].
$BA\_avg\_msg\_size$	bytes	750	Bob is required to communicate information back to Alice on how to fix the errors in her bit buffer. This amount of data required to communicate this information will be no greater than the size of the initial syndrome she sent to Bob.
AB_num_trans	unitless	1	For LDPC, the number of transactions is limited to 1 from Alice→Bob and 1 from Bob→Alice for each block (the number of total executions varies depending on the input buffer).
BA_num_trans	unitless	1	For LDPC, the number of transactions is limited to 1 from Alice→Bob and 1 from Bob→Alice for each block (the number of total executions varies depending on the input buffer).

Entropy Estimation

Table 21. Input Parameters Local to Entropy Estimation

Parameter Name	Units	Value	Justification
$pct\_entropy\_loss\_QBER$	%	12	The percentage of entropy loss on the Quantum Channel was calculated using the Shannon limit for minimum entropy loss. For more information, see Chapter IV discussion on Entropy Estimation.
$pct\_entropy\_loss\_public$	%	12	The percentage of entropy loss on the Classical Channel during Error Reconciliation was calculated using the Shannon limit for minimum entropy loss. For more information, see Chapter IV discussion on Entropy Estimation.
$pct\_entropy\_loss\_multi\_photon$	%	12	The percentage of entropy loss due to multi-photon pulses was calculated using the cumulative Poisson probability of multi-photon pulses. We assume all multi-photon pulses will be eliminated from the final key buffer in order to defend against Photon-Number Splitting (PNS) attacks. For more information, see Chapter IV discussion on Entropy Estimation.
$pct\_entropy\_loss\_safety$	%	4	The percentage of arbitrary safety margin was selected to account for any unaccounted for entropy loss. For more information, see Chapter IV discussion on Entropy Estimation.
$AB\_avg\_msg\_size$	bytes	0	Alice and Bob independently possess the information required to calculate the entropy loss, given that they are aware of the entropy loss calculation procedure before QKD execution begins. It is therefore not necessary for Alice and Bob to communicate during this phase. However, it is a practitioner’s design choice whether or not Alice and Bob should communicate to ensure matching entropy estimates.
$BA\_avg\_msg\_size$	bytes	0	Alice and Bob independently possess the information required to calculate the entropy loss, given that they are aware of the entropy loss calculation procedure before QKD execution begins. It is therefore not necessary for Alice and Bob to communicate during this phase. However, it is a practitioner’s design choice whether or not Alice and Bob should communicate to ensure matching entropy estimates.

AB_num_trans	unitless	0	Alice and Bob independently possess the information required to calculate the entropy loss, given that they are aware of the entropy loss calculation procedure before QKD execution begins. It is therefore not necessary for Alice and Bob to communicate during this phase. However, it is a practitioner's design choice whether or not Alice and Bob should communicate to ensure matching entropy estimates.
BA_num_trans	unitless	0	Alice and Bob independently possess the information required to calculate the entropy loss, given that they are aware of the entropy loss calculation procedure before QKD execution begins. It is therefore not necessary for Alice and Bob to communicate during this phase. However, it is a practitioner's design choice whether or not Alice and Bob should communicate to ensure matching entropy estimates.

Privacy Amplification

Table 22. Input Parameters Local to Privacy Amplification

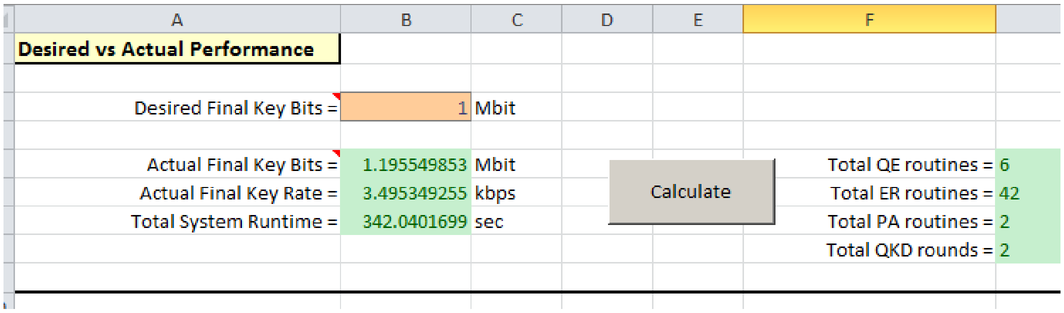
Parameter Name	Units	Value	Justification
$min\_num\_req\_bits$	bits	1000000	To reduce finite key size effects, it is suggested that the input to Privacy Amplification be between 1-100 Mbits [38].
$AB\_avg\_msg\_size$	bytes	1500	Alice and Bob must communicate a seed to prime random number generation to construct a very large matrix used in the calculation of Privacy Amplification. We assume that Alice initiates this communication, selects the seed, and communicates it to Bob. The message(s) she sends to Bob will therefore most likely be larger than the acknowledgment received from Bob.
$BA\_avg\_msg\_size$	bytes	500	Alice and Bob must communicate a seed to prime random number generation to construct a very large matrix used in the calculation of Privacy Amplification. We assume that Alice initiates this communication, selects the seed, and communicates it to Bob. The message(s) she sends to Bob will therefore most likely be larger than the acknowledgment received from Bob.
AB_num_trans	unitless	1	The random seed Alice sends to Bob can be sent to Bob in the amount of data contained in one message.
BA_num_trans	unitless	1	Bob's reply is relatively small and can be contained in one message.

Final Key Generation

Table 23. Input Parameters Local to Final Key Generation

Parameter Name	Units	Value	Justification
$num\_auth\_reservoir\_bits$	bits	10000	We assume Alice and Bob reserve enough bits for Authentication to refill the entire size of the Authentication reservoir buffer, as defined in the Authentication phase.
$AB\_avg\_msg\_size$	bytes	64	We assume that a standard cryptographic hash, such as SHA-512, is being used to complete Final Key Generation. If Alice initiates this hash comparison, she will need to send a message to Bob of at least 512 bits (the output of the hash function).
$BA\_avg\_msg\_size$	bytes	50	After computing the same hash as Alice, Bob is required to confirm that his hash value is identical to Alice by sending her a response message.
AB_num_trans	unitless	1	Alice computes a cryptographic hash, which can fit inside one message, and sends it to Bob.
BA_num_trans	unitless	1	Bob receives the hash from Alice and replies with a confirmation that his calculated hash matches hers.

After establishing reasonable input values for the phase calculations, we can observe the final performance metrics of the configuration by selecting a desired number of final key bits, as shown in the figure below:



A	B	C	D	E	F	
Desired vs Actual Performance						
Desired Final Key Bits =	1	Mbit				
Actual Final Key Bits =	1.195549853	Mbit			Total QE routines =	6
Actual Final Key Rate =	3.495349255	kbps			Total ER routines =	42
Total System Runtime =	342.0401699	sec			Total PA routines =	2
					Total QKD rounds =	2

Figure 29. Readout of Model Performance Metrics

The interpretation of performance in terms of number of QE, ER, and PA routines assumes the execution of Quantum Exchange will continue until Alice's QE memory buffer is filled, ER will run as many times as necessary to process the input as a result of QE and subsequent phases, and PA will only execute if there are enough key bits to meet the minimum requirement (i.e., one million bits). If there are not enough key bits, QE will restart as many times as necessary to meet PA requirements. The flow of this process can be seen in the following figure:

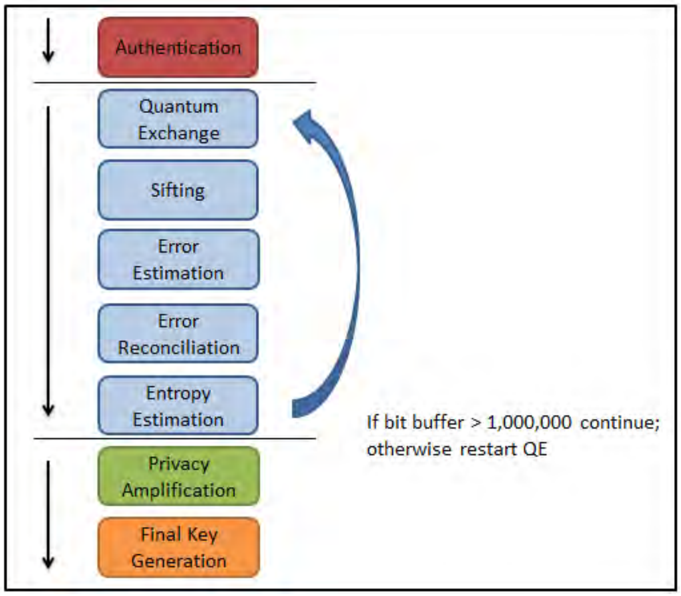


Figure 30. Execution Flow of Practical QKD System

5.4 Use Case: Answering Fundamental Performance Questions

5.4.1 Performance Metrics

In the design of this model, the user is required to enter a desired amount of final key in order to calculate performance metrics. The following tests were performed with 1 Mbit of desired final key using parameters from the Chen configuration and under the assumptions described in Section 5.1. The table below shows the results of increasing the percentage of memory allocated to Alice for Quantum Exchange out of a total of 4GB, incremented from 1 to 50% of her total memory.

Table 24. Table of Performance Metrics Produced By Model

Alice's QE Memory (% of 4GB)	Alice's QE Memory (GB)	Final Key Bits (Mbit)	Final Key Rate (kbps)	System Runtime (sec)	Total QE Routines	Total ER Routines	Total PA Routines	Total QKD Rounds
1	0.04	1.19	2.38	501	120	120	2	2
2	0.08	1.19	3.13	381	60	60	2	2
3	0.12	1.19	3.5	340	40	40	2	2

4	0.16	1.19	3.13	380	30	60	2	2
5	0.2	1.19	3.34	356	24	48	2	2
6	0.24	1.19	3.50	340	20	40	2	2
7	0.28	1.25	3.28	382	18	54	2	2
8	0.32	1.27	3.4	375	16	48	2	2
9	0.36	1.25	3.5	358	14	42	2	2
10	0.4	1.19	3.34	356	12	48	2	2
11	0.44	1.31	3.43	383	12	48	2	2
12	0.48	1.19	3.5	340	10	40	2	2
13	0.52	1.29	3.38	382	10	50	2	2
14	0.56	1.39	3.45	404	10	50	2	2
15	0.6	1.19	3.5	340	8	40	2	2
16	0.64	1.27	3.4	374	8	48	2	2
17	0.68	1.35	3.46	391	8	48	2	2
18	0.72	1.43	3.51	409	8	48	2	2
19	0.76	1.51	3.52	426	8	48	2	2
20	0.8	1.19	3.46	344	6	42	2	2
21	0.84	1.25	3.5	358	6	42	2	2
22	0.88	1.31	3.54	371	6	42	2	2
23	0.92	1.37	3.47	396	6	48	2	2
24	0.96	1.43	3.5	409	6	48	2	2
25	1.0	1.49	3.55	422	6	48	2	2
26	1.04	1.56	3.48	447	6	54	2	2
27	1.08	1.62	3.52	460	6	54	2	2
28	1.12	1.68	3.55	473	6	54	2	2
29	1.16	1.74	3.49	498	6	60	2	2
30	1.2	1.19	3.5	340	4	40	2	2
31	1.24	1.23	3.53	349	4	40	2	2
32	1.28	1.27	3.48	366	4	44	2	2
33	1.32	1.31	3.5	375	4	44	2	2
34	1.36	1.35	3.53	383	4	44	2	2
35	1.4	1.39	3.56	392	4	44	2	2
36	1.44	1.43	3.51	409	4	48	2	2
37	1.48	1.47	3.53	417	4	48	2	2
38	1.52	1.51	3.56	426	4	48	2	2
39	1.56	1.56	3.51	443	4	52	2	2
40	1.6	1.6	3.54	451	4	52	2	2
41	1.64	1.64	3.56	460	4	52	2	2
42	1.68	1.68	3.52	477	4	56	2	2
43	1.72	1.72	3.54	486	4	56	2	2
44	1.76	1.76	3.56	494	4	56	2	2
45	1.8	1.8	3.52	511	4	60	2	2
46	1.84	1.84	3.54	520	4	60	2	2
47	1.88	1.88	3.56	528	4	60	2	2
48	1.92	1.92	3.52	545	4	64	2	2
49	1.96	1.96	3.54	554	4	64	2	2
50	2.0	1.00	3.56	281	2	32	1	1

The table reveals the cyclical nature of achieving a desired number of final key bits. The model assumes that Quantum Exchange will run until all of Alice's memory allocated for QE has been filled. This assumption presents the possibility of a scenario where even if the amount of final key already generated is very close to the desired final key bits, Quantum Exchange will be forced to run again until Alice fills her memory, resulting in a final number of bits in excess of the desired number of bits.

This effect is magnified by the presence of a minimum number of required bits to perform Privacy Amplification (in this case 1 Mbit). For example, when 49% of Alice's memory is allocated for QE, it results in 1.96 Mbit of final key being generated and requires 4 iterations of QE. At 50% memory allocation, exactly 1 Mbit of key is generated with only 2 iterations of QE. This scenario is the result of having to perform QE enough times to perform PA more than once, as opposed to having to perform PA only once.

The following graph illustrates the need for an increased number of ER routines until the amount of allocated memory reaches a threshold such that fewer QE routines are required to attain the desired number of final key bits. We can see the most radical drop-off at 50% where both fewer QE and PA routines are necessary to achieve the desired number of final key bits, which consequently results in fewer required ER routines.

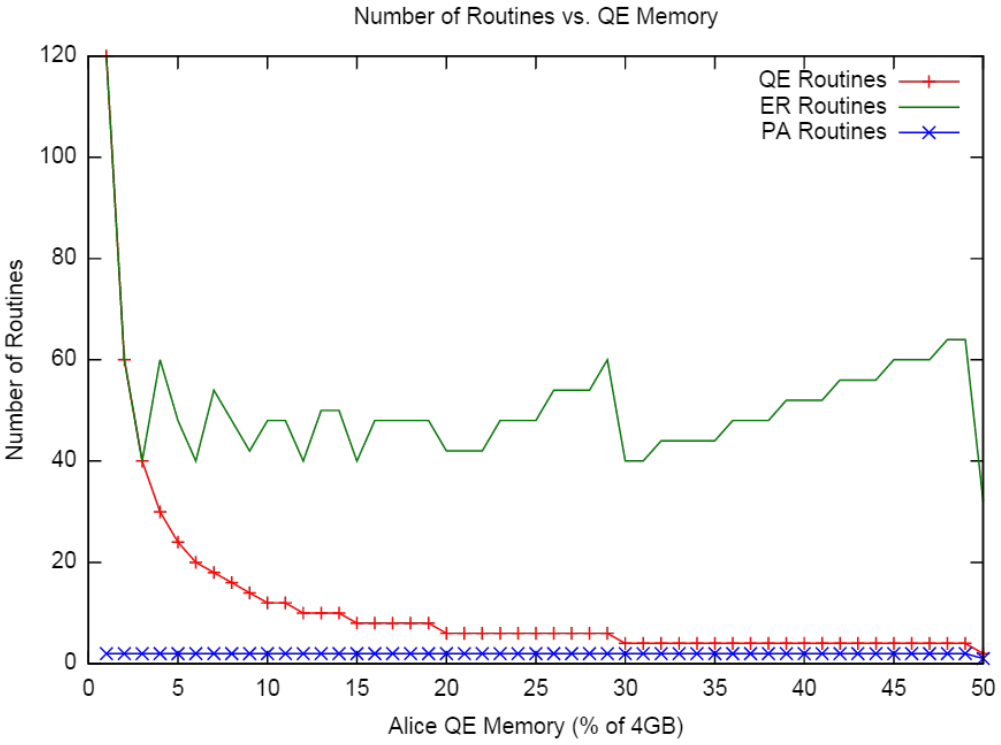


Figure 31. Graph of Routines Required to Achieve 1 Mbit Final Key

5.4.2 Bounds on Final Key Rate

The Chen configuration advertises a sifted key rate of approximately 10.5 kbps [18]. Given the BB84 protocol was used in this configuration, we can assume that the protocol efficiency fraction (applied to sifting) is approximately 0.5. In other words half of Bob's detection rate should result in a reasonable estimation of the sifted key rate. Similarly, doubling the sifted key rate should result in a reasonable estimate for Bob's detection rate. We should expect a detection rate around 21,000 detections per second at Bob in order to achieve a sifted key rate of 10.5 kbps in the Chen configuration.

The detection rate in the model is 20,964 detections per second, which is just under the 21,000 we expected to see for Chen. Therefore we can assume this model provides a reasonable representation of the Chen detection rate. If we apply similar logic from the derivation of the sifted key rate to the losses incurred during other phases of the system, we can establish an upper bound on the final key rate that can be achieved based on the initial detection rate. The following table shows the reduction in potential final key rate as a result of imposed configuration losses incurred during Sifting, Error Estimation, and Privacy Amplification based on a 20,964 det/sec detection rate (for descriptions of incurred loss refer to Section 5.3):

Table 25. Theoretical Bounded Key Rates During Phases With Loss

Phase	**Losses Incurred**	**Resultant Key Rate (kbps)**
Sifting	50%	10.482
Error Estimation	25%	7.861
Privacy Amplification	40%	4.716

Considering there are no major losses incurred during the other phases and barring the inclusion of any communications and/or processing overhead, 4.716 kbps is the theoretical upper bound on the performance of the configuration implemented in the model.

We can now evaluate the final key rate performance of the entire system and compare it to the upper bound. The following graph compares the final key rate against the percentage of Quantum Exchange memory allocated for Alice while attempting to generate 1 Mbit of desired final key (the data contained in this graph can be viewed in Table 23):

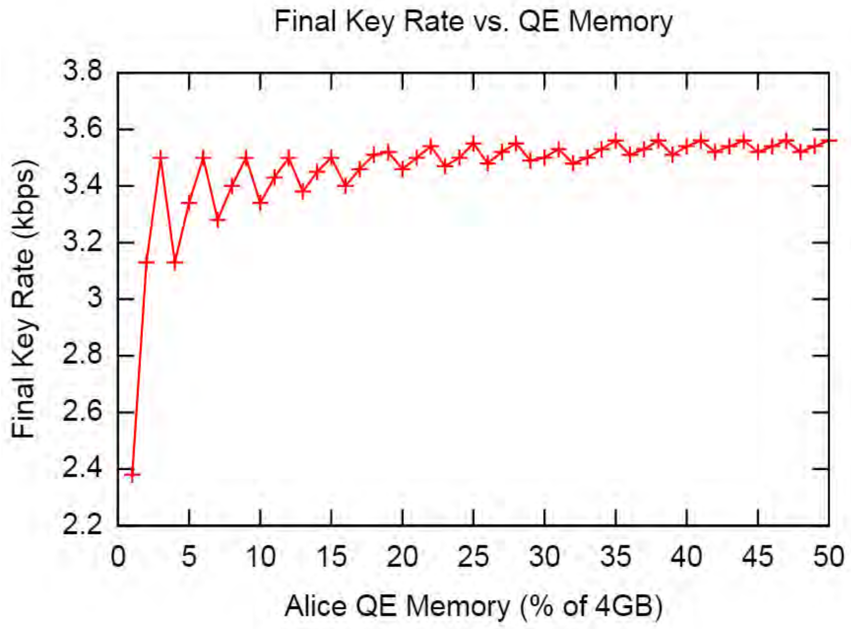


Figure 32. Graph of Alice Memory Effect on Final Key Rate

As the amount of allocated memory increases, the final key rate appears to emulate a sigmoid curve function with an approximate ceiling of 3.6 kbps. We also observe the cyclical fluctuations of the key rate due to the inability to generate exactly 1 Mbit of desired key. It appears that an arbitrarily large amount of memory allocated to Alice will only have a marginal performance benefit as it approaches a limit on performance. To get closer to the asymptotic bound on final key rate, performance in other areas of the system will most likely need to be improved.

5.4.3 Increased Processing Power

In this model we assume that both Alice and Bob have identical processing power. Additionally, we assume the processor is fully available to each phase (i.e., does not suffer from decreased performance due to the existence of concurrent processes). We initially set Alice and Bob's processing power to one billion work units per second. The following table shows the effects on final key rate and system runtime by increasing the power of the CPU from 1 to 16 billion units of work per second while the rest of the system configuration remained unchanged:

Table 26. Table of Performance Metrics Compared to CPU Power

Alice/Bob CPU Power (work units/sec)	Final Key Rate (kbps)	System Runtime (sec)
1.00E+09	3.46	344.97
2.00E+09	3.94	302.96
3.00E+09	4.13	288.96
4.00E+09	4.24	281.96
5.00E+09	4.30	277.76
6.00E+09	4.34	274.96
7.00E+09	4.37	272.96
8.00E+09	4.40	271.46
9.00E+09	4.42	270.29
10.00E+9	4.43	269.36
11.00E+9	4.45	268.60
12.00E+9	4.46	267.96
13.00E+9	4.47	267.42
14.00E+9	4.47	266.96
15.00E+9	4.48	266.56
16.00E+9	4.49	266.21

The following graph shows the effect of increased processing power on the final key rate:

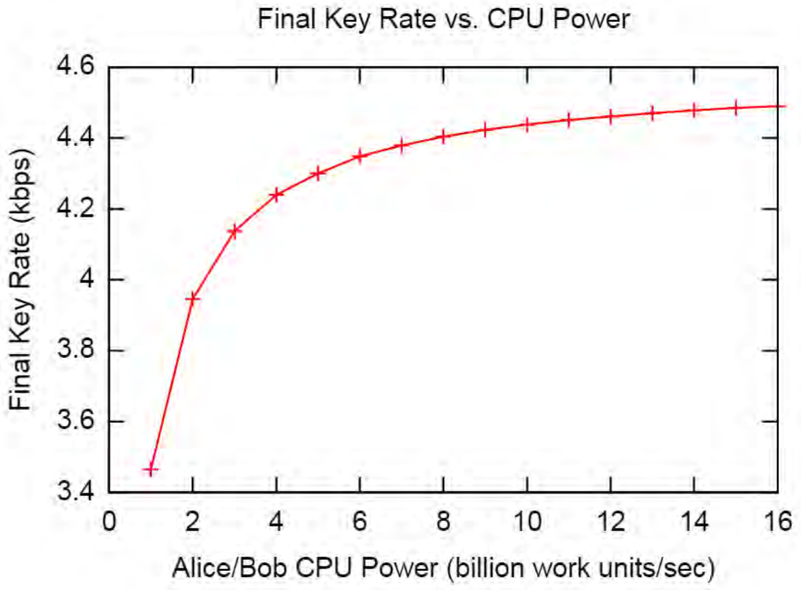


Figure 33. Graph Comparing Final Key Rate to CPU Power

It is clear that processing power and final key rate share an approximately logarithmic relationship. The final key rate quickly approaches the theoretical 4.716 kbps asymptotic bound, but exhibits diminishing returns on performance above 8 billion work units per second. This phenomenon can perhaps best be described by Amdahl’s law on system design [44], which states that system performance will not be indefinitely improved by increasing the performance of one component in the system. An arbitrarily large increase in CPU power will never exceed the asymptotic bound on final key rate. In order to increase the final throughput, the performance bound must be increased. Increasing the performance of one component moves the bottleneck to another component. As we have seen, the performance bound is dictated by the detection rate at Bob, which is ultimately dictated by the pulse rate at Alice, loses on the quantum channel, and the dead time of Bob’s detectors.

The following graph shows the effect of increased processing power on the system runtime:

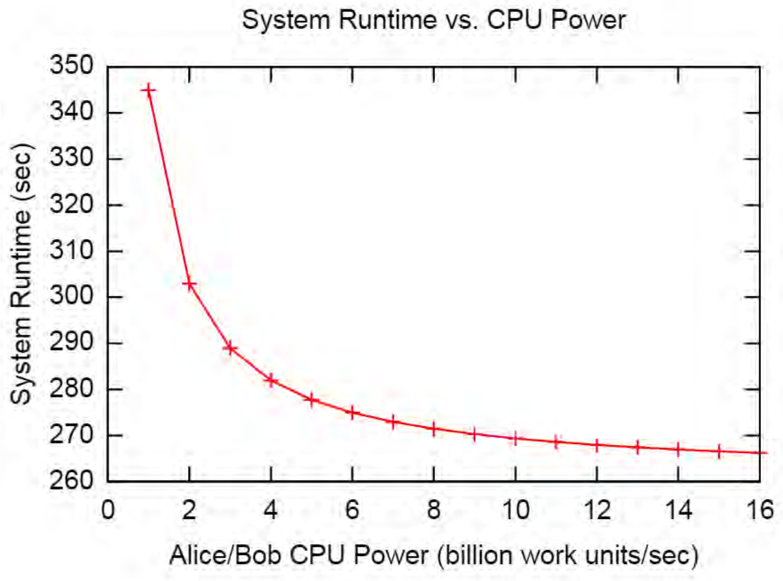


Figure 34. Graph Comparing System Runtime to CPU Power

There appears to be an exponential decrease in the system runtime as processing power is increased. Therefore system runtime and final key rate share an inverse relationship with respect to the processing power of the system. The decrease in runtime exhibits similar behavior to the final key rate – an arbitrarily large increase in CPU power will experience diminishing returns in the total execution time of the system.

VI. Conclusions and Future Work

6.1 Research Relevance

Research in the field of Quantum Key Distribution performance has historically been achieved by the alleviation of one or more bottlenecks in the hopes of increasing final key throughput. These bottlenecks occur at specific chokepoints in the system, such as Bob's detectors or the resource intensive post-processing algorithms. The best way to gauge system throughput performance, aside from observed results, has been the use of various forms of a secure key rate equation. This research models other aspects of QKD systems not captured by these formulas, such as system memory, CPU power, classical communications, and time.

The model presented in this thesis represents the performance characteristics of both the quantum and classical processing of QKD, and is tailorable to implementation-specific systems. The goal was to build a model for QKD practitioners that provides more accurate predictions of throughput performance in the design of systems, and a heuristic method of finding optimality in systems that are already deployed, while at the same time educate the practitioner on the abundance of major system dependencies and trade-offs in design that may be detrimental to key throughput.

6.2 Answering Research Questions

RQ1: What are the parameters that define a Quantum Key Distribution system?

In Chapter IV Section 2, we characterized QKD in terms of several components: Alice, Bob, the classical channel, and the quantum channel. Thus, the parameters that define these components will define the system. A distinction is made between the attributes local to each of these components in a one-way QKD system (i.e., defined as Alice being the sender of photons and Bob the receiver of photons). Alice and Bob both share a total amount of system memory and computational power, and the classical and quantum channels both share a finite distance between Alice and Bob, in addition to a propagation delay per unit of distance. All other parameters are local to each component and describe its behavior.

Since Alice must send photons, she is defined by a pulse rate, a Mean Photon Number, and a percentage of signal states. As the receiver of photons, Bob is defined by loss, the efficiency of his detectors, and the dead time of the detectors. The classical channel is also defined by the total bandwidth of the channel, whereas the quantum channel is defined by a loss associated with transmission.

RQ2: What phases exist in all Quantum Key Distribution systems?

The model presented in Chapter IV was derived from a review of the system processes outlined in QKD literature. It consists of a series of independently modeled phases, which provide an aggregate performance prediction of total final bits, the total system execution time, as well as execution time bit buffer sizes for each intermediate phase. The phases considered in this model include:

1. *Authentication*
2. *Quantum Exchange*
3. *Sifting*
4. *Error Estimation*
5. *Error Reconciliation*
6. *Entropy Estimation*
7. *Privacy Amplification*
8. *Final Key Generation*

RQ3: What are the necessary input and output parameters to define a phase?

System performance in QKD is generally measured in final key bits per second. Thus, the necessary inputs and outputs to each phase must, at minimum, address the issue of key bits and time. The inputs and outputs required to track key bits throughout system execution are the key buffer as an input from the previous phase, and the amount of key that is lost as a result of executing the current phase. The resulting size of the output key buffer will either be the same size as the input buffer (if no bits are lost or sacrificed during the phase) or smaller than the input buffer as a result of lost or sacrificed bits.

The notion of time to execute a phase has two primary components: computational time and transmission time. If the amount of the work required for each phase is determined prior to execution and Alice/Bob possess a computational power that is also defined prior to execution, then the computational time required is known. This time requirement is addressed in the Controls & Mechanisms aspect of the ICOM modeling, considering these predetermined aspects of time are not inputs to the phase itself. The transmission time is defined as inputs to each phase in the form of average message size and total number of transactions that must be passed back and forth between Alice and Bob. The total time of transmission is then dictated by the transmission time equation as a function of message size, number of transactions, and bandwidth of the classical channel.

RQ4: Can the notion of system time and performance be measured at the phase level?

In the answer to RQ3 we established that the primary factor in determining necessary inputs and outputs to a phase is final key throughput, which consists of both time and bit buffer components. In order to calculate the time required to execute QKD, it is necessary to know the execution time of each phase. The total system time can then be determined by the summation of phase time, where each phase is executed an independent number of times (i.e., QE may occur more times than PA). Similarly, computational performance of the phase is predetermined at the phase level based on an assigned measure of work for the phase and the computational power of Alice/Bob. The aggregate performance is then determined by number of times a particular phase is executed.

RQ5: Can a system-level throughput model be developed that incorporates time, system memory, computational power, and the speed of classical communication?

Yes. If computational power, system memory, and the speed of classical communication are dictated by the definition of the system itself (i.e., system characterization in Chapter IV), then these metrics can be used to calculate the time required to complete each of the eight phases of a QKD system, defined in Chapter III. The total execution time of the system can then be determined by the number of times each phase is required to execute.

RQ6: If so, how can it be used to answer fundamental performance questions of QKD such as, "How many Quantum Exchange, Error Reconciliation, and Privacy Amplification routines are necessary to achieve a desired amount final key?"

The answer to fundamental performance questions relies upon a definition of what constitutes a single QKD round. In Chapter V, we defined a serial implementation of QKD, in which a single round consisted of executing Quantum Exchange, Sifting, Error Estimation, Error Reconciliation, and Entropy Estimation multiple times in order to achieve a minimum desired key buffer size as an input to Privacy Amplification. Since we defined the style of execution, we can determine how many rounds of phase will be required to generate a desired number of final key bits.

RQ7: What are the implications of altering the amount of Alice's memory allocated for Quantum Exchange?

As Alice's memory is increased, the number of Quantum Exchange routines will decrease and the number of Error Reconciliation routines will increase proportional to QE. The number of Privacy Amplification routines will remain relatively constant since it cannot execute until a minimum threshold is met. The final key rate also increases as Alice's QE memory increases, but only to an extent. There is an asymptotic upper bound on final key rate that is dictated by the losses imposed on the system at various phases. The final key rate reaches a logistical ceiling below the asymptotic bound and does not increase indefinitely as QE memory is increased.

RQ8: What are the implications of altering computational power for Alice and Bob?

In the answer to RQ7, we recognized that there is an asymptotic upper bound on final key rate in a QKD system, and that increasing the memory allocated to Alice for Quantum Exchange will cause the final key rate to find an intermediate upper bound. Increasing the computational power allocated to Alice and Bob will cause a logarithmic increase final key rate as it quickly approaches the asymptotic upper bound. It is considered asymptotic because regardless of how much computational power is given to both Alice and Bob, the bound will never be reached because the time required for classical communication cannot be eliminated completely. Additionally, the time required for system execution experiences an exponential decrease as computational power is increased.

RQ9: How can this model be used to study Quantum Key Distribution systems?

The answers to research questions 6, 7, and 8 demonstrate several examples of studies that can be performed using the model. In general, the outputs of the model are performance metrics of QKD execution for a particular configuration. Therefore in order to study QKD systems, either alterations to the definition of QKD execution or the configuration itself must be altered to observe new behavior.

6.3 Contributions and Future Work

This research effort began as an investigation into optimal execution patterns for Quantum Key Distribution systems. QKD literature describes the execution of the phases in a serial manner, but there may be more optimal solutions in the form of pipelining, concurrency, or parallelization. It became apparent that the mechanism to study alternative execution patterns did not exist, and this thesis set out to create those mechanisms. As a result, the throughput model that resulted from this research suggests that increasing the size of system memory (in excess of the memory necessary to run the system) only marginally improves performance. A better return on investment may instead be realized by increasing the CPU power of Alice and Bob, although even then the improvement in system performance exhibits the behavior of Amdahl’s law. It appears the most effective way to achieve the highest amount of throughput is to raise the asymptotic bound on final throughput that is dictated by the detection rate at Bob. It is the authoritative speed limit on the rest of the QKD system.

The major contribution of this research includes a mathematical model used to assess the performance of QKD systems. It differs from current interpretations in literature by incorporating post-processing performance characteristics as well as memory constraints that expand upon the protocol-centric focus in evaluating final throughput. This allows for practitioners to make more educated decisions in the design of their systems as well as search for optimization of parameters or design choices in currently implemented systems. In addition, understanding the trade-offs and dependencies that exist within QKD allows designers to achieve a desired throughput rate around constraints, such as the need to use a particular detector or laser source.

Future work in this area could include finding optimality or trade-offs in alternative execution styles, such as continuous Quantum Exchange, ratios of executions between Sifting and Error Correction to Privacy Amplification as it relates to final key throughput, and optimal buffer sizes going into each successive phase. Additionally, the use of Amdahl’s law could be used to quantify the bounds on optimality at the phase level, and how it relates to the theoretical upper bound on throughput if the phases are viewed as “threads” in a parallel system.

Appendix A: Equations

System Time

The total system runtime can be described as:

$$T_{sys} = \sum_{i=0}^{n} \sum_{j=1}^{m} T_{i,j}$$

where *i* is the total time through phase *i*, based on m iterations required for that phase, where T_{Auth}→ T_{FKG} are represented numerically as *i* = 0, 1, 2, …7.

Transmission Time

$$transmission\_time(msg_{size}, bandwidth, num_{trans}) = \frac{msg_{size(bits)}}{bandwidth\left(\frac{bits}{sec}\right)} \cdot num\_trans + t_{class\_prop\_delay}(sec)$$

$$time\_btwn\_sent\_msg = \frac{msg\_size(bits)}{bandwidth\left(\frac{bits}{s}\right)}$$

Authentication, *i* = 0

$$T_{Auth} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}\,, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{Auth}}}{\{A,B\}_{cpu_{power}}}$$

$$auth\_reservoir\_remaining_{size} = (auth\_reservoir_{size} - auth\_key\_required)$$

Quantum Exchange, *i* = 1

Note: In the case of QE, we assume the end of the phase is signaled by the entire free memory buffer being filled.

$$T_{QE} = max\left(\frac{num\_det\_at\_Bob}{actual\_det\_rate} + t_{quant\_prop\_delay}\,,\ \frac{\{A,B\}_{workload_{QE}}}{\{A,B\}_{cpu_{power}}}\right)$$

$$Pois(X \geq 1) = 1 - e^{-\mu}$$

$$num\_pulses\_sent = \frac{A_{mem\_avail}}{mem\_req\_per\_pulse}$$

$$num\_det\_at\_Bob = num\_pulses\_sent \cdot Pois(X \geq 1) \cdot sig_{percent} \cdot \eta_{channel} \cdot \eta_{Bob} \cdot \eta_{det}$$

$$t_{time\_btwn\_pulse\_arrival} = \frac{1}{pulse\_rate \cdot Pois(X \geq 1) \cdot sig_{percent} \cdot \eta_{channel} \cdot \eta_{Bob} \cdot \eta_{det}}$$

$$t_{avg\_time\_btwn\_detection} = ceil\left(\frac{t_{dead}}{t_{time\_btwn\_pulse\_arrival}}\right) \cdot t_{time\_btwn\_pulse\_arrival}$$

$$actual\_det\_rate = \frac{1}{t_{avg\_time\_btwn\_detection}}$$

$$A_{raw_{buffer}} = num\_pulses\_sent \cdot mem\_req\_pulse$$

$$B_{raw_{buffer}} = num\_det\_at\_Bob \cdot mem\_req\_pulse$$

$$A_{QE\_candidate\_key\_bits} = num\_pulses\_sent$$

$$B_{QE\_candidate\_key\_bits} = num\_det\_Bob$$

Sifting, *i* = 2

$$T_{Sift} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{Sift}}}{\{A,B\}_{cpu_{power}}}$$

$$B_{sift_{buffer}} = sift\_eff\_frac \cdot \frac{B_{raw_{buffer}}}{mem\_req\_pulse}$$

$$A_{sift_{buffer}} = B_{sift_{buffer}}$$

Error Estimation, *i* = 3

$$T_{ErrEst} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{ErrEst}}}{\{A,B\}_{cpu_{power}}}$$

$$\{A,B\}_{ErrEst_{buffer}} = (1 - bits\_sacrificed_{pct}) \cdot \{A,B\}_{Sift_{buffer}}$$

Error Reconciliation, *i* = 4

$$T_{ErrRec} = ceil\left(\frac{\{A,B\}_{ErrEst_{buffer}}}{block_{size}}\right) \cdot \left(transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{ErrRec}}}{\{A,B\}_{cpu_{power}}}\right)$$

$$\{A,B\}_{ErrRec_{buffer}} = \{A,B\}_{ErrEst_{buffer}} - (num\_bits\_sacrificed \cdot num\_err\_rec)$$

$$num\_err\_rec = ceil\left(\frac{\{A,B\}_{ErrEst_{buffer}}}{block_{size}}\right)$$

Entropy Estimation, *i* = 5

$$T_{EntEst} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{EntEst}}}{\{A,B\}_{cpu_{power}}}$$

$$\{A,B\}_{EntEst_{buffer}} = \{A,B\}_{ErrRec_{buffer}}$$

$$N_{secure} = (1 - total\_ent\_loss\_pct) \cdot \{A,B\}_{ErrRec_{buffer}}$$

Privacy Amplification, *i* = 6

$$T_{PrivAmp} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{PrivAmp}}}{\{A,B\}_{cpu_{power}}}$$

$$\{A,B\}_{PrivAmp_{buffer}} = N_{secure}$$

Final Key Generation, *i* = 7

$$T_{FKG} = transmission\_time(avg\_msg\_size_{\{AB,BA\}}, bandwidth, num\_trans_{\{AB,BA\}}) + \frac{\{A,B\}_{workload_{FKG}}}{\{A,B\}_{cpu_{power}}}$$

$$\{A,B\}_{FKG_{buffer}} = \{A,B\}_{PrivAmp_{buffer}} - num\_auth\_reservoir\_bits$$

Appendix B: Model Code

The following code is associated with ‘Sheet1’ of the Excel model:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)

Dim num_bytes_in_gigabyte As Double
Dim num_bits_in_gigabyte As Double
Dim num_bits_in_byte As Double

Dim notional_memory_requirement_for_small_operations As Integer
Dim mem_required_per_ER_block As Integer

num_bytes_in_gigabyte = 1073741824
num_bits_in_gigabyte = 8589934592#
num_bits_in_byte = 8

mem_required_per_ER_block = 30 'bytes
notional_memory_requirement_for_small_operations = 1024 'bytes


'Enable events on current worksheet
Application.EnableEvents = False
Target.Activate
Application.EnableEvents = True


'Alice memory allocation check if memory is greater than 100%
If Range("C27").Value > "1" Then

    Cells(27, 3).Interior.ColorIndex = 3
    Cells(10, 10).Interior.ColorIndex = 3

    Else

        Cells(27, 3).Interior.Color = RGB(146, 208, 80)
        Cells(10, 10).Interior.Color = RGB(255, 204, 153)

    End If

'Bob memory allocation check if allocated memory is greater than 100%
If Range("F27").Value > "1" Then

    Cells(27, 6).Interior.ColorIndex = 3
    Cells(10, 14).Interior.ColorIndex = 3

    Else

        Cells(27, 6).Interior.Color = RGB(146, 208, 80)
        Cells(10, 14).Interior.Color = RGB(255, 204, 153)

    End If

'Auth memory configuration check
'Checks if allocated memory is greater than auth reservoir size plus some
notional amount of memory overhead
If (Range("D18").Value * num_bytes_in_gigabyte) >
Worksheets("Sheet2").Range("B15").Value +
notional_memory_requirement_for_small_operations And _
    (Range("G18").Value * num_bytes_in_gigabyte) >
```

```
Worksheets("Sheet2").Range("B15").Value +
notional_memory_requirement_for_small_operations Then

    Cells(18, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(18, 13).Interior.ColorIndex = 3

    End If

'Quantum Exchange memory configuration check
'Checks if allocated memory for Alice is greater than the size of memory
required to store information on a single pulse, the minimum memory required
for quantum exchange
'Checks if the ratio of allocated memory between Alice and Bob is at least the
ratio between sent pulses and detections
If (Range("D19").Value * num_bytes_in_gigabyte >
Worksheets("Sheet2").Range("B34").Value) And _
    (Range("D19").Value / Range("G19").Value <
Worksheets("Sheet2").Range("G41").Value /
Worksheets("Sheet2").Range("G46").Value) Then

    Cells(19, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(19, 13).Interior.ColorIndex = 3

    End If

'Sifting memory configuration check
'Checks if allocated memory is at least greater than sifting_eff_frac *
num_detections_at_bob
If (Range("D20").Value * num_bits_in_gigabyte >
Worksheets("Sheet2").Range("B78").Value *
Worksheets("Sheet2").Range("G46").Value) And _
    (Range("G20").Value * num_bits_in_gigabyte >
Worksheets("Sheet2").Range("B78").Value *
Worksheets("Sheet2").Range("G46").Value) Then

    Cells(20, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(20, 13).Interior.ColorIndex = 3

    End If

'Error Estimation memory configuration check
'Checks if memory allocated is at least greater than some notional amount of
memory overhead
If (Range("D21").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations) And _
    (Range("G21").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations) Then

    Cells(21, 13).Interior.Color = RGB(146, 208, 80)

    Else
```

```
        Cells(21, 13).Interior.ColorIndex = 3

    End If

'Error Reconciliation memory configuration check
'Checks if memory allocated is at least greater than the size of bit buffer
after experiencing sacrifice during Err Est + 30 bytes of computational memory
overhead * block_size
If (Range("D22").Value * num_bits_in_gigabyte >
Worksheets("Sheet2").Range("B102").Value *
Worksheets("Sheet2").Range("N84").Value + mem_required_per_ER_block *
num_bits_in_byte * Worksheets("Sheet2").Range("B125").Value) And _
    (Range("G22").Value * num_bits_in_gigabyte >
Worksheets("Sheet2").Range("B102").Value *
Worksheets("Sheet2").Range("N84").Value + num_mem_bytes_required_per_ER_block *
num_bits_in_byte * Worksheets("Sheet2").Range("B125").Value) Then

    Cells(22, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(22, 13).Interior.ColorIndex = 3

    End If

'Entropy Estimation memory configuration check
'Checks if memory allocated is at least greater than some notional amount of
memory overhead
If (Range("D23").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations) And _
    (Range("G23").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations) Then

    Cells(23, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(23, 13).Interior.ColorIndex = 3

    End If

'Privacy Amplification memory configuration check
'Checks if memory allocated is at least greater than twice the size of the
input into PA
'The matrix used for PA is approx the same size as the input key length into
PA, a smaller matrix results in a compression of the final key size (before
N_secure subtraction) - minmum compression (the ratio between the length of the
output and
'input keys, i.e. the ratio between the number of rows and columns of the
Toeplitz matrix) = 0%
If (Worksheets("Sheet2").Range("N158").Value >
Worksheets("Sheet2").Range("B179").Value) Then

    If (Range("D24").Value * num_bytes_in_gigabyte >
Worksheets("Sheet2").Range("N158").Value * 2) And _
        (Range("D24").Value * num_bytes_in_gigabyte >
Worksheets("Sheet2").Range("N158").Value * 2) Then

        Cells(24, 13).Interior.Color = RGB(146, 208, 80)

        Else
```

```
            Cells(24, 13).Interior.ColorIndex = 3

        End If

    Else

        If (Range("D24").Value * num_bytes_in_gigabyte >
Application.Ceiling(Worksheets("Sheet2").Range("B179").Value /
Worksheets("Sheet2").Range("N158").Value, 1) *
Worksheets("Sheet2").Range("N158").Value * 2) And _
            (Range("G24").Value * num_bytes_in_gigabyte >
Application.Ceiling(Worksheets("Sheet2").Range("B179").Value /
Worksheets("Sheet2").Range("N158").Value, 1) *
Worksheets("Sheet2").Range("N158").Value * 2) Then

           Cells(24, 13).Interior.Color = RGB(146, 208, 80)

        Else

            Cells(24, 13).Interior.ColorIndex = 3

        End If

    End If


'Final Key Generation memory configuration check
'Checks if memory allocated is at least greater than some notional amount of
memory overhead + SHA-512 requirements ~2kb
If (Range("D25").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations + 2048) And _
    (Range("G25").Value * num_bytes_in_gigabyte >
notional_memory_requirement_for_small_operations + 2048) Then

    Cells(25, 13).Interior.Color = RGB(146, 208, 80)

    Else

        Cells(25, 13).Interior.ColorIndex = 3

    End If

End Sub
```

The following code is associated with ‘Sheet2’ of the Excel model:

```
Private Sub CommandButton1_Click()
'This subroutine determines final performance metrics when the Calculate button
is clicked and the desired number of final bits is > 0

Dim num_bits_in_megabit As Double
Dim num_bits_in_kilobit As Double

num_bits_in_megabit = 1000000
num_bits_in_kilobit = 1000

'Determines final performance metrics
If Range("B3").Value > 0 Then
```

```
    Cells(3, 2).Interior.Color = RGB(255, 204, 153)

    If (Range("N158").Value > Range("B179").Value) Then

        'Actual final key bits
        Range("B5").Value = (Application.Ceiling((Range("B3").Value *
num_bits_in_megabit) / Range("N205").Value, 1) * Range("N205").Value) /
num_bits_in_megabit

        'Total PA routines
        Range("G7").Value = Application.Ceiling((Range("B3").Value *
num_bits_in_megabit) / Range("N205").Value, 1)

        'Total QKD rounds
        Range("G8").Value = Range("G7").Value

        'Total QE routines
        Range("G5").Value = Range("G8").Value

        'Total ER routines
        Range("G6").Value = Range("G5").Value * Range("L137")

        'Total System Runtime: add up all of phases that only occur once per
round, then calculate ER time
        Range("B7").Value = (Range("G8").Value * Range("L18").Value + _
                            Range("G8").Value * Range("L38").Value + _
                            Range("G8").Value * Range("L79").Value + _
                            Range("G8").Value * Range("L103").Value + _
                            Range("G8").Value * Range("L153").Value + _
                            Range("G8").Value * Range("L179").Value + _
                            Range("G8").Value * Range("L200").Value) + _
                            Range("G6").Value * Range("L125").Value

        'Actual final key rate
        Range("B6").Value = ((Range("B5").Value * num_bits_in_megabit) /
Range("B7").Value) / num_bits_in_kilobit

        Else

            Dim num_QE_to_reach_PA As Double
            Dim num_bits_end_FKG As Double

            num_QE_to_reach_PA = Application.Ceiling(Range("B179").Value /
Range("N158").Value, 1)

            num_bits_end_FKG = (num_QE_to_reach_PA * Range("N164").Value) -
Range("B199").Value

            'Actual final key bits
            Range("B5").Value = (Application.Ceiling((Range("B3").Value *
num_bits_in_megabit) / num_bits_end_FKG, 1) * num_bits_end_FKG) /
num_bits_in_megabit

            'Total PA routines
            Range("G7").Value = Application.Ceiling((Range("B3").Value *
num_bits_in_megabit) / num_bits_end_FKG, 1)

            'Total QKD rounds
            Range("G8").Value = Range("G7").Value

            'Total QE routines
            Range("G5").Value = num_QE_to_reach_PA * Range("G7").Value

            'Total ER routines
            Range("G6").Value = Range("G5").Value * Range("L137")

            'Total System Runtime: the only phases that occur once per round
are Auth, PA, and FKG; all other pahses run "Total QE" number of times
            Range("B7").Value = (Range("G8").Value * Range("L18").Value + _
                                Range("G8").Value * Range("L179").Value + _
                                Range("G8").Value * Range("L200").Value) + _
                                (Range("G5").Value * Range("L38").Value + _
                                Range("G5").Value * Range("L79").Value + _
                                Range("G5").Value * Range("L103").Value + _
                                Range("G5").Value * Range("L153").Value) + _
                                Range("G5").Value * Range("L125").Value

            'Actual final key rate
            Range("B6").Value = ((Range("B5").Value * num_bits_in_megabit) /
Range("B7").Value) / num_bits_in_kilobit

        End If

    Else

        Cells(3, 2).Interior.ColorIndex = 3

    End If

End Sub


Private Sub Worksheet_Change(ByVal Target As Excel.Range)

'Enable events on current worksheet
Application.EnableEvents = False
Target.Activate
Application.EnableEvents = True

'Checks if the total amount of entropy loss is greater than 100% of the key
If Range("G156").Value > "1" Then

    Cells(156, 7).Interior.ColorIndex = 3

    Else

        Cells(156, 7).Interior.Color = RGB(198, 239, 206)

    End If

End Sub
```

References

[1] P. Shor and J. Preskill, 'Simple Proof of Security of the BB84 Quantum Key Distribution Protocol',*Phys. Rev. Lett.*, vol. 85, no. 2, pp. 441-444, 2000.

[2] S. Wiesner, 'Conjugate coding', *SIGACT News*, vol. 15, no. 1, pp. 78-88, 1983.

[3] C. Bennett and G. Brassard, 'Quantum cryptography: Public key distribution and coin tossing',*Theoretical Computer Science*, vol. 560, pp. 7-11, 2014.

[4] V. Scarani, H. Bechmann-Pasquinucci, N. Cerf, M. Dušek, N. Lütkenhaus and M. Peev, 'The security of practical quantum key distribution', *Reviews of Modern Physics*, vol. 81, no. 3, pp. 1301-1350, 2009.

[5] J. Zhou, B. Liu, B. Zhao and B. Liu, 'A Pipeline Optimization Model for QKD Post-processing System', *Information and Communication Technology*, pp. 472-481, 2014.

[6] C. Elliott, D. Pearson and G. Troxel, 'Quantum cryptography in practice', *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, 2003.

[7] C. Kollmitzer and M. Pivk, 'Applied Quantum Cryptography', *Lecture Notes in Physics*, 2010.

[8] A. Abidin, *Authentication in Quantum Key Distribution: Security Proof and Universal Hash Functions*. Linköping: Linköping University Electronic Press, 2013, p. 55.

[9] R. Hadfield, 'Single-photon detectors for optical quantum information applications', *Nature Photonics*, vol. 3, no. 12, pp. 696-705, 2009.

[10] W. Buttler, S. Lamoreaux, J. Torgerson, G. Nickel, C. Donahue and C. Peterson, 'Fast, efficient error reconciliation for quantum cryptography', *Physical Review A*, vol. 67, no. 5, 2003.

[11] H. Yuen, 'Some physics and system issues in the security analysis of quantum key distribution protocols', *Quantum Inf Process*, vol. 13, no. 10, pp. 2241-2254, 2014.

[12] B. Slutsky, R. Rao, P. Sun, L. Tancevski and S. Fainman, 'Defense Frontier Analysis of Quantum Cryptographic Systems', *Applied Optics*, vol. 37, no. 14, p. 2869, 1998.

[13] C. Erven, C. Couteau, R. Laflamme and G. Weihs, 'Entangled quantum key distribution over two free-space optical links', *Opt. Express*, vol. 16, no. 21, p. 16840, 2008.

[14] C. Fung, X. Ma and H. Chau, 'Practical issues in quantum-key-distribution postprocessing',*Physical Review A*, vol. 81, no. 1, 2010.

[15] J. Carter and M. Wegman, 'Universal classes of hash functions', *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143-154, 1979.

[16] G. Van Assche, *Quantum cryptography and secret-key distillation*. Cambridge: Cambridge University Press, 2006.

[17] C. Bennett, G. Brassard and J. Robert, 'Privacy Amplification by Public Discussion', *SIAM J. Comput.*, vol. 17, no. 2, pp. 210-229, 1988.

[18] T. Chen, H. Liang, Y. Liu, W. Cai, L. Ju, W. Liu, J. Wang, H. Yin, K. Chen, Z. Chen, C. Peng and J. Pan, 'Field test of a practical secure communication network with decoy-state quantum cryptography', *Opt. Express*, vol. 17, no. 8, p. 6540, 2009.

[19] Mart Haitjema, “A Survey of the Prominent Quantum Key Distribution Protocols”, online. Available at http://www.cse.wustl.edu/~jain/cse571-07/ftp/quantum/

[20] Alan Mink and Anastase Nakassis,"LDPC for QKD Reconciliation", *The Computing Science and Technology International Journal*, Vol. 2, No. 2, June, 2012, ISSN (Print) 2162-0660, ISSN (Online) 2162-0687, (June, 2012).

[21] D.S. Pearson and C. Elliott, “On the optimal mean photon number for quantum cryptography,” Eprint quant-ph/0403065 (2004)

[22] M.A. Itzler, M. Entwistle, and X. Jiang, “High-rate photon counting with Geiger-mode APDs,” IEEE Photons Annual Meeting, S1. (2011).

[23] J.M. Myers, T. Wu and D. Pearson (2004), “Entropy estimates for individual attacks on the BB84 protocol for quantum key distribution,” Proceedings of the SPIE Defense and Security 2004, vol. 5105—Quantum Information and Computation, April 2004.

[24] D.R. Stinson, “Universal Hash Families and the Leftover Hash Lemma, and Applications to Cryptography and Computing,” Journal of Combinatorial Mathematics and Combinatorial Computing vol. 42, pp. 3–31, 2002.

[25] I. Mironov. Hash function: Theory, attacks and applications. [Online]. Available: http://research.microsoft.com/pubs/64588/hash_survey.pdf

[26] V. Scarani and C. Kurtsiefer, 'The black paper of quantum cryptography: Real implementation problems', *Theoretical Computer Science*, vol. 560, pp. 27-32, 2014.

[27] C. Bennett, F. Bessette, G. Brassard, L. Salvail and J. Smolin, 'Experimental quantum cryptography', *J. Cryptology*, vol. 5, no. 1, 1992.

[28] H. Singh, D. Gupta and A. Singh, 'Quantum Key Distribution Protocols: A Review', *IOSRJCE*, vol. 16, no. 2, pp. 01-09, 2014.

[29] N. Gisin, G. Ribordy, W. Tittel and H. Zbinden, 'Quantum cryptography', *Reviews of Modern Physics*, vol. 74, no. 1, pp. 145-195, 2002.

[30] P. Shor, 'Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer', *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484-1509, 1997.

[31] W. Wootters and W. Zurek, 'A single quantum cannot be cloned', *Nature*, vol. 299, no. 5886, pp. 802-803, 1982.

[32] E. Rieffel and W. Polak, 'An introduction to quantum computing for non-physicists', *CSUR*, vol. 32, no. 3, pp. 300-335, 2000.

[33] J. Townsend, *A modern approach to quantum mechanics*. Sausalito, Calif.: University Science Books, 2000.

[34] R. Rivest, A. Shamir and L. Adleman, 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[35] W. Diffie and M. Hellman, 'New directions in cryptography', *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.

[36] G. Singh and S. Supriya, 'A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security', *International Journal of Computer Applications*, vol. 67, no. 19, pp. 33-38, 2013.

[37] N. Gigov, 'Quantum Key Distribution Data Post-Processing with Limited Resources: Towards Satellite-Based Quantum Communication', PhD, University of Waterloo, 2013.

[38] A. Dixon and H. Sato, 'High speed and adaptable error correction for megabit/s rate quantum key distribution', *Scientific Reports*, vol. 4, p. 7275, 2014.

[39] Idef.com, 'IDEF0', 2015. [Online]. Available: http://www.idef.com/idef0.htm. [Accessed: 11- Jun- 2015].

[40] K. Sairam, *Optical communications*. New Delhi: Laxmi Publications, 2007.

[41] J. Cederlof and J. Larsson, 'Security Aspects of the Authentication Used in Quantum Cryptography',*IEEE Transactions on Information Theory*, vol. 54, no. 4, pp. 1735-1741, 2008.

[42] M. Wegman and J. Carter, 'New hash functions and their use in authentication and set equality',*Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 265-279, 1981.

[43] C. Shannon, 'A Mathematical Theory of Communication', *Bell System Technical Journal*, vol. 27, no. 3, pp. 379-423, 1948.

[44] G. Amdahl, 'Computer Architecture and Amdahl's Law', *Computer*, vol. 46, no. 12, pp. 38-46, 2

[45] L. Dostálek, A. Kabelová, A. Shirodkar and D. Parekh, *Understanding TCP/IP*. Birmingham, U.K.: Packt Pub., 2006.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE (DD–MM–YYYY) 17-09-2015	2. REPORT TYPE Master's Thesis	3. DATES COVERED (*From — To*) August 2013 – September 2015

4. TITLE AND SUBTITLE A System-Level Throughput Model for Quantum Key Distribution	5a. CONTRACT NUMBER
	5b. GRANT NUMBER 5713400-301-6448
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Cernera, Robert C., Civilian, USAF	5d. PROJECT NUMBER 15 ENV180-24
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-MS-15-S-069
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Laboratory for Telecommunication Sciences Dr. Gerry Baumgartner 8080 Greenmead Drive College Park MD 20740 gbaumgartner@ltsnet.net (240) 373-2743	10. SPONSOR/MONITOR'S ACRONYM(S) LTS
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Quantum Key Distribution (QKD) is an innovative technology which exploits the laws of quantum mechanics to generate and distribute shared secret keying material. QKD systems generate and distribute key by progressing through a number of distinct phases, typically in a serial manner. The purpose of this research is to identify these phases, their relationships to each other, as well as their relationship to time, memory space, computational requirements, and hardware resources. A mathematical model is developed which enables the study of critical system parameters, identifies and demonstrates potential bottlenecks that affect the overall key generation rate of serial implementations, and facilitates the analysis of design trade-offs in terms of parameters associated with specific implementations. Existing models of throughput performance make use of secure key rate equations which do not account for detailed system parameters and performance characteristics, particularly in the post-processing phases. In this research we build a model that is abstract enough to be applied to a wide range of QKD system configurations. The results of the model form an accurate prediction of throughput. The analysis contained herein provides QKD practitioners guidance in system analysis and design.

15. SUBJECT TERMS

Quantum Key Distribution; model; throughput; key rate; dependency; analysis; trade-offs

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Dr. Douglas D. Hodson
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	133	19b. TELEPHONE NUMBER (Include Area Code) (937) 255-3636 x4719; Douglas.Hodson@afit.edu